DeGAS: Gradient-Based Optimization of Probabilistic Programs without Sampling

Francesca Randone[1], Romina Doz[2], Mirco Tribastone[3], and Luca Bortolussi[2]

[1] TU Wien, Wien, Austria francesca.randone@tuwien.ac.at
[2] University of Trieste, Trieste, Italy
[3] IMT School for Advanced Studies Lucca, Lucca, Italy

Abstract. We present DeGAS, a differentiable Gaussian approximate semantics for loopless probabilistic programs that enables sample-free, gradient-based optimization in models with both continuous and discrete components. DeGAS evaluates programs under a Gaussian-mixture semantics and replaces measure-zero predicates and discrete branches with a vanishing smoothing, yielding closed-form expressions for posterior and path probabilities. We prove differentiability of these quantities with respect to program parameters, enabling end-to-end optimization via standard automatic differentiation, without Monte Carlo estimators. On thirteen benchmark programs, DeGAS achieves accuracy and runtime competitive with variational inference and MCMC. Importantly, it reliably tackles optimization problems where sampling-based baselines fail to converge due to conditioning involving continuous variables.

Keywords: Probabilistic Programming · Gaussian Mixtures · Differentiable Semantics · Sample-Free Optimization

1 Introduction

Probabilistic programming languages (PPLs) extend general-purpose languages with probabilistic primitives. In recent years they have received significant attention, motivated by their ability to model and analyze inherently stochastic systems such as randomized algorithms [18], probabilistic cyber-physical systems [23], and machine-learning models [8].

Probabilistic programs (PPs) typically expose parameters that must be tuned to meet goals: maximize likelihood [10, 5], satisfy safety constraints [31], calibrate uncertainty [13], or optimize bespoke utilities [29]. Current practice relies on sampling—either Markov chain Monte Carlo (MCMC) [2] or variational inference (VI) [7]. While often effective, these methods can exhibit high-variance gradient estimates and brittle behavior on discontinuous objectives induced by branching and hard conditioning, frequently necessitating problem-specific reparameterization tricks [34, 28, 1, 14, 26].

S. Junges and G. Katz (Eds.): TACAS 2026, LNCS 16505, pp. 566–585, 2026.
https://doi.org/10.1007/978-3-032-22752-2_29

This paper asks the following: *Can we obtain gradients for PP objectives without sampling?* We answer affirmatively by introducing DeGAS, a differentiable Gaussian-mixture approximate semantics that renders end-to-end evaluation differentiable, enabling optimization with off-the-shelf automatic differentiation (AD) and *without* Monte Carlo estimators. The key idea is a principled, vanishing smoothing of discrete and measure-zero constructs (e.g., point masses and Boolean predicates), which yields differentiable densities and path probabilities throughout a program's execution. This delivers deterministic objective values and gradients with respect to the program parameters.

DeGAS builds on the SOGA (second-order Gaussian approximation) semantics for a class of loop-free probabilistic programs [30]. With SOGA, each program state is represented as a Gaussian mixture (GM) and program constructs update mixtures analytically (affine transforms, products, and conditioning via truncation) to approximate the posterior over program variables. DeGAS smooths the discontinuities that arise under branching and conditioning by injecting small Gaussian noise where needed and relaxing predicates accordingly, while preserving closure under mixture operations. We prove differentiability of posterior distributions and path probabilities with respect to parameters, and we show that as the smoothing parameter tends to zero, the differentiable semantics converges to the unsmoothed SOGA semantics under mild regularity conditions.

We implement DeGAS in PyTorch leveraging its AD capabilities to provide a sample-free gradient-based optimizer. Numerical experiments show that, on thirteen benchmark programs, DeGAS achieves accuracy and runtime competitive with VI and MCMC. More important, we show DeGAS is able to solve instances of optimization problems for which sampling baselines fail to converge due to the presence of constructs such as conditional statements depending on continuous variables.

Structure of the paper. Section 2 reviews related work. Section 3 presents the syntax and defines the differentiable semantics. Section 4 states and proves our main properties (differentiability and convergence). Section 5 reports the experimental results. Section 6 concludes.

2 Related Work

To the best of our knowledge, this is the first work that computes gradients of probabilistic programs without sampling. For deterministic programs, the idea of smoothing to achieve sample-free optimization via gradient-free methods has been explored in [11, 12], and successively adapted for automatic differentiation in [20]. However, as will be shown in Section 4, the smoothing of deterministic programs does not trivially carry over to PPs.

An attempt was made to extend the smoothing to PPs in Leios [22]. Leios takes as input a discrete or hybrid discrete-continuous PP and approximates it with a fully continuous one using smoothing. The authors notice that the relaxation parameters must be tuned to avoid collapsing any probability event to

zero. For example, a variable initially distributed as a Bernoulli after smoothing becomes distributed as a mixture of two components, with means in 0 and 1 and small standard deviations. This implies that the probability of the event $x == 0$ passes from 0.5 to 0. To avoid this, Leios solves an optimization problem to replace all Booleans predicate depending on smoothed variables. Conceptually, we are close in spirit in that we tackle discontinuities by controlled smoothing. However, the goal is fundamentally different: similarly to the previously mentioned works, Leios aids sampling-based inference; instead, we enable sample-free optimization.

For PPs, most of the work has focused on coupling AD with sampling-based inference, designing gradient estimators for discontinuous programs—either to improve MCMC and VI globally [34, 28, 1, 14, 26] or only on the discontinuous parts [35, 21, 24]. These approaches still estimate gradients via sampling and are used to run inference faster or more stably. In contrast, we compute deterministic gradients by evaluating a differentiable program semantics; no Monte Carlo appears in the optimization loop. This lets us optimize likelihood and non-likelihood probabilistic objectives directly (such as reachability), not just expected values under a sampler.

Some methods start from an input program and compute a second program, whose expected return value is the derivative of the input program's expected return value [3, 27]; the new program can then be sampled to obtain gradient estimations. We differ in both goal (optimization rather than derivative-estimation for inference) and mechanism: for our smoothing, we leverage the analytical properties of the SOGA/GM semantics [30], that yield closed-form truncated moments and CDFs for every node, which we then backpropagate through analytically.

3 Syntax and Semantics

SOGA was introduced in [30] as a semantics for a loop-free PPL that manipulates input variables distributed as GMs. SOGA makes the language closed under GMs, providing an analytical representation of the posterior that enables efficient computation of moments, cumulative distribution functions (CDFs), and probability density functions (PDFs). This semantics is the second-order approximation of a family of approximate semantics based on Gaussian Mixtures, guaranteed to convergence to the true one, and demonstrated high accuracy across a variety of programs. Here, we add the possibility of specifying parameters to be used for optimization.

