Accelerating Legacy Numerical Solvers by Non-intrusive Gradient-based Meta-solving

Sohei Arisaka [1 2] **Qianxiao Li** [1]

Abstract

Scientific computing is an essential tool for scientific discovery and engineering design, and its computational cost is always a main concern in practice. To accelerate scientific computing, it is a promising approach to use machine learning (especially meta-learning) techniques for selecting hyperparameters of traditional numerical methods. There have been numerous proposals to this direction, but many of them require automatic-differentiable numerical methods. However, in reality, many practical applications still depend on well-established but non-automatic-differentiable legacy codes, which prevents practitioners from applying the state-of-the-art research to their own problems. To resolve this problem, we propose a non-intrusive methodology with a novel gradient estimation technique to combine machine learning and legacy numerical codes without any modification. We theoretically and numerically show the advantage of the proposed method over other baselines and present applications of accelerating established non-automatic-differentiable numerical solvers implemented in PETSc, a widely used open-source numerical software library.

1. Introduction

Scientific Machine Learning (SciML) is an emerging research area, where scientific data and advancing Machine Learning (ML) techniques are utilized for new data-driven scientific discovery (Baker et al., 2019; Takamoto et al., 2022; Cuomo et al., 2022). One of the important topics in SciML is ML-enhanced simulation that leverages the strengths of both ML and traditional scientific computing, and recently numerous methods have been proposed to this direction (Karniadakis et al., 2021; Thuerey et al., 2021; Vinuesa & Brunton, 2022; Willard et al., 2022). These hybrid methods can be categorized into two groups according to how they are trained: separately or jointly.

In the first group, ML models are trained separately from numerical solvers and combined after training. Typically, they are trained to predict the solution of the target problem, and the solver uses the prediction as an initial guess (Ajuria Illarramendi et al., 2020; Özbay et al., 2021; Huang et al., 2020; Vaupel et al., 2020; Venkataraman & Amos, 2021). However, as shown in Arisaka & Li (2023), using the prediction as an initial guess while ignoring the property of the solver can lead to worse performance than a traditional non-learning choice. In addition to learning initial guesses, ML models can be used to learn other solver parameters such as preconditioners. To train them, one can use certain objectives as the loss fucntion that reflect the performance of the solver, such as the condition number of the preconditioned matrix (Sappl et al., 2019; Calì et al., 2023) and the spectral radius of the iteration matrix (Luz et al., 2020). This approach is effective and efficient because the models can be trained without running solvers. However, it requires some domain knowledge to design effective objectives, and derived methods are applicable to specific problems and solvers.

In the second group, to which this paper belongs, ML models are trained jointly with running numerical solvers. This approach is more general and straightforward than the first group, because the ML model can learn from actual solver's behavior through their gradients that represent how ML model's output affects solver's output. For example, in Um et al. (2020), authors propose a framework to train neural networks with differentiable numerical solvers and report a significant error reduction by correcting solutions using neural networks trained with PDE solvers. In Hsieh et al. (2018) and Chen et al. (2022), neural networks are trained to learn parameters of numerical solvers by minimizing the residual obtained by running the iterative solver with the learned parameters. Furthermore, Arisaka & Li (2023) proposed a general gradient-based learning framework to combine ML and numerical solvers, which can be applied to a wide range of problems and solvers.

[1]Department of Mathematics, National University of Singapore [2]Kajima Corporation, Japan. Correspondence to: Sohei Arisaka <sohei.arisaka@u.nus.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

However, a main limitation of the second training approach is that it requires the gradients of the outputs of the numerical solvers with resepect to their parameters that ML models are learning to tune. There are two difficulties in computing the gradients. First, despite the recent progress in learning-enhanced simulations, numerical computations in many practical applications are still performed by non-automatic-differentiable programs, including established open source libraries such as OpenFOAM (Jasak, 2009) and PETSc (Balay et al., 2023), original legacy codes developed and maintained in each community, and commercial black-box softwares such as ANSYS (Stolarski et al., 2018). Therefore, to compute their gradients, users need to modify existing codes to add the feature or reimplement them in deep learning frameworks for enabling automatic differentiation. It is an arduous task and is even impossible for black-box softwares. Second, even for differentiable numerical methods implemented in deep learning frameworks, it can be difficult to compute the gradients using backpropagation. This is because the computation process of the numerical methods can be very long, and the computation graph for backpropagation can be too large to fit into memory.

To overcome this limitation, we propose a novel methodology, called non-intrusive gradient-based meta-solving (NI-GBMS), to accelerate legacy numerical solvers by combining them with ML without any modification (Figure 1). Our proposed method expands the gradient-based meta-solving (GBMS) algorithm in Arisaka & Li (2023) to handle legacy non-automatic-differentiable codes by a novel gradient estimation technique using an adaptive surrogate model of the legacy codes.

Our main contributions can be summarized as follows:

1. In Section 3, we propose a non-intrusive methodology to train neural networks (meta-solvers) for accelerating legacy numerical solvers jointly with their non-automatic-differentiable black-box routines. To develop the methodology, we introduce an unbaised variance reduction technique for the forward gradient using an adaptive surrogate model to construct the control variates.
2. In Section 4, we theoretically and numerically show that the training using the proposed method converges faster than the training using the original forward gradient in a simple problem setting.
3. In Section 5, we demonstrate an application of the proposed method to accelerate established non-automatic-differentiable numerical solvers inmplemented in PETSc, resulting in a significant speedup. To the best of our knowledge, this is the first successful joint training of neural networks as meta-solvers and legacy solvers.

2. Related Work

ML-enhanced Simulation ML-enhanced simulation is a priority research direction in SciML (Baker et al., 2019). One approach is to use ML to correct the output of traditional methods. For example, Dresdner et al. (2022) use neural networks to learn the correction for traditional spectral solvers in fluid simulations. In Huang et al. (2023), a neural network is used to correct time integration errors and enables to take a larger time step, reusulting in acceleration of solving ODEs. Another complementary approach is to use ML to learn the input of traditional methods. For instance, Guo et al. (2022) develop a learning-enhanced Runge-Kutta method for solving ODEs. Chen et al. (2022) use a neural network to generate smoothers of the Multi-grid Network. In particular, this paper follows the meta-solving framework, which is proposed in Arisaka & Li (2023) to combine ML and numerical methods, and expand it to handle non-automatic-differentiable black-box solver implementations.

Forward Gradient The forward gradient is a projection of the gradient on a radom perturbation vector, and it gives a low-cost unbiased estimator of the gradient (Belouze, 2022). It is expected to be a possible alternative to the backpropagation algorithm, but its high variance is considered as a main limitation of the method (Baydin et al., 2022). To reduce the variance, several approaches have been proposed. Ren et al. (2022) showed that the activity-perturbed forward gradient has lower variance than the original weight-perturbed forward gradient. Bacho & Chu (2024) and Fournier et al. (2023) proposed variance reduction methods by trading off the unbiasedness. In this paper, we propose a novel unbaiased variance reduction approach that is orthogonal to these works.

Black-box Functions and Neural Networks Our control variate approach is inspired by Grathwohl et al. (2017), where the authors construct a gradient estimator using the control variates for a specific class of black-box loss function written as $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(b|\boldsymbol{\theta})}[f(b)]$. It is not applicable to deterministic cases because their method is based on the score-function estimator (Williams, 1992). On the other hand, our method is based on the forward gradient and applicable to any (mathematically) differentiable function. For more general approach, Jacovi et al. (2019) proposed the Estimate and Replace approach, where a neural network is trained by minimizing the difference between its output and the black-box function's output and used to bypass the black-box function during the backward computation. This method is simple but biased whereas ours is (asymptotically) unbiased because we use it for improving the quality of the forward gradient of the black-box function rather than replacing the gradient. We will show our advantage over this simple replacement approach in Section 4 and Section 5.

3. Methodology

3.1. Meta-solving

Our non-intrusive framework can be viewed as a wrapper of a legacy numerical solver to be compatible with deep learning (Figure 1). To describe our method, we follow the meta-solving framework introduced in Arisaka & Li (2023). Suppose that we have task space $(\mathcal{T}, P)$ and numerical solver f with paramter $\boldsymbol{\theta}$. We want to train neural network Ψ with weight $\boldsymbol{\omega}$, called meta-solver, to generate a good parameter $\boldsymbol{\theta}$ of the solver f for each task $\tau \in \mathcal{T}$. The solver performance is measured by loss function $\mathcal{L}$. Then, the meta-solving problem is defined as follows:

Definition 3.1 (Meta-solving problem (Arisaka & Li, 2023))**.** For a given task space $(\mathcal{T}, P)$, loss function $\mathcal{L}$, solver f, and meta-solver Ψ, find $\boldsymbol{\omega}$ which minimizes $\mathbb{E}_{\tau \sim P}\left[\mathcal{L}(\tau, f(\tau; \Psi(\tau; \boldsymbol{\omega})))\right]$.

For instance, τ is to solve a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, f is the Jacobi method starting from an initial guess $\boldsymbol{\theta} = \boldsymbol{x}_0$, Ψ is a neural network to generate a good initial guess $\boldsymbol{\theta}$ of the solver f to solve τ, and $\mathcal{L}$ is a surrogate of the number of iterations to converge. To train Ψ, Arisaka & Li (2023) proposed a gradient-based approach assuming that ∇f can be computed efficiently. However, as mentioned in Section 1, the solver f in practice can be a legacy code or commercial black-box software, which does not support automatic differentiation.

3.2. Forward Gradient

To compute the gradient of legacy numerical solvers without backpropagation, we utilize the forward gradient, a low-cost estimator of the gradient.

Definition 3.2 (Forward gradient (Belouze, 2022))**.** For a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ and a vector $\mathbf{v} \in \mathbb{R}^d$, the forward gradient $\boldsymbol{g}_{\mathbf{v}} : \mathbb{R}^d \to \mathbb{R}^d$ is defined as

$$\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta}) = (\nabla f(\boldsymbol{\theta}) \cdot \mathbf{v})\mathbf{v}. \tag{1}$$

In Section 3 and Section 4, we describe and analyze our method using the definition for scalar functions, but it can be easily extended to vector-valued functions, which will be presented in Section 5. We also note that the original forward gradient is computed using forward mode automatic differentiation (Baydin et al., 2018), but we use the following finite difference approximation for legacy numerical solvers:

$$\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta}) \approx \boldsymbol{g}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta} + \epsilon\mathbf{v}) - f(\boldsymbol{\theta})}{\epsilon}\mathbf{v}, \tag{2}$$

where ϵ is a small positive scalar.

The advantage of the forward gradient is its unbiasedness and low computational cost in terms of both time and memory. It is easily shown that if random vector $\mathbf{v}$'s scalar components v_i are independent and have zero mean and unit variance for all i, then the forward gradient $\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta})$ is an unbiased estimator of $\nabla f(\boldsymbol{\theta})$, i.e. $\mathbb{E}[\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta})] = \nabla f(\boldsymbol{\theta})$. As for the computational cost, the forward gradient of automatic-differentiable function f such as neural networks can be computed in a single forward computation using forward mode automatic differentiation. Even if a function f is implemented in legacy codes without forward mode automatic differentiation, the forward gradient can be computed using Equation (2) at the cost of evaluating f twice. Note that the computation cost is equivalent to the cost of backpropagation, but it can be computed in parallel. Moreover, to compute the forward gradient, we do not need to store the intermediate values for backpropagation.

However, despite the above advantage, the practical use of the forward gradient is limited because it suffers high variance, especially in high-dimensional cases. Baydin et al. (2022) shows that the best possible variance is achieved by choosing v_i's to be independent Rademacher variables, i.e. $P(\mathrm{v}_i = 1) = P(\mathrm{v}_i = -1) = 1/2$ for all i, in which case the variance is equal to

$$\mathbb{E}[\|\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta})\|^2] = (d-1)\|\nabla f(\boldsymbol{\theta})\|^2, \tag{3}$$

which is very large for large d. This high variance in fact hinders the training process, and we will investigate the effect of the variance in Section 4 and Section 5.

3.3. Control Variate Forward Gradient

To alleviate this problem, we introduce a novel approach to reduce the variance while keeping it unbiased. The key idea is to construct control variates using a surrogate model $\hat{f}$ whose gradient $\nabla\hat{f}$ is easy to compute.

