Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **200X**

doi:10.3934/xx.xxxxxxx

pp. **X–XX**

ACCELERATING REINFORCEMENT LEARNING WITH A DIRECTIONAL-GAUSSIAN-SMOOTHING EVOLUTION STRATEGY

Jiaxin Zhang

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Hoang Tran

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Guannan Zhang*

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Abstract. The objective of reinforcement learning (RL) is to find an optimal strategy for solving a dynamical control problem. Evolution strategy (ES) has been shown great promise in many challenging reinforcement learning (RL) tasks, where the underlying dynamical system is only accessible as a black box such that adjoint methods cannot be used. However, existing ES methods have two limitations that hinder its applicability in RL. First, most existing methods rely on Monte Carlo based gradient estimators to generate search directions. Due to low accuracy of Monte Carlo estimators, the RL training suffers from slow convergence and requires more iterations to reach the optimal solution. Second, the landscape of the reward function can be deceptive and may contain many local maxima, causing ES algorithms to prematurely converge and be unable to explore other parts of the parameter space with potentially greater rewards. In this work, we employ a Directional Gaussian Smoothing Evolutionary Strategy (DGS-ES) to accelerate RL training, which is well-suited to address these two challenges with its ability to (i) provide gradient estimates with high accuracy, and (ii) find nonlocal search direction which lays stress on large-scale variation of the reward function and disregards local fluctuation. Through several benchmark RL tasks demonstrated herein, we show that the DGS-ES method is highly scalable, possesses superior wall-clock time, and achieves competitive reward scores to other popular policy gradient and ES approaches.

2020 *Mathematics Subject Classification.* Primary: 37M05, 93E35; Secondary: 60J25.

Key words and phrases. Reinforcement learning, Gaussian smoothing, non-convex optimization, stochastic control, high-dimensional optimization, Guass-Hermite quadrature.

* Corresponding author: Guannan Zhang (zhangg@ornl.gov).

1. **Introduction.** Reinforcement learning is a class of problems which aim to find, through trial and error, a feedback policy that prescribes how an agent should act in an uncertain, complex environment to maximize some notion of cumulative reward [30]. RL can be viewed as a generalization of stochastic optimal control. Traditionally, RL algorithms have mainly been employed for small input and action spaces, and suffered difficulties when scaling to high-dimensional problems. With the recent emergence of deep learning, powerful non-linear function approximators such as deep neural networks (DNN) can be integrated into RL and extend the capability of RL in a variety of challenging tasks which would otherwise be infeasible, ranging from playing Atari from pixels [18], playing expert-level Go [28] to robotic control [2]. Among the most popular current deep RL algorithms are Q-learning methods, policy gradient methods, and evolution strategies. Deep Q-learning algorithms [18] use a DNN to approximate the optimal Q function, yielding policies that, for a given state, choose the action that maximizes the Q-value. Policy gradient methods [26] improve the policies with a gradient estimator obtained from sample trajectories in action space, examples of which are A3C [17], TRPO [25] and PPO [24].

This work concerns RL techniques based on evolution strategies. ES refers to a family of blackbox optimization algorithms inspired by ideas of natural evolution, often used to optimize functions when gradient information is inaccessible. This is exactly the prominent challenge in a typical RL problem, where the environment and policy are usually nonsmooth or can only be accessed via noisy sampling. It is not totally surprising ES has become a convincing competitor to Q-learning and policy gradient in deep RL in recent years. Unlike policy gradient, ES perturbs and performs policy search directly in the parameter space to find an effective policy, which is now generally considered to be superior to action perturbation [27]. The policy search can be guided by a surrogate gradient [23], completely population-based and gradient-free [29], and hybridized with other exploration strategies such as novelty search and quality diversity [7]. It has been shown in those works that ES is easy to parallelize, and requires low communication overhead in a distributed setting. More importantly, being able to achieve competitive performance on many RL tasks, these methods are advantageous over other RL approaches in their high scalability and substantially lower wall-clock time. Given wide availability of distributed computing resources, all environment simulations at each iteration of training can be conducted totally in parallel. Thus, a more reasonable metric for the performance of a training algorithm is the number of non-parallelizable iterations, as opposed to the sample complexity. In this metric, ES is truly an appealing choice.

Nevertheless, there are several challenges that need to be addressed in order to further improve the performance of ES in training complex policies. First, most ES methods cope with the non-smoothness of the objective function by considering a Gaussian-smoothed version of the expected total reward. The gradient of this function is intractable and must be estimated to provide the policy parameter update. In the pioneering work [23], a gradient estimator is proposed based on random parameter sampling. Developing efficient sampling strategies for gradient estimates has become an interest in ES research since then, and several improvements have been proposed, based on imposing structures on parameter perturbation [5], or reusing past evaluations, [6, 15, 16]. Yet, most of these gradient estimators are of Monte Carlo type, therefore arguably affected by the low accuracy of Monte Carlo methods. For faster convergence of training (i.e., reducing the number of iterations), more accurate gradient estimators are desired, particularly in RL tasks where the

policy has a large number of parameters to learn. Another prominent challenge is that the landscape of the objective function is complex and possesses plentiful local maxima. There is a risk for any optimization algorithm to get trapped in some of those points and unable to explore the parameter space effectively. The Gaussian smoothing, with its ability to smooth out function and damps out small, insignificant fluctuations, is a strong candidate in this very challenge. Specifically, with a moderately large smoothing parameter (i.e., strong smoothing effect), we can expect the gradient of the smoothed objective function will be able to look outside unimportant variation in the adjacent area and detect the general trend of the function from a distance, therefore an efficient *nonlocal* search direction. This potential of Gaussian smoothing, however, has not been explored in reinforcement learning.

In this paper, we propose a new strategy to accelerate the time-to-solution of reinforcement learning by exploiting the Directional Gaussian Smoothing Evolution Strategy (DGS-ES) method, developed in our recently work [33]. The DGS-ES method introduced a new directional Gaussian smoothing (DGS) gradient operator, that smooths the original objective function only along d-orthogonal directions in the parameter space. In other words, the DGS gradient requires d one-dimensional Gaussian convolutions, instead of one d-dimensional convolution in the existing ES methods. There are several advantages of using the DGS gradient operator in reinforcement learning. First, each component of the DGS gradient, represented as a one-dimensional integral, can be accurately approximated with various classic numerical integration techniques. When having Gaussian kernels, we use Gauss-Hermite quadrature rule which can provide spectral accuracy (see [1]) in the DGS gradient approximation. Second, the use of Gauss-Hermite quadrature also features embarrassing parallelism as the random sampling used in existing ES methods. Since the communication cost between computing processors/cores is neglectable, the total computing time for each iteration of training does not increase with the number of environment simulations given sufficient computing resources. Third, the directional smoothing approach enables nonlocal exploration which takes into account large variation of the objective function and disregards local fluctuations. This property will greatly help skipping local optima or saddle points during the training. It is demonstrated in §4 that the proposed strategy can significantly reduce the number of iterations in training several benchmark reinforcement learning problems, compared to the state-of-the-art baselines.

