# UNFIXED BIAS ITERATOR: A NEW ITERATIVE FORMAT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Partial differential equations (PDEs) have a wide range of applications in physics and computational science. Solving PDEs numerically is usually done by first meshing the solution region with finite difference method (FDM) and then using iterative methods to obtain an approximation of the exact solution on these meshes, hence decades of research to design iterators with fast convergence properties. With the renaissance of neural networks, many scholars have considered using deep learning to speed up solving PDEs, however, these methods leave poor theoretical guarantees, or sub-convergence. We build our iterator on top of the existing standard hand-crafted iterative solvers. At the operational level, for each iteration, we use a deep convolutional network to modify the current iterative result based on the historical iterative results as a way to achieve faster convergence. At the theoretical level, due to the introduced historical iterative results, our iterator is a new iterative format: Unfixed Bias Iterator. We provide sufficient theoretical guarantees, and theoretically prove that our iterator can obtain correct results with convergence, as well as a better generalization. Finally, sufficient numerical experiments show that our iterator has a convergence speed far beyond that of other iterators and exhibits strong generalization ability.

## 1 INTRODUCTION

Partial differential equations (PDEs) play an important role in science and engineering disciplines, and numerous physical problems are themselves PDEs, such as sound propagation, fluid flow, electromagnetic fields, *etc*.. Numerical solutions to PDEs have been developed over the past few decades and successfully applied to many real-world applications, *i.e*., aerodynamics and weather forecasting. These numerical methods, the most representative of which is the finite difference method (FDM) Mitchell & Griffiths (1980), first grid the solution region of the problem and then obtain an approximation of the exact solution at discrete grid points. Nevertheless, as the size and complexity of the problem increase, the computational complexity of the numerical methods becomes extremely large, and therefore, reducing the computational complexity of the numerical solution of PDEs has become a hot research topic in recent years.

With the renaissance of neural networks, data-driven deep learning methods have made breakthroughs on a variety of tasks Qi et al. (2016); He et al. (2015); Vinod (2019), and as a result, some approaches Bachouch et al. (2018); Beck et al. (2019); Pirmorad et al. (2021); Luz et al. (2020); Berg & Nyström (2017); Chan-Wai-Nam et al. (2019); Han et al. (2017); Huré et al. (2019); Hutzenthaler et al. (2019); Li et al. (2003); Khoo et al. (2018); Lagaris et al. (1998; 2000); Lambrianides et al. (2020); Yang et al. (2020); Lye et al. (2020) have considered using deep learning models to reduce the computational complexity of solving PDEs, and have achieved encouraging results. Some methods Huang et al. (2021a); Li et al. (2021); Huang et al. (2021b); Han & Lee (2021a;b); Ben-Yair et al. (2021); Malik et al. (2020); Stevens & Colonius (2020); Li et al. (2020); Dockhorn (2019) directly output numerical solutions of PDEs by training an end-to-end deep learning model with the conditions of the PDEs as input. The computational complexity of these methods depends entirely on the size of the network, but their accuracy and generalization are not guaranteed given the uninterpretability of the deep models, so these methods are not applicable in some fields where the accuracy of numerical solutions is strictly required.

To solve this problem, some of the methods Shit et al. (2019); He & Xu (2019); Greenfeld et al. (2019); Taghibakhshi et al. (2021); Hsieh et al. (2019) are built directly on top of existing hand-designed iterative methods Frankel (1950); Gräser & Sander (2014); Hackbusch & Wittum (1998), and they can inherit many qualities of existing iterative methods, such as the ability to obtain numerical solutions with arbitrary accuracy, a certain degree of generalization, *etc*., and even some of them Shit et al. (2019); Hsieh et al. (2019) carry out theoretical proofs. Specifically, they train a deep network which can correct the current iteration value by using the residual between the current iteration value and the last iteration value, but using only the residual of two adjacent iterations to estimate the correction value is obviously not accurate enough. In this paper, we consider using the residual generated by all adjacent iterations to estimate the correction value, which can greatly improve the accuracy of the correction value and thus substantially speed up the convergence speed of the iterator. If we regard the residual analogously to the gradient, other methods are similar to the gradient descent method, while our method is similar to the momentum gradient descent method Ruder (2016).

At the operational level, our method only applies the momentum trick compared to other methods, but at the theoretical level, the difference between our method and other methods is that most of the hand-designed iterative methods and deep learning-based iterative methods are by nature a linear iterator Hsieh et al. (2019), while ours is not. A linear iterator has a fixed update matrix ($T$) and bias ($c$). The purpose of the models trained by the deep learning-based method is to find a $T$ and $c$ that makes the iterator converge faster. The disadvantage of this class of iterators is that the bias $c$ cannot be modified adaptively with iteration, limiting the flexibility of the iterator. However, our iterator no longer has a fixed bias due to the introduction of residuals generated by historical iteration values, and its bias can be continuously changed with iterations, enabling it to give a different bias for each iteration, which is more flexible and versatile, and also has faster convergence. We call our iterator an Unfixed Bias Iterator. In fact, the linear iterator is just a special case of an unfixed bias iterator.

Linear iterator is widely used in the industry despite its many drawbacks because it has sufficient theoretical guarantees to determine whether a new linear iterator converges and has generalization. This paper also provides sufficient theoretical guarantees for our proposed new iterative format: the unfixed bias iterator, and analyzes the convergence and generalization of our iterator. Finally, adequate numerical experiments show that our method can converges much faster than other methods.

Our contributions are summarized as follows:

- We introduce a new iterative format: the unfixed bias iterator, and provide sufficient theoretical guarantees for it, after which we prove theoretically that our iterator can converge and has some generalization.
- We propose a novel iterator that can estimate the correction value of the current iteration value using the residuals generated by all adjacent iteration values. The correction value estimated by our iterator is more accurate compared to other methods, thus greatly improving the convergence speed of the iteration.
- Experiments show that our iterator obtains SOTA performance in terms of convergence speed.

## 2 Related Work

### 2.1 End-to-end Methods

The end-to-end methods Huang et al. (2021a); Li et al. (2021); Huang et al. (2021b); Han & Lee (2021a;b); Ben-Yair et al. (2021); Malik et al. (2020); Stevens & Colonius (2020); Li et al. (2020); Dockhorn (2019) usually train a neural network as the solver of PDEs, and different neural networks are also used for different PDEs. Stevens & Colonius (2020) uses a fully convolutional Long Short-Term Memory (LSTM) Greff et al. (2015) network to exploit the spatiotemporal dynamics of PDEs. The neural network serves to enhance finite-difference and finite-volume methods (FDM/FVM) that are commonly used to solve PDEs, allowing the method to maintain guarantees on the order of convergence. Meta-Auto-Decoder (MAD) Huang et al. (2021b) treats solving parametric PDEs as a meta-learning problem and utilizes the Auto-Decoder structure Park et al. (2019) to deal with different tasks/PDEs. Physics-informed losses induced from the PDEs governing equations and boundary conditions is used as the training losses for different tasks. The primary idea of Han & Lee (2021b) is to use graph neural network (GNN) Scarselli et al. (2009) for modeling the spatial domain, and Neural ODE for modeling the temporal domain. The attention mechanism Correia & Colombini (2021) identifies important inputs/features and assign more weightage to the same; this enhances the performance of the proposed framework. Using conditional generative adversarial networks (cGAN) Jaiswal et al. (2017), Farimani et al. (2017) trains models for the direct generation of solutions to steady state heat conduction and incompressible fluid flow purely on observation without knowledge of the underlying governing equations. End-to-end algorithms use mostly uninterpretable neural networks, so they can only verify the accuracy of their methods empirically rather than theoretically. However, our method provides a theoretical explanation.

