mathematics

Article

Fourier Neural Solver for Large Sparse Linear Algebraic Systems

Chen Cui , Kai Jiang *, Yun Liu and Shi Shu *

Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Key Laboratory of Intelligent Computing and Information Processing of Ministry of Education, School of Mathematics and Computational Science, Xiangtan University, Xiangtan 411105, China
* Correspondence: kaijiang@xtu.edu.cn (K.J.); shushi@xtu.edu.cn (S.S.)

Abstract: Large sparse linear algebraic systems can be found in a variety of scientific and engineering fields and many scientists strive to solve them in an efficient and robust manner. In this paper, we propose an interpretable neural solver, the Fourier neural solver (FNS), to address them. FNS is based on deep learning and a fast Fourier transform. Because the error between the iterative solution and the ground truth involves a wide range of frequency modes, the FNS combines a stationary iterative method and frequency space correction to eliminate different components of the error. Local Fourier analysis shows that the FNS can pick up on the error components in frequency space that are challenging to eliminate with stationary methods. Numerical experiments on the anisotropic diffusion equation, convection–diffusion equation, and Helmholtz equation show that the FNS is more efficient and more robust than the state-of-the-art neural solver.

Keywords: Fourier neural solver; fast Fourier transform; local Fourier analysis; convection–diffusion–reaction equation

MSC: 65F10; 65N22; 68T07; 35Q68

check for updates

Citation: Cui, C.; Jiang, K.; Liu, Y.; Shu, S. Fourier Neural Solver for Large Sparse Linear Algebraic Systems. *Mathematics* **2022**, *10*, 4014. https://doi.org/10.3390/math10214014

Academic Editor: Luigi Rodino

Received: 23 September 2022
Accepted: 25 October 2022
Published: 28 October 2022

1. Introduction

Large sparse linear algebraic systems are ubiquitous in scientific and engineering computation, such as discretization of partial differential equations (PDE) and linearization of non-linear problems. Designing efficient, robust, and adaptive numerical methods for solving them is a long-term challenge. Iterative methods are an effective way to resolve this issue. They can be classified into single-level and multi-level methods. There are two types of single-level methods: stationary and non-stationary [1]. Due to sluggish convergence, stationary methods, such as weighted Jacobi, Gauss–Seidel and successive over-relaxation methods [2] are frequently utilized as smoothers in multi-level approaches or as preconditioners. Non-stationary methods typically refer to Krylov subspace methods, such as conjugate gradient (CG) and generalized minimal residual (GMRES) methods [3,4], whose convergence rate is heavily influenced by certain factors, such as the initial value. Multi-level methods mainly comprise the geometric multigrid (GMG) method [5–7] and the algebraic multigrid (AMG) method [8,9]. They are both affected by many factors, such as smoother and coarse grid correction, which heavily affect convergence. Identifying these factors for a concrete problem is an art that requires extensive analysis, innovation, and trial.

In recent years, the technique of automatically picking parameters for Krylov and multi-level methods and constructing a learnable iterative scheme based on deep learning has attracted much interest. Many neural solvers have achieved satisfactory results for second-order elliptic equations with smooth coefficients. Hsieh et al. [10] utilized a convolutional neural network (CNN) to accelerate convergence of the Jacobi method. Luna et al. [11] accelerated the convergence of GMRES with a learned initial value. Zhang et al. [12] combined standard relaxation methods and the DeepONet [13] to target distinct regions in the spectrum of eigenmodes. Significant efforts have also been made in the development of multigrid solvers, such as the learning smoother, the transfer operator [14,15] and coarse-fine splitting [16].

Huang et al. [17] exploited a CNN to design a more sensible smoother for anisotropy elliptic equations. The results showed that the magnitude of the learned smoother was dispersed along the anisotropic direction. Wang et al. [18] introduced a learning-based local weighted least square method for the AMG interpolation operator and applied it to random diffusion equations and one-dimensional small wavenumber Helmholtz equations. Fanaskov [19] produced the learned smoother and transfer operator of GMG in a neural network form.

When the anisotropic strength is mild (within two orders of magnitude), the studies referred to evidence considerable acceleration. Chen et al. [20] proposed the Meta-MgNet to learn a basis vector of Krylov subspace as the smoother of GMG for strong anisotropic cases. However, the convergence rate was still sensitive to the anisotropic strength. For convection–diffusion equations, Katrutsa et al. [21] trained the weighted Jacobi smoother and transfer operator of GMG, which had a positive effect on the upwind discretization system and was also applied to solve a one-dimensional Helmholtz equation. For second-order elliptic equations with random diffusion coefficients, Greenfeld et al. [22] employed a residual network to construct the prolongation operator of AMG for uniform grids. Luz et al. [23] extended it to non-uniform grids using graph neural networks, which outperformed classical AMG methods. For jumping coefficient problems, Antonietti et al. [24] presented a neural network to forecast the strong connection parameter to speed up AMG and used it as a preconditioner for CG. For the Helmholtz equation, Stanziola et al. [25] constructed a fully learnable neural solver, the helmnet, which was built on U-net and a recurrent neural network [26]. Azulay et al. [27] developed a preconditioner based on U-net and shift-Laplacian MG [28] and applied the flexible GMRES [29] to solve the discrete system. For solid and fluid mechanics equations, several neural solver methods for associated discrete systems have been proposed, such as, but not limited to, learning initial values [30,31], constructing preconditioners [32], learning the search directions of CG [33], and learning the parameters of GMG [34,35].

In this paper, we propose a Fourier neural solver (FNS), a deep learning and fast Fourier transform (FFT)-based [36] neural solver. The FNS is made up of two modules: a stationary method and a frequency space correction. Since stationary methods, such as the weighted Jacobi method, have difficulty eliminating low-frequency error, the FNS uses FFT and CNN to learn these modes in the frequency space. Local Fourier analysis (LFA) [5] has shown that the FNS can pick up on the error components in frequency space that are challenging to eradicate using stationary methods. The FNS builds a complementary relationship with the stationary method and CNN to eliminate error. With the help of FFT, the single-step iteration of the FNS has a $O(N \log_2 N)$ computational complexity. All matrix-vector products are implemented using convolution, which is both storage-efficient and straightforward to parallelize. We investigated the effectiveness and robustness of the FNS on three types of convection–diffusion–reaction equations. For anisotropic diffusion equations, numerical experiments showed that the FNS was able to reduce the number of iterations by nearly 10-times compared to the state-of-the-art Meta-MgNet when the anisotropic strength was relatively strong. For non-symmetric systems arising from the convection–diffusion equation discretized by the central difference method, the FNS can converge, while MG and CG methods diverge. In addition, FNS is faster than other algorithms, such as GMRES and BiCGSTAB(ℓ) [37]. For indefinite systems arising from the Helmholtz equation, the FNS outperforms GMRES and BiCGSTAB for medium wavenumbers. In this paper, we apply the FNS to the above three PDE systems. However, the principles employed by the FNS indicate that the FNS has the potential to be useful for a broad range of sparse linear algebraic systems.

The remainder of this paper is organized as follows: Section 2 proposes a general form of linear convection–diffusion–reaction equation and describes the motivation for designing the FNS. Section 3 presents the FNS algorithm. Section 4 examines the performance of the FNS with respect to anisotropy, convection–diffusion, and the Helmholtz equations. Finally, Section 5 describes the conclusions and potential future work.

2. Motivation

We consider the general linear convection–diffusion–reaction equation with a Dirichlet boundary condition

$$\begin{cases} -\varepsilon\nabla\cdot(\boldsymbol{\alpha}(x)\nabla u)+\nabla\cdot(\boldsymbol{\beta}(x)u)+\gamma u=f(x) & \text{in } \Omega \\ u(x)=g(x) & \text{on } \partial\Omega \end{cases} \tag{1}$$

where $\Omega\subseteq\mathbb{R}^d$ is an open and bounded domain. $\boldsymbol{\alpha}(x)$ is the $d\times d-$ order diffusion coefficient matrix. $\boldsymbol{\beta}(x)$ is the $d\times 1$ velocity field that the quantity is moving with. γ is the reaction coefficient. f is the source term.

