SOLVING CONVOLUTION-TYPE INTEGRAL EQUATIONS WITH HIGH-FREQUENCY USING PRECONDITIONED NEURAL OPERATORS*

RAYMOND HON-FU CHAN† AND LINGFENG LI‡

Abstract. The convolution-type integral equations arise from various fields, *e.g.*, finite impulse response filters in signal processing and deblurring problems in image processing. When solving these equations, conventional numerical methods, like the multigrid method, can only efficiently solve the low-frequency components in the error, but not the high-frequency components. In this paper, we apply neural operators to address this issue. We propose a novel training strategy that trains neural operators to solve the high-frequency components efficiently. Then, we combine the neural operators with some classical iterative solvers, like the weighted Jacobi method, to obtain an efficient hybrid iterative algorithm for the integral equations. We also analyze the generalization error of our training strategy and the convergence of the hybrid iterative algorithm. We test our algorithms on large-scale and ill-conditioned linear systems discretized from one- and two-dimensional convolution-type integral equations. Our proposed algorithm significantly outperforms the multigrid method and the preconditioned conjugate gradient method in both iteration numbers and computational time. Besides, the iteration number of our algorithm is more robust to the changes in the size and condition numbers of the underlying linear systems than conventional numerical methods. Though we mainly focus on the integral equations, our method can be directly applied to all kinds of linear problems.

Key words. Integral equation; Neural operator; Multigrid method; hybrid iterative solver

AMS subject classifications. 68T07, 45E10

1. Introduction.

Solving linear systems is one of the most fundamental tasks in scientific computing. In general, there are two types of algorithms for solving linear systems: direct methods and iterative methods. Direct methods can find the exact solutions to linear systems using some factorization techniques, *e.g.*, the LU decomposition method. Although direct methods can find the exact solutions in a finite number of operations, their computational costs increase very fast with the problem size, which makes it impractical to solve large-scale linear systems. Iterative methods, in contrast, solve linear systems by iteratively refining their solutions. They are particularly useful for large-scale problems because of their low computational costs compared to direct methods. The solution of typical iterative methods, such as the Jacobi method, the Gauss-Seidel method, and the Krylov subspace methods, would approach the exact solution as the iteration number increases. The convergence rate usually depends on the condition number of the coefficient matrix. For very ill-conditioned matrices, preconditioning techniques can help accelerate the convergence. One important class of linear systems is the Toeplitz systems, which arise from many applications, especially in the solution of convolution-type integral equations, in signal processing, and in image processing [16, 19, 25]. Toeplitz matrices are matrices with constant diagonals. By exploring this special structure, many efficient preconditioners

*Submitted to the editors DATE.

Funding: This work of R. Chan is partially supported by HKRGC Grants No. CityU11309922, LU13300125, ITF Grant No. MHP/054/22, and LU BGR 105824. The work of Lingfeng Li is supported by the InnoHK project at the Hong Kong Centre for Cerebro-cardiovascular Health Engineering (COCHE) and HKRGC Grants No.LU13300125.

†Department of Data Science and Department of Operations and Risk Management, Lingnan University, Hong Kong; Hong Kong Centre for Cerebro-Cardiovascular Health Engineering, Hong Kong (raymond.chanln.edu.hk).

‡Hetao Institute of Mathematics and Interdisciplinary Sciences (Shenzhen), Shenzhen, China (lilingfeng@himis-sz.cn).

have been proposed [13, 14, 44].

In the following, we use the terminology in [47] to call the eigenvectors corresponding to large (respectively, small) eigenvalues of a matrix the *algebraic* high-frequency (respectively, low-frequency) eigenvectors. If the eigenvector is itself highly-oscillating (respectively, slowly-oscillating), we call the vector a *geometric* high-frequency (respectively, low-frequency) eigenvector. Iterative methods, such as the Jacobi and Gauss-Seidel methods, are called *smoothers* in that they tend to reduce the algebraic high-frequency components (eigenvectors corresponding to large eigenvalues) in the error much faster than the algebraic low-frequency components (eigenvectors corresponding to small eigenvalues). [47].

For PDE problems, the algebraic high-frequency and low-frequency components of the discretized systems coincide with the geometric high-frequency and low-frequency components, respectively, *i.e.*, they are highly oscillatory and slowly oscillatory, respectively. In such cases, we can apply the multigrid method to solve the linear systems efficiently [22]. The idea of multigrid methods is to reduce the geometric high-frequency error using smoothers such as the Jacobi method, and remove the geometric low-frequency error using coarse-grid corrections. The multigrid method is often considered the optimal method for PDE problems, particularly in terms of computational efficiency [5].

Some recent studies borrow the idea from multigrid methods and replace the coarse-grid correction step with neural operator solvers [23, 50, 51]. For example, the DeepONet-based neural operators [33], which are designed for operator learning in continuous spaces, can reduce the geometric low-frequency error [50, 51]. This type of structure is known to be frequency-biased [48], *i.e.*, it tends to learn the geometric low-frequency components in the solution rather than the high-frequency components during training. Therefore, it can be combined with smoothers to solve PDE problems.

There also exist other types of neural operators that learn operators in discrete spaces, like convolutional neural networks (CNN) and Fourier neural operators (FNO) [29]. In [24, 42], the authors use CNNs as multigrid smoothers and apply them with coarse-grid corrections to solve PDE problems. Typically, neural operators require a long time of training before they can be used in applications, so these types of methods are useful when a linear system needs to be solved multiple times for different right-hand-side vectors, *e.g.*, when simulating the evolution of fluid dynamics or heat transfer [3]. Besides, there are also some neural operators that integrate the multigrid framework into the structure directly for solving linear systems [15, 20].

This paper concerns problems where the algebraic high-frequency (respectively, low-frequency) components do not correspond to geometric high-frequency (respectively, low-frequency) components. This occurs in some convolution-type integral equations with smooth kernels [7], where the corresponding linear systems are typically ill-conditioned Toeplitz or block-Toeplitz. For these systems, the multigrid method may fail since both smoothers and coarse-grid corrections tend to reduce the same algebraic high-frequency (geometric low-frequency) components in the error. Finding a good smoother to solve the algebraic low-frequency (geometric high-frequency) components is a challenging task for this type of problem [7].

In this work, we aim to solve the algebraic low-frequency components using neural operators and then combine the operators with smoothers to form a new hybrid iterative algorithm for the integral equations. The main contributions and findings are summarized as follows:

1. We developed a new algorithm that solves algebraic low-frequency compo-

nents using neural operators. The main difference between our method and previous work [23, 50, 51, 24, 42] is that our method is designed to address algebraic frequency components, whereas most previous work focuses on specific geometric frequency components.

2. Inspired by the framework in [50, 51], we apply a hybrid iterative algorithm combining neural operators and classical smoothers to solve convolution-type integral equations.
3. We conduct some theoretical analysis of the proposed algorithm. We first estimate the generalization error when the neural operator is a two-layer ReLU network and derive an upper bound in terms of the number of training samples, problem size, and the optimization error. We then prove the convergence of the hybrid iterative algorithm when the neural operator satisfies some reasonable conditions. Lastly, we analyze how the choice of training loss function would affect the convergence rate of different algebraic frequency components in the training error.
4. We conduct numerical experiments to solve large-scale and ill-conditioned integral equations using the proposed method. Our method outperforms multigrid methods and preconditioned conjugate gradient (PCG) methods significantly, both in terms of the iteration number and evaluation time. Besides, our experiments show that the convergence rate of our method is robust against the change in problem size and condition number.

2. Preliminaries.

2.1. Convolution-type integral equations and Toeplitz matrices.

In image processing and data analysis, we often need to solve integral equations of the form:

$$\alpha R(u)(z) + \int_{\Omega} \mathcal{K}(z - z')u(z')dz' = f(z), \quad z \in \Omega. \tag{2.1}$$

where u is a continuous image or signal, $\alpha > 0$ is the regularization weight parameter, $\mathcal{K}$ is a smooth convolution kernel, Ω is a fixed domain, and $R(u)$ is the regularization term. When $\mathcal{K}$ is a smooth function, *e.g.*, a Gaussian smoothing kernel, the coefficient matrix of $\mathcal{K}$ would be very ill-conditioned, so a regularization term R is required [2]. Equation (2.1) is called the integral equation of the second kind when $R(u) = u$, and is called the integral equation of the first kind when $\alpha = 0$ [36].

Discretizing (2.1) leads to a linear system $A_n\mathbf{x} = \mathbf{y}$ where $A_n \in \mathbb{R}^{n\times n}$ is the coefficient matrix, which usually has some special structures like Toeplitz or Block-Toeplitz-with-Toeplitz-Block (BTTB). In this work, we consider the case when A_n is symmetric positive definite (SPD). Typical choices for the regularization $R(u)$ include the Tikhonov regularization u, isotropic diffusion Δu, and the total variation $-\nabla \cdot (\nabla u/|\nabla u|)$. When the weight parameter α is small, A_n is typically ill-conditioned.

The matrix A_n is called Toeplitz if each diagonal of its coefficient matrices is constant. We denote the value of elements at the main diagonal, k-th supdiagonal, and k-th subdiagonal as a_0, a_k, and a_{-k} respectively. Toeplitz systems are an important class of linear systems, and their solutions can be found efficiently by leveraging the special structure. Many fast solvers, including direct methods [1, 4] and iterative methods [12, 13, 44], have been proposed for solving Toeplitz systems.

A 2π-periodic Lebesgue integrable function $f : \mathbb{R} \to \mathbb{R}$ is called the generating function of A_n if its Fourier coefficients are the entries of A_n: $a_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(\theta)e^{-ik\theta}d\theta$, where $k = 0, \pm1, \pm2, \ldots$. Let $\lambda_1(A_n) \geq \lambda_2(A_n) \geq \cdots \geq \lambda_n(A_n)$ be the eigenvalues

of A_n sorted in descending order. It is proven that the eigenvalues of A_n can be bounded by the range of its generating function [19]:

LEMMA 2.1. *Let f be the generating function of A_n. Then,*

$$\min_{\theta\in[0,2\pi]} f(\theta) < \lambda_n(A_n) \le \lambda_1(A_n) < \max_{\theta\in[0,2\pi]} f(\theta). \tag{2.2}$$

2.2. Algebraic high/low frequency. The terms algebraic high/low frequency are commonly used in the analysis of algebraic multigrid methods [47]. When A_n is symmetric positive definite (SPD), all the eigenvalues are positive, and there exists a set of orthonormal eigenvectors $\{\mathbf{v}_i\}_{i=1}^n$ where $\mathbf{v}_i$ corresponds to $\lambda_i(A_n)$. The $\mathbf{v}_i$ corresponding to the large (resp. small) eigenvalues of A_n are the algebraic high-frequency (resp. low-frequency) modes associated with A_n. Meanwhile, vectors can also be characterized as geometric high-frequency or geometric low-frequency based on the variation of their entries. Geometric high-frequency vectors refer to those vectors with rapidly oscillating entries, while geometric low-frequency vectors refer to those with smooth and slow oscillatory entries. Mathematically, for any given ζ_1, ζ_2 such that $\zeta_2 \ge \zeta_1 > 0$, a vector $\mathbf{v}$ is called ζ_1-algebraic low-frequency if $\|\mathbf{v}\|^2_{A_n D_n^{-1} A_n}/\|\mathbf{v}\|^2_{A_n} \le \zeta_1$, and ζ_2-algebraic high-frequency if $\|\mathbf{v}\|^2_{A_n D_n^{-1} A_n}/\|\mathbf{v}\|^2_{A_n} \ge \zeta_2$, where D_n is the diagonal component of A_n [40, 47].

For any Toeplitz matrix A_n, we have $\|\mathbf{v}\|^2_{A_n D_n^{-1} A_n} = \|\mathbf{v}\|^2_{A_n^2}/a_0$. We can show that

$$\frac{1}{a_0}\|\mathbf{v}\|^2_{A_n^2} = \frac{1}{a_0}\|A_n^{1/2}(A_n^{1/2}\mathbf{v})\|_2^2 \in \left[\frac{\lambda_n(A_n)}{a_0}\|\mathbf{v}\|^2_{A_n}, \frac{\lambda_1(A_n)}{a_0}\|\mathbf{v}\|^2_{A_n}\right].$$

Thus, we need to choose ζ_1 and ζ_2 such that $\frac{\lambda_n(A_n)}{a_0} \le \zeta_1 \le \zeta_2 \le \frac{\lambda_1(A_n)}{a_0}$. For simplicity, in the following, we omit the explicit dependence of ζ_1 (resp. ζ_2) in ζ_1-algebraic low frequency (resp. ζ_2-algebraic high frequency) and simply refer to them as algebraic low frequency (resp. algebraic high frequency).

For some convolution-type integral equations, the algebraic high-frequency (resp. low-frequency) components of the coefficient matrices coincide with the geometric low-frequency (resp. high-frequency) components. In contrast, coefficient matrices of most PDE problems have the opposite spectral property. In Figure 1 (top row), we visualize the eigenvectors corresponding to the large and small eigenvalues of the coefficient matrix of an integral equation (2.1) with $\mathcal{K}$ being a Gaussian smoothing kernel. We observe that the algebraic high-frequency (resp. low-frequency) components correspond to the geometric low-frequency (resp. high-frequency) components. For comparison, we also visualize the eigenvectors of the coefficient matrix of the Poisson equation in Figure 1 (bottom row), where the algebraic high-frequency (resp. low-frequency) components correspond to the geometric high-frequency (resp. low-frequency) components.