### 2.2 Iterative Methods

Iterative methods Shit et al. (2019); He & Xu (2019); Greenfeld et al. (2019); Taghibakhshi et al. (2021); Hsieh et al. (2019) are built on top of existing hand-designed iterative methods, thus inheriting many of their advantages. A neural solverShit et al. (2019) to learn an optimal iterative scheme for a class of PDEs, in a data-driven fashion and attains this objective by modifying an iteration of an existing semi-implicit solver using a deep neural network. Multigrid Network (MgNet) He & Xu (2019) develops an unified model that simultaneously recovers some convolutional neural networks for image classification and multigrid (MG) Hackbusch & Wittum (1998) methods for solving discretized PDEs. Greenfeld et al. (2019) learns a (single) mapping from a family of parameterized PDEs to prolongation operators and trains a neural network for the entire class of PDEs, using an efficient and unsupervised loss function. Taghibakhshi et al. (2021) proposes a method using a reinforcement learning (RL) Puranik (2021) agent based on graph neural networks Scarselli et al. (2009), which can learn to perform graph coarsening on small training graphs and then be applied to unstructured large graphs in Multigrid. Hsieh et al. (2019) proposes an approach to learn a fast iterative solver tailored to a specific domain and achieves this goal by learning to modify the updates of an existing solver using a deep neural network. The iterative methods using nonlinear neural network can not guarantee the accuracy as the end-to-end method. Although the methods using only linear neural network can guarantee the accuracy in theory, they are essentially linear iterators, which limits their performance.

## 3 Methodology

### 3.1 Notations

The purpose of the linear PDEs solver is to find functions $u$ that satisfy a series of partial differential equations and boundary conditions, as follows:

$$\begin{cases} \mathscr{A}u(x) = \mathscr{f}(x), x \in \mathscr{G} \\ u(x) = \mathscr{b}(x), x \in \partial\mathscr{G} \end{cases} \tag{1}$$

where $u \in \mathscr{F} = \{u : \mathbb{R}^k \to \mathbb{R}\}$ is the equation to be solved, $\mathscr{A} : \mathscr{F} \to \mathscr{F}$ is a linear operator, $\mathscr{f} : \mathbb{R}^k \to \mathbb{R}$ and $\mathscr{b} : \mathbb{R}^k \to \mathbb{R}$ are a function that can be obtained naturally according to the problem, and $\mathscr{G}$ is the domain of definition of the function to be solved. The problem domain of PDEs is discretized into an $n \times n \times \cdots \times n$ ($k$ many) uniform Cartesian grid with mesh width $h$ by using FDM Mitchell & Griffiths (1980), the solution of the following equation at discrete grid points is an approximation of the solution of Eq. 1:

$$\begin{cases} G(Au) = Gf \\ (1-G)u = (1-G)b \end{cases} \tag{2}$$

where $G, A \in \mathbb{R}^{n^k \times n^k}$ the matrices of $\mathscr{G}$ and $\mathscr{A}$ after discretization, and $u, b$ and $f \in \mathbb{R}^{n^k}$ the vectors of $u$, $\mathscr{b}$ and $\mathscr{f}$ after discretization. Therefore, a discretized PDE problem consists of five components $(A, G, f, b, n)$. In this paper, we fix $A$ but vary $G, f, b, n$, and are interested in learning an iterator that solves a class of PDE problems governed by the same $A$.

### 3.2 Iterator and Training

The difference between our iterator and other iterators at the operational level is that we apply the residuals generated by all two adjacent iterations to each estimated correction, this idea inspired by the great success of momentum gradient descent in the field of deep model optimization. Specifically, for a fixed PDE problem class $A$, let $\varphi$ be a standard iterative solver, such as Jacobi iterator. We will use more formal notation $\varphi(u; G, f, b, n)$ as $\varphi$ that is a function of $u$, but also depends on $G, f, b, n$. We rely on $\varphi$ to design our new iterator $\psi_H : \mathbb{R}^{n^k} \to \mathbb{R}^{n^k}$ as:

$$\begin{cases} w_{i+1} = \varphi(u_i; G, f, b, n) - u_i \\ z_{i+1} = \theta G H w_{i+1} + (1-\theta) z_i \\ u_{i+1} = \varphi(u_i; G, f, b, n) + z_{i+1} \end{cases} \tag{3}$$

where $z_0 = 0$, $u_0$ is an arbitrary initial vector, $H$ is a learned linear operation and $\theta \in [0, 1]$ is a hyper-parameter. If $\theta = 0$, our iterator degenerates to Jacobi iterator. If $\theta = 1$, our iterator degenerates to the iterator in Hsieh et al. (2019). Therefore, they are special cases of our iterator. In our experiments, $\theta$ is fixed to 0.8. When there is no confusion, we neglect the dependence on $G, f, b, n$ and denote $\varphi(u; G, f, b, n)$ as $\varphi(u)$, and $\psi_H(u; G, f, b, n)$ as $\psi_H(u)$.

We train our iterator $\psi_H(u; G, f, b, n)$ to converge quickly to the ground truth solution on a set of problem instances. For each instance, the ground truth solution $u_*$ is obtained from the existing solver $\varphi$. The loss function is:

$$\min_H \mathbb{E} ||\psi_H^l(u_0; G, f, b, n) - u_*||_2^2, \tag{4}$$

where $u_0 \sim N(0, 1)$. For the training data of each batch, the number of iterations $l$ may be different. In our experiment, $l$ is uniformly chosen from $[1, 20]$.

## 3.3 Theoretical Analysis

Since our iterator is no longer a linear iterator, we first define a new iterator format: unfixed bias iterator and give the sufficient conditions for its convergence, and finally discuss the accuracy and generalization of our iterator at the theoretical level.