Definition 3.3 (Control variate forward gradient)**.** For differentiable functions $f : \mathbb{R}^d \to \mathbb{R}$, $\hat{f} : \mathbb{R}^d \to \mathbb{R}$, and a random vector $\mathbf{v} \in \mathbb{R}^d$, the control variate forward gradient $\boldsymbol{h}_{\mathbf{v}} : \mathbb{R}^d \to \mathbb{R}^d$ is defined as

$$\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta}) = \boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta}) - \hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta}) + \mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta})], \tag{4}$$

where $\hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta}) = (\nabla\hat{f}(\boldsymbol{\theta}) \cdot \mathbf{v})\mathbf{v}$. Similarly, we define its finite difference approximation $\boldsymbol{h}_{\mathbf{v},\epsilon}$ using $\boldsymbol{g}_{\mathbf{v},\epsilon}$ as

$$\boldsymbol{h}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) = \boldsymbol{g}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) - \hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta}) + \mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta})]. \tag{5}$$

Note that we use the finite difference approximation only for $\boldsymbol{g}_{\mathbf{v}}$ but not for $\hat{\boldsymbol{g}_{\mathbf{v}}}$ because the surrogate model $\hat{f}$ will be implemented in deep learning frameworks with automatic differentiation. The method of control variates is a classical and widely used variance reduction technique (Nelson, 1990). In the control variates method, to reduce the variance of random variable $\mathbf{X}$ of interest, we introduce another random variable $\mathbf{Y}$ which has high correlation to

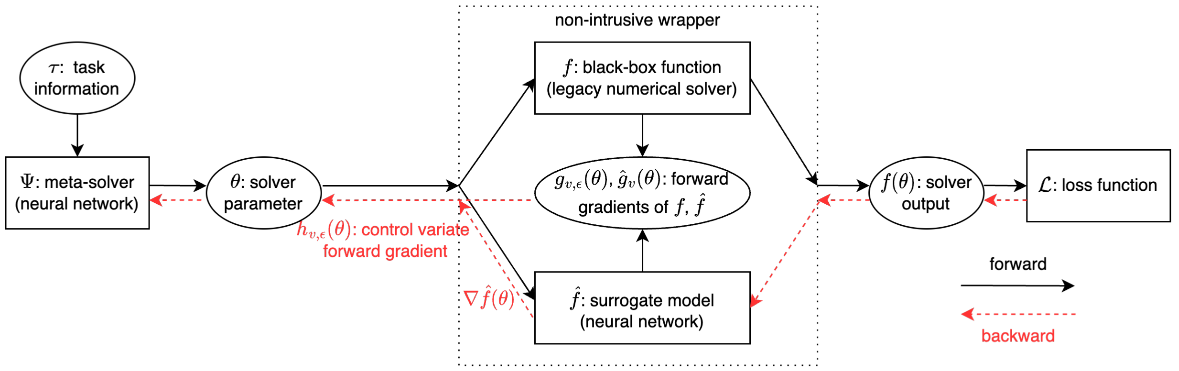

non-intrusive wrapper
τ: task information
Ψ: meta-solver (neural network)
θ: solver parameter
f: black-box function (legacy numerical solver)
$g_{v,\epsilon}(\theta)$, $\hat{g}_v(\theta)$: forward gradients of f, $\hat{f}$
$\hat{f}$: surrogate model (neural network)
$h_{v,\epsilon}(\theta)$: control variate forward gradient
$\nabla\hat{f}(\theta)$
$f(\theta)$: solver output
$\mathcal{L}$: loss function
forward
backward

Figure 1. The overall architecture of non-intrusive gradient-based meta-solving

Algorithm 1 Non-intrusive gradient-based meta-solving

```
Input: $(P,\mathcal{T})$: task space, $\Psi$: meta-solver with weight $\boldsymbol{\omega}$, $f$: legacy solver, $\hat{f}$: surrogate model with weight $\boldsymbol{\phi}$, $\mathcal{L}$: loss function, $\mathrm{Opt}$: optimizer for $\Psi$, $\hat{\mathrm{Opt}}$: optimizer for $\hat{f}$, $\hat{P}$: distribution of $\mathbf{v}$, $\epsilon$: positive scalar, $S$: stopping criterion
repeat
  $\tau \sim P, \mathbf{v} \sim \hat{P}$    ▷sample task $\tau$ and random vector $\mathbf{v}$ from $P$ and $\hat{P}$
  Forward computation:
    $\boldsymbol{\theta} \leftarrow \Psi(\tau;\boldsymbol{\omega})$    ▷generate solver parameter $\boldsymbol{\theta}$ by $\Psi$
    $y \leftarrow f(\boldsymbol{\theta})$, $y^+ \leftarrow f(\boldsymbol{\theta}+\epsilon\mathbf{v})$    ▷solve task $\tau$ twice by $f$ using parameter $\boldsymbol{\theta}$ and $\boldsymbol{\theta}+\epsilon\mathbf{v}$
    $d \leftarrow \frac{y^+-y}{\epsilon}$    ▷compute the directional derivative of $f$ by finite difference approximation
    $\hat{y}, \hat{d} \leftarrow \mathrm{ForwardAD}(\hat{f},\boldsymbol{\theta},\mathbf{v})$    ▷compute the output and directional derivative of $\hat{f}$ by forward mode AD
    $L \leftarrow \mathcal{L}(y)$, $\hat{L} \leftarrow (d-\hat{d})^2$    ▷compute main loss $L$ and surrogate model loss $\hat{L}$
  Backward computation:
    $\boldsymbol{h} \leftarrow d\mathbf{v} - \hat{d}\mathbf{v} + \nabla_{\boldsymbol{\theta}}\hat{f}(\boldsymbol{\theta})$    ▷compute control variate forward gradient
    $\nabla_{\boldsymbol{\omega}} L \leftarrow \mathrm{ModifiedBackprop}(L,\boldsymbol{\omega},\boldsymbol{h})$    ▷compute $\nabla_{\boldsymbol{\omega}} L$ by backpropagation while replacing $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ with $\boldsymbol{h}$
    $\nabla_{\boldsymbol{\phi}}\hat{L} \leftarrow \mathrm{Backprop}(\hat{L},\boldsymbol{\phi})$    ▷compute $\nabla_{\boldsymbol{\phi}}\hat{L}$ by backpropagation
  Update:
    $\boldsymbol{\omega} \leftarrow \mathrm{Opt}(\boldsymbol{\omega},\nabla_{\boldsymbol{\omega}} L)$, $\boldsymbol{\phi} \leftarrow \hat{\mathrm{Opt}}(\boldsymbol{\phi},\nabla_{\boldsymbol{\phi}}\hat{L})$    ▷update $\boldsymbol{\omega}$ and $\boldsymbol{\phi}$ by $\mathrm{Opt}$ and $\hat{\mathrm{Opt}}$ using $\nabla_{\boldsymbol{\omega}} L$ and $\nabla_{\boldsymbol{\phi}}\hat{L}$, respectively
until $S$ is satisfied
```

$\mathbf{X}$ and known mean $\mathbb{E}[\mathbf{Y}]$. Then, $\mathbf{X}-\mathbf{Y}+\mathbb{E}[\mathbf{Y}]$ is used as a variance-reduced estimator of $\mathbf{X}$. The key idea of our method is to use the forward gradient of the surrogate model $\hat{\boldsymbol{g}}_{\mathbf{v}}$ as $\mathbf{Y}$ so that $\mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{g}}_{\mathbf{v}}(\boldsymbol{\theta})] = \nabla\hat{f}(\boldsymbol{\theta})$ can be computed by backpropagation.

The following theorem shows that the control variate forward gradient is still unbiased under the same assumption in Belouze (2022). Furthermore, if the surrogate model $\hat{f}$ is close to the numerical solver f, then the control variate forward gradient $\boldsymbol{h}_{\mathbf{v}}$ successfully reduces the variance of the original forward gradient $\boldsymbol{g}_{\mathbf{v}}$.

Theorem 3.4 (Improvement by $\boldsymbol{h}_{\mathbf{v}}$). *If* v_i*'s are independent and have zero mean and unit variance, then* $\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta})$ *is an unbiased estimator of* $\nabla f(\boldsymbol{\theta})$*, i.e.*

$$\mathbb{E}[\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta})] = \nabla f(\boldsymbol{\theta}). \tag{6}$$

Furtheremore, if v_i*'s are independent Rademacher variables, then the mean squared deviation of* $\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta})$ *is minimized and equal to*

$$\mathbb{E}[\|\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta})-\nabla f(\boldsymbol{\theta})\|^2] = (d-1)\|\nabla f(\boldsymbol{\theta})-\nabla\hat{f}(\boldsymbol{\theta})\|^2. \tag{7}$$

The proof is found in Appendix A.2. Equation (7) shows the improvement from Equation (3) of the original forward gradient. In Equation (3), there is no controllable term, whereas in Equation (7), the variance can be reduced by choosing a good surrogate model $\hat{f}$. We remark that the finite difference version $\boldsymbol{h}_{\mathbf{v},\epsilon}$ is not unbiased due to $\boldsymbol{g}_{\mathbf{v},\epsilon}$ but asymptotically unbiased as ϵ tends to 0. The error of the finite difference version is analyzed in Appendix A.3.

3.4. Non-intrusive Gradient-based Meta-solving

The remaining problem is how to obtain a surrogate model $\hat{f}$. To this end, we propose a novel algorithm, called Non-Intrusive Gradient-Based Meta-Solving (NI-GBMS, Algorithm 1). Its main workflow is the same as the original gradient-based meta-solving (Arisaka & Li, 2023), but it is capable of handling legacy solver f whose gradient is not accessible. In NI-GBMS, we implement surrogate model $\hat{f}$ by a neural network with weight $\boldsymbol{\phi}$ and adaptively update it while training meta-solver Ψ. For training $\hat{f}$, we use

$$\hat{\mathcal{L}}(\boldsymbol{\phi}) = \left(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \cdot \mathbf{v} - \nabla_{\boldsymbol{\theta}} \hat{f}(\boldsymbol{\theta}; \boldsymbol{\phi}) \cdot \mathbf{v}\right)^2 \quad (8)$$

as a surrogate model loss. This loss function is more suitable for our purpose than other losses that minimize the error of outputs $f(\boldsymbol{\theta})$ and $\hat{f}(\boldsymbol{\theta})$ (Jacovi et al., 2019). By contrast, our $\hat{\mathcal{L}}$ is designed to minimize $\|\nabla f(\boldsymbol{\theta}) - \nabla \hat{f}(\boldsymbol{\theta})\|^2 = \|\mathbb{E}[\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta}) - \hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta})]\|^2$, which determines the variance of the control variate forward gradient in Equation (7). Then, the control variate forward gradient is used as an approximation of ∇f to compute $\nabla_{\boldsymbol{\omega}} \mathcal{L}$ for updating meta-solver Ψ.

We remark important characteristics of NI-GBMS. The most significant one is its non-intrusiveness as the name suggests. It enables us to integrate complex legacy codes and commercial black-box softwares into ML as they are. In addition, the adaptive training strategy of the surrogate model $\hat{f}$ is another advantage. By updating $\hat{f}$ during the training of meta-solver Ψ, it can automatically adapt to the changing distribution of $\boldsymbol{\theta}$ generated by the meta-solver Ψ. Finally, we note that the extra computational cost of training $\hat{f}$ is negligible compared to the cost of executing numerical solver f, which is the bottleneck of our target problem setting.

4. Analysis

To investigate the convergence property of NI-GBMS, we consider the simplest case, where a black-box solver f itself is minimized over its parameter $\boldsymbol{\theta}$, i.e. $\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$. This is a special case of Algorithm 1, where the meta-solver Ψ is a constant function that returns its parameter $\boldsymbol{\omega} = \boldsymbol{\theta}$, and the loss function $\mathcal{L}$ is $f(\boldsymbol{\theta})$ itself. This can also be considered that we directly optimize the solver parameter $\boldsymbol{\theta}$ for a single given task τ. The optimization algorithm Opt for training Ψ is a variant of the gradient descent, which uses the control variate forward gradient $\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta})$ instead of the true gradient $\nabla f(\boldsymbol{\theta})$, i.e. $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta}_k)$, where α is the learning rate.