Notation. For a distribution D, let f_D denote its pdf. A normal random variable with mean μ and covariance matrix Σ is denoted by $\mathcal{N}(\mu, \Sigma)$. A GM is a weighted sum of Gaussians, $G_{\pi,\mu,\Sigma} = \sum_{i=1}^{c} \pi_i \mathcal{N}(\mu_i, \Sigma_i)$ where c is the number of *components* and $(\pi_i)_{\{i=1,\ldots,c\}}$ is the vector of *weights* satisfying $0 \leq \pi_i \leq 1$ and $\sum_{i=1}^{c} \pi_i = 1$. A Dirac delta centered on x_0 is denoted by δ_{x_0}. Let $x[x_i \to e]$ denote the vector with e as i-th component, and every other component equal

to x_j. Let $\text{Marg}_y(D)$ denote the marginal of distribution D with respect to subvector y. We denote with $\mathbf{I}_n$ the identity matrix of dimension n and with $\mathbb{I}_A$ the characteristic function of set A. We use $\otimes$ for the product of probability measures: if D_1, D_2 are distributions on $\mathbb{R}$, $D_1 \otimes D_2$ is a distribution on $\mathbb{R}^2$ whose marginal on the first coordinate is D_1 and the marginal on the second coordinate is D_2. For a predicate b, $P_D(b)$ denotes the probability of b under distribution D. $D_{|b}$ denotes distribution D conditioned to b.

3.1 Syntax

Variables. We consider programs defined over a vector of real variables $x = (x_1, \ldots, x_n)$ and a set Θ of parameters, $\Theta = \{\theta_i = d_i\}_{i=1,\ldots,p}$ where θ_i is the name of the parameter and $d_i \in \mathbb{R}$ is an initial value. In addition, for each parameter an interval domain is specified. For instance, we can specify a standard deviation (std) as a parameter called σ, with initial value 1, and interval domain $(0, +\infty)$.

Expressions. Expressions over program variables can either be products between two variables or linear expressions on program variables, where the coefficients of the variables or the zero-th order term can either be constants or parameters:

$$\mathbb{E} = \{x_i x_j \mid i, j = 1 \ldots n\} \cup \{c_1 x_1 + \ldots + c_n x_n + c_0 \mid c_i \in \mathbb{R} \cup \Theta\} \ .$$

Boolean predicates. Boolean predicates are *true* or *false*, or inequalities between a variable and a real number, which can either be a constant or a parameter:

$$\mathbb{B} = \{true, false\} \cup \{x_i \bowtie c \mid c \in \mathbb{R} \cup \Theta, \bowtie \in \{<, \leq, ==, \geq, >\}\}$$

Boolean predicates in the guards of conditional (if) statements are restricted to $\mathbb{B}_{\text{if}}$ with $\bowtie \in \{<, \leq, \geq, >\}$.

The restrictions on expressions and Boolean predicates are inherited by SOGA and are motivated by the need to preserve closure of the Gaussian-mixture semantics under the supported operations. However, our grammar is expressive enough to encode general polynomial assignments and guards.

Language. Our syntax uses the primitive $x \sim gm(\pi, \mu, \sigma)$ for random assignments with GMs, where $\pi = (\pi_1, \ldots, \pi_c)$, $\mu = (\mu_1, \ldots, \mu_c)$, $\sigma = (\sigma_1, \ldots, \sigma_c)$ are vectors of weights, means and standard deviations, respectively, and π_i, μ_i, σ_i can either be real constants or real parameters, with the restriction that $\pi_i, \sigma_i \geq 0$ and $\sum_i \pi_i = 1$. The complete grammar of our programming language is:

$$\begin{aligned}
P ::= \ & \texttt{skip} \mid x \ \texttt{:=} \ e && \text{(deterministic assignments)} \\
& x \sim gm(\pi, \mu, \sigma) && \text{(random assignment)} \\
& \texttt{if } b \ \{ \ P \ \} \ \texttt{else} \ \{ \ P \ \} && \text{(test)} \\
& \texttt{observe } b && \text{(observe)} \\
& P; \ P && \text{(sequential composition)}
\end{aligned}$$

The `observe` b construct represents a hard observe, which conditions the program's distribution on the event specified by b, effectively truncating the support of the resulting distribution to the states satisfying b.

3.2 Semantics

We denote with $[\![\cdot]\!]$ the operator for the *exact semantics* of the program,intended in the classic sense of Kozen's Semantics 2 [19]; instead, $[\![\cdot]\!]^S$ denotes the *Second Order Gaussian Approximation (SOGA) semantics operator* as defined in [30] (both reported in Appedix A for completeness), which closes the language under GMs. The closure is realized by replacing every distribution that would not be a GM in the exact semantics (for instance, when conditioning due to a branch) with a GM. The substitution is performed by a dedicated operator called *the moment-matching operator*, denoted by $\mathcal{T}_{\text{gm}}$, which approximates a density function with a GM by matching its first two moments. In particular, when $\mathcal{T}_{\text{gm}}$ acts on a mixture, it acts on every component; when it acts on a single-component mixture, it maps it to a Gaussian having mean and covariance matrix matching the original distribution, denoted by μ_D, Σ_D. Formally:

$$\mathcal{T}_{\text{gm}}(D) = \begin{cases} \pi_1 \mathcal{T}_{\text{gm}}(D_1) + \ldots + \pi_c \mathcal{T}_{\text{gm}}(D_c) & \text{if } D = \sum_{i=1}^{c} \pi_i D_i \\ \mathcal{N}(\mu_D, \Sigma_D) & \text{else.} \end{cases} . \tag{1}$$

Notably, when acting on Gaussian mixtures, $\mathcal{T}_{\text{gm}}$ leaves them unaltered [30].

Non-differentiability of SOGA semantics. By closing the semantics with respect to GMs, the general pdf of the distribution yielded by a program dependent on parameters Θ can be shown to take the general form

$$f_\Theta(x) = \sum_{i=1}^{C} \frac{\pi_i(\Theta)}{(2\pi)^{n/2}|\Sigma_i(\Theta)|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i(\Theta))^T \Sigma_i^{-1}(\Theta)(x-\mu_i(\Theta))\right).$$

Assuming that $\pi(\Theta), \mu(\Theta)$ and $\Sigma(\Theta)$ are differentiable in Θ, f_Θ can only be nondifferentiable in Θ if a covariance matrix is singular, i.e., $|\Sigma_i(\Theta)| = 0$.

Figure 1 shows four minimal examples in which this may occur. In Figure 1a, y is assigned a constant value c, setting its variance to 0; in Figure 1b, y is assigned with a discrete (Bernoulli) distribution, yielding again 0 variance; in Figure 1c, y is assigned a deterministic function of x, so the distribution of y is completely determined by that of x; reflected in the singularity of the covariance matrix. In Figure 1d, y is observed to take on a real value, having an effect analogous to a constant assignment.

In order to deal with such cases consistently, we smooth the semantics. Smoothing will depend on a hyper-parameter $\epsilon > 0$ that we assume fixed for the rest of this section. Intuitively, ϵ represents the amount of smoothing applied to the program and corresponds to the standard deviation of the Gaussian perturbation applied to smoothed variables.

Next, we discuss how we smooth Boolean predicates and the whole semantics separately. Throughout the section, we let V denote the set of program variables that have been smoothed.

```
x = gm([1.], [0], [1]);
y = c;
```

$\pi = 1, \quad \mu = \begin{bmatrix} 0 \\ c \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

(a) Assignment with a constant.

```
x = gm([1.], [0], [1]);
y = gm([0.5, 0.5], [0, 1], [0, 0]);
```

$\pi_1 = 0.5,\ \mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},\ \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

$\pi_2 = 0.5,\ \mu_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

(b) Assignment with a discrete distribution.

```
x = gm([1.], [0], [1]);
y = x + 1;
```

$\pi = 1, \quad \mu = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

(c) Deterministic function.

```
x = gm([1.], [0], [1]);
y = gm([1.], [0], [1]);
observe(y == 0.5);
```

$\pi = 1, \quad \mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

(d) Observation of a constant value.