### 3.3.1 Unfixed Bias Iterator

As before, denote $\varphi(u_i) = T u_i + c$. Observe that:

$$\begin{aligned} \psi_H(u_i) &= \varphi(u_i) + z_{i+1} \\ &= T u_i + c + \theta G H (T u_i + c - u_i) + (1-\theta) z_i \\ &= (T + \theta G H T - \theta G H) u_i + c + \theta G H c \\ &+ \theta \sum_{j=1}^{i} (1-\theta)^{i-j} H w_j. \end{aligned} \tag{5}$$

Since the offset $(c + \theta G H c + \theta \sum_{j=1}^{i} (1-\theta)^{i-j} H w_j)$ is no longer a constant vector, our iterator is a completely new format, which we call it unfixed bias iterator:

**Definition 3.1** (Unfixed Bias Iterator). An unfixed bias iterator is a function $\phi : \mathbb{R}^{n^k} \to \mathbb{R}^{n^k}$ that can be expressed as:

$$u_{i+1} = \phi(u_i) = T u_i + c_i, \tag{6}$$

where $T$ is called the update matrix, and $c_i$ is a unfixed bias vector.

Whether an unfixed bias iterator converges depends on $T$ and $c_i$, and has the following theorem:

**Theorem 3.2.** *An unfixed bias iterator $\phi$ converges to a unique fixed point from any initial vector if and only if the spectral radius $\rho(T) < 1$ and $\lim_{n\to\infty} c_n = c_*$.*

*Proof.* **Necessity:** If $u_*$ is a fixed point, then $u_* = T u_* + c_*$. Observe that:

$$\begin{aligned} u_{k+1} - u_* &= T(u_k - u_*) + (c_k - c_*) \\ &= T^2 (u_{k-1} - u_*) + T(c_{k-1} - c_*) + (c_k - c_*) \\ &\vdots \\ &= T^{k+1}(u_0 - u_*) + \sum_{i=0}^{k} T^{k-i}(c_i - c_*) \end{aligned} \tag{7}$$

By the arbitrariness of $u_0$, we have that:

$$\lim_{n\to\infty} T^n = O \Rightarrow \rho(T) < 1. \tag{8}$$

According to Lemma A.1, we can infer that:

$$\lim_{n\to\infty} (c_n - c_*) = 0 \Rightarrow \lim_{n\to\infty} c_n = c_*. \tag{9}$$

**Sufficiency:** If $\rho(T) < 1$, then $|I - T| \neq 0$, then the equation $(I - T)u = c_*$ has a unique solution, which is recorded as $u_*$. For any $u_0$, according to Lemma A.2, we have that:

$$\begin{aligned}\lim_{n\to\infty} (u_n - u_*) &= \lim_{n\to\infty} T^n(u_0 - u_*) \\ &+ \lim_{n\to\infty} \sum_{i=0}^{n-1} T^{n-1-i}(c_i - c_*) \\ &= 0 + 0 = 0 \\ &\Rightarrow \lim_{n\to\infty} u_n = u_*.\end{aligned} \tag{10}$$

□

### 3.3.2 Accuracy

The correct PDE solution is a fixed point of $\psi_H$ by the following theorem:

**Theorem 3.3.** *For any PDE problem $(A, G, f, b, n)$ and choice of $H$, $u_* = \lim_{n\to\infty} \psi_H^n(u_0)$, if $u_* = \lim_{n\to\infty} \varphi^n(u_0)$ and $\lim_{x\to 0} Hx = 0$.*

*Proof.* When $\theta = 0$, our iterator degenerates into a Jacobi iterator, and the theorem obviously holds.

When $\theta = 1$, our iterator degenerates into the iterator in Hsieh et al. (2019), and the theorem has been proved by Hsieh et al. (2019).

When $0 < \theta < 1$, since $u_* = \lim_{n\to\infty} \psi_H^n(u_0)$, according to Cauchy criterion, we have that:

$$\lim_{n\to\infty} w_n = 0. \tag{11}$$

Because of $\lim_{x\to 0} Hx = 0$, we know that:

$$\lim_{n\to\infty} Hw_n = 0. \tag{12}$$

If we define $w_0 = 0$, we can obtain:

$$\begin{aligned} z_n &= \theta H w_n + (1-\theta) z_{n-1} \\ &= \theta H w_n + (1-\theta)(\theta H w_{n-1} + (1-\theta) z_{n-2}) \\ &\vdots \\ &= \theta \sum_{i=0}^{n} (1-\theta)^{n-i} H w_i \end{aligned} \tag{13}$$

Since $0 < (1-\theta) < 1$ and $\lim_{n\to\infty} Hw_n = 0$, according to Lemma A.4, we can deduce that:

$$\lim_{n\to\infty} z_n = \theta \lim_{n\to\infty} \sum_{i=0}^{n} (1-\theta)^{n-i} H w_i = 0. \tag{14}$$

To sum up, we have that:

$$\begin{aligned}\lim_{n\to\infty} \psi^n(u_0) &= \lim_{n\to\infty} (\varphi^n(u_0) + z_{n+1}) \\ &= \lim_{n\to\infty} \varphi^n(u_0) + \lim_{n\to\infty} z_n \\ &= u_* + 0 = u_*\end{aligned} \tag{15}$$

□

For any PDE problem, Theorem 3.3 points out that once our iterator converges, the fixed point obtained by our iterator must be accuracy. This means that our iterator can obtain solutions with arbitrary precision just as well as the hand-designed iterator.

### 3.3.3 Generalization

For the PDE problem $(A, G, f, b, n)$, in the case of fixed $A$, even if we can only use a limited combination of $(G, f, b, n)$ to train our model, the model shows surprising generalization properties, which we show in Theorem 3.4:

**Theorem 3.4.** *For fixed $A, G, n$ and fixed $H$, if for one $f_0, b_0$, $\psi_H(u; G, f_0, b_0, n)$ is accuracy for the PDE problem $(A, G, f_0, b_0, n)$, then for all $f$ and $b$, the iterator $\psi_H(u; G, f, b, n)$ is accuracy for the PDE problem $(A, G, f, b, n)$.*

*Proof.* From Theorem 3.2 and Theorem 3.3, our iterator is accuracy if and only if the following conditions holds:

$$\begin{cases} \rho(T + \theta GHT - \theta GH) < 1 \\ \lim_{n\to\infty}(c + \theta GHc + \theta \sum_{i=1}^{n} H(1-\theta)^{n-i} w_i) = c_* \end{cases} \tag{16}$$

Since $T$ depends on $A, G, n$, and $c$ depends on $A, G, f, b, n$, it follows that the establishment of $\rho(T + \theta GHT - \theta GH) < 1$ only depends on $A, G, n$ and has nothing to do with $f, b$.