2.3. Two-grid method. The algebraic multigrid method is a very efficient solver for solving linear systems. It consists of the smoothing steps and coarse-grid correcting steps. Here, we consider a simple two-grid method. The smoothing step is usually performed by some smoothers. We first define an iterative scheme for solving $A_n\mathbf{x} = \mathbf{y}$

$$\mathbf{x}^{(\kappa+1)} = (I_n - M_n^{-1}A_n)\mathbf{x}^{(\kappa)} + M_n^{-1}\mathbf{y}, \quad \kappa = 1, 2, \ldots, \tag{2.3}$$

where $\mathbf{x}^{(0)}$ is the initial vector, I_n is the n-by-n identity matrix, and M_n is an n-by-n matrix whose inverse can be evaluated efficiently. We further define $\mathcal{S}(A_n, \mathbf{y}, \mathbf{x}^{(0)}, \kappa)$

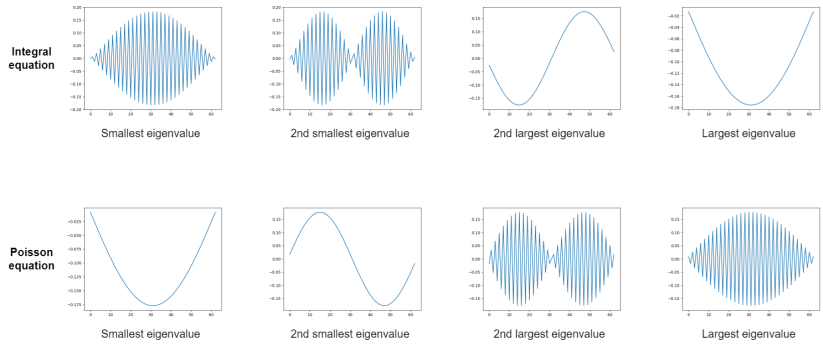


Fig. 1: Eigenvectors corresponding to different eigenvalues of the linear system discretized from the integreal equation (top row) and the Poisson equation (bottom row).

as the procedure of solving $A_n\mathbf{x} = \mathbf{y}$ using κ iterations of (2.3) started from $\mathbf{x}^{(0)}$. Denote the iteration matrix as $S_n := I_n - M_n^{-1}A_n$, then the error of $\mathbf{x}^{(\kappa)}$ can be written as

$$\mathbf{x}^{(\kappa)} - \mathbf{x} = S_n(\mathbf{x}^{(\kappa-1)} - \mathbf{x}) = \cdots = S_n^{\kappa}(\mathbf{x}^{(0)} - \mathbf{x}). \tag{2.4}$$

Many classical iterative methods, like the damped Jacobi iteration, satisfy the smoothing condition [47, 40]: for some $\sigma_1 > 0$ and any $\mathbf{e} \in \mathbb{R}^n$

$$\|S_n\mathbf{e}\|_{A_n}^2 \leq \|\mathbf{e}\|_{A_n}^2 - \sigma_1\|\mathbf{e}\|_{A_n^2}^2/a_0. \tag{2.5}$$

From (2.4) and (2.5), if the error $\mathbf{x}^{(\kappa-1)} - \mathbf{x}$ contains algebraic high-frequency modes, *i.e.*, $\|\mathbf{x}^{(\kappa-1)} - \mathbf{x}\|_{A_n}^2$ and $\|\mathbf{x}^{(\kappa-1)} - \mathbf{x}\|_{A_n^2}^2/a_0$ are comparable in magnitude, after applying the smoothing operator S_n, $\|\mathbf{x}^{(\kappa)} - \mathbf{x}\|_{A_n}^2$ would be efficiently reduced.

The correcting step for $A_n\mathbf{x} = \mathbf{y}$ is typically performed by a coarse grid correction:

$$\hat{\mathbf{x}} = P_H^h(A_H)^{-1}R_h^H\mathbf{y},$$

and the iteration matrix is $T_n := I_n - P_H^h(A_H)^{-1}R_h^H A_n$. Here, P_H^h denotes the prolongation matrix that interpolates a vector on a coarse grid to a fine grid, R_h^H denotes the restriction matrix that restricts a vector on a fine grid to a coarse grid, and $A_H := R_h^H A_n P_H^h$ is the coarse grid coefficient matrix.

The two-grid method solves the linear system $A_n\mathbf{x} = \mathbf{y}$ by alternatively applying the smoothing step and the correcting step (see Algorithm 2.1). By restricting a fine grid vector to a coarse grid, the geometric low-frequency modes are preserved. Therefore, solving the coarse grid problem can eliminate the geometric low-frequency modes efficiently. For most PDE problems, the geometric low-frequency modes are also algebraic low-frequency modes, so the coarse-grid correction step and the smoothing step are complementary to each other. The overall convergence of the two-grid iteration is guaranteed if for some $\sigma_2 > 0$ and any $\mathbf{e} \in \mathbb{R}^n$, T_n satisfies [43]:

$$\|T_n\mathbf{e}\|_{A_n}^2 \leq \frac{\sigma_2}{a_0}\|T_n\mathbf{e}\|_{A_n^2}^2 \text{ and } \|T_n\mathbf{e}\|_{A_n} \leq \|\mathbf{e}\|_{A_n}. \tag{2.6}$$

When σ_2 is sufficiently small, the first inequality implies that the error vectors after the correction step, *i.e.*, $T_n\mathbf{e}$, will not be algebraic low-frequency vectors (by the definition of algebraic low-frequency in Section 2.2). Therefore, it can be further solved efficiently by the smoothing step. Then, by combining (2.5) and (2.6), the two-grid method would converge in a factor of $1-\sigma_1/\sigma_2$:

$$\|S_nT_n\mathbf{e}\|_{A_n}^2 \le \|T_n\mathbf{e}\|_{A_n}^2 - \sigma_1\|T_n\mathbf{e}\|_{A_n^2}^2/a_0 \le \|T_n\mathbf{e}\|_{A_n}^2 - \frac{\sigma_1}{\sigma_2}\|T_n\mathbf{e}\|_{A_n}^2 \le (1-\frac{\sigma_1}{\sigma_2})\|\mathbf{e}\|_{A_n}^2.$$

Algorithm 2.1 Two-grid method

Require: right-hand side vector $\mathbf{y}$ and the tolerance ϵ
  $\tau = 0$, and $\mathbf{x}_0 = 0$
  while $\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2/\|\mathbf{y}\|_2 \ge \epsilon$ **do**
    $\hat{\mathbf{x}}_\tau = T_n\mathbf{x}_\tau + P_H^h(A_H)^{-1}R_h^H\mathbf{y}$ ▷ Correcting step
    $\mathbf{x}_{\tau+1} = \mathcal{S}(A_n, \mathbf{y}, \hat{\mathbf{x}}_\tau, \kappa)$ ▷ Smoothing step
    $\tau = \tau + 1$
  end while
  Return $\mathbf{x}_\tau$

When the algebraic low-frequency modes are not geometrically low-frequency, which is common for integral equations, the two-grid method may fail because the smoothing steps and the correcting steps both reduce the same high-frequency parts of the errors. Detailed experiments and analysis of the difficulties in solving integral equations using multigrid methods can be found in [7].

2.4. Neural operators. Neural operators are neural networks that can approximate mappings between function spaces. Once trained, a neural operator can approximate the solution of the learned mappings directly for various input conditions, which makes them particularly well-suited for solving linear or nonlinear systems [33, 29, 45, 24]. In this work, like the FNO framework [29], we consider the discretized setting. The input and output functions are discretized on mesh grids and are represented by finite-dimensional vectors. Thus, the network actually learns a function between finite-dimensional vector spaces, which can also be viewed as a finite-dimensional approximation of the underlying solution operator. To be consistent with the literature, we still refer to our approach as operator learning.

For solving a linear system $A_n\mathbf{x} = \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are in $\mathbb{R}^n$, we need to construct a neural operator $\mathcal{N}_\theta : \mathbb{R}^n \to \mathbb{R}^n$ that can approximate $\mathbf{x} = A_n^{-1}\mathbf{y}$ by $\mathcal{N}_\theta(\mathbf{y})$, where θ denotes the set of all learnable parameters in the network. Here, $\mathcal{N}_\theta$ can be various types of networks, like the fully connected networks, ResNet [21], U-Net [39], and the FNO [29]. Although A_n^{-1} is linear, we still use a nonlinear network, rather than a general linear model, to approximate it. The reason is that approximating A_n^{-1} by a dense $n \times n$ matrix would require n^2 learnable parameters, which is infeasible for large-scale systems. A nonlinear neural network can use much fewer parameters and allowing the number of parameters to grow more slowly as n increases, especially for large-scale problem. An example is provided in Section 5.3.

Typically, the training of neural operators can be either data-driven or physics-informed. The data-driven method [29] directly minimizes the difference between the network prediction $\mathcal{N}_\theta(\mathbf{y})$ and the true solution $\mathbf{x} = A_n^{-1}\mathbf{y}$: $\min_\theta \frac{1}{N}\sum_{i=1}^N \|\mathcal{N}_\theta(\mathbf{y}_i) - \mathbf{x}_i\|_2^2$. Here $\{\mathbf{x}_i, \mathbf{y}_i = A_n\mathbf{x}_i\}_{i=1}^N$ is a randomly generated set of training samples. The physics-informed method [18, 30, 45] minimizes the residual of the problem:

$\min_\theta \frac{1}{N}\sum_{i=1}^N \|A_n\mathcal{N}_\theta(\mathbf{y}_i)-\mathbf{y}_i\|_2^2$. It is particularly useful when the true solution $\mathbf{x}_i = A_n^{-1}\mathbf{y}_i$ is not available.

3. A hybrid iterative method for integral equations. As we mentioned in the last paragraph of Section 2.3, integral equations of convolution type are difficult to solve using multigrid methods. The main challenge is to find a suitable smoother to efficiently reduce algebraic low-frequency components, *i.e.*, geometric high-frequency components. Motivated by the recent attempts to integrate neural operators into multigrid frameworks [24, 50, 51], we propose a novel framework combining classical smoothers and neural operators to solve convolution-type integral equations. The main idea is to design a new loss function for training neural operators so that they can solve the desired frequency components as expected.

Here, we first clarify the relation between our setting and the well-known frequency principle. The frequency principle discussed in the literature, for example in [48], mainly concerns meshless network structures whose inputs are coordinate variables in a domain. Specifically, one approximates a target function $g(x)$, where x belongs to a given domain Ω, by a neural network whose input is x and whose output is the predicted function value. During training, the frequency principle suggests that such a network tends to learn the geometric low-frequency components of $g(x)$ faster than its geometric high-frequency components.

However, the setting in our manuscript is different from that in [48]. First, we use a mesh-based network structure, whose input and output are functions represented on a fixed mesh grid. Second, the target of the network is not a single function but the inverse operator A_n^{-1}. If one assumes that our network also follows the frequency principle, this would mean that the network can easily learn the "geometric low-frequency components" of A_n^{-1}, which are difficult to define explicitly. The "frequency" of A_n^{-1} is completely different from the frequency of the vector $\mathbf{x}$, which is what we care about. Therefore, the frequency principle does not directly apply to our setting, and we do not need to use special architectures to address this issue.

3.1. Training of neural operators. Here, we consider a general neural network $\mathcal{N}_\theta : \mathbb{R}^n \to \mathbb{R}^n$ for approximating the solution operator A_n^{-1} of the linear systems $A_n\mathbf{x} = \mathbf{y}$. A general expected loss function for training $\mathcal{N}$ is:

$$\mathcal{L}_{\text{exp}}(\theta) := \mathbb{E}_{\mathbf{y}\sim\mu(\mathbb{B}(\mathbb{R}^n))}(\|P_n(\mathcal{N}_\theta(\mathbf{y})-\mathbf{x})\|_2^2). \tag{3.1}$$

Here, $\mathbb{B}(\mathbb{R}^n)$ is the unit sphere $\{\mathbf{y}\in\mathbb{R}^n | \|\mathbf{y}\|_2 = 1\}$, μ is a probability distribution defined on $\mathbb{B}(\mathbb{R}^n)$, and P_n is an $n\times n$ real matrix. Such a preconditioned loss function has been applied for physics-informed neural networks [32] recently. To approximate $A_n^{-1}\mathbf{y}$ for $\|\mathbf{y}\|_2 \neq 1$ using $\mathcal{N}_\theta$, we can first normalize $\mathbf{y}$ and then scale $\mathcal{N}_\theta(\mathbf{y})$ by the same factor, *i.e.*, $\mathcal{N}_\theta(\mathbf{y}/\|\mathbf{y}\|_2)\|\mathbf{y}\|_2$. We suppose the matrix P_n has a set of orthonormal eigenvectors $\{\mathbf{u}_i\}_{i=1}^n$ and the corresponding eigenvalues $\{\lambda_i(P_n)\}_{i=1}^n$. Then, the loss $\mathcal{L}_{\text{exp}}$ can be written as

$$\mathbb{E}_{\mathbf{y}}\left\|P_n\left(\sum_{i=1}^n(\mathbf{e}^\top\mathbf{u}_i)\mathbf{u}_i\right)\right\|_2^2 = \mathbb{E}_{\mathbf{y}}\left\|\sum_{i=1}^n\lambda_i(P_n)(\mathbf{e}^\top\mathbf{u}_i)\mathbf{u}_i\right\|_2^2 = \mathbb{E}_{\mathbf{y}}\sum_{i=1}^n\lambda_i(P_n)^2(\mathbf{e}^\top\mathbf{u}_i)^2 \tag{3.2}$$

where $\mathbf{e} := \mathcal{N}(\mathbf{y})-\mathbf{x}$ is the error vector and $\mathbf{e}^\top\mathbf{u}_i$ represents the projection of $\mathbf{e}$ on the unit eigenvector $\mathbf{u}_i$. From (3.2), we see that the loss function (3.1) is precisely equivalent to a weighted summation of $(\mathbf{e}^\top\mathbf{u}_i)^2$, $i = 1,\ldots,n$, and the weight for each term is the squared eigenvalues $\lambda_i^2(P_n)$. Therefore, the neural operator damps more

on the algebraic high-frequency components associated with P_n in the error $\mathbf{e}$ during the training process.