We can obtain a linear algebraic system once we discretize Equation (13) by the finite element method (FEM) or finite difference method (FDM)

$$\mathbf{A}\mathbf{u}=\mathbf{f}, \tag{2}$$

where $\mathbf{A}\in\mathbb{R}^{N\times N}$, $\mathbf{f}\in\mathbb{R}^N$ and N is the spatial discrete degrees of freedom.

Classical stationary iterative methods, such as Gauss–Seidel and weighted Jacobi methods, have the generic form

$$\mathbf{u}^{k+1}=\mathbf{u}^k+\mathbf{B}\left(\mathbf{f}-\mathbf{A}u^k\right), \tag{3}$$

where $\mathbf{B}$ is an easily computed operator, such as the inverse of the diagonal matrix (Jacobi method) or the inverse of the lower triangle matrix (Gauss–Seidel method). However, the convergence rate of such methods is relatively low. As an example, we utilize the weighted Jacobi method to solve a special case of Equation (1) and use LFA to analyze the rationale.

Taking $\varepsilon=1$, $\boldsymbol{\alpha}(x)=\begin{pmatrix}1&0\\0&1\end{pmatrix}$, $\boldsymbol{\beta}(x)=\begin{pmatrix}0\\0\end{pmatrix}$ and $\gamma=0$, Equation (1) becomes the Poisson equation. With a linear FEM discretization, in stencil notation, the resulting discrete operator reads

$$\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}. \tag{4}$$

In the weighted Jacobi method, $\mathbf{B}=\omega\mathbf{I}/4$, where $\omega\in(0,1]$ and $\mathbf{I}$ is the identity matrix. Equation (3) can be written in the pointwise form

$$u_{ij}^{k+1}=u_{ij}^k+\frac{\omega}{4}\left(f_{ij}-(4u_{ij}^k-u_{i-1,j}^k-u_{i+1,j}^k-u_{i,j-1}^k-u_{i,j+1}^k)\right). \tag{5}$$

Let u_{ij} be the true solution and define error $e_{ij}^k=u_{ij}-u_{ij}^k$. Then, we have

$$e_{ij}^{k+1}=e_{ij}^k-\frac{\omega}{4}(4e_{ij}^k-e_{i-1,j}^k-e_{i+1,j}^k-e_{i,j-1}^k-e_{i,j+1}^k). \tag{6}$$

Expanding error in a Fourier series $e_{ij}^k=\sum_{p_1,p_2}v^k e^{i2\pi\left(p_1x_i+p_2y_j\right)}$, substituting the general term $v^k e^{i2\pi\left(p_1x_i+p_2y_j\right)}$, $p_1,p_2\in[-N/2,N/2)$ into Equation (6), we have

$$\begin{aligned} v^{k+1}&=v^k\left(1-\frac{\omega}{4}\left(4-e^{-i2\pi p_1h}-e^{i2\pi p_1h}-e^{-i2\pi p_2h}-e^{-i2\pi p_2h}\right)\right)\\ &=v^k\left(1-\frac{\omega}{4}(4-2\cos(2\pi p_1h)-2\cos(2\pi p_2h))\right). \end{aligned}$$

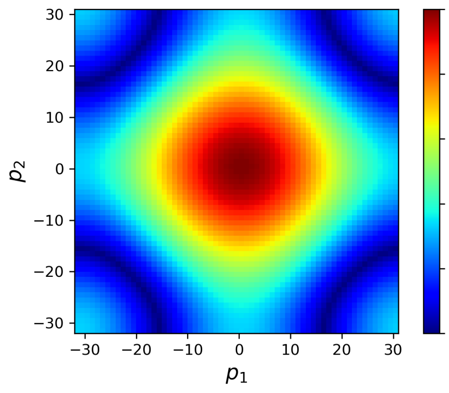
The convergence factor of the weighted Jacobi method (also known as the smoother factor in the MG framework [7]) is

$$\mu_{\text{loc}} := \left|\frac{v^{k+1}}{v^k}\right| = \left|1 - \omega + \frac{\omega}{2}(\cos(2\pi p_1 h) + \cos(2\pi p_2 h))\right|. \tag{7}$$

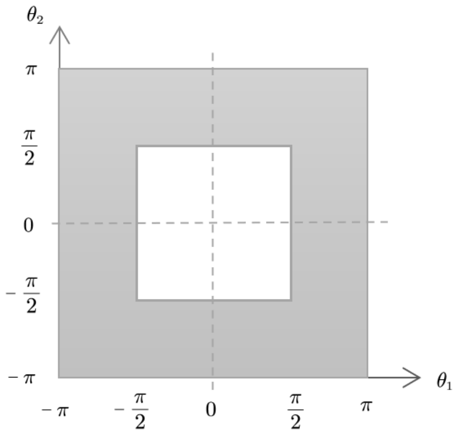
Figure 1a shows the distribution of the convergence factor μ_{loc} of weighted Jacobi ($\omega = 2/3$) in solving a linear system for the Poisson equation. For a better understanding, let $\theta_1 = 2\pi p_1 h$, $\theta_2 = 2\pi p_2 h$, $\boldsymbol{\theta} = (\theta_1, \theta_2) \in [-\pi, \pi)^2$. Define the high and low-frequency regions

$$\begin{aligned} T^{\text{low}} &:= \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2, \\ T^{\text{high}} &:= [-\pi, \pi)^2 \backslash \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2, \end{aligned} \tag{8}$$

as shown in Figure 1b. It can be seen that, in the high-frequency region, μ_{loc} is approximately zero, whereas, in the low-frequency region, μ_{loc} is close to one. As a result, the weighted Jacobi method attenuates high-frequency errors quickly but is mostly ineffective for low-frequency errors.



(**a**) Distribution of convergence factor μ_{loc}



(**b**) High- and low-frequency regions

Figure 1. (**a**) Distribution of the convergence factor μ_{loc} of the weighted Jacobi method ($\omega = 2/3$) in solving a linear system arising from the Poisson equation; (**b**) Low-frequency (white) and high-frequency (gray) regions.

The explanation has two aspects. Firstly, the high-frequency reflects the local oscillation, while the low-frequency reflects the global pattern. Since $\mathbf{A}$ is sparse and the basic operation $\mathbf{Au}$ of the weighted Jacobi method is a local operation, it is challenging to remove low-frequency global error components. Secondly, $\mathbf{A}$ is sparse and $\mathbf{A}^{-1}$ is commonly dense, which means that the mapping $\mathbf{f} \to \mathbf{A}^{-1}\mathbf{f}$ is non-local, making local operations difficult to approximate.

Therefore, we should seek the solution in another space to obtain an effective approximation of the non-local mapping. For example, the Krylov method approximates the solution in a subspace spanned by a basis set. The MG generates a coarse space to broaden the receptive field of the local operation. However, as mentioned in Section 1, these methods require the careful design of various parameters. In this paper, we propose the FNS, a generic solver that uses FFT to learn solutions in frequency space, with the parameters obtained automatically in a data-driven manner.

3. Fourier Neural Solver

Denote stationary iterative methods of (3) in an operator form

$$\mathbf{v}^{k+1} = \Phi(\mathbf{u}^k), \tag{9}$$

and the $k-$th step residual

$$\mathbf{r}^k := \mathbf{f} - \mathbf{A}\mathbf{v}^{k+1}, \tag{10}$$

then the $k-$th step error $\mathbf{e}^k := \mathbf{u} - \mathbf{v}^{k+1}$ satisfies residual equation

$$\mathbf{A}\mathbf{e}^k = \mathbf{r}^k. \tag{11}$$

As shown in the preceding section, the slow convergence rate of stationary methods is due to the difficulty of reducing low-frequency errors. Even high-frequency errors might not be effectively eliminated by Φ in many cases. We employ stationary methods to rapidly erase some components of the error and use FFT to convert the remaining error components to frequency space. The resulting solver is the Fourier neural solver.