For a fixed $A, G, n$ such that $\psi_H(u; G, f_0, b_0, n)$ is accuracy, there must be $\rho(T) < 1$, then we have $u_* = \lim_{n\to\infty} \varphi^n(u_0)$ for all $f, b$. Therefore, according to the proof of Theorem 3.3, we have that:

$$\begin{aligned} &\lim_{n\to\infty}(c + \theta GHc + \theta \sum_{i=1}^{n} H(1-\theta)^{n-i} w_i) \\ &= c + \theta GHc + \lim_{n\to\infty}(\theta \sum_{i=1}^{n} H(1-\theta)^{n-i} w_i) \\ &= c + \theta GHc + 0 = c + \theta GHc. \end{aligned} \tag{17}$$

Since the above equation holds for all $f, b$, that is, the establishment of $\lim_{n\to\infty}(c + \theta GHc + \theta \sum_{i=1}^{n} H(1-\theta)^{n-i} w_i) = c_*$ is independent of $f, b$.

In summary, the accuracy of the iterator is independent with $f$ and $b$. Thus, if the iterator is accuracy for one $f_0$ and $b_0$, then it is accuracy for any choice of $f$ and $b$. $\square$

Theorem 3.4 states that our iterator can be freely generalized to different $f$ and $b$. There is no guarantee that our iterator can generalize to different $G$ and $n$. Generalization to different $G$ and $n$ has to be empirically verified: in our experiments, our learned iterator converges to the correct solution for a variety of grid sizes $n$ and geometries $G$, and we have not observed that any $G, n$ make our iterators non convergent.

Even if some $G, n$ makes our iterator generalization fail, there is no risk of obtaining incorrect results. The iterator will simply fail to converge. This is because according to Theorem 3.3, fixed points of our new iterator is the same as the fixed point of hand designed iterator $\varphi$. Therefore if our iterator is convergent, it is accuracy.

## 4 Experiments

### 4.1 Experimental Setting

For fair comparison, we follow Hsieh et al. (2019) to prepare our experimental setting, including data set, evaluation criterion and the implementation of $H$.

**Data set:** To reemphasize, our goal is to train a model on simple domains where the ground truth solutions can be easily obtained, and then evaluate its performance on more complex geometries and boundary conditions.

For training, we select the simplest homogeneous equation on a 2-dimensional square domain ($k = 2$) with boundary conditions such that each side is a random fixed value. In this case, the equation can be solved quickly by Jacobi method, so we can obtain a large number of training data with low

| Model | Baseline | Square $layers/ops$ | L-shape $layers/ops$ | Cylinders $layers/ops$ | $f \neq 0$ $layers/ops$ |
|---|---|---|---|---|---|
| Conv1 Hsieh et al. (2019) | Jacobi | 0.432/0.702 | 0.432/0.702 | 0.432/0.702 | 0.431/0.701 |
| Conv2 Hsieh et al. (2019) | Jacobi | 0.286/0.524 | 0.286/0.524 | 0.286/0.524 | 0.285/0.522 |
| Conv3 Hsieh et al. (2019) | Jacobi | 0.219/0.424 | 0.219/0.423 | 0.220/0.426 | 0.217/0.421 |
| Conv4 Hsieh et al. (2019) | Jacobi | 0.224/0.449 | 0.224/0.449 | 0.224/0.448 | 0.222/0.444 |
| U-Net2 Hsieh et al. (2019) | Multigrid | 0.091/0.205 | 0.090/0.203 | 0.091/0.204 | 0.079/0.178 |
| U-Net3 Hsieh et al. (2019) | Multigrid | 0.220/0.494 | 0.213/0.479 | 0.201/0.453 | 0.185/0.417 |
| Unfixed-Bias-Conv1 | Jacobi | 0.175/0.328 | 0.206/0.386 | 0.221/0.415 | 0.163/0.306 |
| Unfixed-Bias-Conv2 | Jacobi | **0.146/0.291** | **0.169/0.337** | **0.175/0.350** | **0.142/0.284** |
| Unfixed-Bias-Conv3 | Jacobi | 0.155/0.319 | 0.176/0.362 | 0.185/0.382 | 0.153/0.316 |
| Unfixed-Bias-Conv4 | Jacobi | 0.171/0.360 | 0.194/0.408 | 0.203/0.427 | 0.162/0.340 |
| Unfixed-Bias-U-Net2 | Multigrid | 0.029/0.066 | 0.033/0.074 | 0.032/0.072 | 0.028/0.064 |
| Unfixed-Bias-U-Net3 | Multigrid | **0.014/0.031** | **0.014/0.031** | **0.015/0.033** | **0.014/0.031** |

Table 1: Results in solving Possion Equation.

| Model | Baseline | Square $layers/ops$ | L-shape $layers/ops$ | Cylinders $layers/ops$ | $f \neq 0$ $layers/ops$ |
|---|---|---|---|---|---|
| Conv1 (a=1) Hsieh et al. (2019) | Jacobi | 0.422/0.685 | 0.394/0.640 | 0.423/0.687 | 0.258/0.420 |
| Unfixed-Bias-Conv1 (a=1) | Jacobi | 0.183/0.343 | 0.232/0.434 | 0.258/0.483 | 0.186/0.349 |
| Unfixed-Bias-Conv2 (a=1) | Jacobi | **0.160/0.319** | **0.173/0.346** | **0.181/0.363** | **0.145/0.290** |
| Conv1 (a=2) Hsieh et al. (2019) | Jacobi | 0.396/0.643 | 0.344/0.559 | 0.350/0.569 | 0.270/0.440 |
| Unfixed-Bias-Conv1 (a=2) | Jacobi | 0.178/0.333 | 0.214/0.401 | 0.230/0.431 | 0.172/0.323 |
| Unfixed-Bias-Conv2 (a=2) | Jacobi | **0.145/0.289** | **0.174/0.349** | **0.176/0.353** | **0.139/0.279** |
| Conv1 (a=3) Hsieh et al. (2019) | Jacobi | 0.383/0.622 | 0.405/0.659 | 0.513/0.834 | 0.381/0.619 |
| Unfixed-Bias-Conv1 (a=3) | Jacobi | **0.150/0.281** | **0.161/0.303** | 0.179/0.336 | **0.194/0.363** |
| Unfixed-Bias-Conv2 (a=3) | Jacobi | 0.155/0.309 | 0.170/0.340 | **0.172/0.345** | 0.197/0.393 |

Table 2: Results in solving Helmholtz Equation.

computational cost. This setting is also used in Sharma et al. (2018); Farimani et al. (2017); Hsieh et al. (2019).

For testing, we use larger grid sizes $n$ than training. For example, choose $256 \times 256$ grid sizes to test our model, which is trained on $64 \times 64$ grids. Moreover, we design some challenging geometries to test the generalization of our models: (i) same geometry but larger grid, (ii) L-shape geometry, (iii) Cylinders geometry, and (iv) PDEs in the same geometry, but $f \neq 0$. L-shape and cylinders geometries are designed because the models are trained on square domains and have never seen sharp or curved boundaries. Examples of the four settings are shown in Appendix B.3.

**Implementation of $H$:** We use convolution to realize $A$. Similarly, we use the stack of convolutional layers or U-Net Ronneberger et al. (2015) with linear operation only to implement $H$, and different implementation methods use different data sets.