The rest of the paper is organized as follows: the RL problem under consideration and a brief review about the classic Gaussian smoothing technique is given in Section 2, the DGS gradient and the corresponding DGS-ES algorithm is introduced in Section 3, extensive numerical experiments including tests on benchmark optimization problems and on several benchmark RL problems are provided in Section 4, some concluding remarks are given in Section 5.

2. Problem setting.

We study the continuous-time, continuous-state stochastic control problem via RL. The evolution of the state $\boldsymbol{s}_t \in \mathcal{S}$ (the state space) depends on the control (i.e., the action in RL) $\boldsymbol{a}_t \in \mathcal{A}$ (the action space), by a stochastic differential equation, i.e., the state dynamics,

$$d\boldsymbol{s}_t = b(\boldsymbol{s}_t, \boldsymbol{a}_t)dt + \sigma(\boldsymbol{x}_t, \boldsymbol{a}_t)dw_t, \tag{1}$$

where b is the local drift, σ is the local diffusion, and w_t is the standard Brownian motion. Let $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ denote a control strategy (i.e., the policy in RL) that is a

conditional probability distribution of the action $\boldsymbol{a}_t$ given the current state $\boldsymbol{s}_t$. Then the objective of RL is to maximize a functional $\mathcal{J}$ of the policy π, i.e.,

$$\mathcal{J}(\pi) := \mathbb{E}_{\pi}\left[\int_0^T \gamma_t r(\boldsymbol{s}_t, \boldsymbol{a}_t) dt\right],$$

where T is the terminal time, $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the reward and $0 \leq \gamma \leq 1$ is a discount rate. After applying a temporal discretization scheme to the state process in (1), the continuous objective functional $\mathcal{J}(\pi)$ can be approximated by a discretized functional,

$$J(\pi) := \sum_{n=0}^{N} \mathbb{E}_{(\boldsymbol{s}_{t_n}, \boldsymbol{a}_{t_n})}\left[\gamma^{t_n} r(\boldsymbol{s}_{t_n}, \boldsymbol{a}_{t_n})\right], \tag{2}$$

where $t_n = n\Delta t$ for $n = 0, \ldots, N$.

In policy-based RL, the policy $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta})$ is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$, where the vector $\boldsymbol{\theta} := (\theta_1, \ldots, \theta_d)^\top$ represents the parameters of the policy, e.g., weights of a neural network. Then, the task of learning a good policy π becomes iterative updating the parameter $\boldsymbol{\theta}$ to solve the following optimization problem

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^d} J(\boldsymbol{\theta}), \tag{3}$$

where we denote $J(\boldsymbol{\theta}) := J(\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}))$ by an abuse of notation. The main challenges for the RL problem under consideration lies in three aspects. First, it is usually true that the local gradient of the environment $\mathcal{S}$ is inaccessible, so automatic differentiation or adjoint methods cannot be used to obtain the local gradient of $J(\boldsymbol{\theta})$. Thus, much of the innovation in reinforcement learning algorithms is focused on addressing the lack of access to or the existence of gradients of the environment/policy. Second, the landscape of $J(\boldsymbol{\theta})$ are usually highly non-convex and multi-modal, such that the local gradient (if available) may lead to an unsatisfactory local maximum of $J(\boldsymbol{\theta})$. Third, the dimension d of the parameters are usually on the order of hundreds or thousands (e.g., when using a neural network to define the policy), which makes many existing optimization methods inapplicable. These challenges motivated us to develop a nonlocal gradient operator and the corresponding approximation for the RL problem under consideration.

2.1. **The standard Gaussian smoothing.** We briefly recall the evolution strategy methods, e.g., [13, 23], which use a multivariate Gaussian distribution to generate the population around the current parameter value $\boldsymbol{\theta}_t$ at the t-iteration. When the Gaussian distribution can be factorized to d independent one-dimensional Gaussian distributions, the standard ES method can be mathematically interpreted based on the Gaussian smoothing technique [10, 20]. Specifically, a smoothed version of $J(\boldsymbol{\theta})$ in Eq. (2), denoted by $J_{\boldsymbol{\sigma}}(\boldsymbol{\theta})$, is defined by

$$\begin{aligned} J_{\boldsymbol{\sigma}}(\boldsymbol{\theta}) &:= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{\mathbb{R}^d} J(\boldsymbol{\theta} + \boldsymbol{\sigma} \circ \boldsymbol{u})\, \mathrm{e}^{-\frac{1}{2}\|\boldsymbol{u}\|_2^2}\, d\boldsymbol{u} \\ &= \mathbb{E}_{\boldsymbol{u} \sim \mathcal{N}(0, \mathbf{I}_d)}\left[J(\boldsymbol{\theta} + \boldsymbol{\sigma} \circ \boldsymbol{u})\right], \end{aligned} \tag{4}$$

where $\mathcal{N}(0, \mathbf{I}_d)$ is a d-dimensional standard Gaussian distribution, the notation "$\circ$" represents the element-wise product, and the vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_d)$ controls the smoothing effect. It is well known that $J_{\boldsymbol{\sigma}}$ is always differentiable even if J is not. In addition, most of the characteristics of the original objective function $J(\boldsymbol{\theta})$, e.g., convexity, the Lipschitz constant, are inherited by $J_{\boldsymbol{\sigma}}(\boldsymbol{\theta})$. When $J(\boldsymbol{\theta})$ is differentiable, the difference $\nabla J - \nabla J_{\boldsymbol{\theta}}$ can be bounded by its Lipschitz constant

(see [20], Lemma 3 for details). Thus, the original optimization problem in Eq. (3) can be replaced by a smoothed version, i.e.,

$$\max_{\boldsymbol{\theta}\in\mathbb{R}^d} J_{\boldsymbol{\sigma}}(\boldsymbol{\theta}), \tag{5}$$

where the gradient of $J_{\boldsymbol{\sigma}}(\boldsymbol{\theta})$ is given by

$$\nabla J_{\boldsymbol{\sigma}}(\boldsymbol{\theta}) = \frac{1}{\|\boldsymbol{\sigma}\|_2^{d/2}} \mathbb{E}_{\boldsymbol{u}\sim\mathcal{N}(0,\mathbf{I}_d)} \left[J(\boldsymbol{\theta} + \boldsymbol{\sigma}\circ\boldsymbol{u})\, \boldsymbol{u} \right]. \tag{6}$$

The standard ES method [23] uses Monte Carlo sampling to estimate the gradient $\nabla J_{\boldsymbol{\sigma}}(\boldsymbol{\theta})$ and update the state $\boldsymbol{\theta}$ from iteration n to $n+1$ by

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \frac{\lambda}{M\sigma} \sum_{m=1}^{M} J(\boldsymbol{\theta}_n + \boldsymbol{\sigma}\circ\boldsymbol{u}_m)\boldsymbol{u}_m, \tag{7}$$

where λ is the learning rate, $\boldsymbol{u}_m$ are sampled from the Gaussian distribution $\mathcal{N}(0,\mathbf{I}_d)$.