Fig. 1: Nondifferentiability of SOGA semantics. Examples yielding singular covariance matrices.

Smoothing operator for Boolean predicates. We define the following smoothing operator for Boolean predicates, that takes as input a predicate b and a set of smoothed variables V and returns a new predicate.

$$\mathcal{B}_\epsilon(x \bowtie c, V) = \begin{cases} x > c + \delta_\epsilon & \text{if } \bowtie \text{ is } >, x \in V \\ x > c - \delta_\epsilon & \text{if } \bowtie \text{ is } \geq, x \in V \\ x < c + \delta_\epsilon & \text{if } \bowtie \text{ is } \leq, x \in V \\ x < c - \delta_\epsilon & \text{if } \bowtie \text{ is } <, x \in V \\ (x > c - \delta_\epsilon) \wedge (x < c + \delta_\epsilon) & \text{if } \bowtie \text{ is } == t, x \in V \\ x \bowtie c & \text{else} \end{cases} \tag{2}$$

For now we allow δ_ϵ to vary with ϵ under the only assumption that $\lim_{\epsilon \to 0} \delta_\epsilon = 0$. In the next section we will introduce an additional restriction to guarantee convergence to the SOGA semantics.

DeGAS. As in [30], we introduce the semantics in terms of the program's control flow graph (cfg). Each node in the graph represents a program instruction. We consider six node types: *entry*, *det*, *rnd*, *test*, *observe* and *exit* and use the notation v: *type* to denote that v is of type *type*. Nodes of type *det* are labelled by `skip`

or by a deterministic assignment x_i := e. Nodes of type *rnd* are labelled by a random assignment $x_i \sim gm(\pi, \mu, \sigma)$. Nodes of type *test* are labelled by a Boolean predicate in $\mathbb{B}_{\text{if}}$, while *observe* node are labelled by a Boolean predicate in $\mathbb{B}$. A function *arg* is defined on all labelled nodes v, returning the label of v. In addition, a function *cond* is defined on the set of *det* and *rnd* nodes, returning either *true, false* or *none* and such that $cond(v) = none$ if and only if the parent of v is not a *test* node. Arithmetic expressions and Boolean predicates are interpreted in the standard way. An expression $e \in \mathbb{E}$ denotes a mapping $e\colon \mathbb{R}^n \to \mathbb{R}$. A Boolean predicate $b \in \mathbb{B}$ denotes the set of vectors $x \in \mathbb{R}^n$ that satisfy the predicate.

For a program P we let Ω^P denote the set of paths in its cfg. For each path $\omega \in \Omega^P$ the function $s_\omega(v)$ associates $v \in \omega$ with its succcessor in ω. The semantics of path ω, denoted by $[\![\omega]\!]^{\Delta,\epsilon}$ is defined as a function taking no input and returning a pair (p, D) where $p \in \mathbb{R}_{\geq 0}$ and D is a GM on $\mathbb{R}^n$. The semantics of a path $\omega = v_0..v_n$ is defined as the composition of the semantics of each node in the path: $[\![\omega]\!]^{\Delta,\epsilon} = [\![v_n]\!]^{\Delta,\epsilon}_{\omega} \circ \ldots \circ [\![v_0]\!]^{\Delta,\epsilon}_{\omega}$.

As noted earlier, when computing the semantics of a node, in addition to p and D, we need to keep track of the smoothed variables, so we augment the semantics with a set V, containing the variables that would be discrete in the exact semantics. The semantics of a node v is a function taking as input a triple (p, D, V), and returning a triple (p', D', V') of the same type. The only exceptions are the *entry* node which takes no input and the *exit* node which just outputs the pair (p, D) The semantics of each node type is defined separately.

- If v: *entry*, then $[\![v]\!]^{\Delta,\epsilon}_{\omega} = (1, \mathcal{N}(0, \mathbb{I}_n), \emptyset)$.
- If v: *det*, let D' denote the distribution of $x[x_i \to e]$ and D^ϵ denote the distribution of $x[x_i \to e + z]$, where z is a fresh program variable with distribution $\mathcal{N}(0, \epsilon)$. Let V' be $V \cup \{x_i\}$ if all variables in e are in V and $V' = V$ else. Then:

$$[\![v]\!]^{\Delta,\epsilon}_{\omega}(p, D, V) = \begin{cases} (p, D, V) & arg(v) = \texttt{skip} \\ (p, D^\epsilon, V \cup \{x_i\}) & arg(v) = x_i \texttt{ := } c \\ (p, D^\epsilon, V') & arg(v) = x_i \texttt{ := } e, e \text{ linear}, x_i \text{ not in } e \\ (p, \mathcal{T}_{\text{gm}}(D'), V') & \text{else.} \end{cases}$$

- If v: *rnd* and $arg(v) = x_i \sim gm(\pi, \mu, \sigma)$, let σ' be the vector with $\sigma'_j = \sigma_j$ if $\sigma_j \neq 0$ and $\sigma'_j = \epsilon$ else. Then:

$$[\![v]\!]^{\Delta,\epsilon}_{\omega}(p, D, V) = \begin{cases} (p, \text{Marg}_{x \setminus x_i}(D) \otimes G_{\pi,\mu,\sigma'}, V \cup \{x_i\}) & \sigma_j = 0 \text{ for some } j \\ (p, \text{Marg}_{x \setminus x_i}(D) \otimes G_{\pi,\mu,\sigma}, V) & \text{else.} \end{cases}$$

- If v: *test*, letting $arg(v) = b$ and $b' = \mathcal{B}_\epsilon(b, V)$:

$$[\![v]\!]^{\Delta,\epsilon}_{\omega}(p, D, V) = \begin{cases} (p \cdot P_D(b'), \mathcal{T}_{\text{gm}}(D_{|b'}), V) & cond(s_\omega(v)) = true \\ (p \cdot P_D(\neg b'), \mathcal{T}_{\text{gm}}(D_{|\neg b'}), V) & cond(s_\omega(v)) = false. \end{cases}$$

– If v: *observe*, letting $arg(v) = b$, $I = \int_{\mathbb{R}^{n-1}} f_D(x[x_i \to c])d(x \setminus x_i)$ and $b' = \mathcal{B}_\epsilon(b, V)$:

$$[\![v]\!]^{\Delta,\epsilon}_\omega(p, D, V) = \begin{cases} (p \cdot I, \mathrm{Marg}_{x \setminus x_i}(D_{|b}) \otimes \mathcal{N}(c, \epsilon), V \cup \{x_i\}) & \\ & b \text{ is } x_i == c, x_i \notin V \\ (p \cdot P_D(b'), \mathcal{T}_{\mathrm{gm}}(D_{|b'}), V) & \text{else} \end{cases}$$

– If v: *exit*, $[\![v]\!]^{\Delta,\epsilon}_\omega(p, D, V) = (p, D)$.