Figure 2 shows a flowchart of the $k-$th step for the FNS. The module for solving the residual equation in frequency space is denoted as $\mathcal{H}$. It consists of three steps: FFT→Hadamard product→IFFT. The parameter $\boldsymbol{\vartheta}$ of $\mathcal{H}$ is the output of the hyper-neural network (HyperNN). The input $\boldsymbol{\eta}$ of the HyperNN are the PDE parameters corresponding to the discrete systems. During training, the only parameter $\boldsymbol{\theta}$ of the HyperNN serves as the optimization parameter.

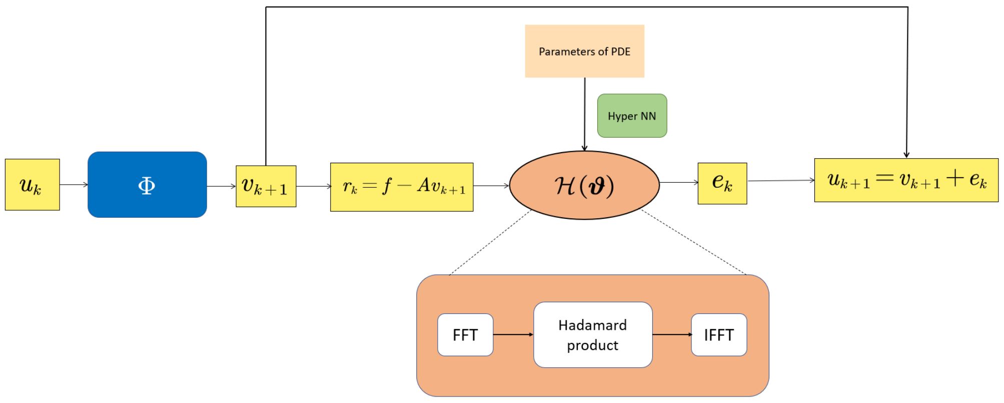


Figure 2. FNS calculation flowchart.

The three-step operation of $\mathcal{H}$ was inspired by the fast Poisson solver [38]. Let eigenvalues and eigenvectors of $\mathbf{A}$ be $\lambda_1, \dots, \lambda_N$ and $\mathbf{q}_1, \dots, \mathbf{q}_N$, respectively. Solving Equation (2) entails three steps:

1. Expand $\mathbf{f}$ as a combination $\mathbf{f} = a_1\mathbf{q}_1 + \cdots + a_N\mathbf{q}_N$ of the eigenvectors
2. Divide each a_k by λ_k
3. Recombine eigenvectors into $\mathbf{u} = (a_1/\lambda_1)\mathbf{q}_1 + \cdots + (a_N/\lambda_N)\mathbf{q}_N$.

In particular, when $\mathbf{A}$ is the system generated by a five-point stencil (4), its eigenvector $\mathbf{q}_k$ is the sine function. The first and third steps above can now be performed with a computational complexity of $O(N \log_2 N)$ using fast sine transform (based on the FFT). The computational complexity of each iteration of the FNS is $O(N \log_2 N)$.

It is worth noting that, although Φ can smooth some components of the error, the components that are removed are indeterminate. As a result, instead of filtering high-frequency modes in frequency space, $\mathcal{H}$ learns the error components that Φ cannot easily eliminate. For Φ with a fixed stencil, we can use LFA to demonstrate that the learned $\mathcal{H}$ is complementary to Φ.

The loss function used here for training is the relative residual

$$\mathcal{L} = \sum_{i=1}^{N_b} \frac{\|\mathbf{f}_i - \mathbf{A}_i \mathbf{u}_i^K\|_2}{\|\mathbf{f}_i\|_2}, \tag{12}$$

where $\{\mathbf{A}_i, \mathbf{f}_i\}$ are the training data. N_b is the batch size. K indicates that the $K-$th step solution $\mathbf{u}^K$ is used to calculate the loss. These specific values will be given in the next section. The training and testing algorithms of the FNS are summarized in Algorithms 1 and 2, respectively.

Algorithm 1: FNS offline traning.

```
Data: PDE parameters $\{\boldsymbol{\eta}_i\}_{i=1}^{N_{train}}$ and corresponding discrete systems $\{\mathbf{A}_i, \mathbf{f}_i\}_{i=1}^{N_{train}}$
Input: $\Phi$, HyperNN($\boldsymbol{\theta}$), K and Epochs
for $epoch = 1, \ldots, Epochs$ do serial
    for $i = 1, \ldots, N_{train}$ do parallel
        $\boldsymbol{\vartheta}_i = $ HyperNN $(\boldsymbol{\eta}_i, \boldsymbol{\theta})$
        $\mathbf{u}_i^0 = $ zeros like $\mathbf{f}_i$
        for $k = 0, \ldots, K-1$ do serial
            $\mathbf{v}_i^{k+1} = \Phi(\mathbf{u}_i^k)$
            $\mathbf{r}_i^k = \mathbf{f}_i - \mathbf{A}_i \mathbf{v}_i^{k+1}$
            $\hat{\mathbf{r}}_i^k = \mathcal{F}(\mathbf{r}_i^k)$
            $\hat{\mathbf{e}}_i^k = \hat{\mathbf{r}}_i^k \circ \boldsymbol{\vartheta}_i$
            $\mathbf{e}_i^k = \mathcal{F}^{-1}(\mathbf{e}_i^k)$
            $\mathbf{u}_i^{k+1} = \mathbf{v}_i^{k+1} + \mathbf{e}_i^k$
        end
    end
    Compute loss function (12)
    Apply Adam algorithm [39] to update parameters $\boldsymbol{\theta}$
end
return learned FNS
```

Algorithm 2: FNS online testing.

```
Data: PDE parameter $\boldsymbol{\eta}$ and corresponding discrete system $\mathbf{A}, \mathbf{f}$
Input: Learned FNS, acceptable tolerance $tol$ and maximum number of iteration
       steps $MaxIterNum$
Setup: $\boldsymbol{\vartheta} = $ HyperNN $(\boldsymbol{\eta}, \boldsymbol{\theta})$
$k = 0$
$\mathbf{u}^0 = $ zeros like $\mathbf{f}$
$res = \frac{\|\mathbf{f} - \mathbf{A}\mathbf{u}^k\|_2}{\|\mathbf{f}\|_2}$
while $res > tol$ and $k < MaxIterNum$ do
    $\mathbf{v}^{k+1} = \Phi(\mathbf{u}^k)$
    $\mathbf{r}^k = \mathbf{f} - \mathbf{A}\mathbf{v}^{k+1}$
    $\hat{\mathbf{r}}^k = \mathcal{F}(\mathbf{r}^k)$
    $\hat{\mathbf{e}}^k = \hat{\mathbf{r}}^k \circ \boldsymbol{\vartheta}$
    $\mathbf{e}^k = \mathcal{F}^{-1}(\mathbf{e}^k)$
    $\mathbf{u}^{k+1} = \mathbf{v}^{k+1} + \mathbf{e}^k$
    $res = \frac{\|\mathbf{f} - \mathbf{A}\mathbf{u}^k\|_2}{\|\mathbf{f}\|_2}$
    $k = k + 1$
end
return solution of $\mathbf{A}\mathbf{u} = \mathbf{f}$
```

4. Numerical Experiments

We used the anisotropic diffusion equation, the convection–diffusion equation, and the Helmholtz equation as examples to demonstrate the performance of the FNS. In all experiments, the matrix-vector products were implemented by CNN based on the Pytorch [40] platform. All the code can be found at https://github.com/cuichen1996/FourierNeuralSolver (accessed on 13 September 2022).

4.1. Anisotropic Diffusion Equation

Consider the anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \tag{13}$$

the diffusion coefficient matrix

$$C = C(\xi, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \xi \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \tag{14}$$

$0 < \xi < 1$ is the anisotropic strength and $\theta \in [0, \pi]$ is the anisotropic direction, $\Omega = (0,1)^2$. We use bilinear FEM to discretize (13) with a uniform $n \times n$ quadrilateral mesh. The associated discrete system is shown in (2), where $N = (n-1) \times (n-1)$. We performed experiments for the two cases described below.