One drawback of the ES method and its variants is the slow convergence of the training process, due to the low accuracy of the MC-based gradient estimator (see [3]), also [33], for extended discussions on the accuracy of gradient approximations using Eq. (7) and related methods). On the other hand, the evaluations of $J(\boldsymbol{\theta}_n + \boldsymbol{\sigma}\circ\boldsymbol{u}_m)$ for $m = 1,\ldots,M$ at the n-th iteration can be generated totally in parallel, which makes it well suited to be scaled up to a large number of parallel workers on modern supercomputers. Therefore, *the motivation of this work is to develop a new gradient operator to replace the one in Eq. (6), such that the new gradient can be approximated in a much more accurate way and the embarrassing parallelism feature can be retained.*

3. The DGS-ES method for RL.

This section introduces our main framework. We start by introducing in Section 3.1 the DGS gradient operator and its approximation using the Gauss-Hermite quadrature rule. In Section 3.2, we describe how to incorporate the DGS gradient operator into the ES for reinforcement learning.

3.1. The DGS gradient operator.

For a given direction $\boldsymbol{\xi}\in\mathbb{R}^d$, the restriction of the objective function $J(\boldsymbol{\theta})$ along $\boldsymbol{\xi}$ can be represented by

$$G(y\,|\,\boldsymbol{\theta},\boldsymbol{\xi}) = J(\boldsymbol{\theta} + y\,\boldsymbol{\xi}), \quad y\in\mathbb{R}, \tag{8}$$

where $\boldsymbol{\theta}$ is the current state of the agent's parameters. Then, we can define the one-dimensional Gaussian smoothing of $G(y)$, denoted by $G_\sigma(y)$, by

$$G_\sigma(y\,|\,\boldsymbol{\theta},\boldsymbol{\xi}) := \mathbb{E}_{v\sim\mathcal{N}(0,1)} \left[G(y+\sigma v\,|\,\boldsymbol{\theta},\boldsymbol{\xi}) \right], \tag{9}$$

which is also the Gaussian smoothing of $J(\boldsymbol{\theta})$ along $\boldsymbol{\xi}$ in the neighbourhood of $\boldsymbol{\theta}$. The derivative of $G_\sigma(y|\boldsymbol{\theta},\boldsymbol{\xi})$ at $y=0$ is given by

$$\mathscr{D}[G_\sigma(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi})] = \frac{1}{\sigma}\, \mathbb{E}_{v\sim\mathcal{N}(0,1)} \left[G(\sigma v\,|\,\boldsymbol{\theta},\boldsymbol{\xi})\, v \right], \tag{10}$$

where $\mathscr{D}$ denotes the differential operator. It is easy to see that $\mathscr{D}[G_\sigma(0\,|\,\boldsymbol{x},\boldsymbol{\xi})]$ only involves the directionally smoothed objective function given in Eq. (9).

We can assemble a new gradient, i.e., the DGS gradient, by putting together the derivatives in Eq. (10) along d orthogonal directions, i.e.,

$$\nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta}) = \boldsymbol{\Xi}^\top \begin{bmatrix} \mathscr{D}\left[G_{\sigma_1}(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi}_1)\right] \\ \vdots \\ \mathscr{D}\left[G_{\sigma_d}(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi}_d)\right] \end{bmatrix}, \tag{11}$$

where $\boldsymbol{\Xi} := (\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_d)^\top$ represents the matrix consisting of d orthonormal vectors. It is important to notice that

$$\nabla J_{\boldsymbol{\sigma}}(\boldsymbol{\theta}) \neq \nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$$

for any $\boldsymbol{\sigma} > 0$, because of the directional Gaussian smoothing used in Eq. (11). However, there is consistency between the two quantities as $\boldsymbol{\sigma} \to 0$, i.e.,

$$\lim_{\boldsymbol{\sigma}\to 0} |\nabla J_{\boldsymbol{\sigma}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})| = 0, \tag{12}$$

for fixed $\boldsymbol{\theta}$ and $\boldsymbol{\Xi}$. If $\nabla J(\boldsymbol{\theta})$ exists, then $\nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ will also converge to $\nabla J(\boldsymbol{\theta})$ as $\boldsymbol{\sigma} \to 0$. Such consistency naturally led to the idea of replacing $\nabla J_{\boldsymbol{\sigma}}(\boldsymbol{\theta})$ with $\nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ in the ES framework.

3.2. **The DGS-ES algorithm.** Since each component of $\nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ in Eq. (11) only involves a one-dimensional integral, we can use Gaussian quadrature rules [22] to obtain spectral convergence. In the case of Gaussian smoothing, a natural choice is the Gauss-Hermite (GH) rule, which is used to approximate integrals of the form $\int_{\mathbb{R}} g(x)\mathrm{e}^{-x^2}dx$. By doing a simple change of variable in Eq. (10), the GH rule can be directly used to obtain the following estimator:

$$\widetilde{\mathscr{D}}^M[G_\sigma(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi})] := \frac{1}{\sqrt{\pi}\sigma}\sum_{m=1}^{M} w_m\, G(\sqrt{2}\sigma v_m\,|\,\boldsymbol{\theta},\boldsymbol{\xi})\sqrt{2}v_m \tag{13}$$

where w_m are the GH quadrature weights defined by

$$w_m = \frac{2^{M+1}M\,!\sqrt{\pi}}{[H'_M(v_m)]^2}, \quad m = 1, \dots, M, \tag{14}$$

v_m are the roots of the Hermite polynomial of degree M

$$H_M(v) = (-1)^M \mathrm{e}^{v^2}\frac{d^M}{dv^M}(\mathrm{e}^{-v^2}), \tag{15}$$

and M is the number of function evaluations, i.e., environment simulations, needed to compute the quadrature in Eq. (13). The weights $\{w_m\}_{m=1}^M$ and the roots $\{v_m\}_{m=1}^M$ can be found in [1]. The approximation error of the GH formula can be bounded by $\widetilde{\mathscr{D}}^M[G_\sigma]$ is

$$\left|\widetilde{\mathscr{D}}^M[G_\sigma] - \mathscr{D}[G_\sigma]\right| \leq C_0\frac{M\,!\sqrt{\pi}}{2^M(2M)\,!}\sigma^{2M-1}, \tag{16}$$

where $M!$ is the factorial of M, and the constant $C_0 > 0$ is independent of M and σ.

Applying the GH quadrature rule $\widetilde{\mathscr{D}}^M$ to each component of $\nabla_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ in Eq. (11), we define the following estimator:

$$\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta}) := \boldsymbol{\Xi}^\top \begin{bmatrix} \widetilde{\mathscr{D}}^M\left[G_{\sigma_1}(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi}_1)\right] \\ \vdots \\ \widetilde{\mathscr{D}}^M\left[G_{\sigma_d}(0\,|\,\boldsymbol{\theta},\boldsymbol{\xi}_d)\right] \end{bmatrix}, \tag{17}$$

which requires a total of $M \times d$ *parallelizable* environment simulations at each iteration of training. The error bound in Eq. (16) indicates that $\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ is an accurate estimator of the DGS gradient for a small M, regardless of the value of $\boldsymbol{\sigma}$. This enables the use of relatively big values of $\boldsymbol{\sigma}$ in Eq. (17) to exploit the *nonlocal* features of the landscape of $J(\boldsymbol{\theta})$ in the training process. The nonlocal exploitation ability of $\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J]$ is demonstrated in §4 to be effective in reducing the necessary number of iterations to achieve a prescribed reward score. An illustration of the

nonlocal exploitation is given in Figure 1 in optimizing the Ackley function (its definition is given in Section 4.1).

