Dimensional Peeking for Low-Variance Gradients in Zeroth-Order Discrete Optimization via Simulation

Philipp Andelfinger
Nanyang Technological University
Singapore, Singapore
philipp.andelfinger@ntu.edu.sg

Wentong Cai
Nanyang Technological University
Singapore, Singapore
aswtcai@ntu.edu.sg

Abstract

Gradient-based optimization methods are commonly used to identify local optima in high-dimensional spaces. When derivatives cannot be evaluated directly, stochastic estimators can provide approximate gradients. However, these estimators' perturbation-based sampling of the objective function introduces variance that can lead to slow convergence. In this paper, we present dimensional peeking, a variance reduction method for gradient estimation in discrete optimization via simulation. By lifting the sampling granularity from scalar values to classes of values that follow the same control flow path, we increase the information gathered per simulation evaluation. Our derivation from an established smoothed gradient estimator shows that the method does not introduce any bias. We present an implementation via a custom numerical data type to transparently carry out dimensional peeking over C++ programs. Variance reductions by factors of up to 7.9 are observed for three simulation-based optimization problems with high-dimensional input. The optimization progress compared to three meta-heuristics shows that dimensional peeking increases the competitiveness of zeroth-order optimization for discrete and non-convex simulations.

CCS Concepts

• **Theory of computation** → **Randomized local search**; • **Mathematics of computing** → **Numerical differentiation**.

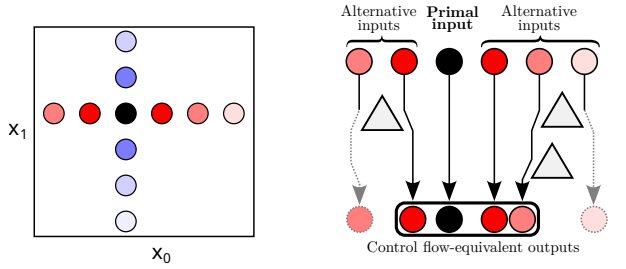
Keywords

Simulation-based optimization, discrete optimization, gradient estimation, variance reduction

ACM Reference Format:
Philipp Andelfinger and Wentong Cai. 2026. Dimensional Peeking for Low-Variance Gradients in Zeroth-Order Discrete Optimization via Simulation. In *40th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '26), June 24–26, 2026, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3806789.3810252

1 Introduction

Simulation-based optimization (SBO) problems appear in many scientific contexts and application domains. Their defining characteristic is that the objective function is a simulation, in contrast to the explicit analytical expression available in other forms of mathematical optimization [13]. In many instances, additional properties of the objective function such as high input dimensionality, stochasticity, non-convexity, high cost of function evaluations, and the unavailability of derivatives make SBO particularly challenging.



(a) Augmented decision variables. **(b) Dimensional peeking for x_0.**

Figure 1: Dimensional peeking in a two-dimensional space. A primal perturbed input (black circle) is augmented by all alternative values of non-negligible probability per dimension. By identifying and grouping input values that follow the same path at branches (triangles), the sampling granularity is lifted from scalars to control flow-equivalent classes.

Commonly, SBO problems are tackled in a black-box manner using response surface methods or meta-heuristics such as evolutionary algorithms [33]. In the past few years, there has been a renewed interest in solving SBO using gradient descent. By various forms of smoothing, gradient estimates can be determined even for simulations involving discontinuities such as those introduced by conditional control flow [2, 5, 8, 15, 18]. Beyond SBO, simulation gradients also enable a more natural integration of simulations into machine learning pipelines, e.g., for reinforcement learning [23].

A significant issue in gradient estimation over simulations is the high variance stemming from the diversity in state trajectories, which can limit the convergence speed. This issue is particularly pressing in derivative-free, or *zeroth-order*, optimization methods [19], which often introduce additional stochasticity by random perturbations to the decision variables to estimate gradients of a smooth approximation of the original objective function.

In the present paper, we propose a variance reduction method for zeroth-order optimization over discrete decision variables. The main idea is to lift the simulation evaluation from the level of points in the input parameter space towards *classes of points* that follow the same control flow path, thereby extracting significantly more information per evaluation. Figure 1 illustrates our method, which we refer to as *dimensional peeking*, on a conceptual level. The simulation output under the alternative perturbations is evaluated in a vectorized fashion alongside a "primal" perturbed evaluation, maintaining low overhead. On each input dimension, all perturbations

SIGSIM-PADS '26, Vienna, Austria

ACM ISBN 979-8-4007-2648-4/26/06
https://doi.org/10.1145/3806789.3810252

equivalent in control flow are considered jointly while exploiting a priori knowledge of the perturbation probabilities. In effect, dimensional peeking eliminates the variance among per-dimension perturbations that do not differ in control flow.

Our main contributions are as follows:

- We present **dimensional peeking**, establish its unbiasedness with respect to an established smoothed gradient estimator [27], and characterize its variance reduction.
- We describe an **efficient implementation** based on operator overloading and vectorization that enables dimensional peeking for simulations in C++ with minimal user effort[1].
- We report **results from extensive experiments** to evaluate the variance reduction, execution time overhead, and optimization performance of dimensional peeking in three discrete SBO problems against three popular meta-heuristics.

The remainder of the paper is structured as follows. In Section 2, we provide background on gradient estimation techniques for discrete simulations and differentiate our method from existing variance reduction techniques. Section 3 introduces the method of dimensional peeking and its efficient implementation. In Section 4, we present experimental results to explore the method's variance reduction, computational overhead, and optimization progress over time in three SBO problems. Section 5 provides a discussion of our results and future work and concludes the paper.

2 Background and Related Work

In the following, we discuss how our approach relates to existing work on gradient estimation for discrete simulations and briefly cover established variance reduction techniques applicable in this context.

2.1 Gradient Estimation for Discrete Simulations

Simulations often involve jump discontinuities originating from conditional branching. Although the output of a stochastic simulation involving branches may still be continuous in expectation, each simulation evaluation may observe a different control flow path. The challenge in estimating gradients over such simulations lies in correctly accounting for the effects of the discontinuities based on a finite sample of trajectories.

2.1.1 Traditional approaches. Classical gradient estimators from the SBO literature rely on manual analysis and problem knowledge to derive unbiased gradient estimators. In *smoothed perturbation analysis* [12], the objective function is separated into continuous parts through a problem-specific conditioning on suitable variables. The *likelihood ratio estimator*, also known as REINFORCE or score function estimator, applies the differentiation rule of the logarithm to allow unbiased gradient estimations with respect to parameters of known distributions [35].