4.1.1. Case 1: Generalization Ability of Anisotropic Strength with Fixed Direction

In this case, we use the same training and testing data as [20]. For fixed $\theta = 0$, we randomly sample 20 distinct parameters ξ from the distribution $\log_{10} \frac{1}{\xi} \sim U[0,5]$ and obtain $\{\mathbf{A}_i\}_{i=1}^{20}$ by discretizing (13) using bilinear FEM with $n = 256$. We randomly select 100 right-hand functions for each $\mathbf{A}_i$. Each entry of $\mathbf{f}$ is sampled from the Gaussian distribution $N(0,1)$. Therefore, there are $N_{train} = 2000$ training data. The hyperparameters used for training, including batch size, learning rate, K in the loss function, and the concrete network structure of HyperNN are listed in Appendix A.1.

The FNS can take various kinds of Φ. In this case, we use weighted Jacobi, Chebyshev semi-iterative (Cheby-semi) and Krylov methods. The weight of the weighted Jacobi method is 2/3. We use a DenseNet [41] to give the basis vector of the Krylov subspace, then approximate the solution in this subspace by least squares as in [20]. For the Chebyshev semi-iterative method, we provide a brief summary here. More details can be obtained in [42,43].

If we vary the parameter of the Richardson iteration at each step

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \tau_k \left(\mathbf{f} - \mathbf{A}u^k \right), \tag{15}$$

and the maximum and minimum eigenvalues of $\mathbf{A}$ are known, then τ_k can be determined as

$$\tau_k = \frac{2}{\lambda_{\max} + \lambda_{\min} - (\lambda_{\min} - \lambda_{\max})x_k}, \quad k = 0, \ldots, m-1, \tag{16}$$

where

$$x_k = \cos\frac{\pi(2k+1)}{2n}, k = 0, \ldots, m-1, \tag{17}$$

are the roots of an $m-$order Chebyshev polynomial. Here, $\lambda_{\max}$ is obtained by the power method [44], but calculating $\lambda_{\min}$ often incurs an expensive computational cost. Therefore, we use $\lambda_{\max}/\alpha$ to replace $\lambda_{\min}$. The resulting method is referred to as a Chebyshev semi-iteration. We take $m = 10, \alpha = 3$. Figure 3 shows the convergence factor obtained by LFA. It can be seen that the smooth effect improves as m increases. However, the high-frequency error along the y direction is also difficult to eliminate. When Φ is the Jacobi method, we implement Φ five-times and transform the residual to frequency space to correct errors. This is because employing the stationary method many times can enhance its smoothing effect.

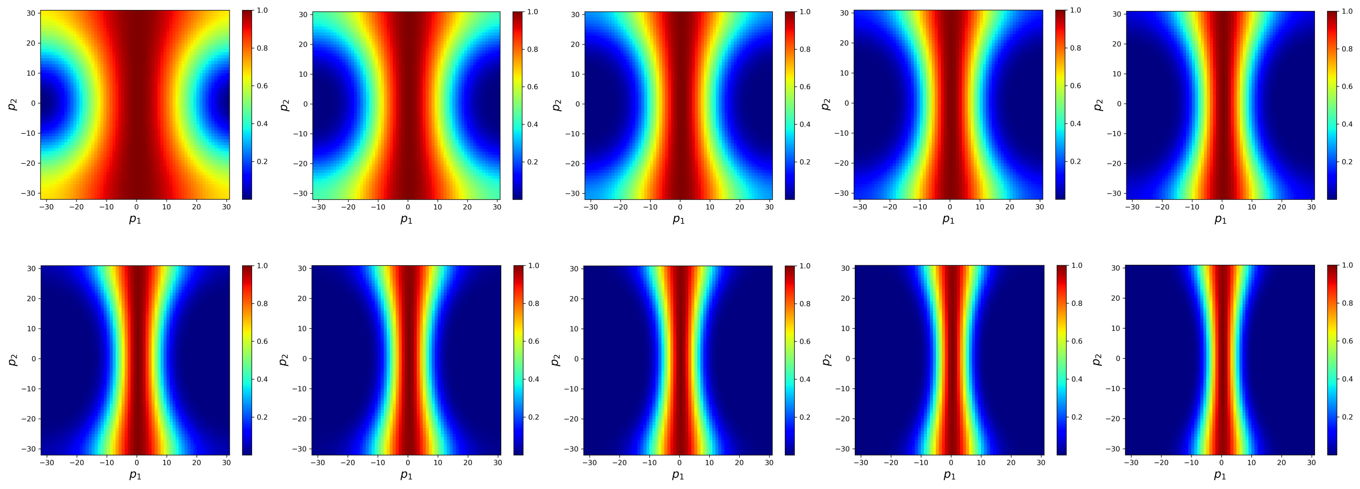


Figure 3. Distribution of convergence factor for Cheby-semi ($m = 10$) when $\xi = 10^{-3}, \theta = 0$.

After training, we choose $\theta = 0$, $\xi = 10^{-1}, \ldots, 10^{-6}$ for testing and generate 10 random right-hand functions for each parameter. The iteration is terminated when the relative residual is less than 10^{-6}. We use "mean $\pm$ std" to show the mean and standard deviation of the iterations over the test set as in [20].

Table 1 shows the test results of different solvers. The iteration steps of all the solvers grow as anisotropic strength increases except for MG (line-Jacobi). The growth of FNS is substantially lower than Meta-MgNet and MG. When the FNS employs the same Φ as Meta-MgNet, the number of iterations is nearly 10-times lower than that of Meta-MgNet at $\varepsilon = 10^{-5}$ with the same computational complexity for a single-step iteration. The line-Jacobi smoother can only be applied to several specific θ, i.e., $0, \pi/4, \pi/2, 3\pi/4, \pi$. However, the FNS can be available for arbitrary θ.

Table 1. Mean and standard deviation of the number of iterations required to achieve the stopping criterion over all tests for the anisotropic diffusion equation case 1. "$-$" means that it cannot converge within 10,000 steps, and " " means that [20] does not provide test results for this parameter.

ξ	FNS (Cheby-Semi)	FNS (Jacobi)	FNS (Krylov)	Meta-MgNet (Krylov) [20]	MG (Jacobi)	MG (Line-Jacobi)
$\xi = 10^{-1}$	67.9 ± 3.81	138.9 ± 11.18	30.0 ± 4.58	7.5 ± 0.50	90.2 ± 0.98	13.0 ± 0.00
$\xi = 10^{-2}$	101.6 ± 8.72	167.8 ± 13.81	38.5 ± 3.83	35.1 ± 1.04	752.8 ± 12.23	13.0 ± 0.00
$\xi = 10^{-3}$	151.0 ± 7.24	221.7 ± 11.56	48.6 ± 3.26	171.6 ± 6.34	5600 ± 119.42	13.0 ± 0.00
$\xi = 10^{-4}$	233.2 ± 5.67	330.1 ± 9.16	65.5 ± 2.80	375.2 ± 5.88	$-$	11.0 ± 0.00
$\xi = 10^{-5}$	340.1 ± 9.43	466.2 ± 13.47	80.7 ± 7.21	797.8 ± 12.76	$-$	11.0 ± 0.00
$\xi = 10^{-6}$	348.1 ± 11.15	477.9 ± 16.10	85.9 ± 7.52		$-$	11.0 ± 0.00

We use $\xi = 10^{-1}, 10^{-6}, n = 64$ and Cheby-semi ($m = 10$) as examples to illustrate the error that $\mathcal{H}$ learned. Figure 4a shows the convergence factor obtained by LFA of Cheby-semi ($m = 10$) for solving the system with $\xi = 10^{-1}, \theta = 0$. It can effectively eliminate the error components except for low-frequency errors. Figure 4b shows the distribution of errors before correction in frequency space. The result is consistent with the guidance of LFA, i.e., the error is concentrated in the low-frequency modes. Figure 4c shows the distribution of errors learned by $\mathcal{H}$ in the frequency space at this time. Its distribution is largely similar to that of Figure 4b. Figure 4d–f show the corresponding situation for

$\xi = 10^{-6}, \theta = 0$. In this case, Cheby-semi ($m = 10$) is unable to eliminate the error along the y direction. However, $\mathcal{H}$ is still capable of learning.