To train the neural operator to effectively solve algebraic low-frequency components associated with A_n, we therefore need to choose P_n such that it has the opposite spectral properties compared with those of A_n, *i.e.*, algebraic high-frequency components of P_n are algebraically low-frequency components with respect to A_n. An ideal choice would be $P_n = A_n^{-1}$, since A_n^{-1} has the same eigenvectors as A_n and its eigenvalues are the reciprocals of A_n's. However, calculating the inverse of A_n directly is not practical as it is our original problem. Here, we propose two practical choices for P_n:

1. Let $P_n = C(A_n)^{-1}$ where $C(A_n)$ denotes a circulant preconditioner of A_n [13]. For Toeplitz matrices, $C(A_n)^{-1}$ is a good approximation to A_n^{-1}, and it can be evaluated efficiently by the fast Fourier transform [11]. It has been proven that the eigenvalues of $C(A_n)^{-1}A_n$ are clustered around 1 [8].
2. Let $P_n = aI_n - A_n$ for some $a > 0$. For the Toeplitz matrix A_n with generating function $f(\theta)$, we can choose a such that $a \geq \max_{[0,2\pi]} f(\theta)$, see (2.2). In this case, an eigenvector of A_n corresponding to the eigenvalue λ_i is also an eigenvector of P_n corresponding to the eigenvalue $a - \lambda_i > 0$.

Using either way to construct P_n, we will be able to train a neural operator that can solve the desired frequency components. We note that when A_n is SPD, P_n is also SPD.

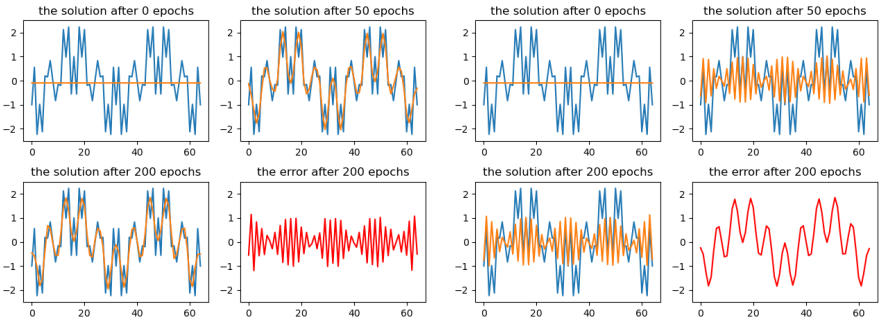
Remark 3.1. *Here, the P_n does not have to be a good preconditioner in the sense of numerical linear algebra, i.e., we do not require $P_n^{-1}A_n \approx I_n$. Instead, we only require P_n has a reverse spectrum with A_n, making our method applicable even to problems without good preconditioners.*

In practice, we can choose between the two constructions of P_n according to the information available for A_n. If a reasonable upper bound $a \geq \lambda_{\max}(A_n)$ is available, we recommend using $P_n = aI_n - A_n$. Since P_n and A_n have the same eigenvectors and the eigenvalue ordering is exactly reversed, the preconditioned loss directly emphasizes the algebraic low-frequency components of A_n. The value of a should not be chosen much larger than $\lambda_{\max}(A_n)$; otherwise $P_n \approx aI_n$ and the loss (3.2) becomes close to a simple data-driven loss.

The alternative choice $P_n = C(A_n)^{-1}$ is mainly useful when such an upper bound is not easily available but an effective preconditioner $C(A_n)$ is known. For Toeplitz-like matrices, there are huge literature studying the construction of preconditioner like [9, 10, 11, 12, 13, 14]. Its performance is comparable to that of $aI_n - A_n$ when $C(A_n)^{-1}$ is a good approximation of A_n^{-1}, so that the spectrum of P_n approximately reverses the spectrum of A_n. If the preconditioner is poor, then the performance can be worse. Thus, $aI_n - A_n$ is preferred when a good upper bound is available, while $C(A_n)^{-1}$ is also a comparable choice when a good preconditioner is available.

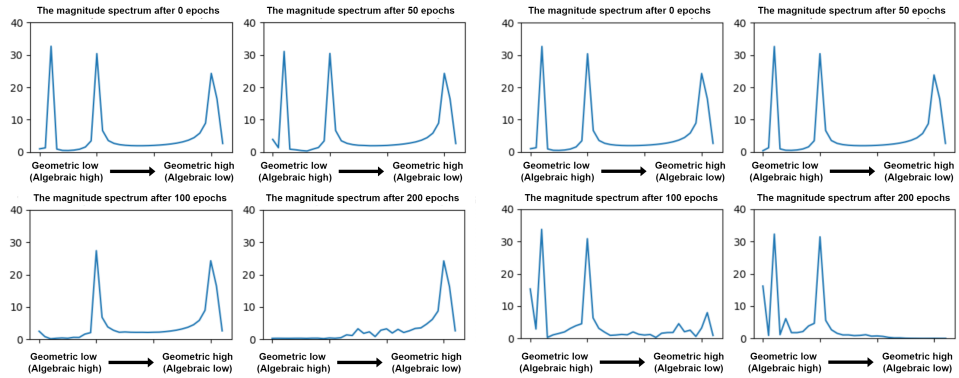
We demonstrate a simple example here to further illustrate the effect of P_n on neural operator training. We consider a discretized one-dimensional Integral equation whose eigenvectors are shown in the first row of Figure 1. In this case, the algebraic high-frequency (resp. algebraic low-frequency) components of A_n correspond to the geometric low-frequency (resp. geometric high-frequency) components. We first construct a vector $\mathbf{x}_0 = \{-\cos(4\pi t/64) + \cos(20\pi t/64) - \cos(60\pi t/64)\}_{t=0}^{64}$ and compute $\mathbf{y}_0 = A_n\mathbf{x}_0$. This vector $\mathbf{x}_0$ consists of three different frequency components. Then, we initialize a neural operator $\mathcal{N}_\theta$ and train it by $\min_\theta \|P_n(\mathcal{N}_\theta(\mathbf{y}_0) - \mathbf{x}_0)\|_2^2$. We choose $P_n = A_n$ and $P_n = C(A_n)^{-1}$ respectively, and compare the predictions of $\mathcal{N}_\theta$.

In Figures 2, we plot the neural operator predictions compare against the ground true solution after different numbers of training epochs and the remaining error after 200 epochs. In Figure 3, we plot the magnitude spectra of the neural operator error after different number of epochs. When $P_n = A_n$, $\mathcal{N}_\theta$ solves the algebraic high-frequency (the same as geometric low-frequency) part associated with A_n and the remaining error is geometric high-frequency. On the contrary, when $P_n = C(A_n)^{-1}$, $\mathcal{N}_\theta$ solves the algebraic low-frequency (the same as geometric high-frequency) part associated with A_n and the remaining error is geometric low-frequency, which is what we desire.



(a) $P_n = A_n$ (b) $P_n = C(A_n)^{-1}$

Fig. 2: (a) and (b) shows the prediction $\mathcal{N}_\theta(\mathbf{y}_0)$ after different numbers of training epochs when $P_n = A_n$ and $P_n = C(A_n)^{-1}$ respectively. Here, the blue curves represent the true solution $\mathbf{x}_0$, the orange curves represent $\mathcal{N}_\theta(\mathbf{y}_0)$, and the red curve represents the remaining error.



(a) $P_n = A_n$ (b) $P_n = C(A_n)^{-1}$

Fig. 3: (a) and (b) show the Fourier magnitude spectra of the error after different numbers of training iterations when $P_n = A_n$ and $P_n = C(A_n)^{-1}$ respectively.

3.2. The modified two-grid method. With a well-trained neural operator solver, we can solve the linear system $A_n\mathbf{x} = \mathbf{y}$ by replacing the coarse-grid correction

in the two-grid method (Algorithm 2.1) by a neural operator correction step. The resulting algorithm is shown in Algorithm 3.1. The overall structure of Algorithm 3.1 is similar to the two-grid method. During each iteration, we first apply κ_1 steps of a classical smoother $\mathcal{S}$ for pre-smoothing. Then we use our trained neural networks developed in Section 3.1 to correct the iterant $\mathbf{x}^{(\kappa+1)}$ in (2.3). Finally, we use κ_2 steps of $\mathcal{S}$ for post-smoothing. In our algorithm, the smoothing steps are used to reduce algebraic high-frequency error, while the neural operator is used to reduce algebraic low-frequency error.

We note that the HINTS method proposed in [50, 51] also replaces the coarse-grid correction in the two-grid method by a neural operator solver DeepONet [33], which tends to fit the geometric low-frequency in the target function first [37]. This phenomenon is probably due to the multi-layer perceptron (MLP) module, which has been proven to favor geometric low-frequency [6], in the structure of DeepONet. Thus, [50, 51] utilize this spectral bias property and train DeepONets to learn geometric low-frequency. The method is good for PDE problems where the smoothers solve the algebraic high-frequency parts of the solution and the DeepONet solves the algebraic low-frequency parts of the solution (which is the same as the geometric low-frequency part), see Figure 1. However, for integral equations, the geometric low-frequency part is also the algebraic high-frequency part, see Figure 1. Hence in the HINTS method, both the smoother and DeepONet will only solve the algebraic high-frequency part of the solution, and therefore the method will not be good for integral equations. Our neural operator in Section 3.1 is designed to exactly recover the algebraic low-frequency part of the solution efficiently.

We also want to mention that there is another way of modifying the two-grid method by replacing the smoothing step by a neural operator and combine it with the coarse-grid correction step. Such a way is good for solving PDEs problems but not suitable for integral equations. The coarser-grid correction is designed to reduce geometric low-frequency components efficiently. For PDE problems, algebraic low-frequency error components correspond to geometric low-frequency components. In this case, a neural operator trained to solve algebraic high-frequency components could be complementary to the coarse-grid correction, and the modified two-grid would work. However, for the integral equations studied here, the algebraic high-frequency components of the error correspond to geometric low-frequency components. If the neural operator were trained to target these algebraic high-frequency components, then both the neural operator and the classical correction step would mainly solve the geometric low-frequency error components. This would be similar to the behavior of classical multigrid methods for integral equations, where the coarse-grid correction is effective for smooth error components but the geometrically oscillatory components are not adequately reduced. The geometric high-frequency components would then be left unsolved. Some numerical examples of solving integral equations by multigrid methods are available in Figure 6 and 7. Therefore, we do not adopt this way of modifying the two-grid method.

Algorithm 3.1

Require: right-hand side vector $\mathbf{y}$, number of pre-smoothing steps v_1, number of post smoothing steps v_2, and the tolerance $\epsilon > 0$.
Set $\tau = 0$ and $\mathbf{x}_0 = 0$
while $\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2/\|\mathbf{y}\|_2 \geq \epsilon$ **do**
$\mathbf{x}_\tau^{(1)} = \mathcal{S}(A_n, \mathbf{y}, \mathbf{x}_\tau, \kappa_1)$ ▷ Pre-smoothing steps
$\mathbf{r}_\tau = (\mathbf{y} - A_n\mathbf{x}_\tau^{(1)})/Z_\tau$ where $Z_\tau = \|\mathbf{y} - A_n\mathbf{x}_\tau^{(1)}\|_2$ ▷ Normalization step
$\mathbf{x}_\tau^{(2)} = \mathcal{N}_\theta(\mathbf{r}_\tau)Z_\tau + \mathbf{x}_\tau^{(1)}$ ▷ Neural operator correction step
$\mathbf{x}_{\tau+1} = \mathcal{S}(A_n, \mathbf{y}, \mathbf{x}_\tau^{(2)}, \kappa_2)$ ▷ Post-smoothing steps
$\tau = \tau + 1$
end while
Return $\mathbf{x}_\tau$

4. Analysis of proposed methods. In this section, we give some theoretical analysis of our proposed methods. We first derive a generalization error estimate for the neural operator training (3.1). Then, we analyze the convergence of Algorithm 3.1 under similar assumptions as the two-grid method [40]. Finally, we study how the choice of P_n in the loss function (3.1) would affect the convergence of different frequency components in the training error.

4.1. Generalization errors analysis of neural operators. We aim to analyze the generalization error when training the neural operator $\mathcal{N}_\theta$. The generalization error reflects how well the model can generalize to new, unseen data, rather than just memorizing the training data. In this section, we derive an analytical upper bound for the generalization error of $\mathcal{N}_\theta$ when $\mathcal{N}_\theta$ is defined as a two-layer ReLU network:

$$\mathcal{N}_\theta(\mathbf{x}) := \sum_{j=1}^{2} c_j \mathrm{ReLU}(W_j\mathbf{x}), \tag{4.1}$$

where $c_j \in \mathbb{R}$, $W_j \in \mathbb{R}^{n\times n}$, $j = 1, 2$, and $\mathrm{ReLU}(z) = \max\{z, 0\}$. Two-layer networks are commonly used in the theoretical study of neural network methods [34, 28, 46] because of their simplicity. We assume the parameter set θ is controlled by a positive contant ρ:

$$\theta \in \Theta(\rho) := \left\{ (c_1, c_2, W_1, W_2) \middle| |c_j| \leq \rho, \|W_j\|_2 \leq 1, j = 1, 2 \right\}.$$

Since $\mathrm{ReLU}(cz) = c\mathrm{ReLU}(z)$ for any $c > 0$, there is no loss of generality of assuming $\|W_j\|_2 \leq 1$.

In practice, we are not able to precisely evaluate the expected loss $\mathcal{L}_{\exp}(\theta)$, so we approximate it with a discrete quadrature instead. Let $\{\mathbf{y}_i\}_{i=1}^N$ be a set of right-hand-side vectors following a probability distribution μ, which is defined on the unit sphere $\mathbb{B}(\mathbb{R}^n)$, and $\{\mathbf{x}_i = A_n^{-1}\mathbf{y}_i\}_{i=1}^N$ be the set of ground true solutions. Here, N is the number of training samples. In practice, we first sample $\mathbf{x}_i$ from some random distribution and then compute $\mathbf{y}_i = A_n\mathbf{x}_i$. Then, we train the network by minimizing the empirical loss function:

$$\mathcal{L}_{\mathrm{emp}}^N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \|P_n(\mathcal{N}_\theta(\mathbf{y}_i) - \mathbf{x}_i)\|_2^2. \tag{4.2}$$

Here $\mathcal{L}^N_{\text{emp}}(\theta)$ is a discrete quadrature approximation to $\mathcal{L}_{\text{exp}}(\theta)$. We also define an error measurement for $\mathcal{N}_\theta$ as

$$\|\mathcal{N}_\theta - A_n^{-1}\|^2_{A_n,\mu} := \mathbb{E}_{\mathbf{y}\sim\mu}\|\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y}\|^2_{A_n} = \int_{\mathbf{y}\sim\mathbb{R}^d} \|\mathcal{N}_\theta(\mathbf{y}) - A^{-1}\mathbf{y}\|^2_{A_n}\mu(\mathbf{y})d\mathbf{y}.$$

Then, the estimation of the generalization error $\|\mathcal{N}_\theta - A_n^{-1}\|^2_{A_n,\mu}$ is given below.