2.1.2 Automatic Differentiation. Recent methods frequently rely on automatic differentiation (AD) [21], which propagates derivative information along the operations involved in a program by repeatedly applying the chain rule. In forward-mode AD, an original program's variables are extended to carry a tangent with respect to an input variable in addition to the original variable value. Reverse-mode AD, a special case of which is the well-known backpropagation algorithm [30], records the operations and computes derivatives back-to-front once the forward execution has terminated. As AD alone cannot account for discontinuities, a variety of works substitute jumps with smooth approximations such as logistic or sigmoid functions [2, 8, 25]. A downside of these approaches is the difficulty of predicting and controlling the bias introduced by this form of smoothing [38], which lacks a clear interpretation.

Some recent works propose methods to compute gradients without the need for manual derivations or problem-specific smoothing. Arya et al. proposed a form of forward-mode AD that computes unbiased gradients for stochastic programs that draw from a set of discrete distributions [5]. Based on a custom chain rule, it suffices to propagate the variable values for a "primal" trajectory and a single alternative trajectory throughout the program. An approach supporting arbitrary conditional branching was proposed by Kreikemeyer et al. [18]. Their estimator records branch condition variables and their derivatives and combines them with pathwise AD gradients to account for the effects of branches. As the methods relies on density estimations, the approach involves a variance-bias tradeoff.

While dimensional peeking does not rely on AD, its realization via operator overloading bears a loose similarity to forward-AD implementations via dual numbers [21], albeit with no similarities in the computation rules or interpretation of the output vectors. The implementation shares the use of vectorization with our prior work on accelerating gradient estimation for agent-based simulations [3]. However, aiming for speedup rather than variance reduction, this existing work focused on the efficient evaluation of sets of random parameter combinations, in contrast to the change in sampling granularity achieved by dimensional peeking.

2.1.3 Stochastic Estimators. Finally, stochastic black-box estimators employ random perturbations to compute smoothed gradient estimates from finite differences across the simulation output. Simultaneous perturbation stochastic approximation [32] is a gradient descent scheme that relies on central finite difference estimates using random perturbations, scaled by a decreasing series over the iterations. A similar estimator described by Polyak [27] employs forward differences and Gaussian perturbations and has been analyzed in the context of random search strategies by Nesterov et al. [24]. We refer to this estimator as Polyak's Gradient Oracle (PGO). Due to the external perturbations, the gradients returned by these stochastic estimators are unbiased with respect to a *smooth approximation* of the original objective function. For instance, Polyak's estimator reflects the gradients of the original objective function after convolution with a Gaussian kernel. While the perturbations introduce a bias, these estimators are generically applicable without the need for prior analysis of the objective function or code adaptations as typically required for AD. Stochastic estimators have been applied to convex and non-convex, smooth and non-smooth, as well as real, discrete, and mixed objectives [4, 7, 10, 14, 16, 34, 37].

In the present paper, we specialize Polyak's estimator for simulations over discrete decision variables and derive dimensional peeking based on this formulation.

[1] Available at https://doi.org/10.5281/zenodo.18081457

2.2 Variance Reduction Methods

We briefly discuss how existing methods for variance reduction relate to dimensional peeking, restricting our discussion to works specific to stochastic black-box estimators. This excludes generic approaches such as control variates, antithetic variates, Rao-Blackwellization, and importance sampling [29], as these are orthogonal to and may be combined with our method.

Petersen et al. [26] evaluated different sampling schemes for the random perturbations used by stochastic gradient estimators. Choosing perturbations based on low-discrepancy sequences rather than i.i.d. sampling achieves better coverage of the neighborhood around the current solution. Similarly, the coverage of the multi-dimensional input space can be improved by choosing orthogonal directions across dimensions [11, 17]. By choosing dependent perturbations, these approaches sacrifice unbiasedness with respect to the smoothed gradient in favor of reduced variance. In contrast, dimensional peeking covers the non-negligible portion of the perturbations' support on all input dimensions entirely, lifting the sampling from individual values of the discrete variables to classes of values that lead to the same control flow.

3 Dimensional Peeking

In the following, we derive the approach of dimensional peeking from an existing stochastic gradient estimator and characterize the achieved variance reduction. Subsequently, we describe an efficient implementation of the required adaptations to lift the evaluation of a program with discrete inputs from the scalar level to classes of values that lead to the same control flow path.

3.1 Derivation

We start with the stochastic forward-differences gradient oracle described by Polyak [27] and analyzed by Nesterov et al. [24], which we refer to as Polyak's Gradient-Free Oracle (PGO). Here, we adapt PGO for the discrete case of a function $f : \mathbb{Z}^d \to \mathbb{R}$ by drawing vectors R of random perturbations from a discretized normal distribution:

$$R \sim \text{Discrete-}\mathcal{N}(0, \sigma^2 I_d)$$

with

$$P(R{=}r) = \int_{r-0.5}^{r+0.5} \frac{1}{\sqrt{2\pi}\,\sigma}\, e^{-\frac{t^2}{2\sigma^2}}\, dt.$$

In this setting, the PGO estimator is:

$$g_{\text{PGO}} = \big(f(x+R) - f(x)\big) R\sigma^{-2}.$$

PGO's expectation is the gradient of a discrete Gaussian approximation of f:

$$\mathbb{E}[g_{\text{PGO}}] = \nabla_x \sum_{r\in\mathbb{Z}^d} f(x+r)\, P(R{=}r)$$

We will now derive a variant of PGO with the same expectation, but per-dimension variance that is typically lower, and at most equal. For this, we express PGO's expectation over a single perturbation dimension i for a given realization of all other dimensions. For brevity, we express this conditioning using the function

$$f_{r_{-i}}(R_i) := f(r_1, \ldots, r_{i-1}, R_i, r_{i+1}, \ldots, r_d).$$

The expectation becomes:

$$\mathbb{E}[g_{\text{PGO},i}] = \mathbb{E}[(f_{-i}(x_i + R_i) - f(x))\sigma^{-2} R_i].$$

By the law of total expectation, equality of this per-variable expectation implies equality of the overall expectation. We now partition the integers into equivalence classes $[y] \subseteq \mathbb{Z}$ according to an equivalence relation $\sim$ to group the perturbations on dimension i. By the definition of expectation and since the equivalence classes are disjoint, we can write:

$$\begin{aligned}\mathbb{E}[g_{\text{PGO},i}] &= \sum_{r_i\in\mathbb{Z}} P(R_i{=}r_i)\big(f_{-i}(x_i+r_i) - f(x)\big)\sigma^{-2} r_i \\ &= \sum_{[r_i]\in\mathbb{Z}/\sim} \sum_{r_i'\in[r_i]} P(R_i{=}r_i')\big(f_{-i}(x_i+r_i') - f(x)\big)\sigma^{-2} r_i' \\ &= \sum_{[r_i]\in\mathbb{Z}/\sim} P(R_i \in [r_i])\,\mathbb{E}\left[\big(f_{-i}(x_i+R_i) - f(x)\big)\sigma^{-2} R_i \,|\, R_i \in [r_i]\right].\end{aligned}$$

The corresponding partial derivative oracle for dimension i is

$$g_{\text{PGO-DP},i} = \frac{1}{P(R_i' \in [R_i])} \sum_{r_i\in[R_i]} P(R_i'{=}r_i)\big(f_{-i}(x_i+r_i) - f(x)\big)\sigma^{-2} r_i,$$

where R_i' is a random variable independent of R_i that follows the same distribution.