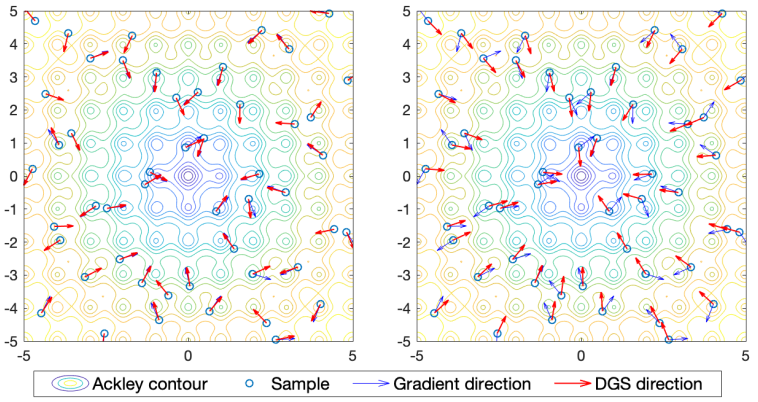

FIGURE 1. Illustration of DGS gradient direction and local gradient direction at 50 random locations on the surface of the Ackley in 2D. (Left) The standard deviation $\boldsymbol{\sigma}$ is set to 0.01, such that the DGS gradient align with the local gradient at most locations. (Right) The standard deviation $\boldsymbol{\sigma}$ is set to 1.0, such that most DGS gradient points to the global minimum at $(0, 0)$.

On the other hand, as the quadrature weights w_m and v_m defined in Eq. (14) and Eq. (15), are deterministic values, the DGS estimator $\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{x})$ in Eq. (17) is also a *deterministic* for fixed $\boldsymbol{\Xi}$ and $\boldsymbol{\sigma}$. To introduce random exploration ability to our approach, we add random perturbations to both $\boldsymbol{\Xi}$ and $\boldsymbol{\sigma}$. For the orthonormal matrix $\boldsymbol{\Xi}$, we add a small random rotation, denoted by $\Delta\boldsymbol{\Xi}$, to the current matrix $\boldsymbol{\Xi}$. The matrix $\Delta\boldsymbol{\Xi}$ is generated as a random skew-symmetric matrix, of which the magnitude of each entry is smaller than a prescribed threshold $\alpha > 0$. The perturbation of $\boldsymbol{\sigma}$ is conducted by drawing random samples from a uniform distribution $U(r - \beta, r + \beta)$ with two hyperparameters r and β with $r - \beta > 0$. The random perturbation of $\boldsymbol{\Xi}$ and $\boldsymbol{\sigma}$ can be triggered by various types of indicators e.g., the magnitude of the DGS gradient, the number of iteration done since last perturbation.

We suggest to set the hyperparameters in Algorithm 1 as follows: $M = 7, \alpha = 2.0, r = 1.0, \beta = 0.2, \gamma = 0.01$. Note that both input and output variables need to be properly normalized before running the DGS-ES algorithm. In practice one can tune the critical hyperparameters (M, r and ℓ_r) given the following suggested range: $M \in [7, 9]$, $r \in [0.5, 1.0]$ and $\ell_r \in [0.01, 0.1]$. Since our method does not have many hyperparameters, we suggest to use a simple grid search for hyperparameter tuning, and special attention should be spent to tuning the smoothing radius σ.

4. **Numerical experiments.**

We evaluate the DGS-ES method using two sets of problems: (a) classic high-dimensional benchmark functions to test the performance of DGS-ES in solving high-dimensional multi-modal optimization problems, and (b) several reinforcement learning benchmark problems.

Algorithm 1: The DGS-ES for reinforcement learning

hyperparameters:
M: the order of GH quadrature rule
α: the scaling factor for controlling the norm of $\Delta\boldsymbol{\Xi}$
r, β: the mean and perturbation scale for sampling $\boldsymbol{\sigma}$
γ: the tolerance for triggering random perturbation
Input:
$\boldsymbol{\theta}_0$: the initial parameter value,
L: the number of parallel workers
Output: the final parameter value $\boldsymbol{\theta}_N$
Initialize the policy π with $\boldsymbol{\theta}_0$
Set $\boldsymbol{\Xi} = \mathbf{I}_d$, and $\sigma_i = r$ for $i = 1, \ldots, d$
Broadcast L copies of $\pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\theta}_0)$ to the L workers
Divide the total GH quadrature points into L subsets, and send each subset to a worker.
for $n = 0, \ldots N-1$ **do**
  Each worker runs Md/L environment simulations at their assigned quadrature points
  Each worker sends Md/L scores to the master
  for $i = 1, \ldots, d$ **do**
    Compute $\widetilde{\mathscr{D}}^M[G_{\sigma_i}(0 \,|\, \boldsymbol{\theta}_n, \boldsymbol{\xi}_i)]$ in Eq. (13)
  end for
  Assemble $\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta}_n)$ in Eq. (17)
  Update $\boldsymbol{\theta}_n$ to $\boldsymbol{\theta}_{n+1}$ using Adam
  if $\|\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta}_n)\|_2 < \gamma$ **then**
    Generate $\Delta\boldsymbol{\Xi}$ and update $\boldsymbol{\Xi} = \mathbf{I}_d + \Delta\boldsymbol{\Xi}$
    Generate $\boldsymbol{\sigma}$ from $U(r-\beta, r+\beta)$
  end if
  Broadcast $\boldsymbol{\theta}_{n+1}$ to the L workers
  Each worker updates the policy to $\pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\theta}_{n+1})$
end for

4.1. **Tests on high-dimensional benchmark functions.** Here we investigate the DGS-ES performance on the high-dimensional functions: Sphere, Ackley, Lévy and Rastrigin functions, whose definitions are given below.

- The Sphere function $F_1(\boldsymbol{x})$ is defined by

$$F_1(\boldsymbol{x}) = \sum_{i=1}^{d} x_i^2,$$

  where d is the dimension and $\boldsymbol{x} \in [-5.12, 5.12]^d$ is the input domain. The global minimum is $f(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = (0, ..., 0)$. It represents *convex and isotropic* landscapes.

- The Ackley function $F_2(\boldsymbol{x})$ is defined by

$$F_2(\boldsymbol{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1),$$

where d is the dimension and $a = 20, b = 0.2, c = 2\pi$ are used in our experiments. The input domain $\boldsymbol{x} \in [-32.768, 32.768]^d$. The global minimum is $f(\boldsymbol{x}^*) = 0$, at $\boldsymbol{x}^* = (0, ..., 0)$. The Ackley function represents *non-convex* landscapes with *nearly flat outer region*. The function poses a risk for optimization algorithms, particularly hill-climbing algorithms, to be trapped in one of its many local minima.