DEFINITION 4.1. *A vector* $\mathbf{x}^*$ *is a* δ**-minimizer** *of a function* $f(\mathbf{x})$ *if* $f(\mathbf{x}^*) \leq \delta + \inf_{\mathbf{x}} f(\mathbf{x})$.

In Definition 4.1, δ bounds the deviation between $f(\mathbf{x}^*)$ and the global minimum, so δ is a measure of the optimization error.

THEOREM 4.2. *Let* $\{\mathbf{y}_i\}_{i=1}^N$ *be a set of training data randomly sampled from a distribution defined on the unit ball* $\mu(\mathbb{B}(\mathbb{R}^n))$, $\mathcal{N}_\theta$ *is a two-layer network defined as (4.1), and* θ^* *is a* δ*-minimizer of the empirical loss:*

$$\min_{\theta\in\Theta(\rho)} \mathcal{L}^N_{\text{emp}}(\theta)$$

where $\rho \geq \|A_n^{-1}\|_2$. *Then, the following inequality holds with probability at least* $1-\epsilon$*:*

$$\|\mathcal{N}_{\theta^*} - A_n^{-1}\|^2_{A_n,\mu} \leq \|A_n P_n^{-2}\|_2 \left(\frac{108n\rho^2\|P_n\|_F^2}{\sqrt{N}} + 9\rho^2\|P_n\|_2^2\sqrt{\frac{2\log(2/\epsilon)}{N}} + \delta \right),$$

where $\|\cdot\|_F$ *denotes the matrix Frobenius norm.*

Remark 4.3. In our error estimates, ρ is chosen to be greater than $\|A_n^{-1}\|_2 = 1/\lambda_{\min}(A_n)$, where $\lambda_{\min}(A_n)$ denotes the smallest eigenvalue of A_n. For the systems considered in this work, the integral equation includes a regularization term, ensuring that the minimum eigenvalue of A_n is bounded away from zero independently of n. For instance, when $R(u) = u$ in (2.1), the coefficient matrix is $A_n = \alpha I_n + K_n$, where I_n is the n-by-n identity matrix and K_n is the discretized Gaussian convolution matrix. Since the eigenvalues of K_n are positive, we have $\lambda_{\min}(A_n) \geq \alpha$. Consequently, one can choose ρ to be any constant greater than $1/\alpha$, which is independent of n.

The proof of Theorem 4.2 requires some results in Appendix A.

Proof. We first construct a set of parameters $\hat{\theta} \in \Theta(\rho)$ as

$$\hat{\theta} = \left(\|A_n^{-1}\|_2, -\|A_n^{-1}\|_2, \frac{A_n^{-1}}{\|A_n^{-1}\|_2}, -\frac{A_n^{-1}}{\|A_n^{-1}\|_2} \right).$$

Then, we have

$$\mathcal{N}_{\hat{\theta}}(\mathbf{y}) = \|A_n^{-1}\|_2 \text{ReLU}\left(\frac{A_n^{-1}}{\|A_n^{-1}\|_2}\mathbf{y}\right) - \|A_n^{-1}\|_2 \text{ReLU}\left(\frac{-A_n^{-1}}{\|A_n^{-1}\|_2}\mathbf{y}\right) = A_n^{-1}\mathbf{y}$$

for any $\mathbf{y} \in \mathbb{R}^n$, which implies that $\mathcal{N}_{\hat{\theta}}$ exactly recover the operator A_n^{-1} and $\mathcal{L}_{\text{exp}}(\hat{\theta}) = \mathcal{L}^N_{\text{emp}}(\hat{\theta}) = 0$.

The generalization errors of $\mathcal{N}_{\theta^*}$ is bounded as follows:

$$\begin{aligned}
(4.3) \qquad & \|\mathcal{N}_{\theta^*} - A_n^{-1}\|^2_{A_n,\mu} = \mathbb{E}_{\mathbf{y}\sim\mu}(\|A_n^{1/2}(\mathcal{N}_{\theta^*}(\mathbf{y}) - A_n^{-1}\mathbf{y})\|_2^2) \\
& = \mathbb{E}_{\mathbf{y}\sim\mu}(\|A_n^{1/2}P_n^{-1}P_n(\mathcal{N}_{\theta^*}(\mathbf{y}) - A_n^{-1}\mathbf{y})\|_2^2) \\
& \leq \|A_n P_n^{-2}\|_2(\mathcal{L}_{\text{exp}}(\theta^*) - \mathcal{L}^N_{\text{emp}}(\theta^*) + \mathcal{L}^N_{\text{emp}}(\theta^*)) \\
& \leq \|A_n P_n^{-2}\|_2(\mathcal{L}_{\text{exp}}(\theta^*) - \mathcal{L}^N_{\text{emp}}(\theta^*) + \mathcal{L}^N_{\text{emp}}(\hat{\theta}) + \delta)
\end{aligned}$$

Here, $\mathcal{L}^N_{\text{emp}}(\hat{\theta}) = 0$ and $\mathcal{L}_{\exp}(\theta^*) - \mathcal{L}^N_{\text{emp}}(\theta^*)$ is bounded by Lemma A.6. We define the function class $\mathcal{H} := \{h(\mathbf{y}) = \|P_n(\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y})\|_2^2 | \theta \in \Theta(\rho)\}$. By Lemma A.6, the error $\mathcal{L}_{\exp}(\theta^*) - \mathcal{L}^N_{\text{emp}}(\theta^*)$ is bounded by $2R_N(\mathcal{H}) + \sup_{h\in\mathcal{H}} \|h\|_\infty \sqrt{\frac{2\log(2/\epsilon)}{N}}$ with probability at least $1-\epsilon$. Consequently, the generalization error is bounded, with probability at least $1-\epsilon$, by

$$\|\mathcal{N}_{\theta^*} - A_n^{-1}\mathbf{y}\|^2_{A_n,\mu} \le \|A_n P_n^{-2}\|_2 (2R_N(\mathcal{H}) + \sup_{h\in\mathcal{H}} \|h\|_\infty \sqrt{\frac{2\log(2/\epsilon)}{N}} + \delta). \tag{4.4}$$

For every $h \in \mathcal{H}$ and $\mathbf{y} \in \mathbb{B}(\mathbb{R}^d)$,

$$|h(\mathbf{y})| \le \|P_n\|_2^2 \|c_1 \text{ReLU}(W_1\mathbf{y}) + c_1\text{ReLU}(W_2\mathbf{y}) - A_n^{-1}\mathbf{y}\|_2^2 \le 9\|P_n\|_2^2\rho^2.$$

What remains is bounding the complexity $R_N(\mathcal{H})$. Notice that

$$\begin{aligned} \|P_n(\mathcal{N}_\theta(y) - A_n^{-1}\mathbf{y})\|_2^2 &= \sum_{j=1}^n \left(\sum_{k=1}^n P_n[j,k](\mathcal{N}(\mathbf{y})_\theta - A_n^{-1}\mathbf{y})[k] \right)^2 \\ &= \sum_{j=1}^n \left(\sum_{k=1}^n \sum_{l=1}^n P_n[j,k]P_n[j,l](\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y})[k](\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y})[l] \right). \end{aligned} \tag{4.5}$$

Here, $[\cdot]$ denotes the indices of a matrix or vector. We define another class of functions $\mathcal{G}$ as

$$\left\{ g(\mathbf{y}) = c_1\text{ReLU}(\mathbf{w}_1^\top\mathbf{y}) + c_2\text{ReLU}(\mathbf{w}_2^\top\mathbf{y}) - \mathbf{w}_3^\top\mathbf{y} \middle| |c_j| \le \rho, \|\mathbf{w}_j\|_2 \le 1, j = 1,2; \|\mathbf{w}_3\|_2 \le \rho \right\}.$$

Then, each $(\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y})[k]$, $k = 1,\dots,n$, belongs to $\mathcal{G}$, and by (4.5),

$$\mathcal{H} \subset \sum_{j=1}^n \left(\sum_{k=1}^n \sum_{l=1}^n P_n[j,k]P_n[j,l](\mathcal{G}\times\mathcal{G}) \right).$$

For any $g \in \mathcal{G}$ and $\mathbf{y} \sim \mu$, we have $|g(\mathbf{y})| \le 3\rho$. Then, using Lemma A.3,

$$\begin{aligned} R_N(\mathcal{H}) &\le \sum_{j=1}^n \left(\sum_{k=1}^n \sum_{l=1}^n P_n[j,k]P_n[j,l] R_N(\mathcal{G}\times\mathcal{G}) \right) && \text{(By Lemma A.3(1-3))} \\ &\le \sum_{j=1}^n \left(\sum_{k=1}^n \sum_{l=1}^n \left(\frac{1}{2}P_n[j,k]^2 + \frac{1}{2}P_n[j,l]^2 \right) R_N(\mathcal{G}\times\mathcal{G}) \right) \\ &\le n\|P_n\|_F^2 (R_N(\mathcal{G}\times\mathcal{G})) \\ &\le 18n\rho\|P_n\|_F^2 R_N(\mathcal{G}) && \text{(By Lemma A.3(6))} \end{aligned} \tag{4.6}$$

We further estimate $R_N(\mathcal{G})$ by $R_N(\mathcal{G}) \le 2R_N(\mathcal{G}_1) + R_N(\mathcal{G}_2)$. Here $\mathcal{G}_1$ is defined as $\{a\text{ReLU}(\mathbf{w}^\top\mathbf{y}) | \|\mathbf{w}\|_2 \le 1, |a| \le \rho\}$ and $\mathcal{G}_2$ is defined as $\{\mathbf{w}^\top\mathbf{y} | \|\mathbf{w}\|_2 \le \rho\}$. Then, we directly apply Lemma A.4 and Lemma A.5 to the above inequality, and get

$$R_N(\mathcal{G}) \le 3\rho \frac{\sup_{\mathbf{y}\sim\mu}\|\mathbf{y}\|_2}{\sqrt{N}} \le \frac{3\rho}{\sqrt{N}}.$$

Then, by applying the above estimate of $R_N(\mathcal{G})$ to (4.6), we have

$$R_N(\mathcal{H}) \leq 54n\rho^2\|P_n\|_F^2/\sqrt{N}. \tag{4.7}$$

Finally, by combining (4.7) and (4.4), we have

$$\begin{aligned}\|\mathcal{N}_{\theta^*} - A_n^{-1}\|_{A_n,\mu}^2 \leq & \|A_nP_n^{-2}\|_2\left(2R_N(\mathcal{H}) + \sup_{h\in\mathcal{H}}\|h\|_\infty\sqrt{\frac{2\log(2/\epsilon)}{N}} + \delta\right)\\ \leq & \|A_nP_n^{-2}\|_2\left(\frac{108n\rho^2\|P_n\|_F^2}{\sqrt{N}} + 9\rho^2\|P_n\|_2^2\sqrt{\frac{2\log(2/\epsilon)}{N}} + \delta\right) \quad \square\end{aligned}$$

with probability at least $1-\epsilon$.

This theorem implies that, for any fixed n, with high probability the generalization error will converge to 0 as N goes to infinity and δ goes to 0. Here, δ corresponds to the optimization error when minimizing (4.2).

In Theorem 4.2, the two-layer ReLU network can represent the linear map A_n^{-1} exactly. This exact representation eliminates the approximation error from the analysis and simplifies the proof. For a more general network class, this exact representation may not hold. In this case, let Θ denotes the feasible parameter set, if θ^* is a δ-minimizer of the empirical loss over Θ, then the proof from (4.3) becomes

$$\begin{aligned}&\|\mathcal{N}_{\theta^*} - A_n^{-1}\|_{A_n,\mu}^2 = \mathbb{E}_{\mathbf{y}\sim\mu}(\|A_n^{1/2}(\mathcal{N}_{\theta^*}(\mathbf{y}) - A_n^{-1}\mathbf{y})\|_2^2)\\ &\leq \|A_nP_n^{-2}\|_2(\mathcal{L}_{\exp}(\theta^*) - \mathcal{L}_{\text{emp}}^N(\theta^*) + \mathcal{L}_{\text{emp}}^N(\theta) + \delta)\\ &= \|A_nP_n^{-2}\|_2(\mathcal{L}_{\exp}(\theta^*) - \mathcal{L}_{\text{emp}}^N(\theta^*) + \mathcal{L}_{\text{emp}}^N(\theta) - \mathcal{L}_{\exp}(\theta) + \mathcal{L}_{\exp}(\theta) + \delta)\end{aligned}$$

where θ is an arbitrary set of parameters in Θ. Both differences $\mathcal{L}_{\exp}(\theta^*) - \mathcal{L}_{\text{emp}}^N(\theta^*)$ and $\mathcal{L}_{\text{emp}}^N(\theta) - \mathcal{L}_{\exp}(\theta)$ can be bounded by the same Rademacher complexity estimate used above:

$$2R_N(\mathcal{H}) + \sup_{h\in\mathcal{H}}\|h\|_\infty\sqrt{\frac{2\log(2/\epsilon)}{N}}.$$

Taking the minimum over all $\theta\in\Theta$ on the right-hand side gives

$$\|A_nP_n^{-2}\|_2\left(4R_N(\mathcal{H}) + 2\sup_{h\in\mathcal{H}}\|h\|_\infty\sqrt{\frac{2\log(2/\epsilon)}{N}} + \min_{\theta\in\Theta}\mathcal{L}_{\exp}(\theta) + \delta\right)$$

as an upper bound for $\|\mathcal{N}_{\theta^*} - A_n^{-1}\|_{A_n,\mu}^2$. The new term $\min_{\theta\in\Theta}\mathcal{L}_{\exp}(\theta)$ is controlled by the approximation error of the network class:

$$\min_{\theta\in\Theta}\mathcal{L}_{\exp}(\theta) \leq \|P_n\|_2^2\min_{\theta\in\Theta}\mathbb{E}_{\mathbf{y}\sim\mu}\|\mathcal{N}_\theta(\mathbf{y}) - A_n^{-1}\mathbf{y}\|_2^2.$$

For standard activation functions and architectures, this approximation error can often be bounded effectively and shown to converge to 0 as the network size increases, as studied in related works [52, 49].