A program P is called *valid* if for at least a path $\omega \in \Omega^P$ $[\![\omega]\!] = (p, D)$ with $p > 0$. The semantics of a valid program P is then defined as a mixture of the semantics of its path:

$$[\![P]\!]^{\Delta,\epsilon} = \sum_{\substack{\omega \in \Omega^P \\ [\![\omega]\!]^{\Delta,\epsilon} = (p,D)}} \frac{p \cdot D}{\sum_{\substack{\omega' \in \Omega^P \\ [\![\omega']\!]^{\Delta,\epsilon} = (p',D')}} p'}.$$

Let us go back to Figure 1, to briefly discuss how $[\![\cdot]\!]^{\Delta,\epsilon}$ avoids the discontinuities. For the program in Figure 1a, $[\![\cdot]\!]^{\Delta,\epsilon}$ effectively assigns y with $c + z$, where z is a freshly defined variable, Gaussianly distributed with mean 0 and std ϵ. This is equivalent to assigning y with $\mathcal{N}(c, \epsilon)$, and yields a non-degenerate covariance matrix $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon^2 \end{bmatrix}$. For the program in Figure 1b, $[\![\cdot]\!]^{\Delta,\epsilon}$ replaces components with zero std with non-degenerate components with std ϵ. Therefore the assignment effectivaly performed would be $y = gm([0.5, 0.5], [0, 1], [\epsilon, \epsilon])$, yielding $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon^2 \end{bmatrix}$. For the program in Figure 1c, $[\![\cdot]\!]^{\Delta,\epsilon}$ acts similarly as in the first case, y is a assigned with $x + 1 + z$, with z fresh Gaussian variable with mean 0 and std ϵ. The covariance matrix then becomes $\Sigma = \begin{bmatrix} 1 & 1 \\ 1 & 1 + \epsilon^2 \end{bmatrix}$. Finally, for the program in Figure 1d, the variance of y after the observation is corrected and put equal to ϵ, yielding $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon^2 \end{bmatrix}$.

To avoid unwanted changes in the behaviour of the probability mass due to the smoothing, $[\![\cdot]\!]^{\Delta,\epsilon}$ uses the operator $\mathcal{B}_\epsilon$ to interpret differently Boolean predicates when they involve smoothed variables, as reflected in the definition of the semantics of *test* and *observe* nodes. This has the effect of avoiding conditioning to zero probability events, since $[\![\cdot]\!]^{\Delta,\epsilon}$ always acts on non-degenerate distributions and $\mathcal{B}_\epsilon(b, V)$ always returns open sets.

In the next section, we will prove that the semantics computed by $[\![\cdot]\!]^{\Delta,\epsilon}$ is indeed differentiable in the program parameters and that by suitably choosing the value of δ_ϵ when applying $\mathcal{B}_\epsilon$ $[\![\cdot]\!]^{\Delta,\epsilon}$ will converge to to $[\![\cdot]\!]^S$.

Example 1. Consider the program P and its cfg in Fig. 2, where θ and σ are parameters with $\theta \in (-\infty, +\infty)$ and $\sigma \in (0, +\infty)$. The exact semantics is:

$$[\![P]\!] = \Phi(\theta/\sigma) \cdot (\mathcal{N}(0, \sigma)_{|x<\theta} \otimes \delta_{-1}) + (1 - \Phi(\theta/\sigma)) \cdot (\mathcal{N}(0, \sigma)_{|x \geq \theta} \otimes \delta_1)$$

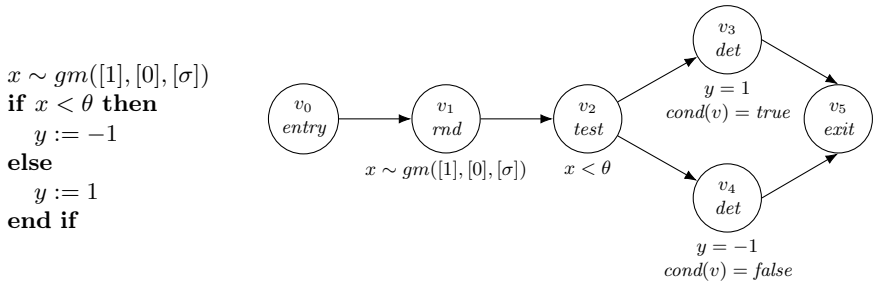

Fig. 2: Example program P (left) and its cfg (right).

In particular, the exact marginal over y is a discrete distribution putting $\Phi(\theta/\sigma)$ probability mass on -1 and $1-\Phi(\theta/\sigma)$ on 1. Therefore, the exact marginal over y is nondifferentiable (in fact, discontinuous) in the parameters.

Using the SOGA semantics, closed with respect to GMs, yields:

$$[\![P]\!]^S = \Phi(\theta/\sigma) \cdot \mathcal{T}_{\mathrm{gm}}(\mathcal{N}(0,\sigma)_{|x<\theta} \otimes \delta_{-1}) + (1-\Phi(\theta/\sigma)) \cdot \mathcal{T}_{\mathrm{gm}}(\mathcal{N}(0,\sigma)_{|x\geq\theta} \otimes \delta_1)$$

However, the marginal over y remains unchanged, and therefore still nondifferentiable. When computing the differentiable semantics, not only is the moment-matching operator applied to close the semantics with respect to GMs, degeneracies are avoided by smoothing the assignments of y. This yields:

$$\begin{aligned}[\![P]\!]^{\Delta,\epsilon} = \Phi(\theta/\sigma) \cdot \mathcal{T}_{\mathrm{gm}}(\mathcal{N}(0,\sigma)_{|x<\theta} \otimes \mathcal{N}(-1,\epsilon)) + \\ + (1-\Phi(\theta/\sigma)) \cdot \mathcal{T}_{\mathrm{gm}}(\mathcal{N}(0,\sigma)_{|x\geq\theta} \otimes \mathcal{N}(1,\epsilon))\end{aligned}$$

For any $\sigma \in (0,+\infty)$, the differentiable marginal over y is given by

$$\Phi(\theta/\sigma) \cdot \mathcal{N}(-1,\epsilon) + (1-\Phi(\theta/\sigma)) \cdot \mathcal{N}(1,\epsilon).$$

The marginal densities of the three semantics, computed for $\sigma = 1$ and $\theta = 0$ are represented in Fig. 3, where the densities of the components of the mixtures are highlighted in orange and for DeGAS we fixed $\epsilon = 0.05$.

4 Properties of $[\![\cdot]\!]^{\Delta,\epsilon}$

In this section we prove two properties of the DeGAS semantics operator $[\![\cdot]\!]^{\Delta,\epsilon}$, namely differentiability of $[\![P]\!]^{\Delta,\epsilon}$ with respect to the parameters and convergence of $[\![P]\!]^{\Delta,\epsilon}$ to $[\![P]\!]^S$ as $\epsilon \to 0$.