- The Rastrigin function $F_3(\boldsymbol{x})$ is defined by

$$F_3(\boldsymbol{x}) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)], \tag{18}$$

where d is the dimension and $\boldsymbol{x} \in [-5.12, 5.12]^d$ is the input domain. The global minimum is $f(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = (0, ..., 0)$. This function represents *multimodal and separable* landscapes.

- The Lévy function $F_4(\boldsymbol{x})$ is defined by

$$F_4(\boldsymbol{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_d - 1)^2[1 + \sin^2(2\pi w_d)],$$

where $w_i = 1 + (x_i - 1)/4$ for $i = 1, \ldots, d$ and $\boldsymbol{x} \in [-10, 10]^d$ is the input domain. The global minimum is $f(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = (0, ..., 0)$. This represents *multi-modal and anisotropic* landscapes.

We compare the DGS-ES method with the following blackbox optimization methods: (a) Evolution Strategy (ES) in [23]; (b) Covariance matrix adaptation evolution strategy (CMA-ES) in [12] (we used the implementation of CMA-ES in the pycma open-source code from https://github.com/CMA-ES/pycma) and (c) the BFGS method in the Scipy library.

Figure 2 shows the scalability of the DGS-ES algorithm with the dimension of the objective function. We perform 20 repeated independent trials for each method to show the statistical performance. For the sphere function, the convergence rate does not change when we increase the dimension from 10 to 1000. Such property empirically carries over to the non-convex Lévy and Rastrigin functions. The reason is that both Lévy and Rastrigin have a globally near-convex structure when smoothing out their local minima. In contrast, the Ackley function is highly concave, as a large part of its surface is almost flat regardless of the small local minima. In this case, it takes many iterations for DGS-ES to search for the global minimum hidden in the middle of the flat surface.

Figure 3 shows the comparison between the DGS-ES and the baselines for optimizing the four benchmark functions in Figure 2 in 2000-dimensional space. For the sphere function, BFGS is a clear winner due to its optimal performance in handling convex functions. Among the three ES-type methods, the DGS-ES has much better performance than its competitors. For the other three functions, the DGS-ES method shows significant advantages over other methods.

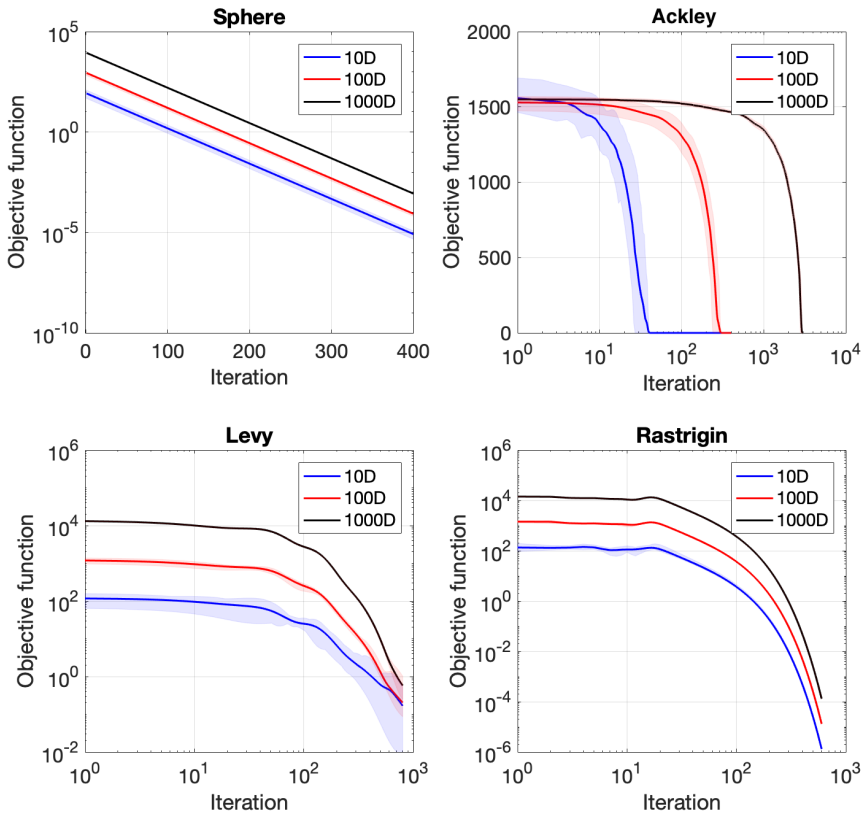

FIGURE 2. Illustration of the dimension dependence of the convergence rate of the DGS-ES method. The convergence rate, i.e., the number of iterations to converge, is *independent* of the dimension for convex functions, e.g., Sphere function, and such property empirically carries over to the non-convex Lévy and Rastrigin functions.

4.2. **Tests on RL benchmark problems.** To evaluate the DGS-ES algorithm, we test its performance on two classes of reinforcement learning environments: three classical control theory problems from OpenAI Gym (`https://github.com/openai/gym`) [4] and three continuous control tasks simulated using PyBullet (2.6.5) (`https://pybullet.org/`) [8] which is an open-source library. Within OpenAI Gym, we demonstrate the proposed approach on three benchmark examples: CartPole-v0 (discrete), MountainCarContinuous-v0 (continuous), Pendulum-v0 (continuous). The maximum time steps for these examples are 200, 999 and 200, respectively. More details about the environment and reward settings can be found in [4]. We also examine the DGS-ES algorithm on the challenging continuous robotic control problems in PyBullet library, namely HopperBulletEnv-v0, InvertedPendulumBulletEnv-v0 and ReacherBulletEnv-v0. In these three tasks, the maximum time steps are 1000, 1000 and 150, respectively. For the purpose of reproducible comparison, we employ the original environment settings from the OpenAI Gym and the PyBullet library without modifying the rewards or the environments.