For the implementation of convolutional layer stacking, we call it Unfixed-Bias-ConvX model. The model is trained on the square domain with $16 \times 16$, and tested on $64 \times 64$ grid sizes.

For the implementation of U-Net, we call it Unfixed-Bias-U-NetX model. The model is trained on the square domain with $64 \times 64$, and tested on $256 \times 256$ grid sizes.

**Evaluation Criterion:** For Unfixed-Bias-Conv models, we compare them against Jacobi method.

On GPU, the Jacobi iterator and our model can both be efficiently implemented as convolution layers. Thus, we measure the computation cost by the number of convolutional layers. Suppose Jacobi method converges after $N$ iterations, and our model converges after $M$ iterations. Then the evaluation criteria $layers$ is calculated as follows:

$$layers = \frac{M(1+L)}{N}, \tag{18}$$

| Model | Baseline | Square $layers/ops$ | L-shape $layers/ops$ | Cylinders $layers/ops$ | $f \neq 0$ $layers/ops$ |
|---|---|---|---|---|---|
| Unfixed-Bias-Conv1 (a=1) | Jacobi | 0.247/0.462 | 0.196/0.368 | 0.208/0.390 | 0.242/0.453 |
| Unfixed-Bias-Conv2 (a=1) | Jacobi | **0.212/0.425** | **0.177/0.354** | **0.187/0.373** | **0.207/0.415** |
| Unfixed-Bias-Conv3 (a=1) | Jacobi | 0.259/0.533 | 0.203/0.419 | 0.220/0.454 | 0.251/0.518 |
| Unfixed-Bias-Conv1 (a=2) | Jacobi | 0.241/0.362 | 0.207/0.311 | 0.202/0.303 | 0.235/0.353 |
| Unfixed-Bias-Conv2 (a=2) | Jacobi | **0.208/0.390** | **0.186/0.352** | **0.198/0.371** | **0.205/0.384** |
| Unfixed-Bias-Conv3 (a=2) | Jacobi | 0.253/0.505 | 0.237/0.473 | 0.227/0.454 | 0.243/0.485 |
| Unfixed-Bias-Conv1 (a=3) | Jacobi | 0.261/0.392 | 0.245/0.367 | 0.243/0.364 | 0.260/0.389 |
| Unfixed-Bias-Conv2 (a=3) | Jacobi | **0.214/0.402** | **0.199/0.373** | **0.202/0.379** | **0.208/0.391** |
| Unfixed-Bias-Conv3 (a=3) | Jacobi | 0.285/0.570 | 0.239/0.479 | 0.244/0.487 | 0.254/0.508 |

Table 3: Results in solving Heat Conduction Equation.

where $L$ represents the number of convolution layers stacked in $H$.

On CPU, the Jacobi iteration requires 4 multiply-add operations, while a $3 \times 3$ convolutional kernel requires 9 operations, so we measure the computation cost by the number of multiply-add operations. Since our iterator has 2 additional multiply-add operation when calculating $z_{i+1}$, the evaluation criteria *ops* is calculated as follows:

$$ops = \frac{M(4+2+9L)}{4N}. \tag{19}$$

For Unfixed-Bias-U-Net models, we compare them against Multigrid method with the same number of sub-sampling and smoothing layers. Therefore, our models have the same number of convolutional layers, and roughly $9/4$ times the number of operations compared to Multigrid.

### 4.2 Numerical Experiments

We have validated the effectiveness of our iterator in solving different PDEs.

#### 4.2.1 Possion Equation

The Possion Equation is written as:

$$\nabla^2 u = f \tag{20}$$

where $\nabla^2$ is the Laplace operator. Table 1 shows results of our iterator in solving Poisson Equation. For Unfixed-Bias-ConvX models, they converge to the correct solution, and require less computation than Jacobi in all settings. Our Unfixed-Bias-Conv2 is the best model, which achieves $5.7 \sim 7.0\times$ faster than Jacobi in terms of layers, and $2.9 \sim 3.5\times$ faster in terms of multiply-add operations. In addition, it is $26\% \sim 53\%$ faster than Conv3 Hsieh et al. (2019).

For Unfixed-Bias-U-NetX models, they also converge to the correct solution, and require less computation than Multigrid in all settings. Our Unfixed-Bias-U-Net3 is $66.7 \sim 71.4\times$ faster than Multigrid in terms of layers, and $30.3 \sim 32.3\times$ faster in terms of multiply-add operations. In addition, it is $464\% \sim 507\%$ faster than U-Net2 Hsieh et al. (2019).

According to Theorem 3.2, if our iterator converges for a geometry, then it is guaranteed to converge to the correct solution for any $f$ and boundary values $b$. The experiment results show that our model not only converges but also converges faster than the standard solver in a variety of grid sizes $n$ and geometries $G$. Empirically, this also shows that our method has strong generalization.

#### 4.2.2 Helmholtz Equation

The Helmholtz Equation is written as:

$$\nabla^2 u + a^2 u = f \tag{21}$$

where $a$ is a constant. Table 2 shows results of our iterator in solving Helmholtz Equation. By changing the value of $a$, we verify three different Helmholtz Equation. In all cases, our Unfixed-Bias-ConvX models can converge and are better than Jacobi method. Unfixed-Bias-Conv2 achieves

$5.5 \sim 7.2\times$ faster than Jacobi in terms of layers, and $2.8 \sim 3.6\times$ faster in terms of multiply-add operations. In addition, our Unfixed-Bias-Conv1 about $\sim 2.5$ times faster than Conv1 Hsieh et al. (2019). Helmholtz Equation is much more difficult to solve than Poisson Equation, but our models still obtain the performance of far more than manual design methods in this challenging equation, which greatly reflects the superiority of our iterators.

### 4.2.3 Heat Conduction Equation

The Heat Conduction Equation is written as:

$$\frac{\partial u}{\partial t} - a\nabla^2 u = f. \tag{22}$$

Table 3 shows results of our iterator in solving Heat Conduction Equation. The Heat Conduction Equation combines the first-order and second-order partial derivatives, which further improves the difficulty of solving. Our Unfixed-Bias-ConvX models can still converge, and both obtain faster convergence speed than the manually designed iterator. Our Unfixed-Bias-Conv2 is still the best model, which achieves $4.6 \sim 5.6\times$ faster than Jacobi in terms of layers, and $2.3 \sim 2.8\times$ faster in terms of multiply-add operations.

In essence, using Poisson Equation, Helmholtz Equation and Heat Conduction Equation to verify the effectiveness of our method is changing $A$. Although we need to retrain a model for different $A$, it also shows that our method can be easily extended to different PDEs, and can obtain an ideal performance. In the future, we will consider training a model that can be generalized to different $A$.