Applying this oracle to all independently perturbed dimensions yields a gradient oracle that considers all per-dimension perturbations in an equivalence class at once, weighted using the known perturbation probabilities.

In practice, we restrict the coverage of alternative perturbations by a radius c around the primal perturbation for efficiency. If a primal perturbation falls outside the radius, we fall back to the original PGO estimator for the current input dimension. With appropriate scaling, limiting the coverage to parts of the encountered equivalence classes does not introduce any bias. Let $S([r_i]) \subseteq [r_i]$ denote the covered subset of the equivalence class $[r_i]$ and let

$$p_S = P(R_i' \in S([r_i]) \mid R_i' \in [r_i]) = \frac{P(R_i' \in S([r_i]))}{P(R_i' \in [r_i])}.$$

Scaling by p_S^{-1} preserves the expectation:

$$\begin{aligned}\mathbb{E}\Big[p_S^{-1} \sum_{r_i\in S([r_i])} P(R_i'{=}r_i)\big(f_{-i}(x_i+r_i) - f(x)\big)\sigma^{-2} r_i \,|\, R_i \in [r_i]\Big] = \\ \mathbb{E}\Big[\sum_{r_i\in[r_i]} P(R_i'{=}r_i)\big(f_{-i}(x_i+r_i) - f(x)\big)\sigma^{-2} r_i \,|\, R_i \in [r_i]\Big].\end{aligned}$$

Hence, the modified estimator $g_{\text{PGO-DP},S,i} =$

$$\frac{1}{P(R_i' \in S([R_i]))} \sum_{r_i\in S([R_i])} P(R_i'{=}r_i)\big(f_{-i}(x_i+r_i) - f(x)\big)\sigma^{-2} r_i$$

has the same expectation as $g_{\text{PGO-DP},i}$.

3.2 Variance Reduction

We now consider the variance reduction achieved by PGO-DP under full coverage of an equivalence class. Using the law of total variance, we can express the original PGO's variance on dimension i as

$$\begin{aligned}\text{Var}(g_{\text{PGO},i}) = \mathbb{E}\big[\text{Var}\big(\big(f_{-i}(x_i+R_i) - f(x)\big)R_i\sigma^{-2} | [R_i]\big)\big] + \\ \text{Var}\big(\mathbb{E}\big[\big(f_{-i}(x_i+R_i) - f(x)\big)R_i\sigma^{-2} | [R_i]\big]\big).\end{aligned}$$

The first summand is the expected variance *within* an equivalence class, the second summand is the variance of the expectation *across* equivalence classes. As PGO-DP yields the same value for any perturbation within the same equivalence class, its variance is only

$$\mathrm{Var}(g_{\text{PGO-DP},i}) = \mathrm{Var}(\mathbb{E}[\big(f_{-i}(x_i + R_i) - f(x)\big)R_i\sigma^{-2}|[R_i]]).$$

The per-dimension variance reduction ratio is thus

$$\frac{\mathrm{Var}(g_{\text{PGO},i})}{\mathrm{Var}(g_{\text{PGO-DP},i})} = 1 + \frac{\mathbb{E}[\mathrm{Var}(\big(f_{-i}(x_i + R_i) - f(x)\big)R_i\sigma^{-2}|[R_i])]}{\mathrm{Var}(\mathbb{E}[\big(f_{-i}(x_i + R_i) - f(x)\big)R_i\sigma^{-2}|[R_i]])}.$$

Note that this holds for any realization of the independent perturbations on other dimensions. If f involves additional random variables beyond R, as is the case in many simulations, the above argumentation applies by the substitutions $f(x) \mapsto f(x|\omega_0)$ and $f_{r_{-i}}(R_i) \mapsto f_{r_{-i}}(R_i|\omega_1)$, where $\omega_0, \omega_1 \in \Omega$ are tuples of realizations of the random variables and Ω their support.

An important case is $\mathbb{Z}/\sim = \{\mathbb{Z}\}$, i.e., all perturbations on a dimension fall into the same class. Then, the variance across classes and thus PGO-DP's total variance stemming from the perturbations is 0, whereas PGO retains the variance within the class.

The impact of the coverage radius c on the variance reduction is assessed in Section 4.3.

3.3 Control Flow Equivalence

The above derivation generically applies to any equivalence relation on the perturbations. However, attaining a variance reduction requires us to determine the function outputs for multiple perturbations in a class. We achieve this without explicit additional sampling by defining the equivalence relation to represent equivalence in terms of control flow. More formally, let $\mathrm{Path}(x) = \langle n_1, n_2, \ldots, n_k \rangle$ be the sequence of nodes in the given program's control-flow graph followed for an input vector x. Input-dependent loops are represented by repeated occurrences of the corresponding nodes in the path sequence. Similarly to f_{-i} above, we define

$$\mathrm{Path}_{-i}(r_i') = \mathrm{Path}\left(x + (r_1, \ldots, r_{i-1}, r_i', r_{i+1}, \ldots, r_d)\right)$$

This is the path followed given the primal perturbed input vector on all dimensions $j \neq i$ and the alternative perturbed input $x_i + r_i'$ on dimension i. We define two perturbations r_i, r_i' to be *control flow-equivalent* if $\mathrm{Path}_{-i}(r_i) = \mathrm{Path}_{-i}(r_i')$ For clarity, we note that control-flow equivalence does not imply equivalence in output, as illustrated by the following simple program snippet.

```
if x[0] > 0:
  return x[0]
...
```

Here, although any input `x[0]` > 0 follows the same control flow path, the output still varies with `x[0]`.

3.4 Implementation

To form a practicable gradient estimator from our definitions of PGO-DP and control flow equivalence, two requirements must be met: First, each program evaluation must be extended beyond the current primal perturbed input vector $x + r$ to alternative perturbations along each input dimension. Second, for each alternative perturbation r_i' along dimension i, control-flow equivalence with r_i must be determined. An efficient implementation of these mechanisms is essential to ensure that a faster convergence through reduced variance is not offset by the function evaluation overhead.

Although we describe our implementation using C++ constructs, the same ideas apply to other languages that support vectorization and operator overloading, such as Python and Julia.