For our implementation of DGS-ES, we define our policies as a two-layer feed-forward neural network with 16 hidden nodes and tanh activation functions. For

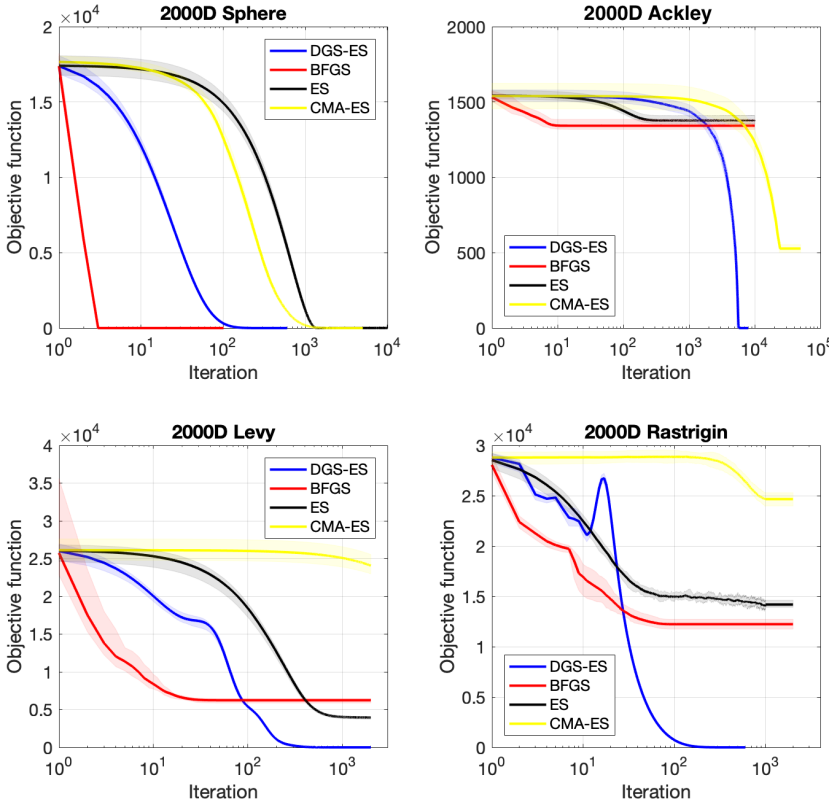

FIGURE 3. Comparison of different blackbox optimization methods on four 2000-dimensional benchmark functions.

gradient-based optimization, we use Adam to adaptively update the network parameters with a learning rate of $\ell_r = 0.1$. We choose the hyperparameters used in Algorithm 1 as follows: $M = 7, \alpha = 2.0, r = 1.0, \beta = 0.2, \gamma = 0.01$. In practice one can tune the critical hyperparameters (M, r and ℓ_r) given the following suggested range: $M \in [7, 9]$, $r \in [0.5, 1.0]$ and $\ell_r \in [0.01, 0.1]$. For each task, our results are performed over 5 repeated independent trials (different random seeds) of the Gym/PyBullet simulators and the network policy initialization.

The DGS-ES algorithm is specifically amenable to parallelization since it only needs to communicate scalars, allowing it to scale to over a large number of parallel workers. We implement a distributed version of Algorithm 1 to the reinforcement learning tasks. The distributed DGS-ES is implemented using PyTorch [21] combined with Ray [19] (`https://github.com/ray-project/ray`), which does not rely on special networking setup and is tested on large-scale high performance computing facilities with thousands of computing nodes/workers.

Comparison metric. As the motivation of this work is to accelerate time-to-solution of reinforcement training under the assumption that sufficient distributed computing resource is available, we use a different metric to evaluate the performance of DGS-ES and the baselines. Specifically, we are interested in the average return $\mathbb{E}[J]$ versus the number of iterations, i.e., N in Algorithm 1, because those iterations cannot be parallelized.

Baseline methods. We compare Algorithm 1 against several RL baselines, including ES, PPO and TRPO, as well as the state-of-the-art algorithms such as ASEBO, DDPG, and TD3. Below is the information of the packages used.

- ES: The Evolution Strategy proposed in [23]. We used the implementation of ES from the open-source code https://github.com/hardmaru/estool.
- ASEBO: Adaptive ES-Active Subspaces for Blackbox Optimization, which was recently developed by [5]. We used the implementation released by the authors at https://github.com/jparkerholder/ASEBO.
- PPO: Proximal Policy Optimization in [24], which is available in OpenAI's baselines repository at https://github.com/openai/baselines [9].
- TRPO: Trust Region Policy Optimization, developed by [25]. We also used the OpenAI's baselines implementation [9].
- DDPG: Deep Deterministic Policy Gradient, proposed by [14]. We used the implementation from https://github.com/georgesung/TD3 where the benchmark DDPG in PyBullet is provided.
- TD3: Twin Delayed Deep Deterministic Policy Gradient [11], which was built upon the DDPG. The original results were reported for the MuJoCo version environments using the implementation from https://github.com/sfujim/TD3, but we used the PyBullet implementation from https://github.com/georgesung/TD3.

The hyperparameters for all algorithms above are set to match the original papers without further tuning to improve performance on the testing benchmark examples.

Comparative evaluation. Figure 4 shows the comparison results of CartPole, Pendulum and MountainCar problems from the OpenAI Gym. We compared the DGS-ES with classical ES and the improved ASEBO method. In general, the DGS-ES method features faster convergence than the baselines. For the simplest CartPole problem, the three methods perform equally well. Discrepancy appears in the Pendulum test, where the DGS-ES method not only converges faster than the baselines, but also achieves a higher average return. There is a much bigger discrepancy between the DGS-ES and the baselines appear in the MountainCar test. According to the guideline provided in the OpenAI Gym, the success threshold is to achieve an average return of 90. The DGS-ES method achieves the threshold within 500 iterations, while the average returns of the ES and ASEBO methods are around zero. It is well known that the challenge of this problem is that the surface of the objective function $J(\boldsymbol{\theta})$ is very flat at most locations in the parameter space, which makes it difficult to capture the peak of $J(\boldsymbol{\theta})$. This test is a good demonstration of the nonlocal exploration ability of the DGS-ES method. Since the mean of the smoothing factor $\boldsymbol{\sigma}$ is set to 1.0, DGS-ES can capture the peak of $J(\boldsymbol{\theta})$ much faster than the baselines. In fact, we can see that it took around 50 iterations for the DGS-ES to find the peak region, while ES and ASEBO needed more than 3000 iterations (not plotted) to move out of the flat region.

Figure 5 shows the comparison results of Hopper-v0, InvertedPendulum-v0 and Reacher-v0 problems from the PyBullet library. We compare the DGS-ES with six baselines including ES, ASEBO, PPO, TRPO, DDPG and TD3. As expected, the DGS-ES method shows better performance in terms of the convergence speed. For the Hopper-v0 problem, the DGS-ES achieves the highest return of all the methods within

