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 Fast Fourier transform. Because the error between the iterative solution and the ground truth involves a wide range of frequency modes, FNS combines a stationary iterative method and frequency space correction to eliminate different components of the error. Local Fourier analysis reveals 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 anisotropy diffusion equation, convection-diffusion equation, and Helmholtz equation show that 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.

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 nonlinear 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 methods [1]. Due to the sluggish convergence, stationary methods such as weighted Jacobi, Gauss-Seidel, successive over relaxation methods [2] are frequently utilized as smoothers in multi-level methods or as preconditioners. Non-stationary methods typically refer to Krylov subspace methods, such as conjugate gradient (CG), generalized minimal residual (GMRES) methods [3, 4], whose convergence rate will be influenced heavily by some factors like initial value. Multi-level methods mainly include geometric multigrid (GMG) method [5, 6, 7] and algebraic multigrid (AMG) method [8, 9]. They are both composed of many factors, such as smoother and coarse grid correction, which heavily affect convergence. Finding these factors for a concrete problem is an art that demands extensive analysis, innovation, and trial.

In recent years, the technique of automatically picking parameters for Krylov and multi-level methods or constructing a learnable iterative scheme based on deep learning has attracted a lot of interest. For second-order elliptic equations with smooth coefficients, many neural solvers have achieved satisfactory results. Hsieh et al. [10] utilized a convolutional neural network (CNN) to accelerate the convergence of Jacobi method. Luna et al. [11] accelerated the convergence of GMRES with a learned initial value. Significant efforts are also made in the development of multigrid solvers, such as learning smoother, transfer operator [12, 13] and coarse-fine splitting [14]. For anisotropy elliptic equations, Huang
et al. [15] exploited a CNN to design a more sensible smoother. Results showed that the magnitude of the learned smoother is dispersed along the anisotropic direction. Wang et al. [16] 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 [17] learned the smoother and transfer operator of GMG in a neural network form. When the anisotropic strength is mild (within two orders of magnitude), previously mentioned works exhibit a considerable acceleration. Chen et al. [18] 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 is still sensitive to the anisotropic strength. For convection-diffusion equations, Katrutsa et al. [19] learned the weighted Jacobi smoother and transfer operator of GMG, which has a positive effect on the upwind discretization scheme and also applied to solve a one-dimensional Helmholtz equation. For second-order elliptic equations with random diffusion coefficients, Greenfeld et al. [20] employed a residual network to construct the prolongation operator of AMG for uniform grids. Luz et al. [21] extended it to non-uniform grids using graph neural networks, which outperforms classical AMG methods. For jumping coefficient problems, Antonietti et al. [22] presented a neural network to forecast the correct parameter to speed up AMG and used it as a preconditioner for CG. For Helmholtz equation, Stanziala et al. [23] constructed a fully learnable neural solver, the helmnet, which is built on U-net and recurrent neural network [24]. Azulay et al. [25] developed a preconditioner based on U-net and shift-Laplacian MG [26] and applied the flexible GMRES [27] to solve the discrete system. For solid and fluid mechanics equations, there are also some neural solvers on associated discrete systems, such as but not limited to, learning initial values [28]  [29], constructing preconditioners [30], learning search directions of CG [31], learning parameters of GMG [32]  [33].