3.4.1 Perturbed Arithmetic. We extend the execution of a program to several perturbations of the decision variables by altering the program to be evaluated on a *vector* per decision variable reflecting several perturbations. For instance, let the original three-dimensional input vector be $x = [3\ 1\ 5]$ and the primal perturbed input vector $x + r = [2\ 1\ 7]$. The range of considered alternative perturbations around each dimension of x is determined by the coverage radius c. For $c = 2$, the perturbed input vector is translated to the three vectors $[1\ \mathbf{2}\ 3\ 4\ 5]$, $[-1\ 0\ \mathbf{1}\ 2\ 3]$, $[3\ 4\ 5\ 6\ \mathbf{7}]$, bold numbers indicating the primal perturbed values. We now propagate the effects of the perturbations through the arithmetic operations of the program, during which new vectors are calculated that may depend on perturbed decision variables on one or more dimensions. For notational consistency, we denote the dependence on dimension i using a subscript $x[i-1]$, e.g., $[1\ \mathbf{2}\ 3\ 4\ 5]_{x[0]}$ for the first input dimension.

For unary operations such as absolute value, negation, and exponentiation, the arithmetic over perturbed variables trivially translates to element-wise arithmetic over all input dependencies and all perturbed values per dependency. Similarly, binary operations between a perturbed variable and a constant, or between two perturbed variables with the same dependencies translates to element-wise operations for the pairs of scalars corresponding to the same dependency and perturbation. For instance:

$$[1\ \mathbf{2}\ 3\ 4\ 5]_{x[0]} \times [3\ \mathbf{5}\ 7\ 9\ 11]_{x[0]} = [3\ \mathbf{10}\ 21\ 36\ 55]_{x[0]}$$

The output of a binary operation between two perturbed variables with differing dependencies is a new perturbed variable carrying the *union* of the decision variables' dependencies. Dependencies on the same decision variable are handled in an element-wise fashion as described above. For dependencies present in only one perturbed operand v_0 but not in another v_1, the binary operation is computed as the element-wise operation between v_0 perturbed values and v_1's scalar primal value. The following is an example of a multiplication between perturbed variables:

$$[1\ \mathbf{2}\ 3\ 4\ 5]_{x[0]} \times [-1\ 0\ \mathbf{1}\ 2\ 3]_{x[1]} = \begin{bmatrix} 1\ \mathbf{2}\ 3\ 4\ 5 \\ -2\ 0\ \mathbf{2}\ 4\ 6 \end{bmatrix}_{x[1]}$$

We implement the perturbed arithmetic by introducing a new data type `pfloat` (perturbed floating point number) that carries the scalar primal variable value together with the perturbed values. This approach bears superficial similarities to forward-mode AD, where variables are extended to include their tangents with respect to the program inputs. However, instead of tangents, dimensional peeking propagates alternative intermediate variable values under different input perturbations.

The `pfloat` type implements arithmetic operations and comparisons via operator overloading, allowing it to be used in the same

manner as C++'s primitive floating point types. The handling of dependencies on d decision variables with n perturbations each could be supported by carrying out element-wise arithmetic over an $d \times n$ matrix. However, in the common case of variables that depend on only a few decision variables, this dense representation would incur many redundant computations. We choose a sparse representation in which each `pfloat` carries a dynamically growing array holding one vector of perturbed values per dependency, and a single vector of pointers representing the dependencies. To avoid the cost of dynamic memory allocation, we employ a stack-based dynamic array type. While its interface is that of a C++ standard template library `vector`, the array reserves space for the maximum of d vectors on the stack and handles item insertion purely by incrementing a size variable and item assignment.

In our sparse representation, the output from a binary operations depends on the union of the decision variables' dependencies. We first search for a's dependencies in b to handle the intersection of the dependencies by an element-wise vector operation. Dependencies not found in b are handled as operations between a's perturbation vector and b's primal value. In doing so, we mark the found intersecting dependencies in b's dependency array. The unmarked dependencies are present only in b and are handled as operations between a' primal value and b's perturbation vector.

As the dependency arrays are unordered, the time complexity of this process is in $O(mn)$ time, m and n being the lengths of a and b's dependency arrays. Several simple heuristics are applied to maintain low cost:

- If one of the operator has no dependencies, the operation is handled as an operation between a `pfloat` and a scalar.
- If b has more dependencies than a, we swap the search order to reduce the cost of checking for unmarked operations.
- To accelerate operations between operands with similar dependencies, we check for dependencies at identical indexes first before resorting to linear search.

In practice, the cost depends not only on the number of dependencies and perturbations, but also on the proportion of operations on `pfloat` instances in the program. The overhead for three simulation models from the literature is evaluated in Section 4.

3.4.2 Determining Control Flow-Equivalence. In a deterministic imperative program (and, equivalently, in a stochastic program with fixed realizations of all random variables) the observed path depends solely on input-dependent control flow in the form of parameter-dependent branching, looping, and jumps. The parameter dependence manifests in conditional expressions that are functions of input parameters. Hence, to establish control flow equivalence between r_i' and r_i throughout a program execution based on r_i, it suffices to ensure that all encountered conditional expressions evaluate to the same truth value for r_i' as for r_i.

As described above, the `pfloat` type contains a dynamic array of pointers to its dependencies. More specifically, each element of the array points to a vector of boolean values representing control flow-equivalence with the primal perturbation for a decision variable. Before evaluating the program, all booleans are initialized to `true`. On each comparison involving a `pfloat`, control flow-equivalence is determined for each dependency, and the booleans are updated accordingly. A comparison operator $\mathrm{cmp}(a, b)$, where a is a `pfloat` and b a primitive numerical type, is overloaded according to pseudo code shown in Algorithm 1.

Here, the truth value of the comparison for the primal value is delegated to an ordinary scalar-scalar comparison. Subsequently, vectors of truth values are gathered by vectorized comparisons for the perturbation vectors associated with each dependency. After each vectorized comparison, the boolean vector of control flow-equivalences is updated. The program execution continues for all perturbations, but finally, only the perturbations identified as control flow-equivalent with the primal perturbation contribute to the gradient estimate, scaled by the perturbation probabilities.

To illustrate an overall program execution under dimensional peeking, we consider the following program snippet:

```
y = x[0] * (2 * x[1] + x[2])
if y < 20:
    ...
```

Figure 2 shows the vectorized arithmetic and identification of control flow-equivalence for the input vector $x = [2\ 1\ 7]$.