Differentiability. First, we focus on the differentiability of the DeGAS semantics with respect to Θ. For this reason, in this subsection we make the dependence of the programs on the parameters explicit, using the notation $P(\Theta)$. For instance, program P from Example 1 would be denoted as $P(\theta,\sigma)$. We assume that the parameters come with assigned initial values and interval domains and that

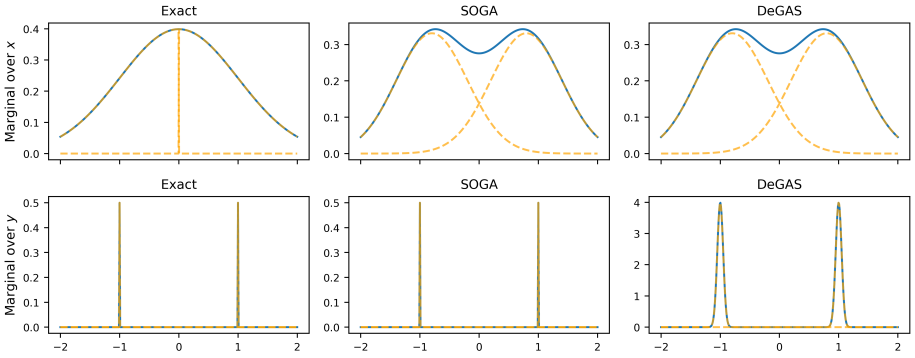

Fig. 3: Marginal posterior densities of program P under different semantics. The orange lines highlight the different components of the mixtures.

they always take values in the interval domains. For instance, for $P(\theta, \sigma)$ we could specify the initial values $d_\theta = 0$ and $d_\sigma = 1$ and the interval domains $\theta \in (-\infty, +\infty)$, $\sigma \in (0, +\infty)$. We say that a distribution $D(\Theta)$ is differentiable in Θ if its pdf is. For example, if $D(\Theta)$ is a GM, $D(\Theta)$ is differentiable in Θ if, for $i = 1, \ldots, c$ (i) $\pi_i(\Theta)$, $\mu_i(\Theta)$ and $\Sigma_i(\Theta)$ are differentiable in Θ; (ii) $\Sigma_i(\Theta)$ is non-singular for every value of Θ that satisfies the constraints.

Theorem 1. *For any valid program $P(\Theta)$, $[\![P(\Theta)]\!]^{\Delta,\epsilon}$ returns $(p'(\Theta), D'(\Theta), V')$ such that $p'(\Theta)$ and $D'(\Theta)$ are differentiable in Θ.*

This is the core theoretical result of the paper (proved in the Appendix), because it allows for a safe application of gradient-based optimization methods to PPs.

Convergence. DeGAS adds an additional layer of approximation, through the Gaussian perturbation, to the already approximated semantics of SOGA $[\![\cdot]\!]^S$. We prove that this additional approximation tends to disappear when the perturbation becomes very small, i.e. in the limit for $\epsilon \to 0$. Coupled with SOGA's convergence result to the true semantics $[\![\cdot]\!]$, this offers a principled way of obtaining sample-free gradients for a class of PPs.

DeGAS' convergence proof (provided in the Appendix) is surprisingly subtle and deserves an intuitive explanation. When smoothing the predicates, the dependence of δ_ϵ on ϵ can jeopardize the convergence. The following example motivates the introduction of an additional requirement, called *consistency*, that is needed to guarantee convergence.

Example 1. We consider two example programs:

$$P_1 \colon x := 0; y \sim gm([1.], [0.], [1.]); \texttt{observe}\ x \geq 0,$$
$$P_2 \colon x := 0; y \sim gm([1.], [0.], [1.]); \texttt{observe}\ x > 0.$$

In SOGA we have $[\![P_1]\!]^S = (1, \delta_0 \times \mathcal{N}(0,1))$ and $[\![P_2]\!]^S = (0, \delta_0 \times \mathcal{N}(0,1))$. In DeGAS, before the observe statement for both programs we have the output triple $(1, D_\epsilon, \{x\})$ where $D_\epsilon = \mathcal{N}(0,\epsilon) \times \mathcal{N}(0,1)$. However, the predicates of the observe node are smoothed differently:

$$\mathcal{B}_\epsilon(x \geq 0, \{x\}) = x > -\delta_\epsilon, \qquad \mathcal{B}_\epsilon(x > 0, \{x\}) = x > \delta_\epsilon.$$

Consider P_1. To compute the differentiable semantics of the observe node we need to compute $P_{D_\epsilon}(x > -\delta_\epsilon)$ and the moments of $(D_\epsilon)_{|x>\delta_\epsilon}$, and we want them to tend to 1 and to the moments of $\delta_0 \times \mathcal{N}(0,1)$, respectively. Here, we focus on the probabilities, since the convergence of the moments are proved using similar arguments. Letting Φ denote the cdf of the standard Gaussian, then $P_{D_\epsilon}(x > -\delta_\epsilon) = 1 - \Phi\left(-\frac{\delta_\epsilon}{\epsilon}\right)$. For this quantity to tend to 1 we need $\frac{\delta_\epsilon}{\epsilon} \to \infty$. Intuitively, this limit forces δ_ϵ to decrease slower than the variance of x (in this case ϵ), so that as ϵ tends to 0 more and more of the probability mass of D_ϵ is inside $\{x > -\delta_\epsilon\}$. Moreover, it is clear that if δ_ϵ is a linear function of ϵ or if it decreases faster than it, the limit cannot hold.

Analogously consider P_2 for which $P_{D_\epsilon}(x > \delta_\epsilon) = 1 - \Phi(\frac{\delta_\epsilon}{\epsilon})$. For it to tend to 0 we need again $\frac{\delta_\epsilon}{\epsilon} \to \infty$. This time the slower convergence of δ_ϵ to 0 allows a larger part of the probability mass of D_ϵ to remain outside $\{x > \delta_\epsilon\}$.

The previous example leads us to the following definition and theorem (we report the full proof in the Appendix).

Definition 1. *The operator $\mathcal{B}_\epsilon$ is applied* consistently *if for every test and observe node with $arg(v) = x_i \bowtie c$ when $\mathcal{B}_\epsilon$ is applied to compute $[\![v]\!]^{\Delta,\epsilon}(p, D, V)$ and $x_i \in V$, δ is chosen such that:*

$$\lim_{\epsilon \to 0} \delta_\epsilon(\epsilon) = 0 \qquad \textit{and} \qquad \lim_{\epsilon \to 0} \frac{\delta_\epsilon(\epsilon)}{\sqrt{\Sigma_{x_i,x_i}(\epsilon)}} = +\infty$$

for every $\Sigma(\epsilon)$ covariance matrix of a component of D. The dependence of Σ from ϵ is guaranteed by the fact that $x_i \in V$.

Theorem 2. *Suppose P is a valid program and $\mathcal{B}_\epsilon$ is applied consinstently when computing $[\![P]\!]^{\Delta,\epsilon}$. Then $\lim_{\epsilon \to 0} [\![P]\!]^{\Delta,\epsilon} = [\![P]\!]^S$ (where convergence for the distributions is intended in the sense of* convergence in distribution*).*

We remark that, while in theory it is possible to let ϵ tend to 0 to make the additional error vanish, in practice we want to use a value of ϵ large enough to enable efficient gradient-based optimization, as will be discussed next.

5 Experimental Evaluation

Setup. We evaluate DeGAS along two axes: (i) we optimize parameters of benchmark probabilistic programs taken from the literature comparing with Variational Inference (VI) and Markov Chain Monte Carlo (MCMC); (ii) we evaluate

DeGAS on programs with discontinuities related to continuous variables, which, as we will show, make standard optimization methods inapplicable. In this case we consider classic models of cyber-physical systems (CPS), whose parameters are synthesized to optimize trajectory likelihoods or reachability properties.