Therefore, using other activation functions or more general network classes simply introduces an additional approximation-error term into the bound, unlike the two-layer ReLU case, where the approximation error is exactly zero because $\hat{\theta}$ can perfectly represent the inverse operator. If we choose a network class with sufficient expressive power, then the approximation error can be made sufficiently small. Thus, the generalization error can still be effectively bounded.

4.2. Convergence analysis of Algorithm 3.1. The convergence of conventional two-grid methods has been well studied in the literature [40, 43] based on reasonable assumptions on the smoothing step and the correcting step. We aim to conduct a similar analysis for our proposed hybrid iterative methods under similar assumptions. We focus on a simple case of Algorithm 3.1 when $\kappa_1 = 0$ and $\kappa_2 = 1$, *i.e.*, only one post-smoothing step and no pre-smoothing step. We can prove the following estimate by making reasonable assumptions on the neural operators $\mathcal{N}_\theta$ similar to assumptions made for coarse-grid correction in [40].

THEOREM 4.4. *Suppose $\mathbf{y} \neq 0$ be a vector in $\mathbb{R}^n$, $\mathcal{N}_\theta$ be a neural network solver, and the iterative solver $\mathcal{S}$ (defined as (2.3) with iteration matrix S_n) satisfies the smoothing condition (2.5) for some $\delta_1 > 0$. Let $\mathbf{x}_\tau$, $\tau = 0, 1, \ldots$, be a sequence generated by Algorithm 3.1 with $\kappa_1 = 0$ and $\kappa_2 = 1$. Then, the following inequality holds*

$$\|\mathbf{x}_{\tau+1} - A_n^{-1}\mathbf{y}\|_{A_n}^2 \leq \left(1 - \frac{\delta_1}{\delta_2}\right)\delta_3 \|\mathbf{x}_\tau - A_n^{-1}\mathbf{y}\|_{A_n}^2$$

if $\mathcal{N}_\theta$ satisfies:

1. *correcting condition: $\|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \frac{\delta_2}{a_0}\|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n^2}^2$ for some $\delta_2 > 0$.*
2. *stability condition: $\|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \delta_3 \|A_n^{-1}\mathbf{r}\|_{A_n}^2$ for some $\delta_3 > 0$.*

Here, $\mathbf{r} = \frac{\mathbf{y} - A_n\mathbf{x}_\tau}{\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2}$.

Proof. Following Algorithm 3.1, $\mathbf{x}_{\tau+1}$ is calculated as

$$\mathbf{r} = \frac{\mathbf{y} - A_n\mathbf{x}_\tau}{\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2}; \quad \hat{\mathbf{x}}_\tau = \mathcal{N}_\theta(\mathbf{r})\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2 + \mathbf{x}_\tau; \quad \mathbf{x}_{\tau+1} = \mathcal{S}(A_n, \mathbf{y}, \hat{\mathbf{x}}_\tau, 1)$$

Then, we have

$$\begin{aligned}
&\|\mathbf{x}_{\tau+1} - A_n^{-1}\mathbf{y}\|_{A_n}^2 = \|S_n(\hat{\mathbf{x}}_\tau - A_n^{-1}\mathbf{y})\|_{A_n}^2 \\
&\leq \|\hat{\mathbf{x}}_\tau - A_n^{-1}\mathbf{y}\|_{A_n}^2 - \frac{\delta_1}{a_0}\|\hat{\mathbf{x}}_\tau - A_n^{-1}\mathbf{y}\|_{A_n^2}^2 \\
&= \|\mathbf{y} - A_n\mathbf{x}_\tau\|_2^2 \left(\|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 - \frac{\delta_1}{a_0}\|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n^2}^2\right) \\
&\leq (1 - \frac{\delta_1}{\delta_2})\|\mathbf{y} - A_n\mathbf{x}_\tau\|_2^2 \|\mathcal{N}_\theta(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \\
&\leq (1 - \frac{\delta_1}{\delta_2})\delta_3 \|\mathbf{y} - A_n\mathbf{x}_\tau\|_2^2 \|A_n^{-1}\mathbf{r}\|_{A_n}^2 = (1 - \frac{\delta_1}{\delta_2})\delta_3 \|\mathbf{x}_\tau - A_n^{-1}\mathbf{y}\|_{A_n}^2
\end{aligned}$$

∎

Consequently, the iteration of Algorithm 2 (with $\kappa_1 = 0$ and $\kappa_2 = 1$) would converge if $(1 - \delta_1/\delta_2)\delta_3 < 1$. Generally, it would be difficult to guarantee that the correcting condition and stability condition stated in Theorem 4.4 hold all the time. Using the Markov's inequality [31] and Theorem 4.2, we can show the stability condition would hold with high probability for well-trained $\mathcal{N}_\theta$.

COROLLARY 4.5. *Given any confidence level $\epsilon > 0$ and $\mathbf{r} \sim \mu(\mathbb{B}(\mathbb{R}^n))$. Let $\mathcal{N}_\theta$ be a two-layer network, and θ^* is a δ-minimizer of the empirical loss:*

$$\min_{\theta \in \Theta(\rho)} \mathcal{L}_{\text{emp}}^N(\theta)$$

where $\rho \geq \|A_n^{-1}\|_2$. Then, the stability condition

$$\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \delta_3 \|A_n^{-1}\mathbf{r}\|_{A_n}^2$$

holds with probability at least $1-\epsilon$ if N is sufficiently large and δ is sufficiently small.

Proof. Since $\|\mathbf{r}\|_2 = 1$, we have

$$\|A_n^{-1}\mathbf{r}\|_{A_n} \geq (\lambda_{\min}(A_n^{-1})\|\mathbf{r}\|_2^2)^{1/2} = \frac{1}{\lambda_{\max}(A_n)^{1/2}} = \frac{1}{\|A_n\|_2^{1/2}}.$$

Then,

$$\begin{aligned}
&\text{Prob}\left(\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \geq \delta_3\|A_n^{-1}\mathbf{r}\|_{A_n}^2\right)\\
&\leq \text{Prob}\left(\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \geq \frac{\delta_3}{\|A_n\|_2}\right)\\
&\leq \frac{\|A_nP_n^{-2}\|_2\|A_n\|_2}{\delta_3}\left(\frac{108n\rho^2\|P_n\|_F^2}{\sqrt{N}} + 9\rho^2\|P_n\|_2^2\sqrt{\frac{2\log(4/\epsilon)}{N}} + \delta\right). \qquad (4.8)
\end{aligned}$$

The last inequality is derived using Markov's inequality [31] and the generalization error estimation in Theorem 4.2, and it holds with probability at least $1-\epsilon/2$. Therefore, by choosing sufficiently large N and solving the empirical loss function sufficiently well such that δ is small enough, we can have

$$\text{Prob}(\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \geq \delta_3\|A_n^{-1}\mathbf{r}\|_{A_n}^2) \leq \epsilon/2,$$

indicating that

$$\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \delta_3\|A_n^{-1}\mathbf{r}\|_{A_n}^2 \qquad (4.9)$$

with probability at least $1-\epsilon/2$. The inequality $\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \delta_3\|A_n^{-1}\mathbf{r}\|_{A_n}^2$ hold if both (4.8) and (4.9) hold simultaneously. By [26], we have

$$\text{Prob}(\|\mathcal{N}_{\theta^*}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2 \leq \delta_3\|A_n^{-1}\mathbf{r}\|_{A_n}^2) \geq (1-\epsilon/2) + (1-\epsilon/2) - 1 = 1-\epsilon. \qquad \square$$

For the correcting condition in Theorem 4.4, we will verify it numerically for some convolution-type integral equations in Section 5.

4.3. The effect of P_n on the training dynamics. In this part, we would like to study how the choice of P_n in the training loss (3.1) would affect the training dynamics of our neural operators. More specifically, we study how the convergence rate of different algebraic frequency components in training error depends on the choice of P_n.

Let $\{\mathbf{v}_i\}_{i=1}^n$ be a set of orthonormal eigenvectors of A_n with corresponding eigenvalues $\{\lambda_i\}_{i=1}^n$ and λ_is are sorted in descending order: $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n > 0$. Then, for each $\mathbf{y}_j$, the projection of the prediction error $\mathcal{N}_\theta(\mathbf{y}_j) - \mathbf{x}_j$ on to each $\mathbf{v}_i$ is

$$f_i^j(\theta) := (\mathcal{N}_\theta(\mathbf{y}_j) - \mathbf{x}_j)^\top \mathbf{v}_i.$$

Here, f_i^j indicates the i-th algebraic frequency component of the prediction error on the j-th sample. We also defined the i-th algebraic frequency components in prediction errors over the entire training set by $F_i(\theta) := \frac{1}{N}\sum_{j=1}^N (f_i^j)^2$. In the previous section, we proposed that P_n can be defined as $aI_n - A_n$ for some $a > \lambda_1$. Next, we will study how $F_i(\theta)$ would be affected by a.

Since the error $\mathcal{N}_\theta(\mathbf{y}_j) - \mathbf{x}_j$ can be decomposed by $\sum_{i=1}^n f_i^j(\theta)\mathbf{v}_i$. Then, the loss function $\mathcal{L}_{\text{emp}}^N(\theta)$ can be written as

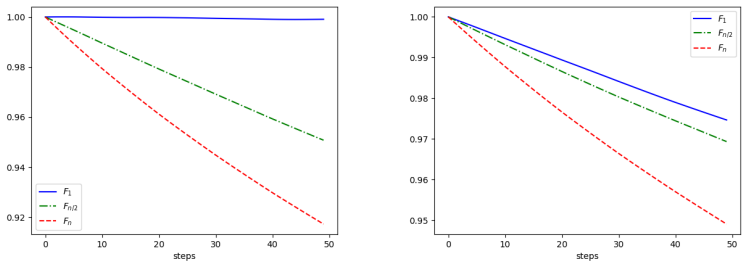
$$\begin{aligned}
\mathcal{L}_{\text{emp}}^N(\theta) &= \frac{1}{N}\sum_{j=1}^N \|P_n(\mathcal{N}_\theta(\mathbf{y}_j) - \mathbf{x}_j)\|_2^2 \\
&= \frac{1}{N}\sum_{j=1}^N \left\| (aI_n - A_n)\left(\sum_{i=1}^n f_i^j(\theta)\mathbf{v}_i\right)\right\|_2^2 \\
&= \frac{1}{N}\sum_{j=1}^N \left\| \sum_{i=1}^n f_i^j(\theta)(a-\lambda_i)\mathbf{v}_i\right\|_2^2 \\
&= \frac{1}{N}\sum_{j=1}^N \sum_{i=1}^n f_i^j(\theta)^2(a-\lambda_i)^2 \\
&= \sum_{i=1}^n (a-\lambda_i)^2 \frac{1}{N}\sum_{j=1}^N f_i^j(\theta)^2 \\
&= \sum_{i=1}^n (a-\lambda_i)^2 F_i(\theta).
\end{aligned}$$

Consequently, for each $F_i(\theta)$, we have an upper bound $F_i(\theta) \le \frac{\mathcal{L}_{\text{emp}}^N(\theta)}{(a-\lambda_i)^2}$ during training. We can compare the upper bound of F_1 and F_n

$$\frac{\mathcal{L}_{\text{emp}}^N(\theta)/(a-\lambda_1)^2}{\mathcal{L}_{\text{emp}}^N(\theta)/(a-\lambda_n)^2} = \frac{(a-\lambda_n)^2}{(a-\lambda_1)^2}.$$

When $a \to \lambda_1^+$, the ratio $\frac{(a-\lambda_n)^2}{(a-\lambda_1)^2}$ converges to $+\infty$, which implies that the algebraic low-frequency components in the training error have significantly lower bounds than the algebraic high-frequency components. Consequently, we expect that algebraic low-frequency components would decay much faster than algebraic high-frequency components during training.

We will further verify it by the following numerical examples. We train a two-layer fully-connected network by minimizing the empirical loss $\mathcal{L}_{\text{emp}}^N$ using the gradient descent method, where the coefficient matrix A_n is discretized from a one-dimensional integral equation (the construction of A_n is the same as (5.1) in Section 5.2.1) The step size is of order $O(a^{-1})$. Here, we use $n = 32$ and the maximum eigenvalue of A_n is smaller than but close to 1.1. Then, we visualize the change of $F_i(\theta^\tau)$ during the first 50 time steps in Figure 4 for $a = 1.1$ and $a = 1.5$. In Figure 4, F_1, $F_{n/2}$, and F_n represent the algebraic high, medium, and low frequency components in the training error. When $a = 1.1$, we observe that F_n and $F_{n/2}$ decay fast while F_1 remains almost unchanged, which implies that $\mathcal{N}_\theta$ is focusing on solving the algebraic low frequency rather than the high frequency. When $a = 1.5$, the convergence rates of different frequency components become comparable, indicating that $\mathcal{N}_\theta$ does not focus on any part of frequency specifically.



(a) $a = 1.1$ (b) $a = 1.5$

Fig. 4: Figures of $F_i(\theta^\tau)/F_i(\theta^0)$, $\tau = 0, 1, \ldots$, during training for different choices of a.