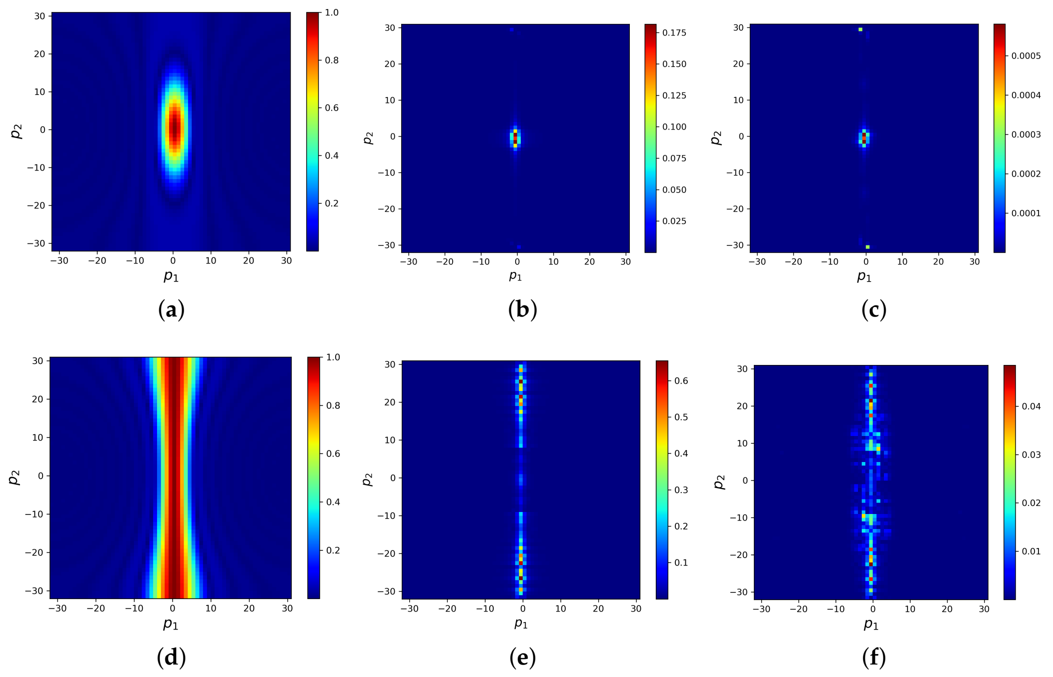


Figure 4. Distribution of convergence factor when $\xi = 10^{-1}, 10^{-6}$. The first column displays the convergence factor of Cheby-semi ($m = 10$). The second column shows the error distribution in the frequency space before correction. The third column shows the error distribution in the frequency space learned by $\mathcal{H}$. (**a**) $\xi = 10^{-1}$: convergence factor of Φ. (**b**) $\xi = 10^{-1}$: $\hat{\mathbf{e}}$, before doing corrections. (**c**) $\xi = 10^{-1}$: learned $\hat{\mathbf{e}}$. (**d**) $\xi = 10^{-6}$: convergence factor of Φ. (**e**) $\xi = 10^{-6}$: $\hat{\mathbf{e}}$, before doing corrections. (**f**) $\xi = 10^{-6}$: learned $\hat{\mathbf{e}}$.

4.1.2. Case 2: Generalization Ability of Anisotropic Direction with Fixed Strength

We randomly sample 20 parameters θ according to the distribution $\theta \sim U[0, \pi]$ with fixed $\xi = 10^{-6}$. The training and testing data are generated in a similar manner as in Section 4.1.1. Table 2 shows the test results. It can be seen that whether Φ is Jacobi or Krylov, the FNS can maintain robust performance in all situations, while the line-smoother is not available for these cases.

Table 2. Mean and standard deviation of the number of iterations required to achieve the stopping criterion over all tests for Equation (13) case 2.

θ	0.1π	0.2π	0.3π	0.4π	0.6π	0.7π	0.8π	0.9π
FNS (Jacobi)	300.2 ± 23.95	252.4 ± 34.81	269.3 ± 36.70	356.4 ± 39.69	338.4 ± 32.07	265.0 ± 29.96	266.5 ± 25.07	316.7 ± 17.26
FNS (Krylov)	58.4 ± 4.45	46.5 ± 4.84	45.1 ± 2.30	64.0 ± 7.01	54.4 ± 5.75	41.3 ± 7.11	43.0 ± 2.83	60.3 ± 3.93

Take $\theta = j\pi/10, j = 1, \ldots, 4, 6, \ldots, 9$, $\xi = 10^{-6}, n = 64$ and Φ is the weighted Jacobi method with weight 2/3. Figure 5 shows the test results. The first row shows the weighted Jacobi method convergence factor μ_{loc} for each θ which is computed by LFA. The region of $\mu_{\text{loc}} \sim 1$ means that the error is difficult to eliminate. These error components are distributed along the anisotropic direction. The second row shows the error distribution in frequency space before correction, which is consistent with the results obtained by the LFA. The third row shows the distribution of the error learned by $\mathcal{H}$. It can be seen that $\mathcal{H}$ can automatically learn the error components that Φ has difficulty eliminating. The line-smoother is not feasible for these θ.

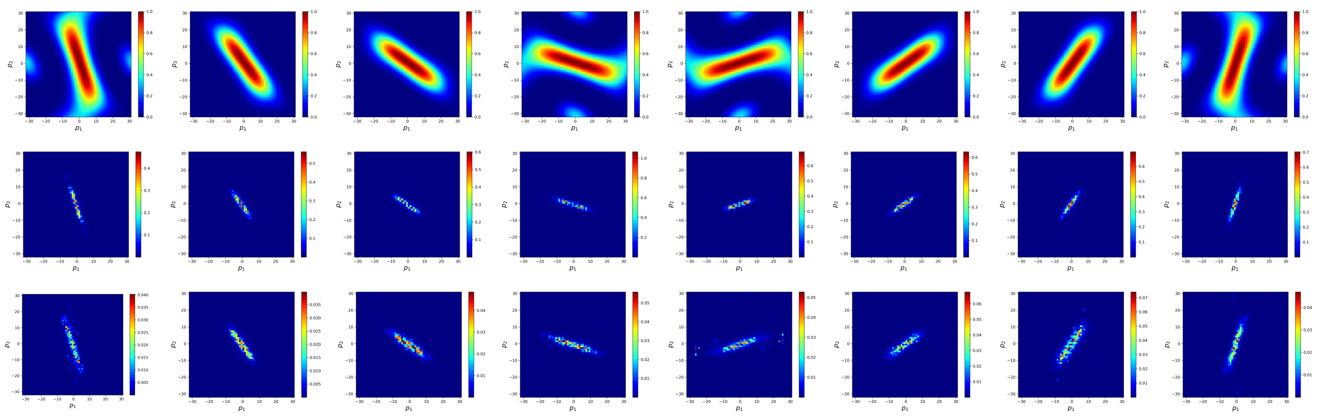


Figure 5. Case 2 of Equation (13) for different anisotropic direction θ. The first row represents the convergence factor of Φ. The second row represents the error distribution in the frequency space before correction. The third row represents the learned error by $\mathcal{H}$.

4.2. Convection–Diffusion Equation

Consider the convection–diffusion equation

$$\begin{cases} -\varepsilon\Delta u + u_x + u_y = 0, & \Omega = (0,1)^2, \\ u = 0, & x = 0, 0 \le y < 1 \text{ and } y = 0, 0 \le x < 1, \\ u = 1, & x = 1, 0 \le y < 1 \text{ and } y = 1, 0 \le x \le 1, \end{cases} \tag{18}$$

We use the central difference method to discretize (18) on a uniform mesh with spatial size h in both x and y directions, which yields a non-symmetric stencil

$$\frac{1}{h^2}\begin{bmatrix} & h/2-\varepsilon & \\ -h/2-\varepsilon & 4\varepsilon & h/2-\varepsilon \\ & -h/2-\varepsilon & \end{bmatrix}. \tag{19}$$

To meet the stability requirement, the central difference scheme needs to satisfy the Peclet condition

$$Pe := \frac{h}{\varepsilon}\max(|a|,|b|) \le 2, \tag{20}$$

which means that the central difference method cannot approximate the PDE solution when ε is extremely small. However, here, we only take into account the solver of the linear system. Thus, we continue to use this discretization method to demonstrate the performance of the FNS. In the following experiments, we explore diffusion- and convection-dominant cases, respectively.