**See Appendix B for more numerical experiments.**

## 5 Conclusion

We build a learned iterator on top of an existing standard iterative solver, which can modify the current iteration result with the help of the historical iteration results, so as to accelerate the convergence speed. At the theoretical level, due to the introduction of the historical iterative results, our iterator is a new iterative format: Unfixed Bias Iterator: the unfixed bias iterator, and we provide sufficient theoretical guarantees for it, after which we prove theoretically that our iterator can converge and has some generalization. Experimental results show that even if our solver is trained only ion simple domains, it can generalize to different grid sizes, geometries and boundary conditions. Moreover, the powerful generalizability of our method is illustrated by solving for different PDEs. Last but not least, our iterator greatly exceeds the existing iterators in convergence speed.

## References

Achref Bachouch, Côme Huré, Nicolas Langrené, and Huyen Pham. Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications. *arXiv e-prints*, art. arXiv:1812.05916, December 2018.

Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *arXiv e-prints*, art. arXiv:1907.03452, July 2019.

Ido Ben-Yair, Gil Ben Shalom, Moshe Eliasof, and Eran Treister. Quantized convolutional neural networks through the lens of partial differential equations. *arXiv e-prints*, art. arXiv:2109.00095, August 2021.

Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *CoRR*, abs/1711.06464, 2017.

Quentin Chan-Wai-Nam, Joseph Mikael, and Xavier Warin. Machine learning for semi linear pdes. *J. Sci. Comput.*, 79(3):1667–1712, 2019.

Alana de Santana Correia and Esther Luna Colombini. Attention, please! a survey of neural attention models in deep learning, 2021. cite arxiv:2103.16775Comment: 66 pages, 24 figures.

Tim Dockhorn. A discussion on solving partial differential equations using neural networks. *CoRR*, abs/1904.07200, 2019.

Amir Barati Farimani, Joseph Gomes, and Vijay S. Pande. Deep learning the physics of transport phenomena. *CoRR*, abs/1709.02432, 2017.

Stanley P Frankel. Convergence rates of iterative treatments of partial differential equations. *Mathematical Tables and Other Aids to Computation*, 4(30):65–75, 1950.

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2415–2423. PMLR, 2019.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey, 2015. cite arxiv:1503.04069Comment: 12 pages, 6 figures.

Carsten Gräser and Oliver Sander. Polyhedral gauß-seidel converges. *J. Num. Math.*, 22(3):221–254, 2014.

Wolfgang Hackbusch and Gabriel Wittum (eds.). *Multigrid Methods V : Proceedings of the Fifth European Multigrid Conference held in Stuttgart, Germany, October 1-4, 1996*, volume 3 of *Lecture Notes in Computational Science and Engineering*, Berlin, 1998. Springer. ISBN 354063133X 9783540631330. doi: 10.1007/978-3-642-58734-4.

Jiequn Han, Arnulf Jentzen, and Weinan E. Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. *CoRR*, abs/1707.02568, 2017.

Jihun Han and Yoonsang Lee. Hierarchical Learning to Solve Partial Differential Equations Using Physics-Informed Neural Networks. *arXiv e-prints*, art. arXiv:2112.01254, December 2021a.

Jihun Han and Yoonsang Lee. Hierarchical Learning to Solve Partial Differential Equations Using Physics-Informed Neural Networks. *arXiv e-prints*, art. arXiv:2112.01254, December 2021b.

Juncai He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *CoRR*, abs/1901.10415, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. cite arxiv:1512.03385Comment: Tech report.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. In *ICLR (Poster)*. OpenReview.net, 2019.

Xiang Huang, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Jing Zhou, Fan Yu, Bei Hua, Lei Chen, and Bin Dong. Solving Partial Differential Equations with Point Source Based on Physics-Informed Neural Networks. *arXiv e-prints*, art. arXiv:2111.01394, November 2021a.

Xiang Huang, Zhanhong Ye, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Jing Zhou, Fan Yu, Bei Hua, Lei Chen, and Bin Dong. Meta-Auto-Decoder for Solving Parametric Partial Differential Equations. *arXiv e-prints*, art. arXiv:2111.08823, November 2021b.

Côme Huré, Huyên Pham, and Xavier Warin. Some machine learning schemes for high-dimensional nonlinear pdes. *CoRR*, abs/1902.01599, 2019.

Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *arXiv e-prints*, art. arXiv:1901.10854, January 2019.

Ayush Jaiswal, Wael Abd-Almageed, Yue Wu, and Premkumar Natarajan. Bidirectional conditional generative adversarial networks. *CoRR*, abs/1711.07461, 2017.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving for high dimensional committor functions using artificial neural networks. *CoRR*, abs/1802.10275, 2018.

Isaac E. Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Networks*, 9(5):987–1000, 1998.

Isaac E. Lagaris, Aristidis Likas, and Dimitris G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans. Neural Networks Learn. Syst.*, 11 (5):1041–1049, 2000.

Panos Lambrianides, Qi Gong, and Daniele Venturi. A new scalable algorithm for computational optimal control under uncertainty. *J. Comput. Phys.*, 420:109710, 2020.

Jianyu Li, Siwei Luo, Yingjian Qi, and Yaping Huang. Numerical solution of elliptic partial differential equation using radial basis function neural networks. *Neural Networks*, 16(5-6): 729–734, 2003.

Yingzhou Li, Jianfeng Lu, and Anqi Mao. Variational training of neural network approximations of solution maps for physical models. *J. Comput. Phys.*, 409:109338, 2020.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *arXiv e-prints*, art. arXiv:2111.03794, November 2021.

Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. *CoRR*, abs/2003.05744, 2020.

Kjetil O. Lye, Siddhartha Mishra, and Deep Ray. Deep learning observables in computational fluid dynamics. *J. Comput. Phys.*, 410:109339, 2020.

Shehryar Malik, Usman Anwar, Ali Ahmed, and Alireza Aghasi. Learning to solve differential equations across initial conditions. *CoRR*, abs/2003.12159, 2020.

A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations*. Wiley, 1980. ISBN 0471276413.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. cite arxiv:1901.05103.

Erfan Pirmorad, Faraz Khoshbakhtian, Farnam Mansouri, and Amir-massoud Farahmand. Deep Reinforcement Learning for Online Control of Stochastic Partial Differential Equations. *arXiv e-prints*, art. arXiv:2110.11265, October 2021.

Shreya Khare — Yogeshchandra Puranik. A review on introduction to reinforcement learning. *International Journal of Trend in Scientific Research and Development*, 5(4):1096–1099, June 2021. ISSN 2456-6470.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016. cite arxiv:1612.00593.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. cite arxiv:1609.04747Comment: 12 pages, 6 figures.

F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009. ISSN 1045-9227, 1941-0093. doi: 10.1109/TNN.2008.2005605.

Rishi Sharma, Amir Barati Farimani, Joe Gomes, Peter Eastman, and Vijay S. Pande. Weakly-supervised deep learning of heat transport via physics informed loss. *CoRR*, abs/1807.11374, 2018.