5. Numerical experiments.

5.1. Training setting. The training of all neural operators in this section is conducted on the NVIDIA RTX A6000 GPU with 48 GB of memory. For one-dimensional problems, we train the neural operators for 1,000 epochs with a learning rate of 10^{-4} and a batch size of 100. For two-dimensional problems, we train for 2,000 epochs with a learning rate of 10^{-4} and a batch size of 10. Details of the neural operator structures are given in Appendix B. In all experiments, we set $\kappa_2 = 0$ in Algorithm 3.1, *i.e.*, we only do the pre-smoothing step. In each batch, we randomly generate training samples $\mathbf{x}$ from a uniform distribution between $[0, 1]$ (using the numpy.random.rand function) and compute the right-hand side vector $\mathbf{y} = A_n\mathbf{x}$. To improve the long-term stability of our methods, we apply a strategy similar to [24] during training. Specifically, after generating random training sample pairs $(\mathbf{x}, \mathbf{y})$, we apply $\hat{\kappa}$ iterations of Algorithm 3.1 with the latest updated $\mathcal{N}_\theta$. Then, we calculate the remaining error $\mathbf{e}$ and the residual $\mathbf{r} = A_n\mathbf{e}$. The pair $(\mathbf{r}, \mathbf{e})$ is further used as training data. We visualize the training process in Figure 5. The number $\hat{\kappa}$ is randomly chosen from 1 to 10.

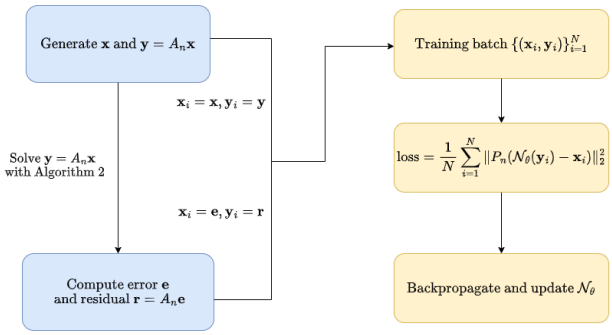


Fig. 5: Training process of one batch

5.2. One-dimensional integral equations.

5.2.1. Tikhonov regularized integral equation. We consider (2.1) where $R(u) = u$, $\mathcal{K}$ is the Gaussian smoothing kernel, and $\Omega = [0, 1]$. The $R(u) = u$ corresponds to the gradient of the Tikhonov regularization term $\|u\|^2_{L^2(\Omega)}$. The coefficient matrix of the discretized system is

$$A_n = \alpha I_n + K_n \tag{5.1}$$

where $\alpha > 0$ is the weight of regularization, I_n is the identity matrix, and K_n is a discrete convolution matrix of a Gaussian smoothing kernel. We set the standard deviation of the Gaussian kernel to be 1.5. Then A_n is a Toeplitz matrix. During training, we define $P_n = (\alpha+1)I_n - A_n$ in the loss function (3.1). Since the sum of each row of K_n is less than or equal to 1, the maximum eigenvalue of K_n is less than 1, which is guaranteed by the Gershgorin circle theorem [17]. Then, the maximum eigenvalue of A_n is less than $1 + \alpha$. During the evaluation, we solve the system $A_n\mathbf{x} = \mathbf{y}$ using our proposed algorithm (Algorithm 3.1 with $\kappa_1 = 5$, $\kappa_2 = 0$, and the pre-smoothing step is performed by the weighted Jacobi iterations with weight 0.3) until the relative residual error is less than 10^{-10}. For comparison, we also applied the two-grid method to solve this equation. Our two-grid method follows Algorithm 2.1 with $\kappa = 5$ and uses the weighted Jacobi iteration with weight 0.3 to perform the smoothing step. We test all methods on different α and n. We also note that the number of required iterations is almost the same for both the two-grid method and the more advanced multigrid methods. This is because the dominant components in the error are the highest geometric frequency (or equivalently, the lowest algebraic frequency), which is only solvable on the finest grid. Consequently, convergence is primarily affected by the number of smoothing steps applied at the finest level, rather than the specific multigrid scheme. The condition numbers of the system A_n corresponding to different α and n are listed in Table 8. The **training time** for our neural operators is about 10**min**, 12**min**, and 16**min** when $n = 256$, 512, and 1024 respectively. For each pair, all methods are evaluated on 10 random right-hand-side vectors. Table 1 reports the average iteration count, evaluation time, and the standard deviation of the number of iterations.

In these experiments, our proposed method converges significantly faster than the two-grid method, requiring fewer iterations and less computational time. Besides, the iteration number of our proposed method is robust to variations in the regularization parameter α, whereas the two-grid method is highly sensitive to changes in α. Thanks to the highly parallelized structure of neural networks, the computational time of our method scales far more gradually with increasing problem size compared to the two-grid approach. A graphical comparison (Figure 6) highlights the efficiency of our hybrid iterative method in resolving both low- and high-frequency components, while the two-grid method reduces only geometric low-frequency modes.

5.2.2. anisotropic diffusion regularized integral equation. Here, we also consider another regularization term $R(u)(z) = -\nabla \cdot (a(z)u(z))$ for the integral equation (2.1), where $a(z) = 1 + 0.5\sin(2\pi z)$. The coefficient matrix of the discretized system is $A_n = \alpha D_n + K_n$, where D_n corresponds to the regularization term and K_n corresponds to the convolution term. Here, K_n is the same as in the previous subsection, and D_n is a tri-diagonal matrix with varying coefficients along the diagonals. During the training, we choose the P_n in the loss function (3.1) to be the inverse of the optimal circulant preconditioner [13]. In this case, though A_n is not a Toeplitz matrix, the optimal circulant preconditioner is a good approximation to A_n if $a(z)$ is uniformly bounded away from zero and from above [9]. The inverse of the circulant

	α	$n = 256$	$n = 512$	$n = 1024$
two-grid method	10^{-3}	2405(1.6s)±24	2407(4s)±14	2413(14s)±13
	10^{-4}	15928(11s)±233	15951(27s)±145	15983(91s)±123
	10^{-5}	45705(26s)±904	45808(76s)±550	46000(311s)±478
proposed method	10^{-3}	6.6(0.05s)±0.4	7.0(0.06s)±0.0	6.8(0.08s) ±0.4
	10^{-4}	6.8(0.05s)±0.4	6.8(0.06s)±0.1	7.0(0.08s)±0.5
	10^{-5}	6.8(0.05s)±0.5	7.0(0.06s)±0.0	6.9(0.08s)±0.3

Table 1: The average number of iterations and evaluation time required for solving the one-dimensional integral equation with Tikhonov regularization.

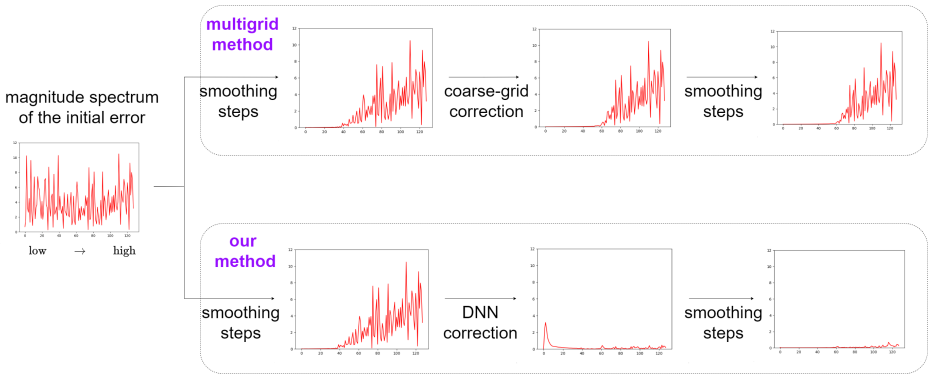


Fig. 6: The magnitude spectra from our proposed method and the two-grid method. Since the magnitude spectra of real vectors are symmetric, here we show half of the spectra. In these figures, the x-axis represents the frequency domain where the left part corresponds to geometric low-frequency and the right part corresponds to geometric high-frequency.

preconditioner can be efficiently implemented using the Fast Fourier transform. We keep the same implementation details as in the Tikhonov case and only change the weight of the weighted Jacobi iteration to 0.025 for both the two-grid method and our proposed method. We compare our methods with the two-grid method for this problem with different n and α. The condition number of each case is listed in Table 8, and the results are reported in Table 2. Our proposed algorithm consistently outperformed the two-grid method in both the iteration number and computational time. The convergence rate of our method is also robust to the change of α compared to the two-grid method.

5.3. Two-dimensional integral equations.

Tikhonov regularized integral equation. We also consider a two-dimensional Integral equation with Tikhonov regularization: $A_{n^2} = \alpha I_{n^2} + K_{n^2}$ where $I_{n^2} = I_n \otimes I_n$, $K_{n^2} = K_n \otimes K_n$, and $\otimes$ denotes the Kronecker product. The size of the system A_{n^2} is $n^2 \times n^2$. The K_n here is the same as Section 5.2. The matrix vector

	α	$n = 256$	$n = 512$	$n = 1024$
two-grid method	10^{-3}	1196(0.8s)±28	1200(2s)±19	1201(7s)±11
	10^{-4}	8550(6s)±288	8640(15s)±202	8678(49s)±117
	10^{-5}	33876(22s)±1724	34338(60s)±1214	34566(196s)±700
proposed method	10^{-3}	5.1(0.05s)±0.3	5.6(0.06s)±0.5	6.0(0.07s)±0.0
	10^{-4}	6.0(0.05s)±0.0	6.0(0.06s)±0.0	6.0(0.07s)±0.0
	10^{-5}	6.0(0.05s)±0.0	6.1(0.06s)±0.3	7.1(0.08s)±0.5

Table 2: The average number of iterations and evaluation time required for solving the one-dimensional integral equation with anisotropic diffusion regularization.

multiplications $A_{n^2}\mathbf{x}$ can be computed efficiently by using this tensor strucutre:

$$A_{n^2}\mathbf{x} = \alpha\mathbf{x} + \text{Vec}(K_n^\top X K_n) \tag{5.2}$$

where $X \in \mathbb{R}^{n\times n}$ is the matricization reshaping of $\mathbf{x}$, and Vec denotes the vectorization function, which is the inverse of matricization, of the input matrix. The multiplication $K_n^\top X K_n$ can further be implemented through the one-dimensional convolution. In the loss function $\mathcal{L}^N_{\text{emp}}(\theta)$, we choose $P_{n^2} = aI_{n^2} - A_{n^2}$ with $a = 1 + \alpha$. We consider different choices of α and n, and the condition numbers of the corresponding systems are shown in Table 8. The **training time** for our neural operators is about 36**min**, 152**min**, and 645**min** when $n = 256$, 512, and 1024 respectively.

To solve this linear system, we apply Algorithm 3.1 with $\kappa_1 = 20$, $\kappa = 0$, and the pre-smoothing step is performed by the weighted Jacobi iteration with weight 0.01. We compare our method against two benchmark approaches: the multigrid method and the preconditioned conjugate gradient (PCG) method, where the preconditioner for the PCG iteration is the block-circulant preconditioner from [10]. During the implementations of all methods, we implement the matrix vector multiplications via (5.2). For the multigrid method, we do 20 steps of the weighted Jacobi iteration for pre-smoothing at each level, and do not perform post-smoothing (This is also called the \-cycle). For $n = 256$, 512, and 1024, the total multigrid level is chosen to be 5, 6, and 7, respectively. Similar to the one-dimensional cases, here the convergence of multigrid methods depends mainly on the number of smoothing steps on the finest grid rather than the specific multigrid scheme, because the main components in the error are only solvable on the finest grid.

The stopping criterion is when the relative residual error is below 10^{-10}. Table 3 presents the average number of iterations and evaluation time over 10 trials with randomly generated right-hand-side vectors. Our method demonstrates significant improvements over both the multigrid and PCG methods in terms of iteration number and computational efficiency. Notably, while the PCG method relies on a carefully constructed preconditioner, our approach does not necessarily require one, which makes our method more broadly applicable. We also visualize the magnitude spectrum of the errors during iterations for our proposed method and the multigrid method in Figure 7. We observe that the multigrid method fails to address geometric high-frequency components, while ours can solve all frequency components efficiently. Besides, the cost of our methods increases very mildly when the matrix size and condition number increase.

We also want to mention that when $n = 256, 512, 1024$, the number of elements in $A_{n^2}^{-1}$ is about $4\times10^9, 7\times10^{10}, 1\times10^{12}$, while the number of parameters in the FNO

	α	$n = 256$	$n = 512$	$n = 1024$
multigrid method	10^{-3}	608(1.2min)±3	611(4.9min)±2	611(21.1min)±1
	10^{-4}	5072(10min)±34	5096(41min)±18	6008 (3hr)±10
	10^{-5}	41207(2hr)±407	41220(8hr)±211	41228(33hr)±110
PCG method	10^{-3}	89(1.3s)±0.7	94.5(7.2s)±0.5	98(26.3s)±0.2
	10^{-4}	162.5(2.4s)±0.5	173.0(11.9s)±0.4	180(46.7s)±0.2
	10^{-5}	276(4.0s)±2.4	294(20.2s)±1.9	307(73.6s)±1.0
proposed method	10^{-3}	8.6(0.19s)±1.0	9.8(1.36s)±0.9	9.8(5.10s)±0.4
	10^{-4}	11.2(0.26s)±2.2	12.1(1.69s)±1.5	12.3(6.5s)±1.4
	10^{-5}	13.8(0.30s)±0.7	15.6(2.10s)±1.8	16.7(8.7s)±1.8

Table 3: The averaged number of iterations and evaluation time required for solving the two-dimensional integral equation.

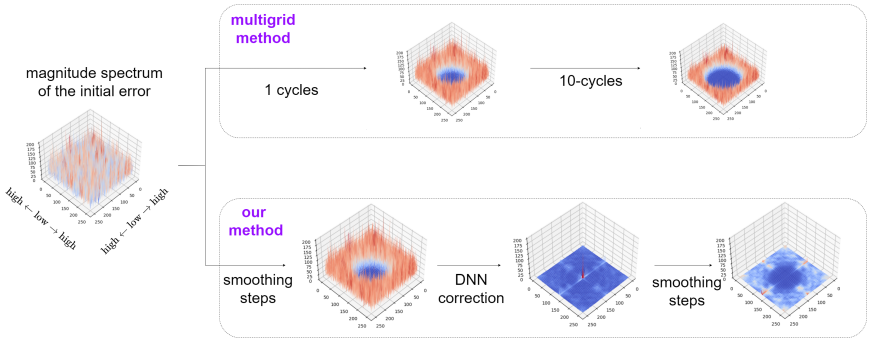


Fig. 7: The magnitude spectra from the proposed method and the multigrid method. In these figures, the x-y plane represents the 2-dimensional shifted frequency domain. On both x and y axises, the central part corresponds to the geometric low frequency and the side parts corresponds to the geometric high-frequency.

is about $5 \times 10^7, 2 \times 10^8, 2 \times 10^8$, respectively (the same network structure is used for the latter two cases). This support the motivation (mentioned in Section 2.4) of using a nonlinear network to approximate the linear inverse opeartor rather than a dense linear model. We did not spend much effort on optimizing the network structure, so the number of parameters may have room for further reduction.