4.1. Theoretical Analysis

For the considered setting, we have the following convergence theorem. Note that it is shown for the exact version of Algorithm 1, where $\boldsymbol{h}_{\mathbf{v}}$ is used instead of its finite difference approximation $\boldsymbol{h}_{\mathbf{v},\epsilon}$. The expectation $\mathbb{E}[\cdot]$ in the theorem is taken over all randomness of $\mathbf{v}$ through the training process.

Theorem 4.1 (Convergence)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be μ-strongly convex and L-smooth. Denote the global minimizer of f by $\boldsymbol{\theta}_*$. Consider the sequence $(\boldsymbol{\theta}_k)_{k\in\mathbb{N}}$ and $(\hat{f}_k)_{k\in\mathbb{N}}$ generated by NI-GBMS.*

*(a) (**No surrogate model**) Suppose that $\hat{f}_k = \hat{f} \equiv 0$ for all $k \in \mathbb{N}$. If $\alpha \in (0, \frac{1}{Ld}]$, then the expected optimality gap satisfies the following inequality for all $k \in \mathbb{N}$:*

$$\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)] \leq (1-\alpha\mu)^k (f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}_*)). \quad (9)$$

*(b) (**Fixed surrogate model**) Instead of $\hat{f} \equiv 0$ in (a), suppose that $f_k = \hat{f}$ for all $k \in \mathbb{N}$ and $\hat{f}$ satisfies $\sup_{\boldsymbol{\theta}\in\mathbb{R}^d}\{\frac{\|\nabla f(\boldsymbol{\theta}) - \nabla \hat{f}(\boldsymbol{\theta})\|^2}{\|\nabla f(\boldsymbol{\theta})\|^2}\} \leq r$ for some $r \in [0, 1)$. Then, we have the same inequality as Equation (9) with an improved range of learning rate $\alpha \in (0, \min\{\frac{1}{\mu}, \frac{1}{Ldr}\}]$.*

*(c) (**Adaptive surrogate model**) Suppose that we have*

$$\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_k) - \nabla \hat{f}_k(\boldsymbol{\theta}_k)\|^2] \leq \xi^k \|\nabla f(\boldsymbol{\theta}_0) - \nabla \hat{f}_0(\boldsymbol{\theta}_0)\|^2 \quad (10)$$

for some $\xi \in (0, 1)$. If $\alpha \in (0, \frac{1}{\mu}]$, then the expected optimality gap satisfies the following inequality for all $k \in \mathbb{N}$:

$$\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)] \leq C\rho^k, \quad (11)$$

where

$$C = \max\{\frac{\alpha L d \|\nabla f(\boldsymbol{\theta}_0) - \nabla \hat{f}_0(\boldsymbol{\theta}_0)\|^2}{\mu}, f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}_*)\} \quad (12)$$

and

$$\rho = \max\{1 - \alpha\mu, \xi\}. \quad (13)$$

The proofs are presented in Appendix A.2. Theorem 4.1 compares our method to the original forward gradient and shows our advantage. In (a), $\hat{f}$ does nothing and the control variate forward gradient $\boldsymbol{h}_{\mathbf{v}}$ reduces to the original forward gradient $\boldsymbol{g}_{\mathbf{v}}$. This case corresponds to the original forward gradient descent (Belouze, 2022). The best convergence rate of (a) is $1 - \frac{\mu}{Ld}$, which becomes worse as the dimension d increases due to the increasing variance of $\boldsymbol{g}_{\mathbf{v}}$. The case of (b) represents the situation of having a fixed surrogate model $\hat{f}$ and shows the improvement by the control forward gradient with the surrogate model $\hat{f}$. In this case, the effective best convergence rate is $1 - \frac{\mu}{Ldr}$. Although it still depends on the dimension d, it is improved by the factor of the relative error r between ∇f and $\nabla \hat{f}$. Note that we have $\frac{1}{Ld} \leq \min\{\frac{1}{\mu}, \frac{1}{Ldr}\}$ because $d \geq 1$, $r \in [0, 1)$, and $\mu \leq L$ (Bottou et al., 2018). The case of (c) represents a more ideal situation where the gradient of the surrogate model $\nabla \hat{f}$ is converging to the gradient of the target function ∇f with convergence rate ξ. In this case, $f(\boldsymbol{\theta}_k)$ converges to $f(\boldsymbol{\theta}_*)$ with the convergence rate $\rho = \xi$, which can be independent of the dimension d. In summary, the convergence rate of the proposed method can be improved as the surrogate model $\hat{f}$ becomes closer to the target black-box solver f, and it can be independent of the dimension d in the ideal case.

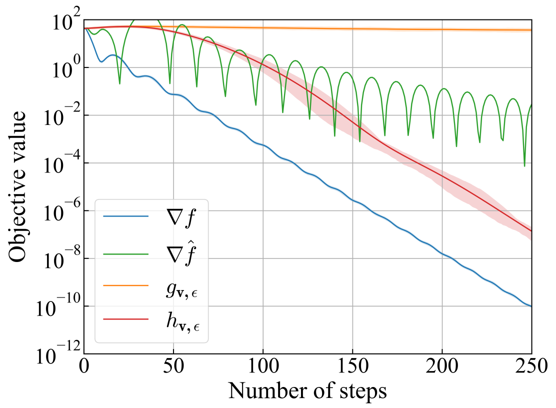
Table 1. Objective values obtained using different gradient estimators. Boldface indicates the best performance except for the case of ∇f.

(a) Sphere function

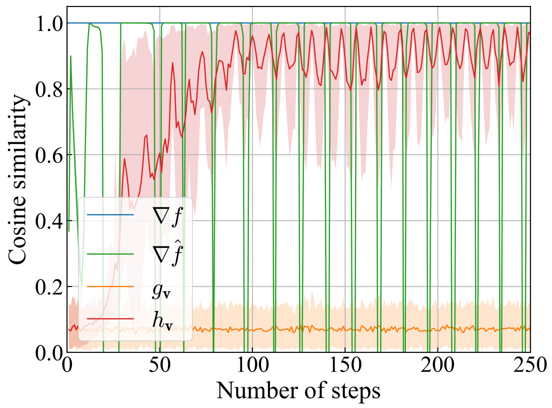
gradient	$d=2$	$d=8$	$d=32$	$d=128$
∇f	1.56e-12	5.88e-12	2.37e-11	9.55e-11
$\nabla \hat{f}$	9.77e-01	1.41e-02	3.67e-07	2.89e-02
$\boldsymbol{g}_{\mathbf{v},\epsilon}$	1.61e-06	6.86e-03	5.88e-01	3.68e+01
$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	**7.33e-12**	**8.63e-11**	**2.36e-09**	**1.36e-07**

(b) Rosenbrock function

gradient	$d=2$	$d=8$	$d=32$	$d=128$
∇f	1.18e-05	5.98e-01	5.61e-01	6.06e-01
$\nabla \hat{f}$	2.23e+01	**6.64e-04**	7.56e-01	1.91e+02
$\boldsymbol{g}_{\mathbf{v},\epsilon}$	9.06e-03	1.08e+00	1.40e+01	1.68e+02
$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	**6.73e-04**	3.40e-01	**4.11e-01**	**7.86e-01**



(a) The convergence plot.



(b) Cosine similarity to the true gradient ∇f.

Figure 2. The convergence plot and cosine similarity (Sphere function $d = 128$). The shadowed area represents a range from 10% to 90%.

4.2. Numerical Experiments

Under the same problem setting, we conduct numerical experiments using two test functions, Sphere function (Molga & Kwietnia):

$$f(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i^2 \tag{14}$$

and Rosenbrock function (Rosenbrock, 1960):

$$f(\boldsymbol{\theta}) = \sum_{i=1}^{d-1} \left[100\left(\theta_{i+1} - \theta_i^2\right)^2 + (\theta_i - 1)^2\right] \tag{15}$$

with the dimension $d = 2, 8, 32, 128$. For surrogate model $\hat{f}$, we use a convolutional neural network because the relation of $\boldsymbol{\theta}$ coordinates is local. To update the surrogate model $\hat{f}$, the Adam optimizer (Kingma & Ba, 2015) with learning rate 0.01 is used as $\hat{\mathrm{Opt}}$. Note that the surrogate model $\hat{f}$ is trained online during minimizing f without pre-training. The main optimizer Opt is also the Adam optimizer with $\alpha = 0.1$ for Sphere function and $\alpha = 0.01$ for the Rosenbrock function. The number of optimization steps is 250 for Sphere function and 50,000 for Rosenbrock function. For comparison, we also test using the true gradient ∇f, the gradient of the surrogate model $\nabla \hat{f}$ (Jacovi et al., 2019), and the forward gradient $\boldsymbol{g}_{\mathbf{v},\epsilon}$ (Belouze, 2022) instead of our control variate forward gradient $\boldsymbol{h}_{\mathbf{v},\epsilon}$. We set finite difference step size $\epsilon = 10^{-8}$. The initial parameter $\boldsymbol{\theta}_0$ is sampled from the uniform distribution on $[-1, 1]^d$. The random vector $\mathbf{v}$ is sampled from the Rademacher distribution. We optimize 100 samples and report the average of their final objective values in Table 1. Other details are presented in Appendix A.4.

Table 1 shows the advantage of our method. For all cases including high dimensional cases, our control variate forward gradient $\boldsymbol{h}_{\mathbf{v},\epsilon}$ successfully minimize the objective function f to comparable values with the best achievable baseline, the true gradient ∇f. By contrast, the performance of the original forward gradient $\boldsymbol{g}_{\mathbf{v},\epsilon}$ degrades quickly as the dimension increases. The performance of the gradient of the surrogate model $\nabla \hat{f}$ is not stable. Although it can perform better than the true gradient ∇f (e.g Rosenbrock function with $d = 8$) thanks to its bias, its performance is worse than our $\boldsymbol{h}_{\mathbf{v},\epsilon}$ for most cases. Figure 2 illustrates the typical behavior of each gradient estimator. In the illustrated case, the original forward gradient $\boldsymbol{g}_{\mathbf{v},\epsilon}$ fails to minimize the objective function f because of its high variance, which is also shown in a low cosine similarity to the true gradient ∇f in Figure 2b. The gradient of the surrogate model $\nabla \hat{f}$ shows unstable convergence behavior in Figure 2a and unstable cosine similarity in Figure 2b. Our control variate forward gradient $\boldsymbol{h}_{\mathbf{v},\epsilon}$ resolves these undesired behaviors and shows stable convergence and high cosine similarity. Although it takes some time for the surrogate model $\hat{f}$ to fit f, the convergence speed after fitting is comparable to the true gradient ∇f.

5. Application: Learning Parameters of Legacy Numerical Solvers

In this section, we present applications of NI-GBMS to learn good initial guesses of iterative solvers implemented in PETSc, the Portable, Extensible Toolkit for Scientific Computation (Balay et al., 2023), which is widely used for large-scale scientific applications but does not support automatic differentiation.

5.1. Toy Example: Poisson Equation

Problem setting Let us consider solving 1D Poisson equations with different source terms repeatedly. Our task τ is to solve the 1D Poisson equation with the homogeneous Dirichlet boundary condition:

$$-\frac{d^2}{dz^2}u(z) = b_\tau(z), \quad z \in (0,1) \tag{16}$$

$$u(0) = u(1) = 0. \tag{17}$$

Using the finite difference scheme, we discretize it to obtain the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}_\tau$, where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ and $\boldsymbol{x}, \boldsymbol{b}_\tau \in \mathbb{R}^N$. Thus, our task τ is represented by $\tau = \{\boldsymbol{b}_\tau\}$. We use two task distributions P and Q described in Appendix A.4. The main loss function is a surrogate loss for the number of iterations $\tilde{\mathcal{L}}_\delta$ presented in (Arisaka & Li, 2023), which is defined recursively as:

$$\tilde{\mathcal{L}}_\delta^{(k+1)}(\tau;\boldsymbol{\omega}) = \tilde{\mathcal{L}}_\delta^{(k)}(\tau;\boldsymbol{\omega}) + \sigma(\mathcal{L}_k(\tau;\boldsymbol{\omega}) - \epsilon), \tag{18}$$

$$\tilde{\mathcal{L}}_\delta^{(0)}(\tau;\boldsymbol{\omega}) = 0, \tag{19}$$

where δ is a given target tolerance, σ is the sigmoid function, and $\mathcal{L}_k$ is the k-th step loss. For $\mathcal{L}_k$, we use the relative residual $\|\boldsymbol{b}_\tau - \boldsymbol{A}\boldsymbol{x}_k\| \,/\, \|\boldsymbol{b}_\tau\|$, where $\boldsymbol{x}_k$ is the solution after k iterations, so the training does not require any pre-computed solutions and is conducted in a self-supervised manner. The solver f is a legacy iterative solver with an initial guess $\boldsymbol{\theta}$ for the Poisson equation. We use the Jacobi method f_{Jac} and the geometric multigrid method f_{MG} (Saad, 2003) implemented in PETSc, and their tolerance δ is set to 10^{-3} and 10^{-8} respectively. The meta-solver Ψ parameterized by $\boldsymbol{\omega}$ generates the initial guess $\boldsymbol{\theta}_\tau$ for each task τ. We use a fully-connected neural network Ψ_{GBMS} for Ψ. We implement surrogate model $\hat{f}$ by a fully-connected neural network. Then, we train Ψ_{GBMS} to minimize $\tilde{\mathcal{L}}_\delta$ using Algorithm 1. Other details, including network architecture, the parameters of PETSc solvers, and training hyperparameters, are presented in Appendix A.4.