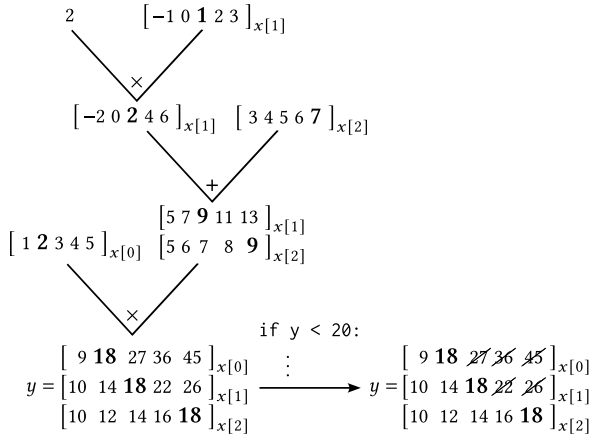


Figure 2: Dimensional peeking over arithmetic and a conditional branch evaluated at the primal perturbed input $x = [2\ 1\ 7]$. Numbers in brackets represent the variable values under different perturbations, with the primal value in bold. Subscripts indicate the decision variables the value depends on. Evaluating the conditional statement `if y < 20` rules out values on two of the three dimensions (crossed-out numbers).

Algorithm 1 Overloaded comparison operator on `pfloat`.

```
function CMP(pfloat a, float b)
    t_primal ← cmp(a.primal, b) // scalar-scalar comparison
    for d ← 0 to |a.value_vecs| − 1 do // iterate dependencies
        // vector-scalar comparison:
        t_perturbations ← (cmp(a.value_vecs[d], b) == t_primal)
        // update control-flow equivalences:
        a.equivalence_vecs[d] ← a.equivalence_vecs[d] &
                                t_perturbations
    return t_primal
```

4 Experiments

Our experiments study the variance reduction achieved using dimensional peeking, its execution time overhead, and finally its benefits when solving three optimization problems in comparison to gradient descent via the original PGO estimator and three popular meta-heuristics.

All execution time measurements employ the same codebase for PGO and PGO-DP, relying on C++ templating to switch between native floating point numbers and our `pfloat` type. The perturbed arithmetic is carried out in a vectorized manner via single-instruction, multiple-data operations implemented via the Fastor library [28]. The baselines in the optimization experiments include the genetic algorithm (GA) implementation of the pyeasyga module[2], particle swarm optimization (PSO) via the PySwarms toolkit [22], and covariance matrix adaptation evolution strategy (CMA-ES) [6] via the pycma module[3]. All baselines rely on native floating point numbers.

The experiments were carried out on a machine equipped with an AMD EPYC 7742 CPU and 256GiB RAM running Ubuntu 20.04.6.

4.1 Optimization Problems

We consider SBO problems using three models from the literature:

CityFlow is a microscopic traffic simulator targeted towards large-scale scenarios and traffic control applications [36]. The simulation engine is written in C++, allowing us to apply dimensional peeking by introducing our `pfloat` data type. We consider a traffic control problem over a scenario covering 4×4 intersections in Hangzhou, China[4]. The objective is the average traveled distance. The simulation involves 144 decision variables representing traffic light phase durations in seconds. Stochasticity is introduced by applying uniform random offsets in $\{0, \ldots, 10\}$s to the trip start times.

Hotel is a revenue maximization problem from the SimOpt suite of SBO problems [9]. The goal is to maximize a hotel's revenue by adjusting the numbers of available "products" comprised of sequences of days in a week and two possible fares. The revenue is counted for one week after one week of warmup. Customers arrive randomly according to product-specific Poisson processes. As the products overlap in time, each successful booking reduces other products' availability. The problem involves 56 decision variables.

DynamNews is also part of the SimOpt suite and represents a newsvendor problem with dynamic demand. Each arriving customer assigns a score to each available product as a sum of a constant product-specific utility and a Gumbel random variable. Each customer chooses the product with the highest utility among the products still in stock. The objective is the overall revenue given by the sum of the sold products' prices, subtracting their cost. The decision variables are the products' prices and the initial inventory levels. We configure the constants according to a setting from the SimOpt web site[5], extending it to 3 000 customers and 1 000 products, corresponding to 1 000 decision variables.

[2]https://github.com/remiomosowon/pyeasyga
[3]https://github.com/CMA-ES
[4]https://traffic-signal-control.github.io/
[5]https://simopt.readthedocs.io/en/development/models/dynamnews.html

A key difference among the problems is that in CityFlow and Hotel, the objective function value depends purely on control flow, i.e., if two sets of decision variable values follow the same control flow, the output is identical. Hence, as discussed in Section 3.1, PGO-DP reduces the variance to that across classes of control-flow equivalent decision variable values. In contrast, perturbations to the products' prices in DynamNews directly propagate to the objective value, even if the control flow is unaffected.

The reported measurements regarding correctness, variance reduction and gradient estimation times are averages over 10^5 repetitions for CityFlow and DynamNews, and 10^6 for Hotel. Optimization curves show averages across 30 replications starting from random parameter combinations for each combination of model and optimizer hyperparameters.

4.2 Verification

We verified the correctness and numerical stability of our implementation by comparing derivative estimates between PGO and PGO-DP. Table 1 shows the mean difference between the estimates with $\sigma = 1$ across all input dimensions with 99% confidence intervals. Setting a coverage radius of $c = 15\sigma$ reflects full coverage of the perturbation probabilities representable as a single precision floating point number. In line with the theoretical results from Section 3.1 showing unbiasedness with respect to the smoothed objective, the statistical results do not indicate a deviation of the estimators' expectations, independently of the coverage radius.

To verify that our implementation of PGO-DP achieves the variance reduction characterized analytically in Section 3.2, we estimate smoothed derivatives of the Heaviside step function

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}.$$

As stated in Section 3.2, PGO's variance can be expressed as the sum of the expected variance *within* a class of control flow-equivalent values ($x + R < 0$ or $x + R \geq 0$) and the variance of the expectation *across* classes. With PGO-DP and sufficiently large coverage radius

Table 1: Verification of our implementation. The percentage difference between PGO and PGO-DP does not indicate a bias at the 1% significance level for any coverage radius c.

c	CityFlow	Hotel	DynamNews
1σ	-0.023 ± 0.217	0.014 ± 0.107	-0.044 ± 0.101
3σ	0.133 ± 0.199	-0.011 ± 0.090	0.012 ± 0.096
5σ	0.011 ± 0.197	0.016 ± 0.090	0.001 ± 0.096
$\mathbf{15\sigma}$	**−0.082 ± 0.196**	**−0.021 ± 0.090**	**0.006 ± 0.095**

Table 2: Analytical and measured variance reduction rate (VRR) between PGO and PGO-DP for the Heaviside function. Larger smoothing factors σ yield a larger VRR.