5.4. Transferability of neural operators. In previous experiments, we independently trained distinct neural operators for each problem size n when solving the same equation. While this approach suffices for solving systems of the form $A_n\mathbf{x} = \mathbf{y}$ with fixed A_n and varying right-hand side y, it becomes computationally expensive when we need to solve an equation discretized at different n, because we need to retrain a new $\mathcal{N}_\theta$ for each n. Consequently, we investigate whether a neural operator trained for a small system can be easily generalized to larger systems discretized from the same equation. Thanks to the resolution-invariant structure of the FNO [29], $\mathcal{N}_\theta$ can directly take input vectors of different sizes. We consider the one-dimensional integral equation (5.1). For each value of $\alpha = 10^{-3}$, 10^{-4}, and 10^{-5}, we first apply the neural operator $\mathcal{N}_\theta$ trained on $n = 256$ directly to systems with $n = 512$ and $n = 1024$

	α	Directly apply	After fine-tunning
$n = 512$	10^{-3}	38.3(0.33s)±5.2	9.7(0.08s)±1.0
	10^{-4}	63.7(0.56s)±4.4	9.9(0.08s)±0.8
	10^{-5}	47.1(0.42s)±2.5	8.8(0.08s)±0.6
$n = 1024$	10^{-3}	51.1(0.60s)±3.6	12.1(0.15s)±1.6
	10^{-4}	113.6(1.32s)±6.9	14.2(0.17s)±4.4
	10^{-5}	79.0(0.92s)±10.4	10.8(0.13s)±0.7

Table 4: The average number of iterations and evaluation time when transferring models trained with $n = 256$ to $n = 512$ and $n = 1024$.

without any modification. Then, we fine-tune $\mathcal{N}_\theta$ on different n, i.e., continue to train $\mathcal{N}_\theta$ for 50 epochs on the new system, to see the improvements. The numerical results are presented in Table 4. We observe that directly applying $\mathcal{N}_\theta$ trained on small n to larger systems can still work, but it takes much more iterations to converge. However, the convergence performance improves significantly after only a few epochs of fine-tuning. This suggests that our neural operator retains transferability across different system sizes, requiring only minimal fine-tuning effort for effective adaptation.

5.5. Validating the convergence rate of the generalization error. In Theorem 4.2, we derive an analytical upper bound for training neural operators. By fixing the structure of neural operators, the expected L_2 error would converge at the order of $O(N^{-1/2})$ if we can find the global minimizer, *i.e.*, $\delta = 0$. Here, we numerically estimate the convergence rate and compare it with the theoretical result.

We consider solving the system $A_n = \alpha I_n + K_n$ (the same as Section 5.2.1) with $n = 32$ and $\alpha = 0.01$. Let $\mathcal{N}_\theta$ be a two-layer network defined as (4.1). We then randomly generate a finite set of training samples and train $\mathcal{N}_\theta$ by minimizing the empirical loss (4.2). Since our generalization error estimate in Theorem 4.2 works for general P_n, we set $P_n = I_n$ in the loss function for simplicity. Due to the existence of the optimization error δ, the actual convergence rate might be lower than the theoretical rate. To mitigate the effect of optimization error, we train $\mathcal{N}_\theta$ with 5 different random initializations and select the model with the minimum empirical loss. The testing error is shown in Table 5. Here, we choose the number of training samples $N_i = 10 \times 2^i$ for $i = 0, 1, \ldots, 4$ and computed the generalization error accordingly. We also estimated the convergence rate for $i = 1, 2, 3, 4$ by

$$\log\left(\frac{\text{error}(N_{i-1})}{\text{error}(N_i)}\right) \Big/ \log\left(\frac{N_{i-1}}{N_i}\right),$$

where $\text{error}(N_i)$ represents the generalization error when training on N_i samples, and shows the convergence rate (marked in parentheses for each N_i) as well. We see the convergence rate is close to the theoretical result, *i.e.*, -0.5, when N is small. When N is large, the optimization error will start to dominate the generalization error, so the observed convergence rate also decreases.

5.6. Validation of Assumption 1 in Theorem 4.4. We also validate the correcting condition assumed in Theorem 4.4. We consider the one-dimensional and two-dimensional integral equation with Tikhonov regularization, and calculate the

N_i	$N_1 = 10$	$N_2 = 20$	$N_3 = 40$	$N_4 = 80$	$N_5 = 160$
error	0.92	0.67(-0.46)	0.47(-0.51)	0.32(-0.55)	0.26(-0.30)

Table 5: Generalization error and estimated convergence rate (marked inside parentheses) with respect to different numbers of training samples.

ratio

$$\frac{\|\mathcal{N}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n}^2}{\|\mathcal{N}(\mathbf{r}) - A_n^{-1}\mathbf{r}\|_{A_n^2}^2/a_0} \tag{5.3}$$

for 100 randomly generated right-hand-sides $\mathbf{r}$. The results are shown in Table 6. In all scenarios, the ratios (5.3) are very stable, indicating that the correcting condition holds with high probability for some δ_2.

	α	$n = 256$	$n = 512$	$n = 1024$
$d = 1$	10^{-3}	$0.267 \pm 2.97 \times 10^{-5}$	$0.267 \pm 2.91 \times 10^{-5}$	$0.267 \pm 2.87 \times 10^{-7}$
	10^{-4}	$0.266 \pm 4.27 \times 10^{-6}$	$0.266 \pm 2.61 \times 10^{-7}$	$0.266 \pm 2.64 \times 10^{-8}$
	10^{-5}	$0.266 \pm 4.99 \times 10^{-6}$	$0.266 \pm 7.73 \times 10^{-6}$	$0.266 \pm 6.39 \times 10^{-5}$
$d = 2$	10^{-3}	$0.290 \pm 1.71 \times 10^{-3}$	$0.304 \pm 2.87 \times 10^{-3}$	$0.493 \pm 9.84 \times 10^{-3}$
	10^{-4}	$0.270 \pm 1.86 \times 10^{-4}$	$0.266 \pm 1.51 \times 10^{-5}$	$0.266 \pm 1.16 \times 10^{-6}$
	10^{-5}	$0.266 \pm 1.93 \times 10^{-6}$	$0.266 \pm 9.97 \times 10^{-6}$	$0.279 \pm 3.01 \times 10^{-4}$

Table 6: Validation of the correcting condition

6. Conclusion and future works. In this work, we combine classical iterative solvers and neural operators to solve large-scale linear systems arising from convolution-type integral equations. We propose a new training strategy for neural operators that can address algebraic low-frequency components efficiently. We also estimate the generalization error for our proposed training strategy with two-layer neural networks, providing an upper bound on the generalization error in terms of the number of training samples, problem size, and the optimization error. In the future, we will also try to estimate the generalization error for some general multi-layer networks like [35]. Another important direction is to tighten the current bound. The current generalization error bound is obtained using a global Rademacher-complexity argument, which may not be optimal. Techniques like local Rademacher complexity or covering-number techniques [38] may lead to sharper and possibly optimal generalization error estimates.

In the numerical experiments, our algorithm outperforms some classical numerical methods, such as the multigrid and the PCG method, by a huge margin in both iteration numbers and computational time. Our method has demonstrated the ability to address large-scale and ill-conditioned linear systems. Thanks to the highly parallel structures of neural networks, the computational time increases gradually when the problem size increases. We also show that the proposed method can be easily transferred to systems discretized at different sizes with mild cost.

Though we focus on a certain type of integral equations (the integral equation of the second kind), the proposed algorithm and training strategy can be generalized to other types of problems directly. We will consider applying the proposed method to

different integral equations, *e.g.*, the integral equation of the first kind, in the following works. One limitation of our proposed method is the requirement for training, which takes a much longer time than evaluation. When the underlying system changes frequently, we need to retrain our neural operators many times. To address this issue, we will try to design multiple-input neural operators, like the Meta-MgNet [15], that can solve a class of linear systems instead of one specific system.

REFERENCES

[1] G. S. Ammar and W. B. Gragg, *Superfast solution of real positive definite Toeplitz systems*, SIAM Journal on Matrix Analysis and Applications, 9 (1988), pp. 61–76.

[2] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 183–202.

[3] J.-M. Bergheau and R. Fortunier, *Finite element simulation of heat transfer*, John Wiley & Sons, 2013.

[4] R. R. Bitmead and B. D. Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra and its Applications, 34 (1980), pp. 103–116.

[5] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, 2000.

[6] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, and Q. Gu, *Towards understanding the spectral bias of deep learning*, in 30th International Joint Conference on Artificial Intelligence (IJCAI 2021), International Joint Conferences on Artificial Intelligence, 2021, pp. 2205–2211.

[7] R. Chan, T. F. Chan, and W. Wan, *Multigrid for differential-convolution problems arising in image processing*, in 6th Workshop on Scientific Computing, Springer-Verlag, 1997, pp. 58–72.

[8] R. H. Chan, *The spectrum of a family of circulant preconditioned Toeplitz systems*, SIAM Journal on Numerical Analysis, 26 (1989), pp. 503–506.

[9] R. H. Chan and T. F. Chan, *Circulant preconditioners for elliptic problems*, Department of Mathematics, University of California, Los Angeles, 1990.

[10] R. H. Chan and X.-Q. Jin, *A family of block preconditioners for block systems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 1218–1235.

[11] R. H. Chan and M. K. Ng, *Conjugate gradient methods for Toeplitz systems*, SIAM Review, 38 (1996), pp. 427–482.

[12] R. H. Chan and M.-C. Yeung, *Circulant preconditioners constructed from kernels*, SIAM Journal on Numerical Analysis, 29 (1992), pp. 1093–1103.

[13] T. F. Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 766–771.

[14] T. F. Chan and J. A. Olkin, *Circulant preconditioners for Toeplitz-block matrices*, Numerical Algorithms, 6 (1994), pp. 89–101.

[15] Y. Chen, B. Dong, and J. Xu, *Meta-mgnet: Meta multigrid networks for solving parameterized partial differential equations*, Journal of Computational Physics, 455 (2022), p. 110996.

[16] C. Chui and A. Chan, *Application of approximation theory methods to recursive digital filter design*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 30 (1982), pp. 18–24.

[17] G. H. Golub and C. F. Van Loan, *Matrix computations*, JHU press, 2013.

[18] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis, *A physics-informed variational deeponet for predicting crack path in quasi-brittle materials*, Computer Methods in Applied Mechanics and Engineering, 391 (2022), p. 114587.

[19] U. Grenander, *Toeplitz forms and their applications*, California Monographs in Mathematical Sciences/University of California Press, (1958).

[20] J. He, X. Liu, and J. Xu, *Mgno: Efficient parameterization of linear operators via multigrid*, in 12th International Conference on Learning Representations, ICLR 2024, 2024.

[21] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Procedings of IEEE/CVF Computer Vision and Pattern Recognition Conference, 770–778, 2016.

[22] P. W. Hemker and P. M. de Zeeuw, *Some implementations of multigrid linear system solvers*, Stichting Mathematisch Centrum, 1984.

[23] J. Hu and P. Jin, *A hybrid iterative method based on MIONet for pdes: Theory and numerical examples*, arXiv preprint arXiv:2402.07156, (2024).

[24] R. Huang, R. Li, and Y. Xi, *Learning optimal multigrid smoothers via neural networks*, SIAM Journal on Scientific Computing, 45 (2022), pp. S199–S225.

[25] R. King, M. Ahmadi, R. Gorgui-Naguib, A. Kwabwe, and M. Azimi-Sadjadi, *Digital filtering in one and two dimensions: design and applications*, Springer, 1989.

[26] S. M. Kwerel, *Bounds on the probability of the union and intersection of m events*, Advances in Applied Probability, 7 (1975), pp. 431–448.

[27] M. Ledoux and M. Talagrand, *Probability in Banach Spaces: Isoperimetry and processes*, Springer Science & Business Media, 2013.

[28] L. Li, X.-C. Tai, J. Yang, and Q. Zhu, *A priori error estimate of deep mixed residual method for elliptic PDEs*, Journal of Scientific Computing, 98 (2024), p. 44.

[29] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Fourier neural operator for parametric partial differential equations*, in International Conference on Learning Representations, 2021, https://openreview.net/forum?id=c8P9NQVtmnO.

[30] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, *Physics-informed neural operator for learning partial differential equations*, ACM/JMS Journal of Data Science, 1 (2024), https://doi.org/10.1145/3648506, https://doi.org/10.1145/3648506.

[31] Z. Lin, *Probability Inequalities*, Springer, 2010.

[32] S. Liu, C. Su, J. Yao, Z. Hao, H. Su, Y. Wu, and J. Zhu, *Preconditioning for physics-informed neural networks*, arXiv preprint arXiv:2402.00531, (2024).

[33] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*, Nature Machine Intelligence, 3 (2021), pp. 218–229.

[34] Y. Lu, J. Lu, and M. Wang, *A priori generalization analysis of the deep ritz method for solving high dimensional elliptic partial differential equations*, in Conference on learning theory, PMLR, 2021, pp. 3196–3241.

[35] I. Ohn and Y. Kim, *Smooth function approximation by deep neural networks with general activation functions*, Entropy, 21 (2019), p. 627.

[36] P. Polyanin and A. V. Manzhirov, *Handbook of integral equations*, Chapman and Hall/CRC, 2008.

[37] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, *On the spectral bias of neural networks*, in International conference on machine learning, PMLR, 2019, pp. 5301–5310.

[38] A. Rakhlin, K. Sridharan, and A. B. Tsybakov, *Empirical entropy, minimax regret and minimax risk*, Bernoulli, 23 (2017), p. 789–824.