Baselines For the purpose of comparison, we have five baselines: Ψ_0, Ψ_{SL}, Ψ_{GBMS} trained with ∇f, Ψ_{GBMS} trained with $\nabla\hat{f}$, and Ψ_{GBMS} trained with $\boldsymbol{g}_{\mathbf{v},\epsilon}$. Ψ_0 is a non-learning baseline that always gives initial guess $\boldsymbol{\theta} = \boldsymbol{0}$, which is a reasonable choice because $u_\tau(0) = u_\tau(1) = 0$ and $\mathbb{E}_{\tau\sim P}[u_\tau] = \boldsymbol{0}$. Ψ_{SL} is an ordinary supervised learning baseline whose network architecture is the same as Ψ_{NN}. It is trained independently of the solver f by minimizing the relative error $\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\tau*}\| \,/\, \|\boldsymbol{x}_{\tau*}\|$, where $\hat{\boldsymbol{x}}$ is the prediction of Ψ and $\boldsymbol{x}_{\tau*}$ is the pre-computed reference solution of task τ. Note that Ψ_{SL} does not require the gradient of the solver f. Ψ_{GBMS} trained with ∇f is considered as the best achievable baseline, which is available when the solver f is implemented in deep learning frameworks. Ψ_{GBMS} trained with $\nabla\hat{f}$ is a baseline trained using the gradient of the surrogate model $\nabla\hat{f}$, which can be computed easily by backpropagation but is biased. This corresponds to the training approach proposed in Jacovi et al. (2019). Ψ_{GBMS} trained with $\boldsymbol{g}_{\mathbf{v},\epsilon}$ is a baseline trained with the forward gradient $\boldsymbol{g}_{\mathbf{v},\epsilon}$, which is applicable to legacy solvers but suffers high variance. We also check the difference between the exact forward gradients $(\boldsymbol{g}_{\mathbf{v}}, \boldsymbol{h}_{\mathbf{v}})$ and the finite difference approximation $(\boldsymbol{g}_{\mathbf{v},\epsilon}, \boldsymbol{h}_{\mathbf{v},\epsilon})$. For training the baselines requiring automatic differentiation, we implement the Jacobi method and the geometric multigrid method using PyTorch so that they are differentiable and equivalent to the solvers in PETSc.

Table 2. The average number of iterations to converge. Boldface indicates the best performance except for the case of ∇f.

Ψ	gradient	$f = f_{\text{Jac}}$	$f = f_{\text{MG}}$
Ψ_0	-	173.00	90.06
Ψ_{SL}	-	175.71	78.62
Ψ_{GBMS}	∇f	53.64	39.29
	$\nabla\hat{f}$	106.65	52.59
	$\boldsymbol{g}_{\mathbf{v}}$	66.76	59.06
	$\boldsymbol{g}_{\mathbf{v},\epsilon}$	66.85	59.09
	$\boldsymbol{h}_{\mathbf{v}}$	**44.92**	**42.76**
	$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	45.15	43.06

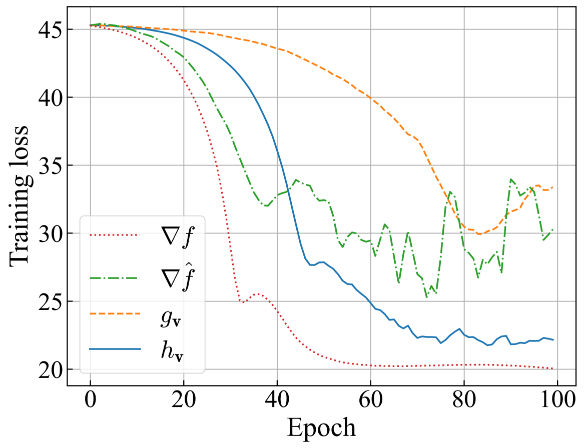


Figure 3. The learning curves of the case f_{MG}. They are trained using a same random seed and hyperparameters except for the gradient estimator.

Results Table 2 presents the average number of iterations to converge starting from the initial guesses generated by each trained meta-solver. For both Jacobi and multigrid solvers, the proposed method achieves comparable results to the best achievable baseline (Ψ_{GBMS} trained with ∇f) and outperforms other baselines by a large margin. The gap between Ψ_{SL} and Ψ_{GBMS} trained with ∇f shows the importance of incorporating the solver information into the training of Ψ and necessity of the proposed method for legacy solvers. Table 2 also shows that there is no significant difference between the exact forward gradients and the finite difference approximation. Figure 3 shows the learning curves of the case of the multigrid method. Their behaviors are similar to the ones shown in Figure 2a: $\boldsymbol{g}_{\mathbf{v},\epsilon}$ suffers slow training, $\nabla \hat{f}$ is unstable, and $\boldsymbol{h}_{\mathbf{v},\epsilon}$ converges similarly to ∇f after some delay for fitting $\hat{f}$. In summary, Ψ_{GBMS} is successfully trained jointly with non-automatic-differentiable solvers using the proposed method, and reduces the number of iterations from the solver-independent supervised learning baseline Ψ_{SL} by 74% for the Jacobi method and 45% for the multigrid method.

5.2. Advanced Examples

To demonstrate the versatility and performance of NI-GBMS, we also consider more advanced examples, 2D biharmonic equation and 3D linear elasticity equations, using FEniCS (Alnæs et al., 2015). In these examples, we accelerate the algebraic multigrid method f_{AMG} in PETSc, which is one the most practical linear solvers (Stüben, 2001), by learning initial guesses using NI-GBMS. Details are presented in Appendix A.4.

Biharmonic Equation The biharmonic equation is a fourth-order elliptic equation of the form: $\nabla^4 u = b_\tau$ in Ω, where ∇^4 is the biharmonic operator, Ω is a domain of interest. Let Ω be $[0,1]^2$ and boundary conditions be $u = \nabla^2 u = 0$ on $\partial\Omega$. The source term b_τ is sampled from a task distribution P presented in Appendix A.4. The equation is discretized using finite elements, and the discretized linear system is solved by the algebraic multigrid method f_{AMG} in PETSc.

Linear Elasticity Equations The linear elasticity problem is described by the following equations:

$$-\nabla \cdot \sigma(u) = b_\tau \quad \text{in } \Omega, \tag{20}$$

$$\sigma(u) = \lambda \operatorname{tr}(\epsilon(u)) I + 2\mu\epsilon(u) \tag{21}$$

$$\epsilon(u) = \frac{1}{2}\left(\nabla u + (\nabla u)^T\right) \tag{22}$$

where $\Omega = [0,1]^3$ is an elastic body, u is the displacement field, σ is the stress tensor, ϵ is strain-rate tensor, λ and μ are the Lamé elasticity parameters, and b_τ is the body force.

Table 3. The average number of iterations to reach relative residual tolerance $\delta = 10^{-5}$.

	Biharmonic Eq.	Elasticity Eq.
Default PETSc	73.74	88.71
Enhanced by NI-GBMS	**31.54**	**25.61**

We consider clamped bodies deformed by their own weight by letting $u = (0,0,0)^T$ at $x = 0$ and $b_\tau = (0, 0, -\rho_\tau g)$, where ρ_τ is the density and g is the gravitational acceleration. The density ρ_τ is sampled from the log uniform distribution on $[10^{-2}, 10^2]$, which determines our task space. Then, the equations are discretized using finite elements and solved by f_{AMG}.

Results The performance improvement by NI-GBMS is shown in Table 3, where the average number of iterations is reduced from the default initial guess, which is the zero vector, by 57% for the biharmonic equation and 71% for the linear elasticity equations. This result demonstrates the versatility and performance of NI-GBMS for more advanced problem settings.

6. Discussion

Wall-clock Time Although we have not conducted detailed wall-clock time analysis, we note the actual computation time for the case of $f = f_{\mathrm{MG}}$ in Section 5.1. As for the inference time, the non-learning baseline Ψ_0 takes 9.0 seconds for solving 10,000 tasks while Ψ_{GBMS} trained with $\boldsymbol{h}_{\mathbf{v},\epsilon}$ takes 5.5 seconds. Note that this is similar to the ratio of the number of iterations, $\Psi_0 : \Psi_{\mathrm{GBMS}} = 90.06 : 43.06$, because the inference cost of Ψ is neglectable compared to the cost of f. About the training time, training Ψ_{GBMS} for 100 epochs takes 24 minutes with $\boldsymbol{h}_{\mathbf{v},\epsilon}$ and 27 minutes with $\boldsymbol{g}_{\mathbf{v},\epsilon}$. Interestingly, training with $\boldsymbol{h}_{\mathbf{v},\epsilon}$ is faster than $\boldsymbol{g}_{\mathbf{v},\epsilon}$ even though $\boldsymbol{h}_{\mathbf{v},\epsilon}$ requires additional cost of evaluating and training surrogate model $\hat{f}$. This reduction of training time comes from the acceleration of f during training. Since $\boldsymbol{h}_{\mathbf{v},\epsilon}$ gives better approximation of ∇f than $\boldsymbol{g}_{\mathbf{v},\epsilon}$, meta-solver Ψ can faster learn good initial guesses. As a result, the computational cost of f is reduced more quickly, leading to a decrease in total training time, even considering the additional costs associated with the surrogate model. This advantage can be more significant in practical problems where the cost of f is more dominant in the total computational cost. Note that the experiments are conducted using the following hardwares: Intel Xeon W-3335 CPU @ 3.40GHz and NVIDIA GeForce RTX 3090.

Total Cost Saving Considering the result of $f = f_{\mathrm{Jac}}$ in Section 5.1 under two different scenarios, we estimate how much NI-GBMS can save the total number of iterations. The first scenario is Train-and-Solve, where the cost of training and the saving by the trained model are considered separately. As for training cost, we train the meta-solver for 100 epochs, and each epoch contains 5,000 tasks. During training, we need to solve each task twice to evaluate $\boldsymbol{g}_{\mathbf{v},\epsilon}$, and each solution requires about 105 iterations on average. In total, the training requires 105 million iterations. After training, the trained model saves 128 iterations per task. Thus, if the model is used for more than 0.82 million tasks, the training cost pays off. Note that it is a reasonable number in practical applications such as incompressible flow simulations. The second scenario is Train-by-Solve, where we train the meta-solver while solving actual target tasks. In other words, there is no separation of training and deployment. It is possible because the training can be done in unsupervised manner. Suppose we want to solve similar one million Poisson equations. Without NI-GBMS approach, solving all equations takes 173 million iterations. However, with NI-GBMS, we can solve them faster even considering the training cost. For example, we train the meta-solver while solving the first half, which requires 105 million iterations. Then, we stop training and use it for solving the last half, which requires 22.5 million iterations. In total, solving all equations takes 127.5 million iterations, which is 26 % less than the case without NI-GBMS. Furthermore, we obtain the trained meta-solver which can be used for other tasks for free.