σ	Expected in-class var.	Var. across class means	Analytical VRR	Measured VRR
1	0.069	0.327	1.212	1.203
2	0.122	0.232	1.525	1.524
4	0.151	0.193	1.781	1.799
8	0.166	0.176	1.946	1.939

Table 3: Variance reduction ratios for the simulation models, varying the coverage radius c.

c	CityFlow	Hotel	DynamNews
1σ	1.36	1.66	1.17
3σ	**2.55**	**7.53**	**1.45**
5σ	2.62	7.88	1.46
15σ	**2.62**	**7.90**	**1.44**

Table 4: Slowdown factors, varying the coverage radius c.

c	CityFlow	Hotel	DynamNews
1σ	1.11 ± 0.00	1.26 ± 0.00	1.22 ± 0.00
3σ	**1.09 ± 0.00**	**1.28 ± 0.00**	**1.22 ± 0.00**
5σ	1.12 ± 0.00	1.23 ± 0.00	1.35 ± 0.00
15σ	**1.12 ± 0.00**	**1.43 ± 0.00**	**1.39 ± 0.00**

c, the in-class variance is eliminated. We analytically determined the two summands for $x = 0$ with $c = 15\sigma$ and different smoothing factors σ based on the known perturbation probabilities (cf. Appendix A) and compared the results to measurements via PGO and PGO-DP over 10^5 estimations. The results in Table 2 show that the measured variance reduction ratio (VRR) is in line with the analytical results. With larger smoothing factor σ, the in-class variance increases, whereas the variance across class means decreases, yielding a larger VRR.

4.3 Impact of Coverage Radius

In contrast to the estimates' expectation, the variance reduction achieved by PGO-DP is affected by the coverage radius c. Table 3 shows the VRR for the three simulation models used in our optimization experiments using $\sigma = 1$. The variance reduction is substantial for all models, the largest VRR being 7.9 for the Hotel model. Beyond $c = 3\sigma$, only modest increases in VRR are observed, making this value sufficient for the optimization experiments of Section 4.4.

The coverage radius also affects PGO-DP's computational cost. Table 4 shows that since c affects the size of the vectors involved in the perturbed arithmetic, the overhead increases moderately with larger c. When extending the coverage to the full support of the perturbations up to machine precision by setting $c = 15\sigma$, the maximum overhead is 43% for Hotel. With $c = 3\sigma$, the maximum overhead is 28%. The comparatively larger overhead for Hotel and DynamNews is explained by their larger proportion of operations on `pfloat` variables.

We also measured the overhead in memory consumption, which depends on the number of resident `pfloat` values during a simulation run and the size of each `pfloat` according to the coverage radius. For CityFlow, memory consumption increased by factors of 2.2 to 2.6 with $c = 1\sigma$ and $c = 15\sigma$ compared to PGO. In contrast, Hotel and DynamNews showed increases of 2% or less.

4.4 Optimization Progress

To evaluate the effects of PGO-DP when solving overall optimization problems, we compare its convergence behavior against four baseline methods: gradient descent using PGO, genetic algorithm

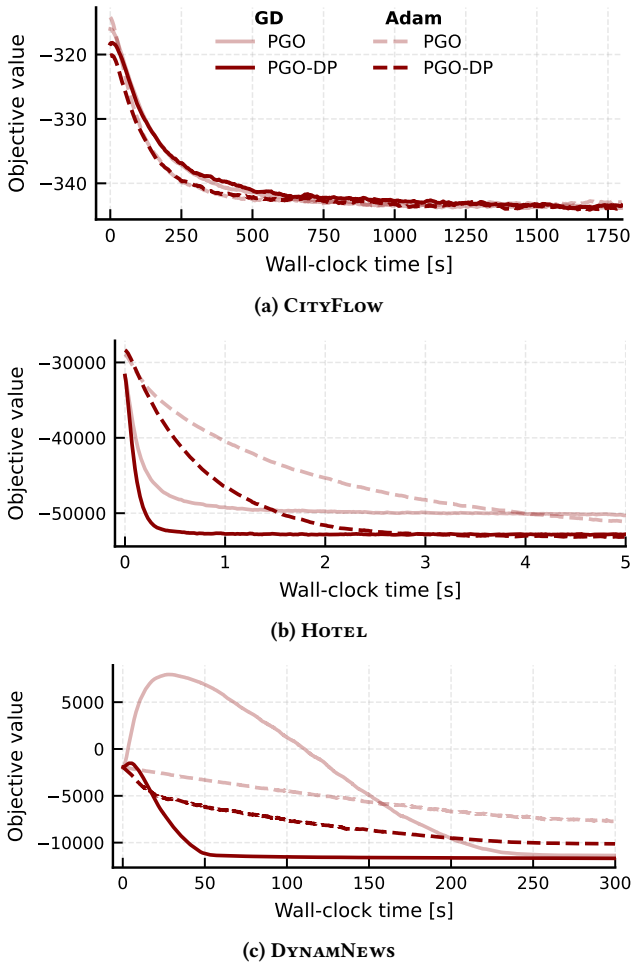


Figure 3: Comparison of the optimization progress over time between PGO and PGO-DP with $\sigma = 1$ and learning rates of 0.01 and 0.1 for SGD and Adam. For Hotel and DynamNews, PGO-DP significantly outperforms PGO.

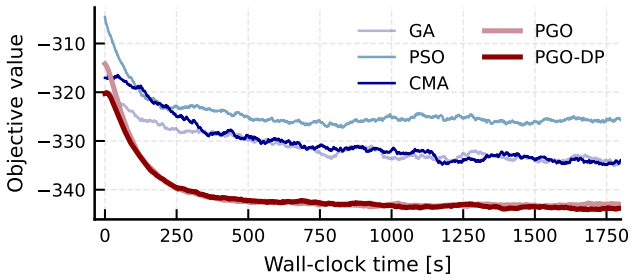
(GA), particle swarm optimization (PSO), and covariance matrix adaptation evolution strategy (CMA-ES).

To account for the optimizers' dependence on hyperparameters, we conducted a limited parameter sweep. For GA, we configured a population size of 30 and enabled elitism. The mutation function shifts a decision variable by a Gaussian random variable with $\mu = 0$ and $\sigma \in \{1, 2, 4\}$. PSO uses 30 particles and the parameters $c_1 = 1.5, c_2 = 1.5, w \in \{0.4, 0.65, 0.9\}$. For CMA-ES, the standard deviation was set to 1, 2, and 4. Similarly, for PGO and PGO-DP, we set the smoothing factor σ to 1, 2, and 4. The gradient estimates are applied in traditional gradient descent (GD) and using the Adam optimizer [1] with learning rates in $\{0.001, 0.05, 0.01\}$ and $\{0.01, 0.05, 0.1\}$.