In this paper, we propose the Fourier Neural Solver (FNS), a deep learning and Fast Fourier Transform (FFT) [34] based neural solver. FNS is made up of two modules: the stationary method and the frequency space correction. Since stationary methods like weighted Jacobi method are difficult to get rid of low-frequency error, FNS uses FFT and CNN to learn these modes in the frequency space. Local Fourier analysis (LFA) [5] reveals that FNS can pick up on the error components in frequency space that are challenging to eradicate by stationary methods. Therefore, FNS builds a complementary relationship by stationary method and CNN to eliminate error. With the help of FFT, the single-step iteration of FNS has a \( \mathcal{O}(N \log_2 N) \) computational complexity. All matrix-vector products are implemented using convolution, which is both storage-efficient and straightforward to parallelize. We investigate the effectiveness and robustness of FNS on three types of convection-diffusion-reaction equations. For anisotropic equations, numerical experiments show that FNS can reduce the number of iterations by nearly 10 times compared to the state-of-the-art Meta-MgNet when the anisotropic strength is relatively strong. For the non-symmetric systems arising from the convection-diffusion equations discretized by central difference method, FNS can converge while MG and CG methods diverge. And FNS is faster than other algorithms such GMRES and BiCGSTAB(\ell) [35]. For the indefinite systems arising from the Helmholtz equations, FNS outperforms GMRES and BiCGSTAB at medium wavenumbers. In this paper, we apply FNS to above three PDE systems. However, the principles employed by FNS indicate that FNS has the potential to be useful for a broad range of sparse linear algebraic systems.

The rest 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 FNS. Section 3 presents the FNS algorithm. Section 4 examines the performance of FNS to anisotropy, convection-diffusion, and Helmholtz equations. Finally, Section 5 draws the conclusions and future work.

### 2 Motivation

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

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

(2.1)

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

We can obtain a linear algebraic system once we discretize Eq. (2.1) by finite element method (FEM) or finite difference method (FDM)

\[
Au = f,
\]  

(2.2)

where \( A \in \mathbb{R}^{N \times N}, f \in \mathbb{R}^N \) 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

\[
u^{k+1} = u^k + B (f - Au^k),
\]  

(2.3)
where $\mathbf{B}$ is an easily computed operator such as the inverse of diagonal matrix (Jacobi method), the inverse of 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 Eq. (2.1) and use LFA to analyze the reason.

Taking $\varepsilon = 1$, $\alpha(x) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $\beta(x) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\gamma = 0$, Eq. (2.1) becomes the Poisson equation. With a linear FEM discretization and in stencil notation, the resulting discrete operator reads

\[
\begin{bmatrix}
-1 & -1 \\
4 & -1
\end{bmatrix}.
\]

In weighted Jacobi method, $\mathbf{B} = \omega \mathbf{I}/4$, where $\omega \in (0, 1]$ and $\mathbf{I}$ is the identity matrix. Eq. (2.3) can be written in the pointwise form

\[
u_{ij}^{k+1} = \nu_{ij}^k + \frac{\omega}{4} \left( f_{ij} - (4 \nu_{ij}^k - \nu_{i-1,j}^k - \nu_{i+1,j}^k - \nu_{i,j-1}^k - \nu_{i,j+1}^k) \right).
\]

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

\[
\mu_{\text{loc}} := \frac{|v_{ij}^{k+1} - v_{ij}^k|}{|v_{ij}^k|} = 1 - \omega + \frac{\omega}{4} \left( 4 - 2 \cos (2\pi p_1h) - 2 \cos (2\pi p_2h) \right).
\]

Figure 1: (a) The distribution of convergence factor $\mu_{\text{loc}}$ of weighted Jacobi method ($\omega = 2/3$) in solving linear system arisen by Poisson equation; (b) Low-frequency (white) and high-frequency (gray) regions.
as shown in Figure 1(b). It can be seen that in the high-frequency region, $\mu_{\text{loc}}$ is approximately zero, whereas in the low-frequency region, $\mu_{\text{loc}}$ is close to one. As a result, weighted Jacobi method attenuates high-frequency errors quickly but is mostly useless towards low-frequency errors.

The reason is attributed to two aspects. Firstly, the high-frequency reflects the local oscillation, while the low-frequency reflects the global pattern. Since $A$ is sparse and the basic operation $Au$ of weighted Jacobi method is a local operation, which makes it challenging to remove low-frequency global error components. Secondly, $A$ is sparse, $A^{-1}$ is commonly dense, which means that the mapping $f \rightarrow A^{-1}f$ 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 methods approximate the solution in a subspace spanned by a set of basis. MG generates a coarsen space to broaden the receptive field of the local operation. However, as mentioned in Section 1, these methods have various parameters to be carefully designed. In this paper, we propose the FNS, a generic solver that uses FFT to learn solutions in frequency space, with the parameters automatically obtained in a data-driven manner.