7. Conclusion

In this paper, we proposed NI-GBMS, a novel methodology to combine meta-solvers parametrized by neural networks and non-automatic-differentiable legacy solvers without any modification. To develop this, we introduced the control variate forward gradient, which is unbiased and has lower variance than the original forward gradient. Furthermore, we proposed a practical algorithm to construct the control variate forward gradient using an adaptive surrogate model. It was theoretically and numerically shown that the proposed method has better convergence property than other baselines. We applied NI-GBMS to learn initial guesses of established iterative solvers in PETSc, resulting in a significant reduction of the number of iterations to converge. Our proposed method expands the range of applications of neural networks, and it can be a fundamental building block to combine modern neural networks and legacy numerical solvers.

In future work, we will explore more sophisticated architectures of the surrogate model, which can leverage our prior knowledge of the solver and problem and lead to better performance. For example, one can build surrogate model $\hat{f} = \hat{f}_2 \circ \hat{f}_1$ using two building blocks $\hat{f}_1$ and $\hat{f}_2$, where $\hat{f}_1$ is the same algorithm as legacy solver f but implemented in a deep learning framework and working on a smaller problem size, and $\hat{f}_2$ is a neural network to correct the difference between f and $\hat{f}_1$. In addition, we will apply the proposed method to learn multiple hyperparameters simultaneously (e.g. initial guess and preconditioner). Toward this end, a preliminary experiment for learning preconditioners is presented in Appendix A.5. Beyond scientific computing applications, our proposed method can be useful to the problems of learning to optimize (Chen et al., 2021), where memory limitation often becomes a practical bottleneck because the computation graph tends to be too large for backpropagation. We expect that this limitation can be overcome by the proposed method using the high-quality gradient estimation without storing intermediate values.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments. This project is supported by the National Research Foundation, Singapore, under the NRF fellowship (NRF-NRFF13-2021-0005). S. Arisaka is supported by Kajima Corporation, Japan.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

DOLFIN documentation — DOLFIN documentation. https://fenicsproject.org/olddocs/dolfin/2019.1.0/python/index.html. URL https://fenicsproject.org/olddocs/dolfin/2019.1.0/python/index.html. Accessed: 2024-2-2.

Ajuria Illarramendi, E., Alguacil, A., Bauerheim, M., Misdariis, A., Cuenot, B., and Benazera, E. Towards an hybrid computational strategy based on Deep Learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*, AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, June 2020. doi: 10.2514/6.2020-3058. URL https://doi.org/10.2514/6.2020-3058.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. The FEniCS Project Version 1.5. *Anschnitt*, 3(100), December 2015. ISSN 0003-5238. doi: 10.11588/ans.2015.100.20553. URL https://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553.

Arisaka, S. and Li, Q. Principled Acceleration of Iterative Numerical Methods Using Machine Learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1041–1059. PMLR, 2023. URL https://proceedings.mlr.press/v202/arisaka23a.html.

Bacho, F. and Chu, D. Low-variance Forward Gradients using Direct Feedback Alignment and momentum. *Neural networks: the official journal of the International Neural Network Society*, 169:572–583, January 2024. ISSN 0893-6080, 1879-2782. doi: 10.1016/j.neunet.2023.10.051. URL https://linkinghub.elsevier.com/retrieve/pii/S0893608023006172.

Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., and Lee, S. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. *DOE*, February 2019. doi: 10.2172/1478744. URL https://www.osti.gov/servlets/purl/1478744.

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E. M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., Kong, F., Kruger, S., May, D. A., McInnes, L. C., Mills, R. T., Mitchell, L., Munson, T., Roman, J. E., Rupp, K., Sanan, P., Sarich, J., Smith, B. F., Zampini, S., Zhang, H., Zhang, H., and Zhang, J. PETSc Web page. https://petsc.org/, 2023. URL https://petsc.org/.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic Differentiation in Machine Learning: a Survey. *Journal of machine learning research: JMLR*, 18(153):1–43, 2018. ISSN 1532-4435, 1533-7928. URL https://jmlr.org/papers/v18/17-468.html.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. Gradients without Backpropagation. February 2022. URL http://arxiv.org/abs/2202.08587.

Belouze, G. Optimization without Backpropagation. September 2022. URL http://arxiv.org/abs/2209.06302.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM review. Society for Industrial and Applied Mathematics*, 60(2): 223–311, January 2018. ISSN 0036-1445, 1095-7200. doi: 10.1137/16m1080173. URL https://epubs.siam.org/doi/10.1137/16M1080173.

Calì, S., Hackett, D. C., Lin, Y., Shanahan, P. E., and Xiao, B. Neural-network preconditioners for solving the Dirac equation in lattice gauge theory. *Physical Review D*, 107 (3):034508, February 2023. doi: 10.1103/PhysRevD.107.034508. URL https://link.aps.org/doi/10.1103/PhysRevD.107.034508.

Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., and Yin, W. Learning to optimize: A primer and A benchmark. March 2021. URL https://jmlr.org/papers/volume23/21-0308/21-0308.pdf.

Chen, Y., Dong, B., and Xu, J. Meta-MgNet: Meta multigrid networks for solving parameterized partial differential equations. *Journal of computational physics*, 455:110996, April 2022. ISSN 0021-9991. doi: 10.1016/j.jcp.2022.110996. URL https://www.sciencedirect.com/science/article/pii/S0021999122000584.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *Journal of scientific computing*, 92(3):88, July 2022. ISSN 0885-7474, 1573-7691. doi: 10.1007/s10915-022-01939-z. URL https://doi.org/10.1007/s10915-022-01939-z.

Dresdner, G., Kochkov, D., Norgaard, P., Zepeda-Núñez, L., Smith, J. A., Brenner, M. P., and Hoyer, S. Learning to correct spectral methods for simulating turbulent flows. July 2022. URL http://arxiv.org/abs/2207.00556.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks: the official journal of the International Neural Network Society*, 107: 3–11, November 2018. ISSN 0893-6080, 1879-2782. doi: 10.1016/j.neunet.2017.12.012. URL http://dx.doi.org/10.1016/j.neunet.2017.12.012.

Fournier, L., Rivaud, S., Belilovsky, E., Eickenberg, M., and Oyallon, E. Can Forward Gradient Match Backpropagation? In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10249–10264. PMLR, 2023. URL https://proceedings.mlr.press/v202/fournier23a.html.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. October 2017. URL https://openreview.net/pdf?id=SyzKd1bCW.

Guo, Y., Dietrich, F., Bertalan, T., Doncevic, D. T., Dahmen, M., Kevrekidis, I. G., and Li, Q. Personalized Algorithm Generation: A Case Study in Learning ODE Integrators. *SIAM Journal of Scientific Computing*, 44 (4):A1911–A1933, August 2022. ISSN 1064-8275. doi: 10.1137/21M1418629. URL https://doi.org/10.1137/21M1418629.

Hendrycks, D. and Gimpel, K. Gaussian Error Linear Units (GELUs). June 2016. URL http://arxiv.org/abs/1606.08415.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning Neural PDE Solvers with Convergence Guarantees. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/pdf?id=rklaWn0qK7.

Huang, J., Wang, H., and Yang, H. Int-Deep: A deep learning initialized iterative method for nonlinear problems. *Journal of computational physics*, 419:109675, October 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109675. URL https://www.sciencedirect.com/science/article/pii/S0021999120304496.

Huang, Z., Liang, S., Zhang, H., Yang, H., and Lin, L. On fast simulation of dynamical system with neural vector enhanced numerical solver. *Scientific reports*, 13 (1):15254, September 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-42194-y. URL http://dx.doi.org/10.1038/s41598-023-42194-y.

Jacovi, A., Hadash, G., Kermany, E., Carmeli, B., Lavi, O., Kour, G., and Berant, J. Neural network gradient-based learning of black-box function interfaces. January 2019. URL https://openreview.net/pdf?id=r1e13s05YX.

Jasak, H. OpenFOAM: Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89–94, December 2009. ISSN 2092-6782. doi: 10.2478/IJNAOE-2013-0011. URL https://www.sciencedirect.com/science/article/pii/S2092678216303879.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, May 2021. ISSN 2522-5820, 2522-5820. doi: 10.1038/s42254-021-00314-5. URL https://www.nature.com/articles/s42254-021-00314-5.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980.

Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I. Learning Algebraic Multigrid Using Graph Neural Networks. In Iii, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6489–6499. PMLR, 2020. URL https://proceedings.mlr.press/v119/luz20a.html.

Molga, M. and Kwietnia, C. S. . Test functions for optimization needs. https://marksmannet.com/RobertMarks/Classes/ENGR5358/Papers/functions.pdf. URL https://marksmannet.com/RobertMarks/Classes/ENGR5358/Papers/functions.pdf. Accessed: 2023-8-29.

Nelson, B. L. Control Variate Remedies. *Operations research*, 38(6):974–992, December 1990. ISSN 0030-364X. doi: 10.1287/opre.38.6.974. URL https://doi.org/10.1287/opre.38.6.974.

Özbay, A. G., Hamzehloo, A., Laizet, S., Tzirakis, P., Rizos, G., and Schuller, B. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering*, 2, 2021. ISSN 2632-6736. doi: 10.1017/dce.2021.7.

Polyak, B. T. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, January 1963. ISSN 0041-5553. doi: 10.1016/0041-5553(63)90382-3. URL https://www.sciencedirect.com/science/article/pii/0041555363903823.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. Scaling forward gradient with local losses. October 2022. URL https://github.com/google-research/.

Rosenbrock, H. H. An Automatic Method for Finding the Greatest or Least Value of a Function. *Computer Journal*, 3(3):175–184, January 1960. ISSN 0010-4620. doi: 10.1093/comjnl/3.3.175. URL https://academic.oup.com/comjnl/article-pdf/3/3/175/988633/030175.pdf.

Saad, Y. *Iterative Methods for Sparse Linear Systems: Second Edition*. Other Titles in Applied Mathematics. SIAM, April 2003. ISBN 9780898715347. doi: 10.1137/1.9780898718003. URL https://play.google.com/store/books/details?id=qtzmkzzqFmcC.

Sappl, J., Seiler, L., Harders, M., and Rauch, W. Deep Learning of Preconditioners for Conjugate Gradient Solvers in Urban Water Related Problems. June 2019. URL http://arxiv.org/abs/1906.06925.

Stolarski, T., Nakasone, Y., and Yoshimoto, S. *Engineering Analysis with ANSYS Software*. Butterworth-Heinemann, January 2018. ISBN 9780081021651. URL https://play.google.com/store/books/details?id=50IyDwAAQBAJ.

Stüben, K. A review of algebraic multigrid. *Journal of computational and applied mathematics*, 128(1):281–309, March 2001. ISSN 0377-0427. doi: 10.1016/S0377-0427(00)00516-1. URL https://www.sciencedirect.com/science/article/pii/S0377042700005161.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBENCH: An extensive benchmark for Scientific machine learning. pp. 1596–1611, October 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/0a9747136d411fb83f0cf81820d44afb-Paper-Datasets_and_Benchmarks.pdf.

Thuerey, N., Holl, P., Mueller, M., Schnell, P., Trost, F., and Um, K. *Physics-based Deep Learning*. WWW, 2021. URL https://physicsbaseddeeplearning.org.

Um, K., Brand, R., Fei, Y. r., Holl, P., and Thuerey, N. Solver-in-the-loop: learning from differentiable physics to interact with iterative PDE-solvers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, number Article 513 in NIPS'20, pp. 6111–6122, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 9781713829546. URL https://dl.acm.org/doi/abs/10.5555/3495724.3496237.

Vaupel, Y., Hamacher, N. C., Caspari, A., Mhamdi, A., Kevrekidis, I. G., and Mitsos, A. Accelerating nonlinear model predictive control through machine learning. *Journal of process control*, 92:261–270, August 2020. ISSN 0959-1524. doi: 10.1016/j.jprocont.2020.06.012. URL https://www.sciencedirect.com/science/article/pii/S0959152420302481.

Venkataraman, S. and Amos, B. Neural Fixed-Point Acceleration for Convex Optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=Vxbpb6XvGgH.

Vinuesa, R. and Brunton, S. L. Enhancing computational fluid dynamics with machine learning. *Nature computational science*, 2(6):358–366, June 2022. ISSN 2662-8457. doi: 10.1038/s43588-022-00264-7. URL http://dx.doi.org/10.1038/s43588-022-00264-7.

Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V. Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems. *ACM Comput. Surv.*, 55(4):1–37, November 2022. ISSN 0360-0300. doi: 10.1145/3514228. URL https://doi.org/10.1145/3514228.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

A. Appendix

A.1. Implementation

The source code of the experiments is available at `https://github.com/arisakaso/nigbms`.

A.2. Proofs

Proof of Theorem 3.4. To lighten the notation, we omit $\boldsymbol{\theta}$ dependence. Denoting ith component of $\boldsymbol{g}_{\mathbf{v}}$ by $g_{\mathbf{v}i}$,

$$\begin{aligned}
\mathbb{E}[{g_{\mathbf{v}i}}^2] &= \mathbb{E}\left[(\nabla f\cdot\mathbf{v})^2\mathrm{v}_i^2\right] && (23)\\
&= \mathbb{E}\left[\left(\frac{\partial f}{\partial\theta_i}\right)^2{\mathrm{v}_i}^4 + \sum_{j\neq i}\left(\frac{\partial f}{\partial\theta_j}\right)^2{\mathrm{v}_i}^2{\mathrm{v}_j}^2 + 2\sum_{k<l}\frac{\partial f}{\partial\theta_k}\frac{\partial f}{\partial\theta_l}{\mathrm{v}_i}^2\mathrm{v}_k\mathrm{v}_l\right] && (24)\\
&= \left(\frac{\partial f}{\partial\theta_i}\right)^2\mathbb{E}[{\mathrm{v}_i}^4] + \sum_{j\neq i}\left(\frac{\partial f}{\partial\theta_j}\right)^2\mathbb{E}[{\mathrm{v}_i}^2]\mathbb{E}[{\mathrm{v}_j}^2] + 2\sum_{k<l}\frac{\partial f}{\partial\theta_k}\frac{\partial f}{\partial\theta_l}\mathbb{E}[{\mathrm{v}_i}^2]\mathbb{E}[\mathrm{v}_k]\mathbb{E}[\mathrm{v}_l] && (25)\\
&= \left(\frac{\partial f}{\partial\theta_i}\right)^2(1+\mathrm{Var}[{\mathrm{v}_i}^2]) + \sum_{j\neq i}\left(\frac{\partial f}{\partial\theta_j}\right)^2 && (26)\\
&= \left(\frac{\partial f}{\partial\theta_i}\right)^2\mathrm{Var}[{\mathrm{v}_i}^2] + \|\nabla f\|^2. && (27)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbb{E}[{h_{\mathbf{v}i}}^2] &= \mathbb{E}\left[\left(g_{\mathbf{v}i} - \hat{g_{\mathbf{v}i}} + \mathbb{E}[\hat{g_{\mathbf{v}i}}]\right)^2\right] && (28)\\
&= \mathbb{E}\left[\left((\nabla f\cdot\mathbf{v})\mathrm{v}_i - (\nabla\hat{f}\cdot\mathbf{v})\mathrm{v}_i + \frac{\partial\hat{f}}{\partial\theta_i}\right)^2\right] && (29)\\
&= \mathbb{E}\left[\left((\nabla f - \nabla\hat{f})\cdot\mathbf{v})\mathrm{v}_i + \frac{\partial\hat{f}}{\partial\theta_i}\right)^2\right] && (30)\\
&= \mathbb{E}\left[\left((\nabla f-\nabla\hat{f})\cdot\mathbf{v}\right)^2\mathrm{v}_i^2 + 2\left((\nabla f-\nabla\hat{f})\cdot\mathbf{v}\right)\mathrm{v}_i\frac{\partial\hat{f}}{\partial\theta_i} + \left(\frac{\partial\hat{f}}{\partial\theta_i}\right)^2\right] && (31)\\
&= \mathbb{E}\left[\left((\nabla f-\nabla\hat{f})\cdot\mathbf{v}\right)^2\mathrm{v}_i^2\right] + 2\mathbb{E}\left[((\nabla f-\nabla\hat{f})\cdot\mathbf{v})\mathrm{v}_i\right]\frac{\partial\hat{f}}{\partial\theta_i} + \left(\frac{\partial\hat{f}}{\partial\theta_i}\right)^2 && (32)\\
&= \left(\frac{\partial f}{\partial\theta_i} - \frac{\partial\hat{f}}{\partial\theta_i}\right)^2\mathrm{Var}[{\mathrm{v}_i}^2] + \|\nabla f-\nabla\hat{f}\|^2 + 2\left(\frac{\partial f}{\partial\theta_i} - \frac{\partial\hat{f}}{\partial\theta_i}\right)\frac{\partial\hat{f}}{\partial\theta_i} + \left(\frac{\partial\hat{f}}{\partial\theta_i}\right)^2 && (33)\\
&= \left(\frac{\partial f}{\partial\theta_i} - \frac{\partial\hat{f}}{\partial\theta_i}\right)^2\mathrm{Var}[{\mathrm{v}_i}^2] + \|\nabla f-\nabla\hat{f}\|^2 + 2\frac{\partial f}{\partial\theta_i}\frac{\partial\hat{f}}{\partial\theta_i} - \left(\frac{\partial\hat{f}}{\partial\theta_i}\right)^2. && (34)
\end{aligned}$$

Therefore,

$$
\begin{aligned}
&\mathbb{E}[\|\boldsymbol{h}_{\mathbf{v}} - \nabla f\|^2] && (35)\\
=&\mathbb{E}[\|\boldsymbol{h}_{\mathbf{v}}\|^2] - \|\nabla f\|^2 && (36)\\
=&\sum_{i=1}^{d} \mathbb{E}[{h_{\mathbf{v}}}_i^2] - \|\nabla f\|^2 && (37)\\
=&\sum_{i=1}^{d} \left(\frac{\partial f}{\partial \theta_i} - \frac{\partial \hat{f}}{\partial \theta_i}\right)^2 \mathrm{Var}[\mathrm{v}_i^2] + d\|\nabla f - \nabla \hat{f}\|^2 + 2\nabla f \cdot \nabla \hat{f} - \|\nabla \hat{f}\|^2 - \|\nabla f\|^2 && (38)\\
=&\sum_{i=1}^{d} \left(\frac{\partial f}{\partial \theta_i} - \frac{\partial \hat{f}}{\partial \theta_i}\right)^2 \mathrm{Var}[\mathrm{v}_i^2] + (d-1)\|\nabla f - \nabla \hat{f}\|^2 && (39)
\end{aligned}
$$

This is minimized when $\mathrm{Var}[\mathrm{v}_i^2] = 0$ for all i, which implies v_i's are independent Rademacher variables. Then, the minimized deviation is

$$\mathbb{E}[\|\boldsymbol{h}_{\mathbf{v}} - \nabla f\|^2] = (d-1)\|\nabla f - \nabla \hat{f}\|^2. \tag{40}$$

□

Proof of Theorem 4.1. To prove Theorem 4.1, we introduce Lemma A.1 and Lemma A.2.

Lemma A.1. *If $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, then for all $k \in \mathbb{N}$, we have*

$$\mathbb{E}[f(\boldsymbol{\theta}_{k+1})] - f(\boldsymbol{\theta}_k) \leq -\alpha\|\nabla f(\boldsymbol{\theta}_k)\|^2 + \frac{\alpha^2 Ld}{2}\|\nabla f(\boldsymbol{\theta}_k) - \nabla \hat{f}_k(\boldsymbol{\theta}_k)\|^2. \tag{41}$$

Proof. Since f is L-smooth, we have

$$
\begin{aligned}
f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_k) &\leq \nabla f(\boldsymbol{\theta}_k) \cdot (\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k) + \frac{L}{2}\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\| && (42)\\
&= \alpha \nabla f(\boldsymbol{\theta}_k) \cdot \boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta}_k) + \frac{\alpha^2 L}{2}\|\boldsymbol{h}_{\mathbf{v}}(\boldsymbol{\theta}_k)\| && (43)
\end{aligned}
$$

Then, taking expectations with respect to $\mathbf{v}$ at the kth step and using Equation (6) and Equation (7), we have the desired inequality. □

Lemma A.2 (Polyak (1963))**.** *μ-strong convexity implies μ-Polyak-Łojasiewicz inequality:*

$$2\mu(f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}_*)) \leq \|\nabla f(\boldsymbol{\theta})\|^2 \tag{44}$$

Proof. The proof can be found in Bottou et al. (2018). □

Now let us prove Theorem 4.1.

(a) When $\hat{f}_k \equiv 0$, Equation (41) reduces to

$$
\begin{aligned}
\mathbb{E}[f(\boldsymbol{\theta}_{k+1})] - f(\boldsymbol{\theta}_k) &\leq -\alpha\|\nabla f(\boldsymbol{\theta}_k)\|^2 + \frac{\alpha^2 Ld}{2}\|\nabla f(\boldsymbol{\theta}_k)\|^2 && (45)\\
&= -\alpha(1 - \frac{\alpha Ld}{2})\|\nabla f(\boldsymbol{\theta}_k)\|^2 && (46)\\
&\leq -\frac{\alpha}{2}\|\nabla f(\boldsymbol{\theta}_k)\|^2 && (47)\\
&\leq -\alpha\mu(f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)). && (48)
\end{aligned}
$$

Subtracting $f(\boldsymbol{\theta}_*)$, taking total expectations, and rearranging, this yields

$$\mathbb{E}[f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_*)] \leq (1 - \alpha\mu)\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)]. \tag{49}$$

By applying this inequality repeatedly, the desired inequality follows.

(b) When $\sup_{\boldsymbol{\theta}\in\mathbb{R}^d}\{\frac{\|\nabla f(\boldsymbol{\theta})-\nabla \hat{f}(\boldsymbol{\theta})\|^2}{\|\nabla f(\boldsymbol{\theta})\|^2}\} \leq r$, Equation (41) reduces to

$$\mathbb{E}[f(\boldsymbol{\theta}_{k+1})] - f(\boldsymbol{\theta}_k) \leq -\alpha\|\nabla f(\boldsymbol{\theta}_k)\|^2 + \frac{\alpha^2 Ldr}{2}\|\nabla f(\boldsymbol{\theta}_k)\|^2. \tag{50}$$

Then, the rest of the proof is the same as (a)

(c) By Lemma A.2, Equation (41) gives

$$\mathbb{E}[f(\boldsymbol{\theta}_{k+1})] - f(\boldsymbol{\theta}_k) \leq -2\alpha\mu(f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)) + \frac{\alpha^2 Ld}{2}\|\nabla f(\boldsymbol{\theta}_k) - \nabla \hat{f}_k(\boldsymbol{\theta}_k)\|^2. \tag{51}$$

Subtracting $f(\boldsymbol{\theta}_*)$, taking total expectations, and rearranging, this yields

$$\mathbb{E}[f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_*)] \leq (1-2\alpha\mu)\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)] + \frac{\alpha^2 Ld}{2}\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_k) - \nabla \hat{f}_k(\boldsymbol{\theta}_k)\|^2]. \tag{52}$$

Using the assumption Equation (10), we have

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_*)] &\leq (1-2\alpha\mu)\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)] + \frac{\alpha^2 Ldr}{2}\xi^k\|\nabla f(\boldsymbol{\theta}_0) - \nabla \hat{f}_0(\boldsymbol{\theta}_0)\|^2 && (53)\\ &\leq (1-2\alpha\mu)\mathbb{E}[f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}_*)] + \alpha\mu C\xi^k, && (54)\end{aligned}$$

where

$$C = \max\{\frac{\alpha Ld\|\nabla f(\boldsymbol{\theta}_0) - \nabla \hat{f}_0(\boldsymbol{\theta}_0)\|^2}{\mu}, f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}_*)\}. \tag{55}$$

Using this inequality, we prove Equation (11) by induction. For $k = 0$, we have

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_*)] &\leq (1-2\alpha\mu)(f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}_*)) + \alpha\mu C && (56)\\ &\leq \rho C. && (57)\end{aligned}$$

Assume Equation (11) holds for k. Then, we have

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_*)] &\leq (1-2\alpha\mu)C\rho^k + \alpha\mu C\xi^k && (58)\\ &\leq C\rho^k\left(1 - 2\alpha\mu + \alpha\mu\left(\frac{\xi}{\rho}\right)^k\right) && (59)\\ &\leq C\rho^k\left(1 - 2\alpha\mu + \alpha\mu\right) && (60)\\ &\leq C\rho^{k+1}. && (61)\end{aligned}$$