Suprosanna Shit, Abinav Ravi Venkatakrishnan, Ivan Ezhov, Jana Lipková, Marie Piraud, and Bjoern H. Menze. Implicit neural solver for time-dependent linear pdes with convergence guarantee. *CoRR*, abs/1910.03452, 2019.

Ben Stevens and Tim Colonius. Finitenet: A fully convolutional lstm network architecture for time-dependent partial differential equations. *CoRR*, abs/2002.03014, 2020.

Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-Based Algebraic Multigrid Coarsening Using Reinforcement Learning. *arXiv e-prints*, art. arXiv:2106.01854, June 2021.

P. Rajeshwari — P. Abhishek — P. Srikanth — T. Vinod. Object detection an overview. *International Journal of Trend in Scientific Research and Development*, 3(3):1663–1665, March 2019. ISSN 2456-6470. doi: https://doi.org/10.31142/ijtsrd23422.

Yunlei Yang, Muzhou Hou, Hongli Sun, Tianle Zhang, Futian Weng, and Jianshu Luo. Neural network algorithm based on legendre improved extreme learning machine for solving elliptic partial differential equations. *Soft Comput.*, 24(2):1083–1096, 2020.

## A LEMMAS AND PROOFS

This chapter gives the lemmas we use and their proofs.

**Lemma A.1.** $\lim_{n\to\infty} a_n = 0$, *if* $\lim_{n\to\infty}\sum_{i=0}^{n} B^{n-i}a_i = 0$, *where* $B$ *is a matrix and* $a_i$ *is a vector.*

*Proof.* Since $\lim_{n\to\infty}\sum_{i=0}^{n} B^{n-i}a_i = 0$, we know that:

$$\begin{aligned}\lim_{n\to\infty}\sum_{i=0}^{n} B^{n-i}a_i &= \lim_{n\to\infty}(\sum_{i=0}^{n-1} B^{n-i}a_i + a_n) = 0\\ &\Rightarrow \lim_{n\to\infty}\sum_{i=0}^{n-1} B^{n-i}a_i = -\lim_{n\to\infty} a_n\end{aligned} \tag{23}$$

When $n$ is sufficiently large, we have that:

$$a_n = -\sum_{i=0}^{n-1} B^{n-i}a_i, \tag{24}$$

$$\begin{aligned}a_{n+1} &= -\sum_{i=0}^{n} B^{n+1-i}a_i\\ &= -\sum_{i=0}^{n-1} B^{n+1-i}a_i - Ba_n\\ &= -\sum_{i=0}^{n-1} B^{n+1-i}a_i + B\sum_{i=0}^{n-1} B^{n-i}a_i\\ &= -\sum_{i=0}^{n-1} B^{n+1-i}a_i + \sum_{i=0}^{n-1} B^{n+1-i}a_i = 0\\ &\Rightarrow \lim_{n\to\infty} a_n = 0.\end{aligned} \tag{25}$$

□

**Lemma A.2.** $\lim_{n\to\infty}\sum_{i=0}^{n} B^{n-i}a_i = 0$, *if the spectral radius* $\rho(B) < 1$ *and* $\lim_{n\to\infty} a_n = 0$, *where* $B$ *is a matrix and* $a_i$ *is a vector.*

*Proof.* Let $\epsilon$. Then since $\lim_{n\to\infty} a_n = 0$ there is an $N_1 > 0$ such that if $m > N_1$ we know that:

$$||a_m|| < \epsilon. \tag{26}$$

By dividing $\sum_{i=0}^{n} B^{n-i} a_i$ into two parts with $N_1$ as the boundary, we can obtain:

$$||\sum_{i=0}^{n} B^{n-i} a_i|| \leq ||\sum_{i=0}^{N_1} B^{n-i} a_i|| + ||\sum_{i=N_1}^{n} B^{n-i} a_i||. \tag{27}$$

According to Eq.26, we have that:

$$\begin{aligned} ||\sum_{i=N_1}^{n} B^{n-i} a_i|| &< \epsilon||\sum_{i=N_1}^{n} B^{n-i}|| = \epsilon||\sum_{i=0}^{n-N_1-1} B^{i}|| \\ &= \epsilon||(I-B)^{-1}(I-B^{n-N_1})|| \\ &\leq \epsilon||(I-B)^{-1}|| \end{aligned} \tag{28}$$

If $||a_*|| \geq ||a_i||$ for all $i \leq N_1$, then we have that:

$$||\sum_{i=0}^{N_1} B^{n-i} a_i|| \leq ||\sum_{i=0}^{N_1} B^{n-i} a_*|| < N_1||B^{n-N_1} a_*||. \tag{29}$$

Obviously, there is an $N_2 > N_1$ such that if $m > N_2$ we know that:

$$N_1||B^{m-N_1} a_*|| < \epsilon. \tag{30}$$

To sum up, if we take $N = N_2$, then for all $m > N$, we have that:

$$|\sum_{i=0}^{m} B^{m-i} a_i| < \epsilon(||(I-B)^{-1}|| + 1). \tag{31}$$

Therefore, $\lim_{n\to\infty} \sum_{i=0}^{n} B^{n-i} a_i = 0$. □

**Lemma A.3.** $\lim_{n\to\infty} \sum_{i=0}^{n} \beta^{n-i} \alpha_i = 0$, *if* $0 < \beta < 1$ *and* $\lim_{n\to\infty} \alpha_n = 0$.

*Proof.* Let $\epsilon > 0$. Then since $\lim_{n\to\infty} \alpha_n = 0$ there is an $N_1 > 0$ such that if $m > N_1$ we know that:

$$|\alpha_m| < \epsilon. \tag{32}$$

By dividing $\sum_{i=0}^{n} \beta^{n-i} \alpha_i$ into two parts with $N_1$ as the boundary, we can obtain:

$$|\sum_{i=0}^{n} \beta^{n-i} \alpha_i| \leq |\sum_{i=0}^{N_1} \beta^{n-i} \alpha_i| + |\sum_{i=N_1}^{n} \beta^{n-i} \alpha_i|. \tag{33}$$

According to Eq.32, we have that:

$$\begin{aligned} |\sum_{i=N_1}^{n} \beta^{n-i} \alpha_i| &< \epsilon|\sum_{i=N_1}^{n} \beta^{n-i}| = \epsilon|\sum_{i=0}^{n-N_1-1} \beta^{i}| \\ &= \epsilon\frac{1-\beta^{n-N_1-1}}{1-\beta} < \frac{\epsilon}{1-\beta}. \end{aligned} \tag{34}$$

If we take $\alpha_* = \max_{0\leq i\leq N_1} \alpha_i$, we have that:

$$|\sum_{i=0}^{N_1} \beta^{n-i} \alpha_i| \leq |\alpha_* \sum_{i=0}^{N_1} \beta^{n-i}| < |\alpha_* N_1 \beta^{n-N_1}|. \tag{35}$$