[39] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional networks for biomedical image segmentation*, in Procedings of Medical Image Computing and Computer Assisted Intervention, 234–241, 2015.

[40] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, SIAM, 1987, pp. 73–130.

[41] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[42] S. Song, T. Mukerji, and D. Zhang, *Physics-informed multi-grid neural operator: theory and an application to porous flow simulation*, Journal of Computational Physics, 520 (2025), p. 113438.

[43] H.-W. Sun, R. H. Chan, and Q.-S. Chang, *A note on the convergence of the two-grid method for Toeplitz systems*, Computers & Mathematics with Applications, 34 (1997), pp. 11–18.

[44] E. E. Tyrtyshnikov, *Optimal and superoptimal circulant preconditioners*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 459–473.

[45] S. Wang, H. Wang, and P. Perdikaris, *Learning the solution operator of parametric partial differential equations with physics-informed DeepONets*, Science Advances, 7 (2021), p. eabi8605.

[46] E. Weinan, C. Ma, and L. Wu, *A priori estimates of the population risk for two-layer neural networks*, Communications in Mathematical Sciences, 17 (2019), pp. 1407–1425.

[47] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numerica, 26 (2017), pp. 591–721.

[48] Z.-Q. J. Xu, *Frequency principle: Fourier analysis sheds light on deep neural networks*, Communications in Computational Physics, 28 (2020), pp. 1746–1767.

[49] Y. Yang and J. He, *Deep neural networks with general activations: Super-convergence in sobolev norms*, arXiv preprint arXiv:2508.05141, (2026), https://doi.org/10.48550/arXiv.2508.05141, http://arxiv.org/abs/2508.05141. arXiv:2508.05141 [cs.LG].

[50] E. Zhang, A. Kahana, E. Turkel, R. Ranade, J. Pathak, and G. E. Karniadakis, *A hybrid iterative numerical transferable solver (HINTS) for PDEs based on deep operator network and relaxation methods*, arXiv preprint arXiv:2208.13273, (2022).

[51] E. Zhang, A. Kahana, E. Turkel, R. Ranade, J. Pathak, and G. E. Karniadakis, *Blending neural operators and relaxation methods in PDE numerical solvers*, Nature Machine Intelligence, (2024), https://doi.org/10.1038/s42256-024-00910-x.
[52] S. Zhang, J. Lu, and H. Zhao, *Deep network approximation: Beyond relu to diverse activation functions*, Journal of Machine Learning Research, 25 (2024), p. 1687–1725.

Appendix A. Rademacher complexity and generalization error. In this section, we introduce the definition of the Rademacher complexity and how to estimate the generalization error using this complexity.

Definition A.1. *Let $\{\mathbf{x}_i \in \mathbb{R}^n\}_{i=1}^N$ be a set of i.i.d. random variables drawn from certain distribution $\mathcal{X}$ and $\{\varepsilon_i\}_{i=1}^N$ be a set of i.i.d Rademacher variables (i.e., only taking value of 1 and -1 with equal probability). Then the* ***empirical Rademacher Complexity*** *of the function class $\mathcal{F}$ is defined as*

$$\hat{R}_N(\mathcal{F}) := \mathbb{E}_\varepsilon \left(\sup_{f\in\mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^N \varepsilon_i f(\mathbf{x}_i) \right) \right).$$

Taking its expectation over $\mathcal{X}$ yields the ***Rademacher Complexity*** *of the function class $\mathcal{F}$*

$$R_N(\mathcal{F}) := \mathbb{E}_{\mathcal{X}} \left(\hat{R}_N(\mathcal{F}) \right) = \mathbb{E}_{\mathcal{X}} \mathbb{E}_\varepsilon \left(\sup_{f\in\mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^N \varepsilon_i f(\mathbf{x}_i) \right) \right).$$

Remark A.2. In the rest of this section, we assume that all $\mathbf{x}_i$ sampled from $\mathcal{X}$ are uniformly bounded: $\|\mathbf{x}_i\|_2 \le 1$.

Some calculation rules of the Rademecher complexity are given below

Lemma A.3. *Let $\mathcal{F}, \mathcal{G}$ be function classes and a, b be constants. Then*

1. $R_N(\mathcal{F}) \le R_N(\mathcal{G})$ *if* $\mathcal{F} \subseteq \mathcal{G}$
2. $R_N(\mathcal{F} + \mathcal{G}) = R_N(\mathcal{F}) + R_N(\mathcal{G})$.
3. $R_N(a\mathcal{F}) = |a| R_N(\mathcal{F})$.
4. *(contraction lemma [27]) Assume that $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is a L-Lipschitz function, then $R_N(\sigma(\mathcal{F})) \le L R_N(\mathcal{F})$.*
5. *If $\|f\|_\infty$ is uniformly bounded for any $f \in \mathcal{F}$, then $R_N(\mathcal{F}^2) \le 2\|f\|_\infty R_N(\mathcal{F})$ where $\mathcal{F}^2 := \{f(\mathbf{x})^2 | f \in \mathcal{F}\}$.*
6. *If $\|f\|_\infty$ is uniformly bounded for any $f \in \mathcal{F}$, then $R_N(\mathcal{F}\times\mathcal{F}) \le 6\|f\|_\infty R_N(\mathcal{F})$ where $\mathcal{F} \times \mathcal{F} := \{f_1(\mathbf{x}) f_2(\mathbf{x}) | f_1, f_2 \in \mathcal{F}\}$.*

Proof. The first 3 computation rules can be directly derived from the definition. The proof of the contraction lemma can be found in [41, Lemma 26.9]. Rule 5 is a direct application of the contraction lemma. Though $\sigma(x) = (x)^2$ is not a Lipschitz function when $x \in \mathbb{R}$, its derivative $2x$ is bounded when x is bounded. Since $\sup_{f\in\mathcal{F}} \|f\|_\infty$ is assumed to be finite, we can apply the contraction lemma to estimate $R_N(\mathcal{F}^2)$ directly. What remains is to prove property 6. For any $f_1 f_2$, we can write it as $f_1 f_2 = \frac{1}{2}(f_1 + f_2)^2 - \frac{f_1^2}{2} - \frac{f_2^2}{2}$, so we have $\mathcal{F}\times\mathcal{F} \subseteq \frac{1}{2}(\mathcal{F}+\mathcal{F})^2 - \frac{1}{2}\mathcal{F}^2 - \frac{1}{2}\mathcal{F}^2$. Then, applying calculation rules 1 to 5, we have

$$R_N(\mathcal{F}\times\mathcal{F}) \le R_N\left(\frac{1}{2}(\mathcal{F}+\mathcal{F})^2 - \frac{1}{2}\mathcal{F}^2 - \frac{1}{2}\mathcal{F}^2\right) \le R_N\left(\frac{1}{2}(\mathcal{F}+\mathcal{F})^2\right) + R_N(\mathcal{F}^2)$$

$$\le \frac{1}{2} 2(\sup_{f\in\mathcal{F}} \|f\|_\infty + \sup_{f\in\mathcal{F}} \|f\|_\infty)(2R_N(\mathcal{F})) + 2\sup_{f\in\mathcal{F}} \|f\|_\infty R_N(\mathcal{F}) = 6\sup_{f\in\mathcal{F}} \|f\|_\infty R_N(\mathcal{F}).$$

The same technique is used in the proof of [28, Lemma 4.4]. ∎

Lemma A.4 ([41, Lemma 26.10]) gives an upper bound for the Rademacher complexity for a class of linear functions.

LEMMA A.4. *Let $\mathcal{G}$ be the linear transformation function class defined by*

$$\mathcal{G} := \{g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} | \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq C\},$$

where $C > 0$. Then we have $R_N(\mathcal{G}) \leq \frac{\sup_{\mathbf{x}\sim\mathcal{X}} \|\mathbf{x}\|_2 C}{\sqrt{N}}$.

Using Lemma A.4, we can estimate the Rademacher complexity of the composition of a ReLU function and linear transformations.

LEMMA A.5. *Let $\mathcal{G}$ be defined by*

$$\mathcal{G} := \{g(\mathbf{x}) = a\mathrm{ReLU}(\mathbf{w}^\top \mathbf{x}) | \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq 1, |a| \leq C\},$$

where $C > 0$. Then we have $R_N(\mathcal{G}) \leq \frac{\sup_{\mathbf{x}\sim\mathcal{X}} \|\mathbf{x}\|_2 C}{\sqrt{N}}$.

Proof. Let $\{\mathbf{x}_i\}_{i=1}^N$ be a set of randomly generated vectors. In Definition A.1, we have

$$\sup_{g\in\mathcal{G}} \left(\frac{1}{N} \sum_{i=1}^N \epsilon_i g(\mathbf{x}_i) \right) = \sup_{a\in[-C,C], \|\mathbf{w}\|_2 \leq 1} \left(\frac{a}{N} \sum_{i=1}^N \epsilon_i \mathrm{ReLU}(\mathbf{w}^\top \mathbf{x}_i) \right)$$

$$\leq C \sup_{\|\mathbf{w}\|_2 \leq 1} \left(\frac{1}{N} \sum_{i=1}^N \epsilon_i \mathrm{ReLU}(\mathbf{w}^\top \mathbf{x}_i) \right),$$

which implies that $R_N(\mathcal{G}) \leq C R_N(\{\mathrm{ReLU}(\mathbf{w}^\top \mathbf{x}) | \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq 1\})$. Then, using the Lipschitz continuity of the ReLU function and the contraction lemma (Lemma A.3(3)), we get

$$R_N(\mathcal{G}) \leq C R_N(\{\mathbf{w}^\top \mathbf{x} | \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq 1\}) \leq \frac{C \sup_{\mathbf{x}} \|\mathbf{x}\|_2}{\sqrt{N}},$$

where the last inequality is derived from Lemma A.4. ∎

Let $l : \mathbb{R} \to \mathbb{R}$ be a given continuous function and $\mathcal{L} := \{l(f(\mathbf{x})) | f \in \mathcal{F}\}$ be a function class. In machine learning problems, l usually represents the loss function that measures the difference between the model f and a given ground truth label. For any $f \in \mathcal{F}$, we define

$$L_{\mathcal{X}}(f) := \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}(l(f(\mathbf{x}))) \text{ and } L_N(f) := \frac{1}{N} \sum_{i=1}^N (l(f(\mathbf{x}_i))). \tag{A.1}$$

The difference $|L_N(f) - L_{\mathcal{X}}(f)|^2$ is called the generalization error, which can be bounded by the Rademacher complexity using Lemma A.6 ([41, Theorem 26.5]).

LEMMA A.6. *Let $\mathcal{F}$ be a set of functions, $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ be i.i.d. random variables sampled from $\mathcal{X}$. Then, for any $\epsilon > 0$ and $f \in \mathcal{F}$, the following inequality holds with probability at least $1 - \epsilon$,*

$$L_{\mathcal{X}}(f) - L_N(f) \leq 2R_N(l \circ \mathcal{F}) + \sup_{l(f)\in l\circ\mathcal{F}} \|l(f)\|_\infty \sqrt{\frac{2\log(2/\epsilon)}{N}}, \tag{A.2}$$

where $l \circ \mathcal{F} := \{l(f) | f \in \mathcal{F}\}$.

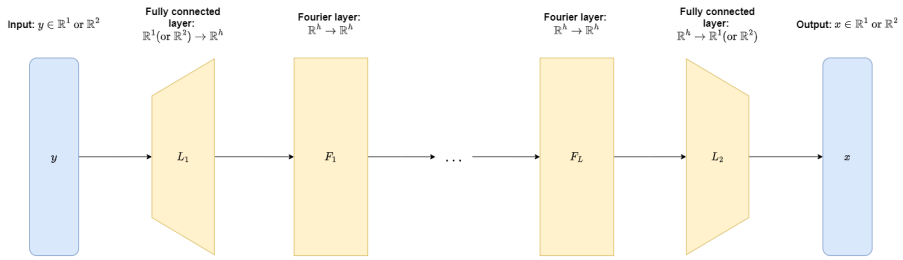


Fig. 8: The structure of Fourier neural operators

Appendix B. Network structures. In Section 5, we adopt the Fourier neural operators (FNO) [29]. The general structure is shown in Figure 8. Here, the first fully connected layer L_1 aims to lift the input dimension to $\mathbb{R}^h$ where h is the hidden dimension specified by users. The last fully connected layer L_2 aims to change the output dimension back to $\mathbb{R}^1$ or $\mathbb{R}^2$. The Fourier layer is introduced in [29] and is defined as $\sigma(\text{FFT}^{-1} \circ R_l \circ \text{FFT}(\mathbf{z}) + W_l\mathbf{z} + \mathbf{b})$, $l = 1, \ldots, L$, where σ is a nonlinear activation function, R_l is a complex linear transformation applied only to the first k low-frequency components in each dimension, W_l is a linear transformation, and $\mathbf{b}$ is a bias vector. More details of the implementation of the FNO can be found in [29]. Here, W_l is implemented as a convolution operator with learnable convolutional kernels. The hyperparameters of networks used for Section 5 are listed in Table 7.

problem dimension	problem size	h	L	k	kernel size
$d = 1$	$n = 256$	64	6	64	7
	$n = 512$	64	6	128	7
	$n = 1024$	64	6	128	7
$d = 2$	$n = 256$	64	5	32	5
	$n = 512$	64	6	64	7
	$n = 1024$	64	6	64	7

Table 7: Hyperparameters in neural operators

Appendix C. Table 8: Condition number of linear systems used in Section 5.

	α	$n = 256$	$n = 512$	$n = 1024$
Section 5.2.1	10^{-3}	971	972	972
	10^{-4}	7678	7683	7685
	10^{-5}	24849	24904	24918
Section 5.2.2	10^{-3}	488	490	492
	10^{-4}	4225	4286	4316
	10^{-5}	19159	19546	19746
Section 5.3	10^{-3}	999.7	999.9	1000.0
	10^{-4}	9996.5	9999.1	9999.7
	10^{-5}	99957.2	99982.5	99988.8

Table 8: Condition number of linear systems corresponding to different choices of α and n in Section 5