4.2.1. Case 1: $\varepsilon \in (0.01, 1)$

We utilize the weighted Jacobi method as Φ in this case. Taking $\varepsilon = 0.1, h = 1/64$ as an example, Figure 6a illustrates the convergence factor obtained by LFA of the weighted Jacobi method ($\omega = 4/5$) for solving the system. We use five-times the consecutive weighted Jacobi method as Φ. Figure 6b–e show that such a Φ is a good smoother. Figure 6f shows the distribution of error learned by $\mathcal{H}$ in the frequency space. It can be seen that this is essentially complementary to Φ.

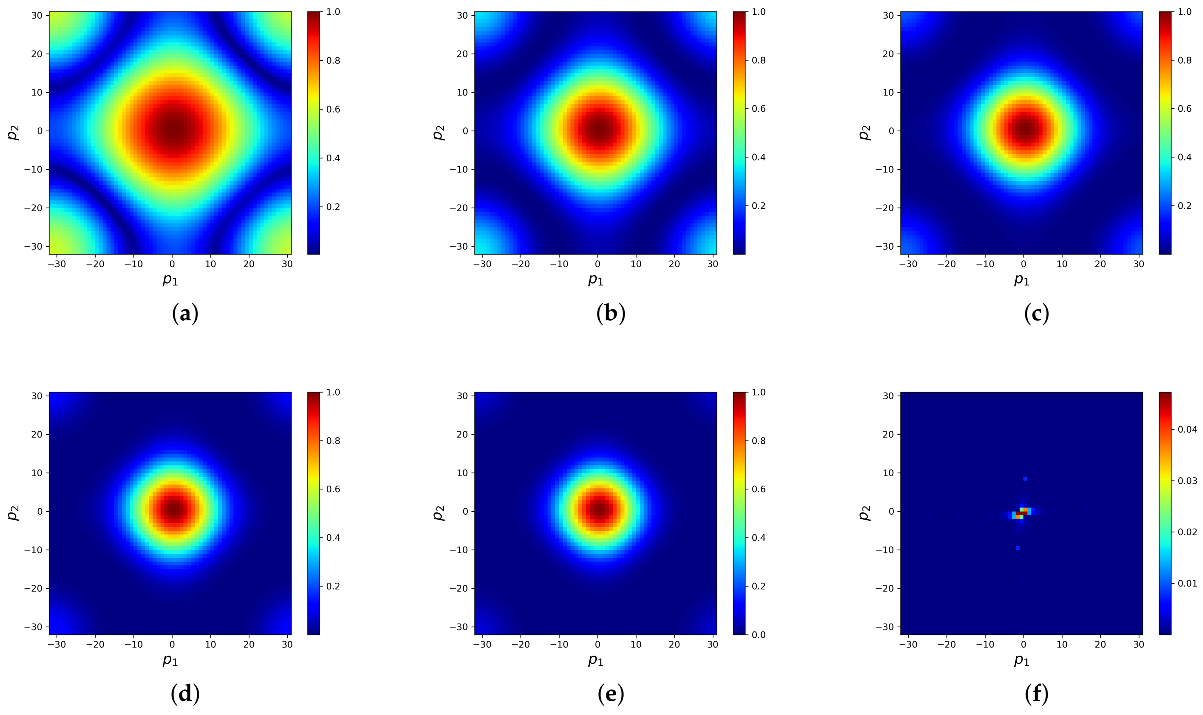


Figure 6. Distribution of convergence factor of five-times consecutive weighted Jacobi method. The last plot shows the distribution of errors learned by $\mathcal{H}$ in frequency space. (**a**) Convergence factor of one times weighted Jacobi ($\omega = 4/5$). (**b**) Two times. (**c**) Three times. (**d**) Four times. (**e**) Five times. (**f**) Learned $\hat{\mathbf{e}}$.

Figure 7 uses $\varepsilon = 0.5, 0.1, 0.05$ as examples to show the relative residual of the FNS and the weighted Jacobi method. It can be seen that the FNS has acceleration and the weighted Jacobi method ramps up as ε decreases. This is because when ε declines, the diagonal element 4ε becomes small, and the weight along the gradient direction increases. However, the Jacobi and other gradient descent algorithms will diverge as long as ε continues to decrease unless the weight is drastically reduced.

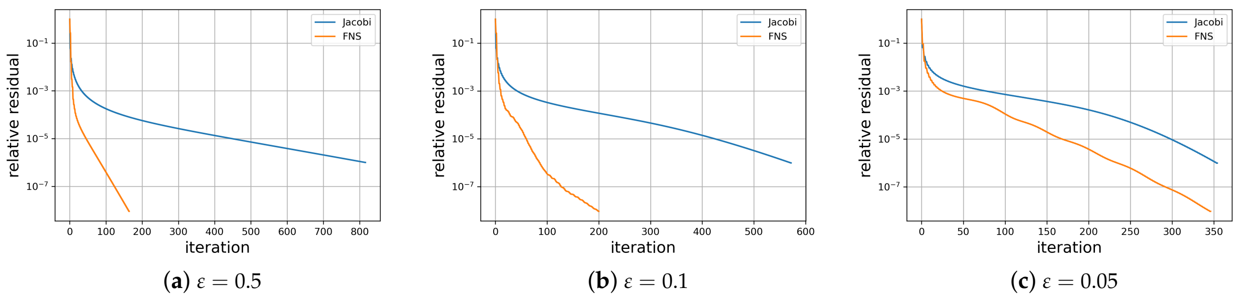


Figure 7. The relative residual with the FNS and weighted Jacobi method, where Jacobi denotes five-times the consecutive weighted Jacobi method.

4.2.2. Case 2: $\varepsilon \in [10^{-6}, 10^{-3}]$

In this case, since the diagonal elements of the discrete system are notably less than the off-diagonal elements, the system is non-symmetric. Many methods, such as Jaocbi, CG, and MG (Jacobi), might diverge. Figure 8 shows the convergence factor of the weighted Jacobi ($\omega = 4/5$) for solving the system when $\varepsilon = 10^{-2}, 10^{-3}, 10^{-6}$. It can be seen that, when ε is small, the convergence factor of the weighted Jacobi method for most frequency modes is larger than 1, which causes this iterative method to diverge and to be unsuitable as a smoother.

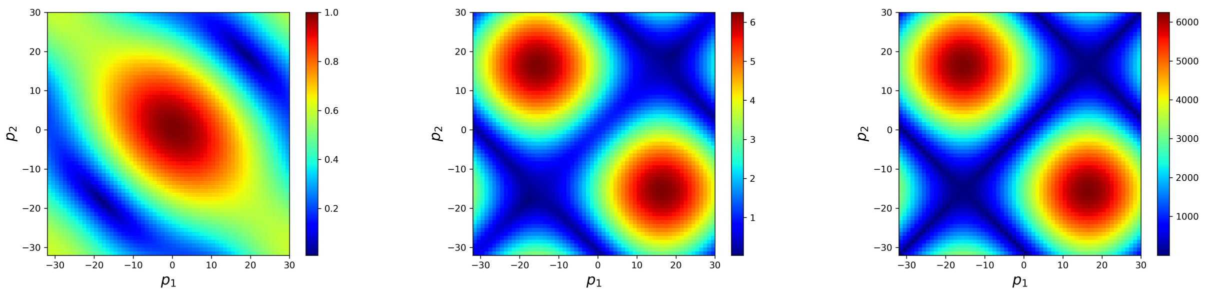
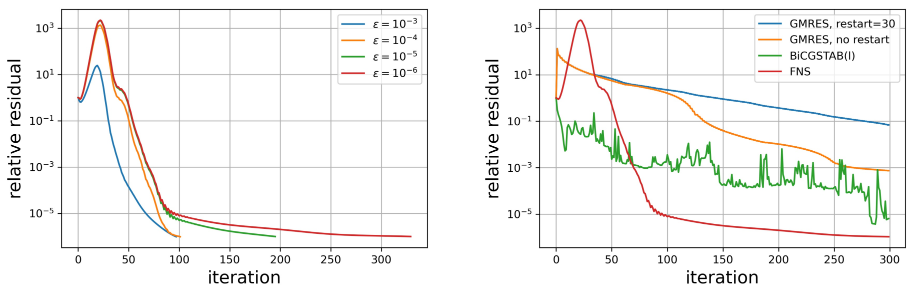

(**a**) $\varepsilon = 10^{-2}$ (**b**) $\varepsilon = 10^{-3}$ (**c**) $\varepsilon = 10^{-6}$

Figure 8. Distribution of convergence factor for weighted Jacobi method ($\omega = 4/5$) when solving the system corresponding to $\varepsilon = 10^{-2}, 10^{-3}, 10^{-6}$.