### 3 Fourier Neural Solver

Denote stationary iterative methods of (2.3) in a operator form
\[ v^{k+1} = \Phi(u^k), \tag{3.1} \]
and the $k$-th step residual
\[ r^k := f - Av^{k+1}, \tag{3.2} \]
then the $k$-th step error $e_k := u - v^{k+1}$ satisfies residual equation
\[ Ae_k = r^k. \tag{3.3} \]

As shown in the preceding section, the slow convergence rate of stationary methods is due to the difficulty in reducing low-frequency errors. In fact, even high-frequency errors might be not effectively eliminated by $\Phi$ for many cases. Now, 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 of FNS. The module for solving the residual equation in frequency space is denoted as $\mathcal{H}$. It consists of three steps: FFT $\rightarrow$ Hadamard product $\rightarrow$ IFFT. The parameter $\vartheta$ of $\mathcal{H}$ is the output of the Hyper-neural network (HyperNN). The input $\eta$ of HyperNN is the PDE parameters corresponding to the discrete systems. During training, the only parameter $\theta$ of HyperNN serves as our optimization parameter.

![Figure 2: FNS calculation flow chart.](image)

The three-step operation of $\mathcal{H}$ is inspired by the fast Poisson solver [36]. Let eigenvalues and eigenvectors of $A$ be $\lambda_1, \ldots, \lambda_N$ and $q_1, \ldots, q_N$, respectively. Solving Eq. (2.2) includes three steps:

1. Expand $f$ as a combination $f = a_1 q_1 + \cdots + a_N q_N$ of the eigenvectors
2. Divide each $a_k$ by $\lambda_k$
3. Recombine eigenvectors into $u = (a_1/\lambda_1) q_1 + \cdots + (a_N/\lambda_N) q_N$.

In particular, when $A$ is the system generated by five-point stencil (2.4), its eigenvector $q_k$ is the sine function. The above first and third steps can now be done 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 FNS is $O(N \log_2 N)$.

It is worth noting that, although $\Phi$ 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, $H$ learns error components that $\Phi$ cannot easily eliminate. For $\Phi$ with a fixed stencil, we can use LFA to demonstrate that the learned $H$ is complementary to $\Phi$.

The loss function used here for training is the relative residual

$$\mathcal{L} = \sum_{i=1}^{N_b} \frac{\|f_i - A_i u^K_i\|_2}{\|f_i\|_2}, \quad (3.4)$$

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

---

**Algorithm 1: FNS offline training**

**Data:** PDE parameters $\{\eta_i\}_{i=1}^{N_{\text{train}}}$ and corresponding discrete systems $\{A_i, f_i\}_{i=1}^{N_{\text{train}}}$

**Input:** $\Phi$, HyperNN($\theta$), $K$ and Epochs

1. for epoch = 1, ..., Epochs do serial
2.   for $i = 1, \ldots, N_{\text{train}}$ do parallel
3.     $\vartheta_i = \text{HyperNN} (\eta_i, \theta)$
4.     $u_0^i = \text{zeros like } f_i$
5.     for $k = 0, \ldots, K - 1$ do serial
6.       $v_{i}^{k+1} = \Phi (u_{i}^{k})$
7.       $r_{i}^{k} = f_i - A_i v_{i}^{k+1}$
8.       $\tilde{r}_{i}^{k} = \mathcal{F}(r_{i}^{k})$
9.       $\tilde{e}_{i}^{k} = \tilde{r}_{i}^{k} \circ \vartheta_i$
10.  $e_{i}^{k} = \mathcal{F}^{-1}(\tilde{e}_{i}^{k})$
11.  $u_{i}^{k+1} = v_{i}^{k+1} + e_{i}^{k}$
12. end
13. end
14. Compute loss function (3.4)
15. Apply Adam algorithm [37] to update parameters $\theta$
16. end
17. return learned FNS
Algorithm 2: FNS online testing