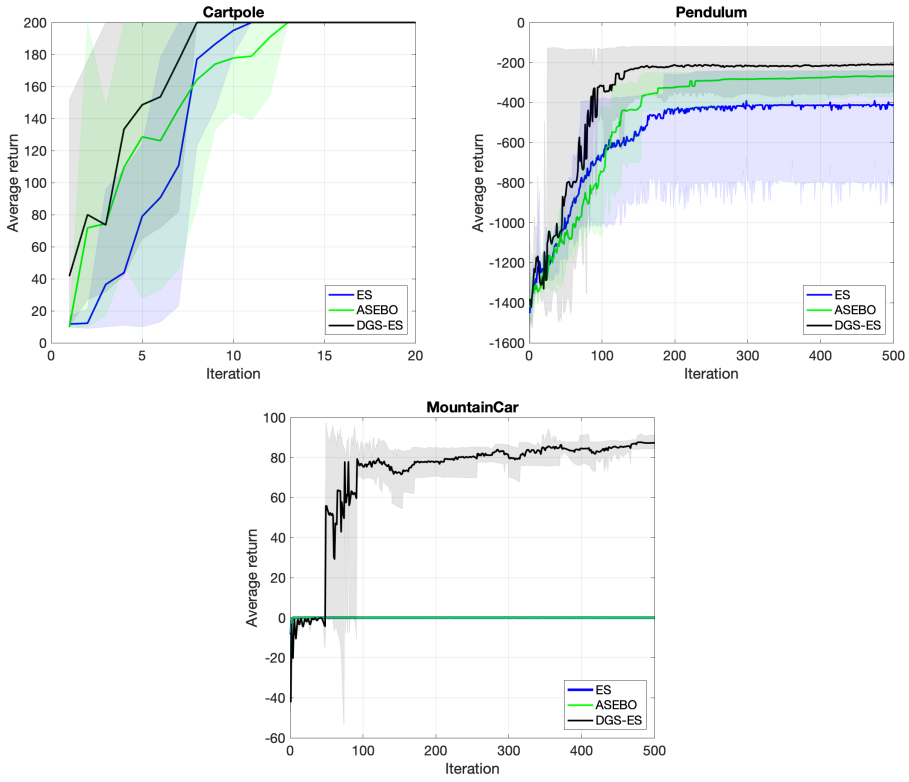

FIGURE 4. Comparison between the DGS-ES and two baselines, i.e., ES and ASEBO, for solving the three problems from OpenAI Gym. The colored curves are the average return over 5 repeated runs with different random seeds, and the corresponding shade represents the interval between the maximum and minimum return.

400 iterations. It is worth pointing out that some of the baselines will eventually catch up with the DGS-ES given a sufficiently large number of iterations. For example, DDPG and TD3 do not provide much improvement within 400 iterations, but DDPG could reach 1650 average return with over 3000 iterations, and TD3 could reach even higher, around 2200 average return, with over 6000 iterations, according to the baselines provided by OpenAI [9] and [11]. This phenomenon illustrates the fast convergence feature of DGS-ES. For the InvertedPendulum-v0 problem, DGS-ES can achieve the maximum return 1000 (default value in the PyBellet library) around 30 iterations. In comparison, ES and ASEBO can reach the maximum return but with more iterations than DGS-ES. According to the benchmark results for PyBullet environments in [11], DDPG and TD3 cannot converge to the maximum return even with a large number of iterations. For the Reacher-v0 problem, DGS-ES and ASEBO are still the top performers, and the advantage of DGS-ES is, again, faster convergence.

Figure 6 illustrates the effect of the radius $\boldsymbol{\sigma}$ of $\widetilde{\nabla}^{M}_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ on the performance of the DGS-ES method in solving the Reacher-v0 problem. All the simulations are done using the same initialization. We set the mean of , i.e., the hyperparameter r

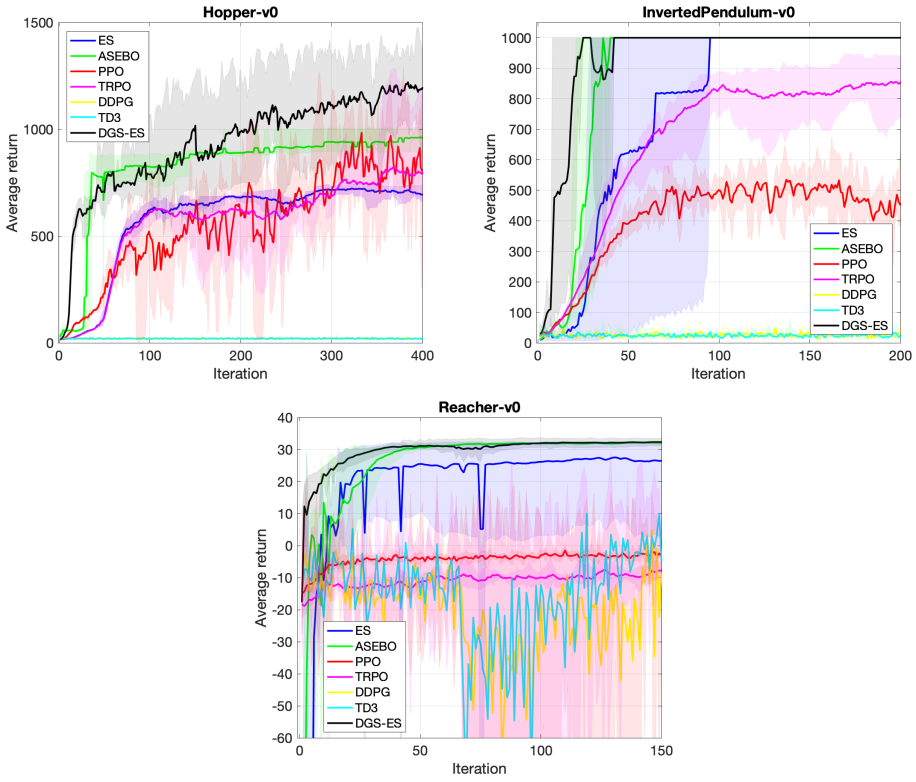

FIGURE 5. Comparison between the DGS-ES method and the baselines, i.e., ES, ASEBO, PPO, TRPO, DDPG and TD3, for solving the three problems from the PyBullet library. The colored curves are the average return over 20 runs with different random seeds, and the corresponding shade represents the interval between the maximal and minimal rewards.

in Algorithm 1, to 0.5, 0.05, 0.01, and 0.005. It is easy to see that the performance of DGS-ES deteriorates with the decrease of $\boldsymbol{\sigma}$. Since it is evident that the surface of $J(\boldsymbol{\theta})$ is not convex and may have many local maxima, a relatively big radius $\boldsymbol{\sigma}$ is necessary to help DGS-ES skip the local maxima. As $\boldsymbol{\sigma}$ becomes smaller, the DGS gradient $\widetilde{\nabla}^M_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ converges to the local gradient $\nabla J(\boldsymbol{\theta})$, which may get the optimizer trapped in a local maximum.

4.3. **Aerospace hierarchical stiffened shell design.** Now we demonstrate the DGS-ES method on a real-world stiffened shell design problem. Due to its high strength and stiffness, the hierarchical stiffened shell has been widely used in aerospace engineering [31, 32]. However, it is challenging to fully explore its optimal buckling load-carrying capacity. The goal of this work is to improve the load-carrying capacity by optimizing the representative unit cell of hierarchical stiffened shell where the inputs are 9 size variables (widths, heights and thickness) for major and minor stiffeners, as shown in Figure 7(a), and the output is the carrying-load capability factor which is calculated by high-fidelity numerical simulation, e.g., finite element method (FEM), see Figure 7 (b). Although the high-fidelity FEM simulation is