□

A.3. Analysis for the finite difference approximation

Theorem A.3. *Let v_i's be independent Rademacher variables and $f: \mathbb{R}^d \to \mathbb{R}$ be L-smooth. Then, we have the following bounds for the control variate forward gradient $\boldsymbol{h}_{\mathbf{v},\epsilon}(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \mathbb{R}^d$:*

$$\|\mathbb{E}[\boldsymbol{h}_{\mathbf{v},\epsilon}(\boldsymbol{\theta})] - \nabla f(\boldsymbol{\theta})\| \leq \frac{\epsilon L d^{3/2}}{2}, \tag{62}$$

and

$$\mathbb{E}[\|\boldsymbol{h}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta})\|^2] \leq \frac{\epsilon^2 L^2 d^3}{4} + 2(d-1)\frac{\epsilon L d^{3/2}}{2}\left\|\nabla f(\boldsymbol{\theta}) - \nabla \hat{f}(\boldsymbol{\theta})\right\| + (d-1)\left\|\nabla f(\boldsymbol{\theta}) - \nabla \hat{f}(\boldsymbol{\theta})\right\|^2 \tag{63}$$

Proof. Since f is L-smooth, we have

$$|f(\boldsymbol{x}) - f(\boldsymbol{y}) - \nabla f(\boldsymbol{y}) \cdot (\boldsymbol{x} - \boldsymbol{y})| \leq \frac{L}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 \quad (\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d). \tag{64}$$

This inequality with $\boldsymbol{\theta} + \epsilon \mathbf{v}$ and $\boldsymbol{\theta}$ yields

$$\begin{aligned} |f(\boldsymbol{\theta} + \epsilon \mathbf{v}) - f(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta}) \cdot \epsilon \mathbf{v}| &\leq \frac{L}{2} \|\epsilon \mathbf{v}\|^2 & (65) \\ &= \frac{\epsilon^2 L d}{2}. & (66) \end{aligned}$$

We denote $D_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) := \frac{f(\boldsymbol{\theta}+\epsilon\mathbf{v}) - f(\boldsymbol{\theta})}{\epsilon} - \nabla f(\boldsymbol{\theta}) \cdot \mathbf{v}$. Note that $|D_{\mathbf{v},\epsilon}(\boldsymbol{\theta})| \leq \epsilon L d/2$. Using this inequality, we can show inequality (62) as follows:

$$\begin{aligned} \|\mathbb{E}[\boldsymbol{h}_{\mathbf{v},\epsilon}(\boldsymbol{\theta})] - \nabla f(\boldsymbol{\theta})\| &= \|\mathbb{E}[\boldsymbol{g}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) - \hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta}) + \mathbb{E}[\hat{\boldsymbol{g}_{\mathbf{v}}}(\boldsymbol{\theta})]] - \mathbb{E}[\boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta})]\| & (67) \\ &= \|\mathbb{E}[\boldsymbol{g}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) - \boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta})]\| & (68) \\ &\leq \mathbb{E}[\|\boldsymbol{g}_{\mathbf{v},\epsilon}(\boldsymbol{\theta}) - \boldsymbol{g}_{\mathbf{v}}(\boldsymbol{\theta})\|] & (69) \\ &= \mathbb{E}[\left\| \frac{f(\boldsymbol{\theta} + \epsilon \mathbf{v}) - f(\boldsymbol{\theta})}{\epsilon} \mathbf{v} - (\nabla f(\boldsymbol{\theta}) \cdot \mathbf{v}) \mathbf{v} \right\|] & (70) \\ &= \mathbb{E}[|D_{\mathbf{v},\epsilon}(\boldsymbol{\theta})| \, \|\mathbf{v}\|] & (71) \\ &\leq \frac{\epsilon L d}{2} d^{1/2} & (72) \\ &= \frac{\epsilon L d^{3/2}}{2}. & (73) \end{aligned}$$

Next, we show inequality (63). To simplify the notation, we omit the argument $\boldsymbol{\theta}$. Then, we have

$$\begin{aligned} \mathbb{E}[\|\boldsymbol{h}_{\mathbf{v},\epsilon} - \nabla f\|^2] &= \mathbb{E}[\|(\boldsymbol{g}_{\mathbf{v},\epsilon} - \boldsymbol{g}_{\mathbf{v}}) + (\boldsymbol{h}_{\mathbf{v}} - \nabla f)\|^2] & (74) \\ &= \mathbb{E}[\|D_{\mathbf{v},\epsilon} \mathbf{v} + (\boldsymbol{h}_{\mathbf{v}} - \nabla f)\|^2] & (75) \\ &= \mathbb{E}[\|D_{\mathbf{v},\epsilon} \mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v} \cdot (\boldsymbol{h}_{\mathbf{v}} - \nabla f)] + \mathbb{E}[\|\boldsymbol{h}_{\mathbf{v}} - \nabla f\|^2] & (76) \\ &= \mathbb{E}[{D_{\mathbf{v},\epsilon}}^2 \|\mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v} \cdot (\boldsymbol{g}_{\mathbf{v}} - \hat{\boldsymbol{g}}_{\mathbf{v}} + \nabla \hat{f} - \nabla f)] + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (77) \\ &= \mathbb{E}[{D_{\mathbf{v},\epsilon}}^2 \|\mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v} \cdot (\boldsymbol{g}_{\mathbf{v}} - \hat{\boldsymbol{g}}_{\mathbf{v}})] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v}] \cdot (\nabla \hat{f} - \nabla f) + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (78) \\ &= \mathbb{E}[{D_{\mathbf{v},\epsilon}}^2 \|\mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v} \cdot ((\nabla f - \nabla \hat{f}) \cdot \mathbf{v}) \mathbf{v}] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v}] \cdot (\nabla \hat{f} - \nabla f) + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (79) \\ &= \mathbb{E}[{D_{\mathbf{v},\epsilon}}^2 \|\mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon}((\nabla f - \nabla \hat{f}) \cdot \mathbf{v}) \|\mathbf{v}\|^2] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v}] \cdot (\nabla \hat{f} - \nabla f) + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (80) \\ &= d\mathbb{E}[{D_{\mathbf{v},\epsilon}}^2] + 2d\mathbb{E}[D_{\mathbf{v},\epsilon}((\nabla f - \nabla \hat{f}) \cdot \mathbf{v})] + 2\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v}] \cdot (\nabla \hat{f} - \nabla f) + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (81) \\ &= d\mathbb{E}[{D_{\mathbf{v},\epsilon}}^2] + 2(d-1)\mathbb{E}[D_{\mathbf{v},\epsilon} \mathbf{v}] \cdot (\nabla f - \nabla \hat{f}) + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (82) \\ &\leq d(\frac{\epsilon L d}{2})^2 + 2(d-1) \frac{\epsilon L d^{3/2}}{2} \left\|\nabla f - \nabla \hat{f}\right\| + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2 & (83) \\ &= \frac{\epsilon^2 L^2 d^3}{4} + 2(d-1) \frac{\epsilon L d^{3/2}}{2} \left\|\nabla f - \nabla \hat{f}\right\| + (d-1) \left\|\nabla f - \nabla \hat{f}\right\|^2. & (84) \end{aligned}$$

□

A.4. Details of numerical examples

A.4.1. Details of Section 4.2

Surrogate Model In Section 4.2, our surrogate model $\hat{f}$ has two convlutional layers with 64 filters, followed by a global average pooling and a fully connected layer with 64 units. The kernel size is 1 for the Sphere function and 3 for the Rosenbrock function. The activation function is GELU (Hendrycks & Gimpel, 2016).

Results The standard deviation of the objective values are presented in Table 4, and convergence plots are shown in Figure 4.

A.4.2. Details of Section 5.1

Task In both of P and Q, solution u_τ has the following form:

$$u_\tau(z) = \sum_{i=1}^{N} c_i \sin(i\pi z), \tag{85}$$

where c_i's are sampled from different distributions. The task distribution P used for the Jacobi solver is the same as the one in Arisaka & Li (2023). For the Jacobi solver, the task difficulty is determined by how much of each frequency $\sin(i\pi z)$ is contained in the solution u because the solver reduces the error in each frequency $\sin(i\pi z)$ at a rate of $|\cos(i\pi/(N+1))|$. Considering this property, the distribution P is defined as the mixture of two distributions P_1 and P_2, where P_1 is designed to give large coefficients c_i for i with large $|\cos(i\pi/(N+1))|$ and small coefficients for i with small $|\cos(i\pi/(N+1))|$, and P_2 is designed oppositely. Specifically, P_1 gives $c_i \sim \mathcal{N}(0, \left|\frac{N+1-2i}{N-1}\right|)$, P_2 gives $c_i \sim \mathcal{N}(0, 1 - \left|\frac{N+1-2i}{N-1}\right|)$, and $P = 0.01P_1 + 0.99P_2$. Thus, P generates difficult tasks from P_1 with low probability and easy tasks from P_2 with high probability. Arisaka & Li (2023) showed that GBMS has a significant advantage in such situations where the task difficulty varies widely. Based on the same design principle, we designed Q for the multigrid solver with the weighted Jacobi smoother ($\omega = 2/3$). Because the weighted Jacobi solver reduces frequency $\sin(i\pi z)$ at a rate of $|1 - \omega + \omega\cos(i\pi/(N+1))|$, we modified Q accordingly. Specifically, the task distribution Q is a mixture of two distributions $Q = 0.01Q_1 + 0.99Q_2$, where Q_1 gives $c_i \sim \mathcal{N}(0, 1 - \frac{i}{N})$ and Q_2 gives $c_i \sim \mathcal{N}(0, \frac{i}{N})$. We sampled 15,000 tasks from each distribution of P and Q, and evenly and randomly split them into training, validation, and test sets. Since each task is sampled i.i.d., there is no data leakage issue. The discretization size is $N = 31$ in the experiments.

Solver The solver f_{Jac} solves $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by updating the solution $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{D}^{-1}\boldsymbol{b}$, where $\boldsymbol{D}$ is the diagonal of $\boldsymbol{A}$. The solver f_{MG} is the simplest two-level multigrid solver, which updates the solution $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$ by the following procedure:

1. Pre-smoothing: 2 iterations of the weighted Jacobi method for the current solution $\boldsymbol{x} = \boldsymbol{x}_k$.
2. Compute the residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$.
3. Restrict the residual $\boldsymbol{r}_c = \boldsymbol{I}_h^{2h}\boldsymbol{r}$ to the coarse grid.
4. Solve $\boldsymbol{A}_c\boldsymbol{e}_c = \boldsymbol{r}_c$ by the weighted Jacobi method with 4 iterations.
5. Prolongate the error $\boldsymbol{e}_c$ to the fine grid $\boldsymbol{e} = \boldsymbol{I}_{2h}^{h}\boldsymbol{e}_c$.
6. Correct the solution $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{e}$.
7. Post-smoothing: 2 iterations of the weighted Jacobi method to obtain $\boldsymbol{x}_{k+1}$.

Table 4. The standard deviations of objective values obtained using different gradient estimators.

(a) Sphere function

gradient	$d = 2$	$d = 8$	$d = 32$	$d = 128$
∇f	1.61e-12	3.02e-12	5.23e-12	1.04e-11
$\nabla \hat{f}$	2.09e-03	7.38e-06	1.36e-07	1.88e-04
$\boldsymbol{g}_{\mathbf{v},\epsilon}$	1.57e-05	2.71e-02	3.44e-01	5.86e+00
$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	9.11e-12	7.28e-11	6.42e-10	1.17e-07

(b) Rosenbrock function

gradient	$d = 2$	$d = 8$	$d = 32$	$d = 128$
∇f	4.49e-05	1.42e+00	1.38e+00	1.43e+00
$\nabla \hat{f}$	1.44e-01	9.66e-07	3.77e-02	6.72e+01
$\boldsymbol{g}_{\mathbf{v},\epsilon}$	3.59e-02	3.47e+00	9.85e+00	1.94e+01
$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	9.95e-04	1.08e+00	1.20e+00	1.52e+00

Table 5. The standard deviations of the results in Table 2.