**Data:** PDE parameter $\eta$ and corresponding discrete system $A, f$

**Input:** Learned FNS, acceptable tolerance $tol$ and maximum number of iteration steps $MaxIterNum$

1. Setup: $\vartheta = \text{HyperNN}(\eta, \theta)$
2. $k = 0$
3. $u^0 = \text{zeros like } f$
4. $res = \frac{\|f - Au^k\|_2}{\|f\|_2}$
5. while $res > tol$ and $k < MaxIterNum$ do
6. $v^{k+1} = \Phi(u^k)$
7. $r^k = f - Av^{k+1}$
8. $\hat{v}^k = F(r^k)$
9. $\hat{e}^k = \hat{v}^k \circ \vartheta$
10. $e^k = F^{-1}(\hat{e}^k)$
11. $u^{k+1} = v^{k+1} + e^k$
12. $res = \frac{\|f - Au^k\|_2}{\|f\|_2}$
13. $k = k + 1$
14. end
15. return solution of $Au = f$

4 Numerical Experiments

We take the anisotropy equation, convection-diffusion equation, and Helmholtz equation as examples to demonstrate the performance of FNS. In all experiments, matrix-vector products are implemented by CNN based on Pytorch [38] platform. All code can be found at [https://github.com/cuichen1996/FourierNeuralSolver](https://github.com/cuichen1996/FourierNeuralSolver).

4.1 Anisotropy equation

Consider the anisotropy diffusion equation
\[
\begin{cases}
-\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\
u = 0, & \text{on } \partial\Omega,
\end{cases}
\]

(4.1)

the diffusion coefficient matrix
\[
C = C(\xi, \theta) = \begin{pmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & \xi
\end{pmatrix}
\begin{pmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{pmatrix},
\]

(4.2)

$0 < \xi < 1$ is the anisotropic strength, $\theta \in [0, \pi]$ is the anisotropic direction, $\Omega = (0, 1)^2$. We use bilinear FEM to discretize (4.1) with a uniform $n \times n$ quadrilateral mesh. The associated discrete system is shown in (2.2), where $N = (n - 1) \times (n - 1)$. We will carry out experiments for the following two cases.

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 [18]. For fixed $\theta = 0$, we randomly sample 20 distinct parameters $\xi$ from the distribution $\log_{10} \frac{1}{2} \sim U[0, 5]$ and obtain $\{A_i\}_{i=1}^{20}$ by discretizing (4.1) using bilinear FEM with $n = 256$. We randomly select 100 right-hand functions for each $A_i$. And each entry of $f$ is sampled from the Gaussian distribution $N(0, 1)$. Therefore there are $N_{\text{train}} = 2000$ 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 6.1.

FNS can take various kinds of $\Phi$. In this case, we use weighted Jacobi, Chebyshev semi-iterative (Cheby-semi), and Krylov methods. The weight of weighted Jacobi method is $2/3$. Krylov method uses the same subspace as [18]. Its basis vector is the output of a DenseNet [39]. For Chebyshev semi-iterative method, we provide a brief summary here. More details can refer to [40, 41].

If we vary the parameter of Richardson iteration at each step
\[
u^{k+1} = u^k + \tau_k (f - Au^k),
\]

(4.3)
and the maximum and minimum eigenvalues of $A$ are known, then $\tau_k$ can be determined as

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

(4.4)

where

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

(4.5)

are the roots of an $m$-order Chebyshev polynomial. Here, $\lambda_{\max}$ is obtained by the power method \[42\], 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 Chebyshev semi-iteration. We take $m = 10$, $\alpha = 3$. Figure 3 shows the convergence factor obtained by LFA. It can be observed that the smooth effect improves as $m$ increases. However, the high-frequency error along $y$ direction is also difficult to eliminate. When $\Phi$ is Jacobi method, we implement $\Phi$ five-times, and transform residual to frequency space to correct error. This is because employing the stationary method multi-times can enhance its smoothing effect.