Obviously, there is an $N_2 > N_1$ such that if $m > N_2$ we know that:

$$|\alpha_* N_1 \beta^{m-N_1}| < \epsilon. \tag{36}$$

To sum up, if we take $N = N_2$, then for all $m > N$, we have that:

$$|\sum_{i=0}^{m} \beta^{m-i} \alpha_i| < \frac{\epsilon}{1-\beta} + \epsilon. \tag{37}$$

Therefore, $\lim_{n\to\infty} \sum_{i=0}^{n} \beta^{n-i} \alpha_i = 0$. □

**Lemma A.4.** $\lim_{n\to\infty}\sum_{i=0}^{n}\beta^{n-i}a_i = 0$, *if* $0<\beta<1$, $\lim_{n\to\infty} a_n = 0$, *where* $a_i$ *is a vector.*

*Proof.* Since $a_i$ is a vector in linear space $\mathbb{R}^k$, where:

$$a_i = (a_i^{(1)}, a_i^{(2)}, \cdots, a_i^{(k)}). \tag{38}$$

Then $\sum_{i=0}^{n}\beta^{n-i}a_i$ can be rewritten as:

$$(\sum_{i=0}^{n}\beta^{n-i}a_i^{(1)}, \sum_{i=0}^{n}\beta^{n-i}a_i^{(2)}, \cdots, \sum_{i=0}^{n}\beta^{n-i}a_i^{(k)}) \tag{39}$$

According to Lemma A.3, $\lim_{n\to\infty}\sum_{i=0}^{n}\beta^{n-i}a_i^{(j)} = 0$ holds for any $j$.

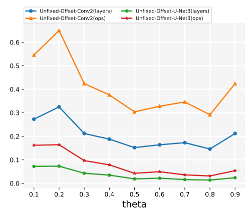
Therefore, $\lim_{n\to\infty}\sum_{i=0}^{n}\beta^{n-i}a_i = 0$. □

## B More Experiments

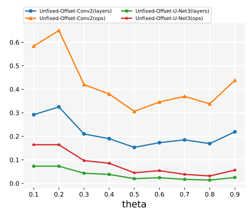
### B.1 The Sensitivity of Hyper-parameters

In this section, we discuss the influence of the hyper-parameter $\theta$ and the number of iterations $k$ of the model.

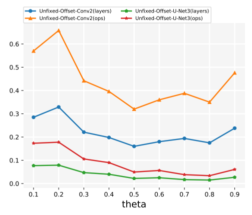
**Hyper-parameter $\theta$:** Taking models Unfixed-Bias-Conv2 and Unfixed-Bias-U-Net3 as examples, we test their sensitivity to $\theta$. The results are shown in Figure 1. When $\theta$ is between 0.5 and 0.8, there is no obvious fluctuation in the convergence speed of the models. When $\theta$ is less than 0.5, the convergence speed of the models is greatly reduced. Because $\theta$ is too small, the correction value is too dependent on the historical correction value. Although the correction value is relatively stable, the update is not real-time enough, resulting in the decline of the accuracy of the correction value. When $\theta$ is greater than 0.8, the convergence speed of the model also decreases significantly. The main reason is that when $\theta$ is too large, the correction value mainly refers to the current correction value. Although the update of the correction value is timely, the stability is greatly reduced, and finally the accuracy of the correction value is reduced. In our experiments, $\theta = 0.8$ is a suitable value for all models.



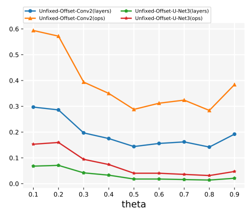
(a) Square.



(b) L-shape.



(c) Cylinders.



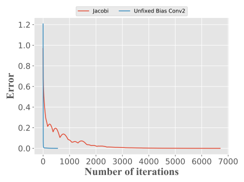
(d) $f \neq 0$.

Figure 1: Sensitivity of the models to $\theta$.

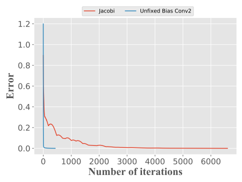
**Number of Iterations $l$:** When we change the value range of iteration number $k$, no obvious change in the convergence speed of the model is observed. Only in some extreme settings, for example, $l \in [1, 2]$. This setting requires that the model can converge with a minimum number of iterations, but it is obviously impossible, which leads to model training collapse. However, although the models are not sensitive to the value range of $l$, the range of smaller value can reduce the training time of the models. Therefore, $l \in [1, 20]$ is a recommended setting.

### B.2 Errors

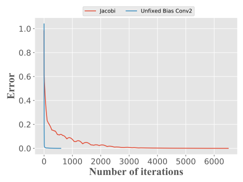
Figure 2 shows that the errors decrease with iteration under different iterators. Obviously, our iterator can reduce the high frequency error at a very fast speed. After reducing the errors to a certain extent, the speed of Jacobi iterator to reduce the low-frequency errors is much lower than our iterator. From the speed of error reduction, we can see why our method can achieve much faster convergence than Jacobi iterator, which further explains the superiority of our iterator.



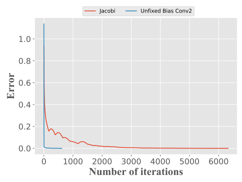
(a) Square.



(b) L-shape.

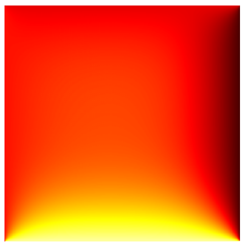


(c) Cylinders.

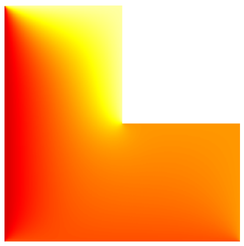


(d) $f \neq 0$.

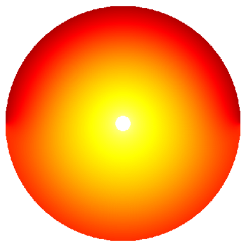
Figure 2: Errors under different iteration times.



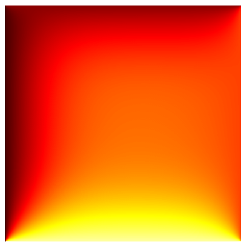
(a) Square.



(b) L-shape.



(c) Cylinders.



(d) $f \neq 0$.

Figure 3: The ground truth solutions of examples in different settings.

### B.3 VISUALIZATION

We design some challenging geometries to test the generalization of our models: (i) same geometry but larger grid, (ii) L-shape geometry, (iii) Cylinders geometry, and (iv) PDEs in the same geometry, but $f \neq 0$. L-shape and cylinders geometries are designed because the models are trained on square domains and have never seen sharp or curved boundaries. Examples of the four settings are shown in Figure 3.