We implemented DeGAS in Pytorch, leveraging automatic differentiation for the gradient computation (cf. Appendix C) and employing Adam as optimizer. For all the experiments we set $\delta_\epsilon = \sqrt{\epsilon}$ and report the results for $\epsilon = 0.001$. The choice of δ_ϵ is coherent for the consistency assumptions. Furthermore, we did not observe significant accuracy changes with respect to the SOGA approximation when varying ϵ, as long as it stays below 0.1, as reported in Appendix D.1. This is further confirmed by the results reported in Appendix D.6, pointing to the fact that the main source of error is that induced by SOGA, while ϵ has a negligible empirical effect for the considered values. For a dedicated analysis of the SOGA error for inference tasks, including discussion on increasing the Gaussian approximation order, we refer the reader to [30]. We run all the experiments on a machine equipped with an Apple M2 processor with 8 cores running at 3.49 GHz and 8 GB RAM. For all the experiments we set a time-out threshold of 600s.

Comparison with VI and MCMC. We start by comparing DeGAS with Variational Inference (VI) and Markov Chain Monte Carlo (MCMC) as implemented in Pyro [5] on a set of 13 case studies taken from the PPL literature ([10, 15, 16]). As a standard benchmark optimization task, we also consider a Proportional Integral Derivative (PID) controller, described in Appendix D.2.

For each case study, some of the significant variables are set as optimizable parameters while others as observables. For the observables a dataset D of size $N = 1000$ is generated for the true value of the parameters. For the PID case study, data correspond to idealized representations of stable dynamics, maintaining the system at its equilibrium point. For each program, the posterior density computed by DeGAS is used to compute the negative log-likelihood $l(\Theta) = -\log f_\Theta(x)$ of the dataset D; then, we optimize to find $\Theta^* = \text{argmin}_\Theta l(\Theta)$.

For both VI and DeGAS, we report the results for the value of the learning rate chosen among $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.2\}$ that achieves the minimum error. The optimization is stopped upon convergence within a tolerance of 10^{-8} with a patience of 30 iterations, with a maximum of 500 steps for DeGAS (except 1000 for the PID model) and 1000 for VI. For VI, we use the `Trace_ELBO` loss[4]. For MCMC, we run the NUTS kernel with 4 chains, initially running inference with 500 samples and 50 warm-up steps. If $\hat{R}$ exceeds 1.05, the procedure is repeated, increasing the number of samples by 500 and the warm-up steps by 50 when the total number of steps is between 2000 and 5000, and by 100 thereafter, up to a maximum of 6000 total steps. All hyperparameter values are reported in Appendix D.3.

Table 1 collects the results, where 'time' refers to the average runtimes (in s) out of 10 executions and 'error' refers to the average relative difference between the optimized value and the real value across all the optimizable parameters of

[4] https://docs.pyro.ai/en/dev/inference_algos.html

Model	# Par.	# Var.	VI		MCMC		DeGAS	
			Time	Error	Time	Error	Time	Error
Bernoulli	1	2	0.204	0.024	10.968	0.023	1.986	0.001
Burglary	2	7	3.244	0.913	13.014	0.141	14.347	1.686
ClickGraph	1	6	1.635	0.376	62.546	0.096	21.600	0.086
ClinicalTrial	3	6	1.055	1.980	120.492	2.536	17.070	0.657
Grass	4	6	3.077	0.306	58.151	0.015	317.582	0.036
MurderMistery	1	1	0.792	8.015	14.036	2.303	2.508	0.203
SurveyUnbiased	2	2	0.794	0.013	17.030	0.013	11.570	0.008
TrueSkills	3	6	2.056	0.003	t.o.	t.o.	1.664	0.003
TwoCoins	2	1	0.095	0.864	49.120	0.873	3.410	0.688
AlterMu	3	5	1.007	1.043	t.o.	t.o.	0.640	0.312
AlterMu2	2	3	0.718	0.556	384.730	0.139	0.618	0.096
NormalMixtures	3	3	1.061	0.909	345.409	0.053	2.941	0.386
PID	2	55	10.778	–	nc	nc	213	–

Table 1: Comparison of inference methods across models. Models highlighted in gray indicate cases where DeGAS outperformed both other methods in either accuracy or execution time. For the last model, PID, we are not able to quantify the error, as the true parameters are unknown, but we show the result of the optimization in Figure 7. MCMC fails to converge for this case study ($\hat{R} \gg 1$.)

the program. As the table shows, our approach generally achieves performance comparable to the baseline methods, and in several benchmarks it outperforms them in terms of accuracy. Two models particularly challenging for DeGAS are Grass and Burglary. Both models contain a large number of nested if statements, that give rise to a high number of components in the GM approximation. This problem is also known for SOGA and can be mitigated using a pruning strategy [30], whose tuning is outside the scope of this paper.

Many benchmark programs include conditional statements with Boolean guards over discrete variables. When these conditions are enumerable, both VI MCMC can correctly perform inference through enumeration.[5] However, when the condition depends on continuous variables, the resulting model becomes discontinuous and nondifferentiable, breaking the smoothness assumptions of gradient-based VI and affecting the convergence of MCMC. This behavior is clearly visible in the program on the right inset adapted from [25], where μ_1 and μ_2 are optimizable parameters and whose true value is 0.5 and 1. Figure 4 shows that VI has oscillatory loss values (similarly, MCMC does not reach a value of $\hat{R} < 1.05$, indicating nonconvergence), while DeGAS optimizes successfully. This highlights the complementarity of DeGAS with existing methods:

$v \sim gm([1],[\mu_1],[5])$
if $v > 0$ **then**
  $y \sim gm([1],[\mu_2],[1])$
else
  $y \sim gm([1],[-2],[1])$
end if

[5] https://pyro.ai/examples/enumeration.html

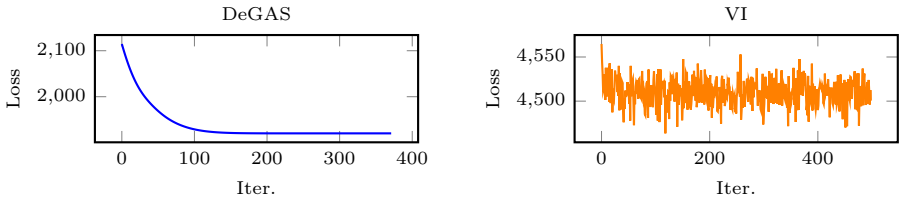

Fig. 4: Comparison of DeGAS and VI loss curves for the program in Section 5.

while it may incur longer runtimes on some benchmarks, it can also enable efficient optimization in challenging discontinuous settings, where VI and MCMC are known to struggle.