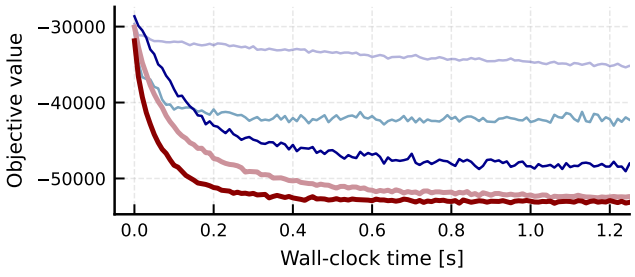
As the execution time per function evaluation and the number of evaluations per optimization step differ between the methods, we plot the optimization progress over wall-clock time. All problems are cast as minimization problems by negating the objective values.

We first compare PGO and PGO-DP using the same smoothing factor of $\sigma = 1$ and learning rates of 0.01 and 0.1 for GD and Adam. Figure 3 shows no substantial difference in convergence between PGO and PGO-DP in the CityFlow problem. Although Table 3 showed a VRR of 2.6, the absolute variance of CityFlow is around three orders of magnitude lower than that of the other problems. We conjecture that for this reason, the optimization progress is not sensitive to further variance reductions.

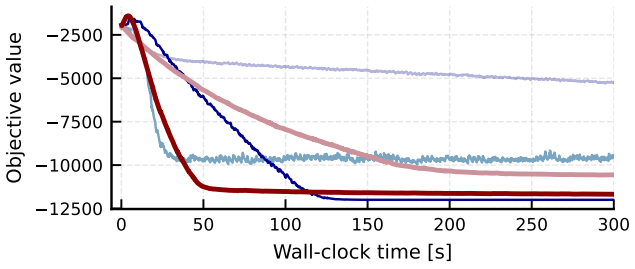
In contrast, both Hotel and DynamNews benefit strongly from the reduced variance, both with GD and Adam. For DynamNews, we observe that initially, GD is vastly misled by PGO's higher-variance estimates, leading to a strong decline in solution quality. With PGO-DP, a minuscule initial decline is observed, after which the solution quality improves quickly. The solution quality at the end of the time budget is consistently higher with PGO-DP.



(a) CityFlow



(b) Hotel



(c) DynamNews

Figure 4: Optimization progress over time with the best hyperparametrization for each method. Gradient descent via PGO and PGO-DP excels at Hotel and is competitive at DynamNews. PGO-DP consistently outperforms PGO.

Table 5: Speedup of PGO-DP over PGO in approaching the best found objective value. PGO never reached 99% of PGO-DP's improvement over the starting solution.

Simulation model	75%	90%	95%	99%
CityFlow	1.02	0.96	0.96	-
Hotel	2.60	2.77	3.09	-
DynamNews	4.10	-	-	-

Finally, we plot the results with the best-performing hyperparameter combination for PGO-DP and all baselines in Figure 4. We define the better optimization performance as achieving a lower area under the optimization curve. For PGO and PGO-DP, we report results with the better-performing optimizer, GD or Adam. In the CityFlow problem, PGO and PGO-DP outperform the meta-heuristic baselines, but comparing PGO and PGO-DP, no significant improvement is observed.

For Hotel, we constrain the time axis to more clearly show the initial optimization steps, during which most of the progress occurs. Here, PGO-DP achieves the fastest convergence and reaches the best solution quality of all methods. The objective value of -50 000 is reached more than a factor 2 sooner than with PGO. Both PGO-DP and PGO substantially outperform the meta-heuristics.

In DynamNews, PSO and PGO-DP achieved the fastest initial improvement, with PGO-DP converging to a much better solution quality after about 50s. CMA-ES required about 125s to reach convergence, but achieved a slightly better solution quality. As we saw in Figure 3, the lower variance of PGO-DP allows for fast convergence at higher learning rates compared to PGO. Accordingly, the best-performing learning rate for PGO-DP was 0.001 using GD, whereas PGO performed best with a learning rate of 0.01.

In Table 5, we quantify the faster optimization progress with PGO-DP by comparing the times after which a certain improvement over the initial solution quality has been reached. Using PGO-DP as the reference, the percentage improvement at optimization step s is computed as $\frac{100\,y_s}{y_\Omega - y_A}$, where y_s, y_A, and y_Ω are the objective values at step s, the beginning, and the end of PGO-DP's optimization trajectory. We compare the best-performing hyperparametrization of PGO and PGO-DP for each problem, including the choice of optimizer. As previously observed in Figure 3, the results for CityFlow differ only slightly between PGO and PGO-DP. For the other problems, the differences in solution qualities are more substantial, and PGO-DP reaches almost all improvement levels significantly faster, the largest speedup being 4.1 at 75% improvement for DynamNews. The baseline PGO was not able to reach 99% of PGO-DP's improvement within the time budget for any of the models.

5 Conclusions

We presented dimensional peeking, a novel variance reduction method for zeroth-order discrete optimization via simulation. Our experiments showed that at an execution time overhead between 9% and 28%, dimensional peeking reduces the variance of gradient estimates for three simulation-based optimization problems by factors between 1.5 to 7.5 over estimations on the scalar level. In two of the three problems, the reduced variance translated to substantially improved convergence behavior. Overall, the presented approach

improves the competitiveness of zeroth-order optimization via gradient descent as compared to GA, PSO, and CMA-ES.

Our future work aims to extend the scope of dimensional peeking beyond gradient descent alone. One promising avenue is its integration into optimization methods for non-convex problems. For instance, trust-region algorithms such as TRON [20] and ASTRO-DF [31] could benefit from low-variance gradient estimates when constructing local approximations of the objective function.

Finally, outside of static parameter optimization, dimensional peeking could accelerate the training of reinforcement learning policies over discrete action spaces by enhancing or substituting established high-variance gradient estimator such as REINFORCE [35].

Acknowledgments

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031).