Ψ	gradient	$f = f_{\text{Jac}}$	$f = f_{\text{MG}}$
Ψ_{SL}	-	1.83	0.86
Ψ_{GBMS}	∇f	0.70	0.45
	$\nabla \hat{f}$	1.35	3.85
	$\boldsymbol{g}_{\mathbf{v}}$	1.70	0.22
	$\boldsymbol{g}_{\mathbf{v},\epsilon}$	1.68	0.19
	$\boldsymbol{h}_{\mathbf{v}}$	1.09	0.60
	$\boldsymbol{h}_{\mathbf{v},\epsilon}$ (ours)	0.71	0.62

Here, $\boldsymbol{I}_{2h}^{h}$ is the linear prolongation

$$\boldsymbol{I}_{2h}^{h} = \frac{1}{2}\begin{pmatrix} 1 & & & & \\ 2 & & & & \\ 1 & 1 & & & \\ & 2 & & & \\ & 1 & 1 & & \\ & & 2 & \ddots & \\ & & 1 & \ddots & \\ & & & & 1 \\ & & & 2 & \\ & & & & 1 \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N/2-1)}, \tag{86}$$

$\boldsymbol{I}_{h}^{2h}$ is the weighted restriction

$$\boldsymbol{I}_{h}^{2h} = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 & & & & & & & & \\ & & 1 & 2 & 1 & & & & & & \\ & & & & 1 & 2 & 1 & & & & \\ & & & & & & & \ddots & & & \\ & & & & & & & & 1 & 2 & 1 \end{pmatrix} \in \mathbb{R}^{(N/2-1)\times(N-1)}, \tag{87}$$

and $\boldsymbol{A}_c = \boldsymbol{I}_{h}^{2h}\boldsymbol{A}\boldsymbol{I}_{2h}^{h}$ is the restriction of $\boldsymbol{A}$ to the coarse grid.

Meta-solver The meta-solver Ψ is a fully-connected neural network with one hidden layer with 512 units and SiLU activation function (Elfwing et al., 2018). It takes the right-hand side b_τ as input and gives the coefficients c_i's in Equation (85) to construct initial guess θ_τ.

Surrogate Model In Section 5.1, our surrogate model $\hat{f}$ is a fully-connected neural network with three hidden layers with 1024 units and GELU activation function. It takes the initial guess $\boldsymbol{\theta}_\tau$ and the solver's output $\hat{u}_\tau$ as input.

Training We use the Adam optimizer for all training. For the Jacobi method, we use learning rate 10^{-5} for the meta-solver and 5.0×10^{-4} for the surrogate model. For the multigrid method, we use learning rate 10^{-6} for the meta-solver and 5.0×10^{-4} for the surrogate model. For the supervised baseline, we use learning rate 10^{-4}. The batch size is set to 256 for all training, and the number of epochs is set to 100. The best model is selected based on the validation loss. The finite difference step size ϵ is set to 10^{-12} for all training. We conducted the experiments with 3 different random seeds.

Results The standard deviations of the results Table 2 are presented in Table 5.

A.4.3. Details of Section 5.2

Task The problem formulation is based on the demos in the DOLFIN documentation (noa). For the Biharmonic equation, we use a triangular mesh of the 2D unit square $[0, 1]^2$, where we have 3 cells in each direction, so the total number of triagles is $2 \times 3 \times 3 = 18$. For the finite element family, we use the standard Lagrange family with degree 2. For solving the biharmonic equation, a discontinuous Galerkin approach is used to weakly impose continuity of the normal derivative. For the linear elasticity, we use a tetrahedral mesh of the 3D cube $[0, 1]^3$, where we have 3 cells in each direction, so the total number of tetrahedra is $6 \times 3 \times 3 \times 3 = 162$. For the finite element family, we use the standard Lagrange family with degree 1.

For the biharmonic problem, the source term b_τ has the following form:

$$b_\tau(x, y) = c_1 \sin(c_2 \pi x) \sin(c_3 \pi y), \tag{88}$$

where

$$c_1 \sim \mathrm{LogUniform}([10^{-2}, 10^2]), \tag{89}$$
$$c_2 \sim \mathrm{DiscreteUniform}([1, 2, 3, 4, 5]), \tag{90}$$
$$c_3 \sim \mathrm{DiscreteUniform}([1, 2, 3, 4, 5]). \tag{91}$$

The number of tasks for training, validation, and test are all $5,000$ for both problem settings.

Solver The algebraic multigrid solver f_{AMG} is constructed by Geometric algebraic multigrid preonditioner `PCGAMG` and Richardson iterative method `KSPRICHARDSON` in PETSc (Balay et al., 2023). Because AMG is implemented in PETSc as a preconditioner, we need to use it with the Richardson method: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{B}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k)$, where $\boldsymbol{B}$ is the application of the preconditioner. We use `PCGAMG` and `KSPRICHARDSON` with their default parameters, which gives a standard v-cycle algebraic multigrid solver. Specifically, the solver f_{AMG} is constructed by setting the following PETSc options:

- `-ksp_type richardson`
- `-ksp_initial_guess_nonzero True`
- `-pc_type gamg`

Other options are set to default values.

Meta-solver The meta-solver Ψ is a fully-connected neural network with one hidden layer with 1024 units and SiLU activation function for both biharmonic and elasticity problem.

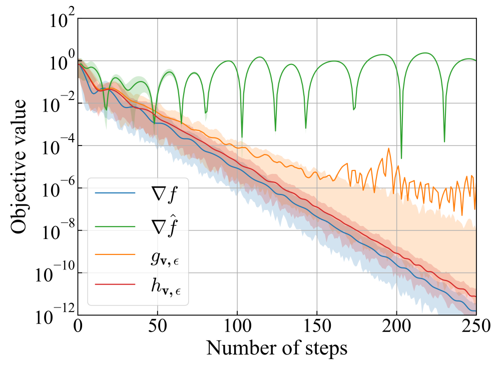
Surrogate Model Surrogate model $\hat{f}$ is a fully-connected neural network with three hidden layers with 512 units and GELU activation function for both biharmonic and elasticity problem. It takes the initial guess $\boldsymbol{\theta}_\tau$ and the solver's solution as input.

Training We use the Adam optimizer for all training. For the biharmonic problem, we use learning rate 10^{-5} for the meta-solver and 5.0×10^{-4} for the surrogate model. For the elasticity problem, we use learning rate 10^{-4} for the meta-solver and 10^{-4} for the surrogate model. The batch size is set to 256 for all training. We train them for 1,000 epochs and reduce the learning rate by a factor of 0.1 at 500 and 750 epochs. The best model is selected based on the validation loss.

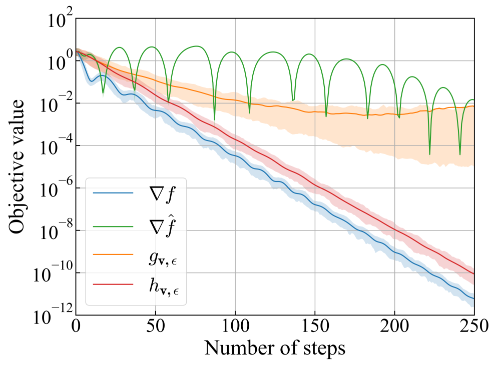
A.5. Learning Preconditioners

Beyond learning initial guesses, NI-GBMS can also learn preconditioners. Here, we use NI-GBMS to solve linear elasticity problems similar to the one in Section 5.2. The difference from Section 5.2 is that we use the conjugate gradient (CG) method with incomplete Cholesky preconditioner as the solver f and learn the diagonal shift parameter of the incomplete Cholesky preconditioner. We follow the same experimental settings as in the section 4.2 of Arisaka & Li (2023) except for the solver implementation and training methodology. Against the differentiable solver in Arisaka & Li (2023), which is implemented in a deep learning framework, our solver f is implemented in PETSc and non-automatic-differentiable. Thus, we train the meta-solver Ψ using NI-GBMS. Note that it has only one scalar parameter (diagonal shift) resulting in $\boldsymbol{h}_{\mathbf{v}} = \boldsymbol{g}_{\mathbf{v}}$.

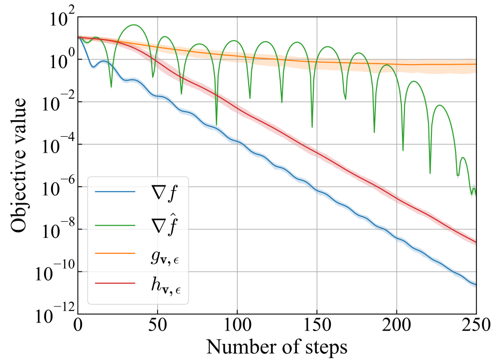
The performance of NI-GBMS is almost the same as the performance reported in Arisaka & Li (2023). The meta-solver trained using NI-GBMS reduces the number of iterations by 54% from a non-learning baseline, while the one in Arisaka & Li (2023) reduces it by 55%. Although this experiment is simple, it shows that NI-GBMS is applicable to learning other hyperparameters beyond initial guesses.



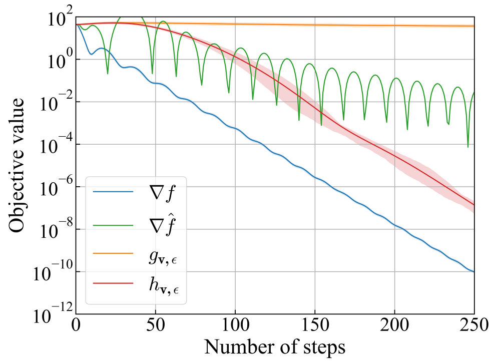
(a) sphere2D



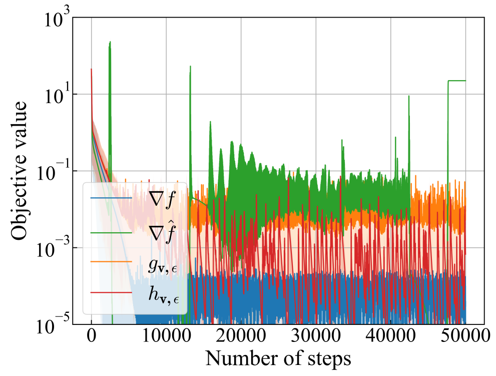
(b) sphere8D



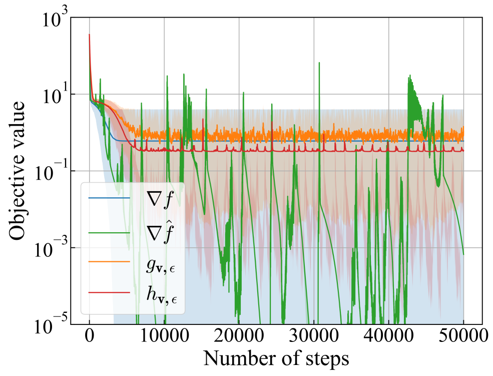
(c) sphere32D



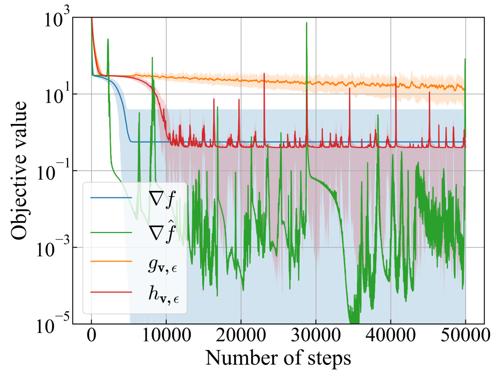
(d) sphere128D



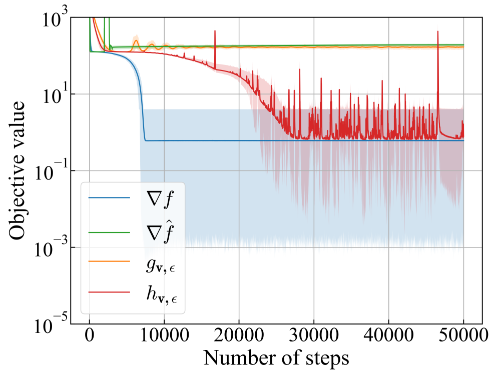
(e) rosenbrock2D



(f) rosenbrock8D



(g) rosenbrock32D



(h) rosenbrock128D

Figure 4. Convergence plots. The horizontal axis is the number of Adam steps, and the vertical axis is the objective value.