Parameter synthesis for cyberphysical systems. We now consider models of cyberphysical systems (CPS) where control-flow branches depend on continuous random variables such that the previously described nondifferentiability issue arises. DeGAS, instead, provides an analytic form for the loss functions: we show examples of both data likelihoods and complex reachability objectives. We consider the following models taken from [32, 11] and fully described in Appendix D.4:

1. **Thermostat** represents a thermostat maintaining fixed temperature by switching an heater on or off when the temperature crosses the thresholds t_{on} and t_{off} (optimizable parameters). The temperature evolves according to a first-order decay process with an added heating term when the heater is active and stochastic fluctuations due to noise.
2. **Gearbox** represents a car's transmission system. We consider a model with three gears. The controller determines when to shift from one gear to the next, allowing only consecutive transitions. The thresholds s_i (optimizable parameters) specify the velocity at which gear i is released and gear $i+1$ engages. Additionally, gear shifts are not instantaneous: in our model, each transition lasts 0.3 seconds, during which the gearbox remains disengaged. The velocity grows linearly when the gear is engaged and decreases quadratically when disengaged. In both cases it is subject to stochastic fluctuations.
3. **Controlled Bouncing Ball** represents a ball dropped onto a platform with a spring and damper. The spring pushes the ball back with a force determined by the compliance C, and the damper slows the ball according to the coefficient R (both are optimizable parameters). When the ball reaches the platform, the system transitions to a reflection mode, returning to free fall once the height is positive.

Parameter synthesis by data likelihoods. As in the previous case studies, the goal of the optimization is to maximize the negative log-likelihood with respect to a set of 100 trajectories generated from the models using the true parameter values. Table 2 and Figure 5 show that DeGAS consistently recovers parameter values close to the ground truth. The optimized mean trajectories (in red) accurately

Table 2: Results for the CPS optimization from trajectories

Case Study	Params	lr	Steps	Time	Init	Target	Result
Thermostat	t_{ON}, t_{OFF}	0.1	40	61.964	$(15, 22)$	$(17, 20)$	$(16.78, 19.97)$
Gearbox	s_1, s_2	0.15	200	167.643	$(8, 12)$	$(10, 20)$	$(8.69, 19.24)$
Bouncing Ball	$R, 1/C$	0.8	100	500.621	$(-1, 450)$	$(7, 400)$	$(5.27, 408.99)$

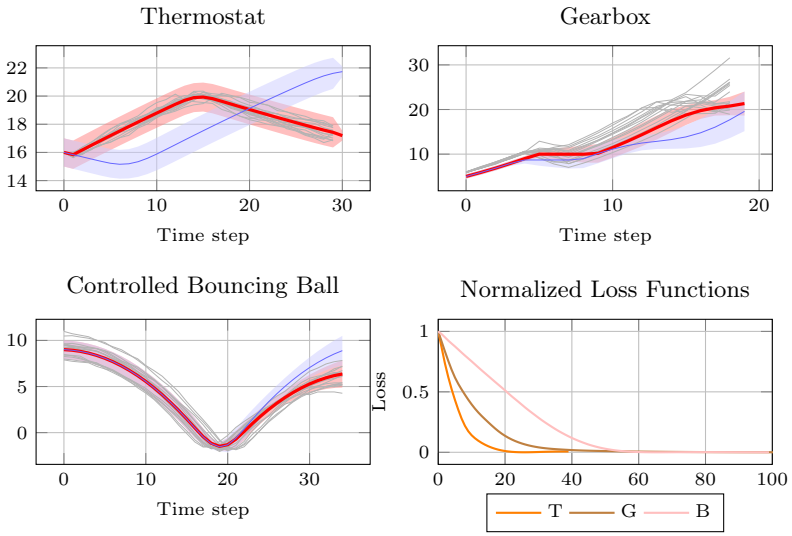

Fig. 5: Mean trajectories with one standard deviation (red) obtained using optimized parameters, compared to trajectories generated with initial parameters (blue). The gray trajectories represent the samples used for optimization.

capture the behavior of the stochastic simulations (in grey), even when the initial trajectories (in blue) are far from the correct one. The normalized loss curves confirm convergence in all three cases.

Synthesis by reachability A complementary synthesis objective consists in optimizing the parameters of the CPS to maximize the probability of reaching specific target states or satisfying certain behavioral constraints. We consider three such cases. Thermostat: the goal is to maximize the probability of reaching the region $T(t) \in [T_{min}, T_{max}] = [19.9, 20.1]$ in mode "ON" at various time points ($\tau_1 = 0.6, \tau_2 = 1.8$ and $\tau_3 = 2.4$); gearbox: we maximize the probability of maintaining velocity below 16 across the whole trajectory; bouncing ball: we want to maximize the probability that the ball reaches $H \geq 7$ when falling after making one bounce. The optimizable parameters are the same as the ones in the previous section.

Table 3: Results for the CPS optimization from reachability properties

Case Study	lr	Steps	Time(s)	Init	Init Loss	Result	Fin. Loss
Thermostat	0.1	100	2770	$(16.5, 22)$	$-2e^{-12}$	$(19.04, 21.30)$	-5695
1Gearbox	0.5	40	52.642	$(10, 20)$	1.000	$(1.92, -1.57)$	0.966
Bounc. Ball	0.2	50	184.264	$(7, 200)$	-0.002	$(-3.87, 210.14)$	-0.979

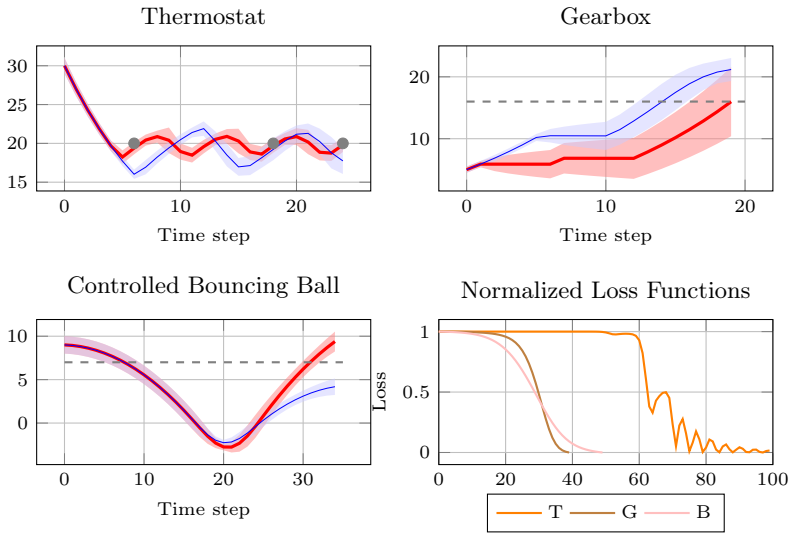

Fig. 6: Mean trajectories with one standard deviation (red) obtained by optimizing the target represented in gray are compared with initial trajectories (blue)

We report the results in Table 3 and Figure 6. The table reports the initial and final loss values, since the true optimal parameters are unknown. As shown in the figure, for all models the optimized trajectories (in red) satisfy the desired behavioral constraints, unlike the initial trajectories (in blue). The optimization curves further confirm convergence across all three models.

6 Conclusions

We introduced DeGAS, a method to enable sample-free optimization of probabilistic programs via a differentiable program semantics. DeGAS smoothens SOGA, a semantics which represents a program evolution through Gaussian-mixture updates (affine maps, products, and truncated moments) that enable an analytic—but, in general, discontinuous, representation of the posterior distribution. We demonstrated the potential of DeGAS in tackling synthesis tasks that cannot be directly solved through classic optimization methods such as variational inference and Markov chain Monte Carlo, due to the presence of discontinuities hindering convergence.