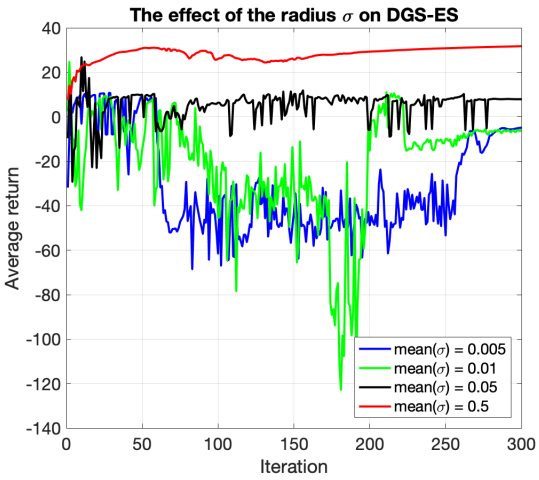

FIGURE 6. Illustration of the effect of the radius $\boldsymbol{\sigma}$ of $\widetilde{\nabla}^{M}_{\boldsymbol{\sigma},\boldsymbol{\Xi}}[J](\boldsymbol{\theta})$ on the performance of the DGS-ES method in solving the Reacher-v0 problem.

time-consuming, many open-source codes and commercial software have improved the computational efficiency via scalable parallelism and GPU acceleration. It is thus feasible to apply the DGS-ES method by implementing parallel FEM simulations in supercomputers. Figure 7 presents the comparison between DGS-ES and the other blackbox optimization methods. We observe that DGS-ES outperforms all other algorithms and achieves faster convergence using only 100 iterations. The robustness of DGS-ES also outperforms the alternatives.

5. **Conclusion.** Despite the successful demonstration shown in §4, there are several limitations with the DGS-ES method for reinforcement learning. First, it requires a powerful enough cluster or distributed computing resources to show superior performance. Even though more and more parallel environments for complex RL tasks, those parallel codes might not compatible with a cluster/supercomputer with a specific architecture. This will need some extra effort to modify environment codes, in order to exploit the advantage of the DGS-ES approach. Second, asynchronization between different environment simulations may drag down the total performance. In a distributed computing system, all the parallel workers receive the same number of environment simulations. However, as different parameter values may lead to different termination times of the environment simulations, there will be a potential waste of computing resources due to such asynchronization. Thus, a better scheduling algorithm is needed to further improve the performance of DGS-ES in RL. Third, even though the performance of the DGS-ES is not very sensitive to the hyperparameters, especially the radius $\boldsymbol{\sigma}$ in the experiments conducted in this work, optimal or even viable hyperparameters of the DGS-ES method are still problem dependent, which means hyperparameter tuning may be needed when applying the method to another RL problem.

Acknowledgement. This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contract and award numbers ERKJ352,

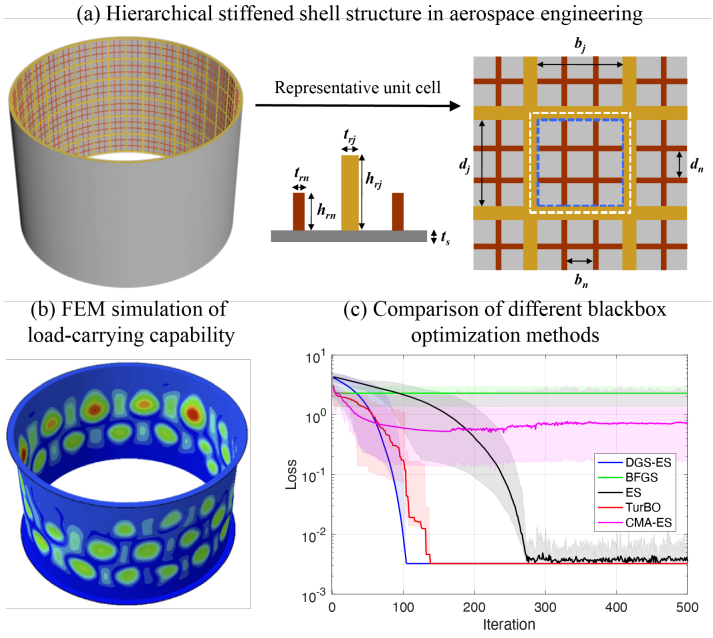

FIGURE 7. Illustration and design optimization of hierarchical stiffened shell structures in aerospace engineering, e.g. rocket.

by the DOE SciDac FastMath project, and by the Artificial Intelligence Initiative at the Oak Ridge National Laboratory (ORNL). ORNL is operated by UT-Battelle, LLC., for the U.S. Department of Energy under Contract DE-AC05-00OR22725.

REFERENCES.

[1] M. Abramowitz and I. Stegun, editors. *Handbook of Mathematical Functions*. Dover, New York, 1972.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[3] A. S. Berahas, L. Cao, K. Choromanskiv, and K. Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv:1905.01332*, 2019.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Krzysztof Choromanski, Aldo Pacchiano, Jack Parker-Holder, , and Yunhao Tang. From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization. *NeurIPS*, 2019.

[6] Krzysztof Choromanski, Aldo Pacchiano, Jack Parker-Holder, , and Yunhao Tang. Provably robust blackbox optimization for reinforcement learning. *arXiv:1903.02993*, 2019.

[7] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies

for deep reinforcement learning via a population of novelty-seeking agents. *NIPS*, 2018.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[9] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[10] Abraham D. Flaxman, Adam Tauman Kalai, Adam Tauman Kalai, and H. Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. *Proceedings of the 16th Annual ACM-SIAM symposium on Discrete Algorithms*, pages 385–394, 2005.

[11] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[12] Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[13] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016.

[15] Niru Maheswaranathan, Luke Metz, George Tucker, Dami Choi, and Jascha Sohl-Dickstein. Guided evolutionary strategies: Augmenting random search with surrogate gradients. *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[16] Florian Meier, Asier Mujika, Marcelo Matheus Gauy, and Angelika Steger. Improving gradient estimation in evolutionary strategies with past descent directions. *Optimization Foundations for Reinforcement Learning Workshop at NeurIPS*, 2019.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ICML*, pages 1928–1937, 2016.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[20] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, pages 1–40, 2015.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[22] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, volume 332. Springer Science Business Media &, 2007.

[23] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. From complexity to simplicity as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. *ICML*, pages 1889–1897, 2015.

[26] F. Sehnke, C. Osendorfer, T. Ruckstieb, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 2010.

[27] Olivier Sigaud and Freek Stulp. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and et al. M. Lanctot. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[29] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[30] R.S. Sutton and A. G. Barto, editors. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, New York, 1998.

[31] Kuo Tian, Lei Huang, Yu Sun, Kaifan Du, Peng Hao, and Bo Wang. Fast buckling load numerical prediction method for imperfect shells under axial compression based on pod and vibration correlation technique. *Composite Structures*, 252:112721, 2020.

[32] Kuo Tian, Li Zengcong, Lei Huang, K. Du, Liangliang Jiang, and Bo Wang. Enhanced variable-fidelity surrogate-based optimization framework by gaussian process regression and fuzzy clustering. *Computer Methods in Applied Mechanics and Engineering*, 366:113045, 2020.

[33] Jiaxin Zhang, Hoang Tran, Dan Lu, and Guannan Zhang. Enabling long-range exploration in minimization of multimodal functions. *Proceedings of 37th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.

E-mail address: zhangj@ornl.gov
E-mail address: tranha@ornl.gov
E-mail address: zhangg@ornl.gov