Consequently, we learn the Φ in Equation (3), where $\mathbf{B}$ is a two-layer linear CNN with channels $1 \to 8 \to 1$, and the kernel size is 3×3. The Φ is trained together with $\mathcal{H}$; the training hyperparameters are listed in Appendix A.2. Figure 9a illustrates how the learned FNS is able to solve the linear system when $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. It is evident that the FNS converges rapidly. Figure 9b shows the change in the relative residual for the FNS, GMRES, and BiCGSTAB(ℓ) ($\ell = 15$) with $\varepsilon = 10^{-6}$. It is clear that the FNS has the fastest convergence rate.



(**a**) different ε (**b**) $\varepsilon = 10^{-6}$

Figure 9. (**a**) Change in relative residual with FNS iteration steps for different ε. (**b**) Comparison of FNS and GMRES, BiCGSTAB(ℓ) when $\varepsilon = 10^{-6}$.

4.3. Helmholtz Equation

The Helmholtz equation we consider here is

$$-\Delta u(x) - \kappa^2 u(x) = g(x), \quad x \in \Omega, \tag{21}$$

where $\Omega = (0,1)^2$, κ is the wavenumber. We currently only take into account the zero Dirichlet boundary condition. We use the second-order FDM to discretize (21) on a uniform mesh with spatial size h. The corresponding stencil reads

$$\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4-\kappa^2 h^2 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{22}$$

We examine the FNS performance at a low wavenumber ($\kappa = 25$) and medium wavenumber ($\kappa = 125$). For $\kappa = 25$, we take $h = 1/64$ and $h = 1/256$ for $\kappa = 125$. Using

Krylov in [20] as Φ and $g(x) = 1$; the training hyperparameters are listed in Appendix A.3. Figure 10 shows how the relative residual decreases with different solvers. For $\kappa = 25$, the FNS performs best for the first 300 steps; however, BiCGSTAB performs better at the end. For $\kappa = 125$, the FNS outperforms BiCGSTAB. The GMRES results were too poor to display for this case.

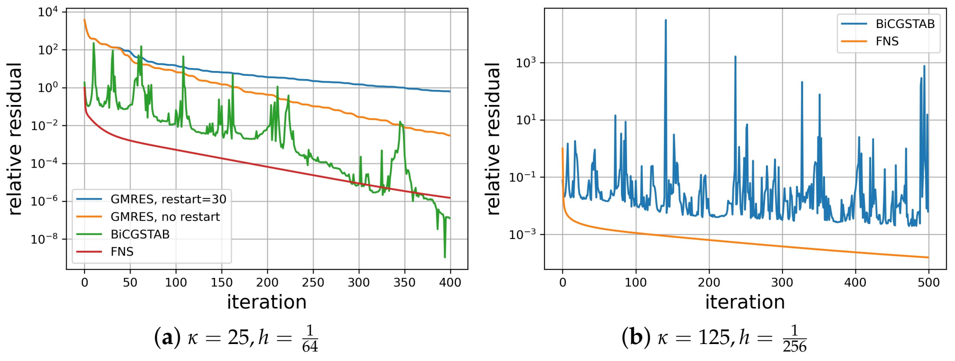


Figure 10. Change in relative residual for the FNS and other solvers when $\kappa = 25$ and $\kappa = 125$, respectively.

5. Conclusions And Future Work

This paper proposes an interpretable FNS to solve large sparse linear systems. It is composed of a stationary method and a frequency correction, which are used to eliminate errors in different Fourier modes. Numerical experiments undertaken showed that the FNS was more effective and robust than other solvers in solving the anisotropic diffusion equation, the convection–diffusion equation and the Helmholtz equation. The core concepts discussed here are relevant to a broad range of systems.

There is still a great deal of work to do. First, we only considered uniform mesh in this paper. We intend to generalize the FNS to non-uniform grids by exploiting geometric deep learning tools, such as graph neural networks and graph Fourier transform. Secondly, as previously discussed, the stationary method converges slowly or diverges in some situations, which has prompted researchers to approximate solutions in other transform space. This is true for almost all advanced iterative methods, including MG, Krylov subspace methods and the FNS. This specified space, however, may not always be the best choice. In the future, we will investigate additional potential transforms, such as Chebyshev, Legendre transforms, and potentially learnable transforms based on data.

Author Contributions: Conceptualization, C.C., K.J. and S.S.; methodology, C.C., K.J. and S.S.; software, C.C. and Y.L.; validation, C.C. and Y.L.; formal analysis, C.C., K.J. and S.S.; investigation, C.C.; resources, C.C., K.J. and S.S.; data curation, C.C.; writing—original draft preparation, C.C., K.J. and S.S.; writing—review and editing, C.C. and K.J.; visualization, C.C.; supervision, K.J. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (12171412, 11971414). K.J. is partially supported by the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (2021JJ10037). C.C. is supported by the Hunan Provincial Innovation Foundation For Postgraduates.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Code and data are available at https://github.com/cuichen1996/FourierNeuralSolver (accessed on 13 September 2022).

Acknowledgments: We would like to thank Chen et al. [20] for sharing data for the anisotropic diffusion equation and the Krylov method.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PDE	Partial differential equation
FFT	Fast Fourier transform
FNS	Fourier neural solver
CG	Conjugate gradient
GMRES	Generalized minimal residual
GMG	Geometric multigrid
AMG	Algebraic multigrid
CNN	Convolutional neural network
LFA	Local Fourier analysis
FEM	Finite element method
FDM	Finite difference method
HyperNN	Hyper-neural network

Appendix A. Training Hyperparameters

Appendix A.1. Anisotropic Diffusion Equation

Table A1. Training hyperparameters for anisotropic diffusion equation.

	Learning Rate	Batch Size	K	Xavier Init	Grad Clip
FNS (Cheby-semi)	10^{-4}	100	10	10^{-2}	false
FNS (Jacobi)	10^{-4}	100	10	10^{-2}	false
FNS (Krylov)	10^{-4}	100	10	10^{-2}	false

Table A2. HyperNN architecture parameters for anisotropic diffusion equation. Notations in ConvTranspose2d are: i: in channels; o: out channels; k: kernel size; s: stride; p: padding.

ConvTranspose2d(i = 1,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 2,k = 3,s = 2,p = 2)
AdaptiveAvgPool2d(n)

Appendix A.2. Convection–Diffusion Equation

Table A3. Training hyperparameters for convection–diffusion equation.

	Learning Rate	Batch Size	K	Xavier Init	Grad Clip
FNS(Jacobi)	10^{-4}	100	10	10^{-2}	false
FNS(Conv)	10^{-4}	100	$1\sim100$	10^{-2}	1.0

Table A4. HyperNN architecture parameters for convection–diffusion equation. Notations in ConvTranspose2d are: i: in channels; o: out channels; k: kernel size; s: stride; p: padding.

Φ	Hyper NN
Conv(i = 1,o = 8, k = 3, s = 2, p = 1)	ConvTranspose2d(i = 1,o = 4,k = 3,s = 2,p = 1) + Relu()
Conv(i = 8,o = 1, k = 3, s = 2, p = 1)	ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
	ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
	ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
	ConvTranspose2d(i = 4,o = 2,k = 3,s = 2,p = 2)

Appendix A.3. Helmholtz Equation

Table A5. Training hyperparameters for Helmholtz equation.