In principle, the main ideas behind DeGAS can be applied to any semantics that (i) represents program states in a tractable form, (ii) supports closed-form (or deterministic-quadrature) updates for transforms and conditioning, and (iii) admits a smooth relaxation of discrete or measure-zero constructs whose parameter $\varepsilon \to 0$ recovers the original semantics. We conjecture that under these conditions, all statements about differentiability hold for $\varepsilon > 0$, and convergence back to the unsmoothed semantics follows by similar arguments as for SOGA. Thus, an interesting direction for future work is to study other semantics such as probabilistic circuits, mixture of exponentials or general symbolic PPs that provide analytic or semi-analytic representations of distributions.

Data Availability Statement

DeGAS is publicly available on Github at https://github.com/frarandone/DeGAS. This paper was accompanied by an artifact for the replication of the experimental results which can be found at: https://zenodo.org/records/18197807.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 101205923 (MSCA Post-doctoral Fellowship EMBEr).

References

1. Afshar, H.M., Domke, J.: Reflection, refraction, and hamiltonian monte carlo. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. pp. 3007–3015 (2015)
2. Andrieu, C., De Freitas, N., Doucet, A., Jordan, M.I.: An introduction to mcmc for machine learning. Machine learning **50**(1), 5–43 (2003)
3. Arya, G., Schauer, M., Schäfer, F., Rackauckas, C.: Automatic differentiation of programs with discrete randomness. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022)
4. Billingsley, P.: Convergence of probability measures. John Wiley & Sons (2013)
5. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., Goodman, N.D.: Pyro: Deep universal probabilistic programming. Journal of machine learning research **20**(28), 1–6 (2019)
6. Bishop, C.M., Nasrabadi, N.M.: Pattern recognition and machine learning, vol. 4. Springer (2006)

7. Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: A review for statisticians. Journal of the American statistical Association **112**(518), 859–877 (2017)
8. Borgström, J., Gordon, A.D., Greenberg, M., Margetson, J., Van Gael, J.: Measure transformer semantics for bayesian machine learning. In: European symposium on programming. pp. 77–96. Springer (2011)
9. Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
10. Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A.: Stan: A probabilistic programming language. Journal of statistical software **76**, 1–32 (2017)
11. Chaudhuri, S., Solar-Lezama, A.: Smooth interpretation. ACM Sigplan Notices **45**(6), 279–291 (2010)
12. Chaudhuri, S., Solar-Lezama, A.: Smoothing a program soundly and robustly. In: International Conference on Computer Aided Verification. pp. 277–292. Springer (2011)
13. Chong, A., Menberg, K.: Guidelines for the bayesian calibration of building energy models. Energy and Buildings **174**, 527–547 (2018)
14. Dinh, V., Bilge, A., Zhang, C., IV, F.A.M.: Probabilistic path hamiltonian monte carlo. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, vol. 70, pp. 1009–1018. PMLR (2017), http://proceedings.mlr.press/v70/dinh17a.html
15. Gehr, T., Misailovic, S., Vechev, M.: Psi: Exact symbolic inference for probabilistic programs. In: International Conference on Computer Aided Verification. pp. 62–83. Springer (2016)
16. Huang, Z., Dutta, S., Misailovic, S.: Aqua: Automated quantized inference for probabilistic programs. In: International Symposium on Automated Technology for Verification and Analysis. pp. 229–246. Springer (2021)
17. Kan, R., Robotti, C.: On moments of folded and truncated multivariate normal distributions. Journal of Computational and Graphical Statistics **26**(4), 930–934 (2017)
18. Karp, R.M.: An introduction to randomized algorithms. Discrete Applied Mathematics **34**(1-3), 165–201 (1991)
19. Kozen, D.: Semantics of probabilistic programs. In: 20th Annual Symposium on Foundations of Computer Science (sfcs 1979). pp. 101–114. IEEE (1979)
20. Kreikemeyer, J.N., Andelfinger, P.: Smoothing methods for automatic differentiation across conditional branches. IEEE Access **11**, 143190–143211 (2023)
21. van Krieken, E., Tomczak, J.M., ten Teije, A.: Storchastic: A framework for general stochastic automatic differentiation. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 7574–7587 (2021)
22. Laurel, J., Misailovic, S.: Continualization of probabilistic programs with correction. In: Müller, P. (ed.) Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12075, pp. 366–393. Springer (2020). https://doi.org/10.1007/978-3-030-44914-8_14, https://doi.org/10.1007/978-3-030-44914-8_14

23. Lee, E.A., Seshia, S.A.: Introduction to embedded systems: A cyber-physical systems approach. MIT press (2016)
24. Lee, W., Rival, X., Yang, H.: Smoothness analysis for probabilistic programs with application to optimised variational inference. Proc. ACM Program. Lang. **7**(POPL), 335–366 (2023). https://doi.org/10.1145/3571205, https://doi.org/10.1145/3571205
25. Lee, W., Yu, H., Rival, X., Yang, H.: Towards verified stochastic variational inference for probabilistic programs. Proceedings of the ACM on Programming Languages **4**(POPL), 1–33 (2019)
26. Lee, W., Yu, H., Yang, H.: Reparameterization gradient for non-differentiable models. Advances in Neural Information Processing Systems **31** (2018)
27. Lew, A.K., Huot, M., Staton, S., Mansinghka, V.K.: ADEV: sound automatic differentiation of expected values of probabilistic programs. Proc. ACM Program. Lang. **7**(POPL), 121–153 (2023). https://doi.org/10.1145/3571198, https://doi.org/10.1145/3571198
28. Pakman, A., Paninski, L.: Auxiliary-variable exact hamiltonian monte carlo samplers for binary distributions. In: Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. pp. 2490–2498 (2013)
29. Rainforth, T., Le, T.A., van de Meent, J.W., Osborne, M.A., Wood, F.: Bayesian optimization for probabilistic programs. Advances in Neural Information Processing Systems **29** (2016)
30. Randone, F., Bortolussi, L., Incerto, E., Tribastone, M.: Inference of probabilistic programs with moment-matching gaussian mixtures. Proc. ACM Program. Lang. **8**(POPL), 1882–1912 (2024). https://doi.org/10.1145/3632905, https://doi.org/10.1145/3632905
31. Sato, T., Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., Hsu, J.: Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, bayesian inference, and optimization. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)
32. Shmarov, F., Zuliani, P.: Probreach: Verified probabilistic delta-reachability for stochastic hybrid systems (2015), https://arxiv.org/abs/1410.8060
33. Wick, G.C.: The evaluation of the collision matrix. Physical review **80**(2), 268 (1950)
34. Zhang, Y., Sutton, C., Storkey, A.J., Ghahramani, Z.: Continuous relaxations for discrete hamiltonian monte carlo. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. pp. 3203–3211 (2012)
35. Zhou, Y., Gram-Hansen, B.J., Kohn, T., Rainforth, T., Yang, H., Wood, F.: LF-PPL: A low-level first order probabilistic programming language for non-differentiable models. In: Chaudhuri, K., Sugiyama, M. (eds.) The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan. Proceedings of Machine Learning Research, vol. 89, pp. 148–157. PMLR (2019), http://proceedings.mlr.press/v89/zhou19b.html