References

[1] Kingma DP Ba J Adam et al. 2014. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* 1412, 6 (2014).
[2] Philipp Andelfinger. 2023. Towards differentiable agent-based simulation. *ACM Transactions on Modeling and Computer Simulation* 32, 4 (2023), 1–26.
[3] Philipp Andelfinger and Wentong Cai. 2025. Slight Stochastic Shifts Suffice: Cross-Trajectory Vectorized Estimation of Simulation Gradients. In *39th ACM SIGSIM Principles of Advanced Discrete Simulation (PADS)*. https://dl.acm.org/doi/10.1145/3726301.3728410
[4] Philipp Andelfinger and Justin N Kreikemeyer. 2024. Automatic gradient estimation for calibrating crowd models with discrete decision making. In *International Conference on Computational Science*. Springer, 227–241.
[5] Gaurav Arya, Moritz Schauer, Frank Schäfer, and Christopher Rackauckas. 2022. Automatic differentiation of programs with discrete randomness. *Advances in Neural Information Processing Systems* 35 (2022), 10435–10447.
[6] Anne Auger and Nikolaus Hansen. 2012. Tutorial CMA-ES: evolution strategies and covariance matrix adaptation. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 827–848.
[7] Krishnakumar Balasubramanian and Saeed Ghadimi. 2022. Zeroth-order nonconvex stochastic optimization: Handling constraints, high dimensionality, and saddle points. *Foundations of Computational Mathematics* 22, 1 (2022), 35–76.
[8] Ayush Chopra, Jayakumar Subramanian, Balaji Krishnamurthy, and Ramesh Raskar. 2023. Agenttorch: Agent-based modeling with automatic differentiation. In *Second Agent Learning in Open-Endedness Workshop*.
[9] David J Eckman, Shane G Henderson, and Sara Shashaani. 2023. SimOpt: A testbed for simulation-optimization experiments. *INFORMS Journal on Computing* 35, 2 (2023), 495–508.
[10] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. 2018. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. *Advances in neural information processing systems* 31 (2018).
[11] Yasong Feng and Tianyu Wang. 2023. Stochastic zeroth-order gradient and Hessian estimators: variance reduction and refined bias bounds. *Information and Inference: A Journal of the IMA* 12, 3 (2023), 1514–1545.
[12] Michael C Fu. 2006. Gradient estimation. *Handbooks in operations research and management science* 13 (2006), 575–616.
[13] Michael C Fu et al. 2015. *Handbook of simulation optimization*. Vol. 216. Springer.
[14] Saeed Ghadimi and Guanghui Lan. 2013. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization* 23, 4 (2013), 2341–2368.
[15] Wei-Bo Gong and Yu-Chi Ho. 1987. Smoothed (conditional) perturbation analysis of discrete event dynamical systems. *IEEE Trans. Automat. Control* 32, 10 (1987), 858–866.
[16] Kaiyi Ji, Zhe Wang, Yi Zhou, and Yingbin Liang. 2019. Improved zeroth-order variance reduced algorithms and analysis for nonconvex optimization. In *International conference on machine learning*. PMLR, 3100–3109.
[17] David Kozak, Cesare Molinari, Lorenzo Rosasco, Luis Tenorio, and Silvia Villa. 2023. Zeroth-order optimization with orthogonal random directions. *Mathematical Programming* 199, 1 (2023), 1179–1219.
[18] Justin N Kreikemeyer and Philipp Andelfinger. 2023. Smoothing methods for automatic differentiation across conditional branches. *IEEE Access* (2023).
[19] Jeffrey Larson, Matt Menickelly, and Stefan M Wild. 2019. Derivative-free optimization methods. *Acta Numerica* 28 (2019), 287–404.
[20] Chih-Jen Lin and Jorge J Moré. 1999. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization* 9, 4 (1999), 1100–1127.
[21] Charles C Margossian. 2019. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9, 4 (2019), e1305.
[22] Lester James Miranda. 2018. PySwarms: a research toolkit for Particle Swarm Optimization in Python. *Journal of Open Source Software* 3, 21 (2018), 433.
[23] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. 2021. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*. PMLR, 7805–7817.
[24] Yurii Nesterov and Vladimir Spokoiny. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics* 17, 2 (2017), 527–566.
[25] Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. 2024. A review of differentiable simulators. *IEEE Access* (2024).
[26] Felix Petersen, Christian Borgelt, Aashwin Mishra, and Stefano Ermon. 2024. Generalizing stochastic smoothing for differentiation and gradient estimation. *arXiv preprint arXiv:2410.08125* (2024).
[27] Roman A Polyak et al. 2021. *Introduction to continuous optimization*. Vol. 172. Springer.
[28] Roman Poya, Antonio J Gil, and Rogelio Ortigosa. 2017. A high performance data parallel tensor contraction framework: Application to coupled electro-mechanics. *Computer Physics Communications* 216 (2017), 35–52.
[29] Christian P Robert, George Casella, and George Casella. 1999. *Monte Carlo statistical methods*. Vol. 2. Springer.
[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.
[31] Sara Shashaani, Fatemeh S Hashemi, and Raghu Pasupathy. 2018. ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free stochastic optimization. *SIAM Journal on Optimization* 28, 4 (2018), 3145–3176.
[32] James C Spall. 2002. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Autom. Control* 37, 3 (2002), 332–341.
[33] Eylem Tekin and Ihsan Sabuncuoglu. 2004. Simulation optimization: A comprehensive review on theory and applications. *IIE trans.* 36, 11 (2004), 1067–1081.
[34] Long Wang, Qi Wang, James C Spall, and Jingyi Zhu. 2025. Simultaneous perturbation stochastic approximation for mixed variables. *IEEE Trans. Automat. Control* (2025).
[35] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.
[36] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. 2019. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*. 3620–3624.
[37] Haixiang Zhang, Zeyu Zheng, and Javad Lavaei. 2023. Gradient-based algorithms for convex discrete optimization via simulation. *Operations research* 71, 5 (2023), 1815–1834.
[38] Yaofeng Desmond Zhong, Jiequn Han, and Georgia Olympia Brikis. 2022. Differentiable physics simulations with contacts: Do they have correct gradients wrt position, velocity and control? *arXiv preprint arXiv:2207.05060* (2022).

A Variance Reduction for Heaviside Step Function

In Section 4.2, we verified our implementation by comparison to analytical determined variance reductions for the Heaviside step function H. Here, we briefly show the steps to compute the references values for derivative estimates at $x = 0$. Let $R \sim \text{Discrete-}\mathcal{N}(0, \sigma^2)$. The probabilities for negative and positive perturbations are:

$$p = \sum_{k=-\infty}^{-1} \frac{\exp(-k^2/(2\sigma^2))}{\sum_{m=-\infty}^{\infty} \exp(-m^2/(2\sigma^2))},$$

$$q = 1 - p.$$

As $H(x) - H(0) = 0 \quad \forall x \in \mathbb{N}^+$, positive perturbations do not contribute to the derivative estimates. The mean and variance in the negative class are:

$$\mu_{<0} = \frac{1}{p} \sum_{k=-\infty}^{-1} (-k) \frac{\exp(-k^2/(2\sigma^2))}{\sum_{m=-\infty}^{\infty} \exp(-m^2/(2\sigma^2))},$$

$$\sigma^2_{<0} = \frac{1}{p} \sum_{k=-\infty}^{-1} k^2 \frac{\exp(-k^2/(2\sigma^2))}{\sum_{m=-\infty}^{\infty} \exp(-m^2/(2\sigma^2))} - \mu^2_{<0}.$$

Decomposing the total variance into the in-class variance and the variance of the expectation across classes, we obtain the variance reduction rate (VRR):

$$\mathrm{Var}(\mathrm{PGO}) = p\sigma^2_{<0} + pq\mu^2_{<0},$$

$$\mathrm{VRR} = 1 + \frac{p\sigma^2_{<0}}{pq\mu^2_{<0}}.$$