	Learning Rate	**Batch Size**	K	**Xavier Init**	**Grad Clip**
FNS(Krylov)	10^{-4}	100	$1\sim 100$	10^{-2}	1.0

Table A6. HyperNN architecture parameters for Helmholtz equation. Notations in ConvTranspose2d are: i: in channels; o: out channels; k: kernel size; s: stride; p: padding.

ConvTranspose2d(i = 1,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 4,k = 3,s = 2,p = 1) + Relu()
ConvTranspose2d(i = 4,o = 2,k = 3,s = 2,p = 2)
AdaptiveAvgPool2d(n)

References

1. Barrett, R.; Berry, M.; Chan, T.F.; Demmel, J.; Donato, J.; Dongarra, J.; Eijkhout, V.; Pozo, R.; Romine, C.; Van der Vorst, H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*; SIAM: Singapore, 1994.
2. Saad, Y. *Iterative Methods for Sparse Linear Systems*; SIAM: Singapore, 2003.
3. Hestenes, M.R.; Stiefel, E. Methods of conjugate gradients for solving. *J. Res. Natl. Bur. Stand.* **1952**, *49*, 409. [CrossRef]
4. Saad, Y.; Schultz, M.H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **1986**, *7*, 856–869. [CrossRef]
5. Brandt, A. Multi-level adaptive solutions to boundary-value problems. *Math. Comput.* **1977**, *31*, 333–390. [CrossRef]
6. Briggs, W.L.; Henson, V.E.; McCormick, S.F. *A Multigrid Tutorial*; SIAM: Singapore, 2000.
7. Trottenberg, U.; Oosterlee, C.W.; Schuller, A. *Multigrid*; Elsevier: Amsterdam, The Netherlands, 2000.
8. Falgout, R.D. *An Introduction to Algebraic Multigrid*; Technical Report; Lawrence Livermore National Lab. (LLNL): Livermore, CA, USA, 2006.
9. Xu, J.; Zikatanov, L. Algebraic multigrid methods. *Acta Numer.* **2017**, *26*, 591–721. [CrossRef]
10. Hsieh, J.T.; Zhao, S.; Eismann, S.; Mirabella, L.; Ermon, S. Learning neural PDE solvers with convergence guarantees. *arXiv* **2019**, arXiv:1906.01200.
11. Luna, K.; Klymko, K.; Blaschke, J.P. Accelerating gmres with deep learning in real-time. *arXiv* **2021**, arXiv:2103.10975.
12. Zhang, E.; Kahana, A.; Turkel, E.; Ranade, R.; Pathak, J.; Karniadakis, G.E. A Hybrid Iterative Numerical Transferable Solver (HINTS) for PDEs Based on Deep Operator Network and Relaxation Methods. *arXiv* **2022**, arXiv:2208.13273.
13. Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; Karniadakis, G.E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* **2021**, *3*, 218–229. [CrossRef]
14. Weymouth, G.D. Data-Driven Multi-grid Solver for Accelerated Pressure Projection. *arXiv* **2021**, arXiv:2110.11029.
15. Tomasi, C.; Krause, R. Construction of Grid Operators for Multilevel Solvers: A Neural Network Approach. *arXiv* **2021**, arXiv:2109.05873.
16. Taghibakhshi, A.; MacLachlan, S.; Olson, L.; West, M. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 12129–12140.
17. Huang, R.; Li, R.; Xi, Y. Learning optimal multigrid smoothers via neural networks. *arXiv* **2021**, arXiv:2102.12071.
18. Wang, F.; Gu, X.; Sun, J.; Xu, Z. Learning-Based Local Weighted Least Squares for Algebraic Multigrid Method. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4110904 (accessed on 16 May 2022).
19. Fanaskov, V. Neural Multigrid Architectures. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, Shenzhen, China, 18–22 July 2021; pp. 1–8.
20. Chen, Y.; Dong, B.; Xu, J. Meta-mgnet: Meta multigrid networks for solving parameterized partial differential equations. *J. Comput. Phys.* **2022**, *455*, 110996. [CrossRef]
21. Katrutsa, A.; Daulbaev, T.; Oseledets, I. Black-box learning of multigrid parameters. *J. Comput. Appl. Math.* **2020**, *368*, 112524. [CrossRef]
22. Greenfeld, D.; Galun, M.; Basri, R.; Yavneh, I.; Kimmel, R. Learning to optimize multigrid PDE solvers. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 10–15 June 2019; pp. 2415–2423.
23. Luz, I.; Galun, M.; Maron, H.; Basri, R.; Yavneh, I. Learning algebraic multigrid using graph neural networks. In Proceedings of the International Conference on Machine Learning, PMLR, Online, 26–28 August 2020; pp. 6489–6499.
24. Antonietti, P.F.; Caldana, M.; Dede, L. Accelerating Algebraic Multigrid Methods via Artificial Neural Networks. *arXiv* **2021**, arXiv:2111.01629.

25. Stanziola, A.; Arridge, S.R.; Cox, B.T.; Treeby, B.E. A Helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *J. Comput. Phys.* **2021**, *441*, 110430. [CrossRef]
26. Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; Dabney, W. Recurrent experience replay in distributed reinforcement learning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2018.
27. Azulay, Y.; Treister, E. Multigrid-Augmented Deep Learning Preconditioners for the Helmholtz Equation. *arXiv* **2022**, arXiv:2203.11025.
28. Erlangga, Y.A.; Oosterlee, C.W.; Vuik, C. A novel multigrid based preconditioner for heterogeneous Helmholtz problems. *SIAM J. Sci. Comput.* **2006**, *27*, 1471–1492. [CrossRef]
29. Calandra, H.; Gratton, S.; Langou, J.; Pinel, X.; Vasseur, X. Flexible variants of block restarted GMRES methods with application to geophysics. *SIAM J. Sci. Comput.* **2012**, *34*, A714–A736. [CrossRef]
30. Um, K.; Brand, R.; Fei, Y.R.; Holl, P.; Thuerey, N. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 6111–6122.
31. Nikolopoulos, S.; Kalogeris, I.; Papadopoulos, V.; Stavroulakis, G. AI-enhanced iterative solvers for accelerating the solution of large scale parametrized linear systems of equations. *arXiv* **2022**, arXiv:2207.02543.
32. Stanaityte, R. ILU and Machine Learning Based Preconditioning for the Discretized Incompressible Navier-Stokes Equations. Ph.D. Thesis, University of Houston, Houston, TX, USA, 2020.
33. Kaneda, A.; Akar, O.; Chen, J.; Kala, V.; Hyde, D.; Teran, J. A Deep Gradient Correction Method for Iteratively Solving Linear Systems. *arXiv* **2022**, arXiv:2205.10763.
34. Margenberg, N.; Hartmann, D.; Lessig, C.; Richter, T. A neural network multigrid solver for the Navier-Stokes equations. *J. Comput. Phys.* **2022**, *460*, 110983. [CrossRef]
35. Margenberg, N.; Jendersie, R.; Richter, T.; Lessig, C. Deep neural networks for geometric multigrid methods. *arXiv* **2021**, arXiv:2106.07687.
36. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
37. Sleijpen, G.L.; Fokkema, D.R. BiCGstab (ell) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.* **1993**, *1*, 11–32.
38. Swarztrauber, P.N. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Rev.* **1977**, *19*, 490–501. [CrossRef]
39. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
40. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
41. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
42. Golub, G.H.; Varga, R.S. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods. *Numer. Math.* **1961**, *3*, 157–168. [CrossRef]
43. Adams, M.; Brezina, M.; Hu, J.; Tuminaro, R. Parallel multigrid smoothing: Polynomial versus Gauss–Seidel. *J. Comput. Phys.* **2003**, *188*, 593–610. [CrossRef]
44. Mises, R.; Pollaczek-Geiringer, H. Praktische Verfahren der Gleichungsauflösung. *Zamm-J. Appl. Math. Mech. FÜR Angew. Math. Mech.* **1929**, *9*, 58–77. [CrossRef]