Figure 3: The 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 iterations over test set as \[18\] does.

Table 1 shows test results of different solvers. It can be found that iteration steps of all solvers grow as anisotropic strength increases except for MG (line-Jacobi). The growth of FNS is substantially lower than Meta-MgNet and MG. When FNS employs the same $\Phi$ 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 of single-step iteration. It is also worth mentioning that the line-Jacobi smoother can only be applied to several specific $\theta$, i.e. $0, \pi/4, \pi/2, 3\pi/4, \pi$. However, FNS can be available for arbitrary $\theta$.

Table 1: The mean and standard deviation of the number of iterations required to achieve the stopping criterion over all tests for the anisotropy equation case 1. "-" means that it cannot converge within 10000 steps, and "\#" means that \[18\] does not provide test results for this parameter.

| $\xi$  | FNS(Cheby-semi) | FNS(Jacobi) | FNS(Krylov) | Meta-MgNet(Krylov) | MG(Jacobi) | MG(line-Jacobi) |
|-------|-----------------|-------------|-------------|--------------------|------------|-----------------|
| $\xi = 10^{-4}$ | 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 | - | - | - |

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 4(a) 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. Figure 4(b) shows the distribution of error before correction in frequency space. The result is consistent with the guidance of LFA, i.e., the error concentrated in the low-frequency modes. Figure 4(c) shows the distribution of error learned by $\mathcal{H}$ in the frequency space at this
time. Its distribution is largely similar to that of Figure 4(b). Figures 4(d-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](image)

Figure 4: The 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 $\mathcal{H}$.

### 4.1.2 Case 2: Generalization ability of anisotropic direction with fixed strength

We randomly sample 20 parameters $\theta$ 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 Section 4.1.1 does. Table 2 shows test results. It can be seen that whether $\Phi$ is Jacobi or Krylov, FNS can maintain robust performance in all situations, while the line-smoother is not available for these cases.

| $\theta$ | FNS(Jacobi) | FNS(Krylov) |
|---------|-------------|-------------|
| $0.1\pi$ | $300.2 \pm 23.95$ | $58.4 \pm 4.45$ |
| $0.2\pi$ | $252.4 \pm 34.81$ | $46.5 \pm 4.84$ |
| $0.3\pi$ | $269.3 \pm 36.70$ | $45.1 \pm 2.30$ |
| $0.4\pi$ | $356.4 \pm 39.69$ | $64.0 \pm 7.01$ |
| $0.5\pi$ | $338.4 \pm 32.07$ | $54.4 \pm 5.75$ |
| $0.6\pi$ | $265.0 \pm 29.96$ | $41.3 \pm 7.11$ |
| $0.7\pi$ | $266.5 \pm 25.07$ | $43.0 \pm 2.83$ |
| $0.8\pi$ | $316.7 \pm 17.26$ | $60.3 \pm 3.93$ |

Take $\theta = j\pi/10, j = 1, \ldots, 4, 6, \ldots, 9, \xi = 10^{-6}, n = 64$ and $\Phi$ is the weighted Jacobi method with weight $2/3$, Figure 5 shows the test results. The first row shows the weighted Jacobi method’s convergence factor $\mu_{\text{loc}}$ for each $\theta$ which is computed by LFA. The region of $\mu_{\text{loc}} \sim 1$ means that the error is difficult to eliminate. It can be found that these error components are distributed along the anisotropic direction. The second row depicts 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 $\Phi$ is difficult to eliminate. Line-smoother is not feasible for these $\theta$. 

![Figure 5](image)
Fourier Neural Solver for large sparse linear algebraic systems

4.2 Convection-diffusion equation

Consider the convection-diffusion equation

\[
\begin{cases}
  -\varepsilon \Delta u + u_x + u_y = 0, & \Omega = (0,1)^2, \\
  u = 0, & x = 0, 0 \leq y < 1 \text{ and } y = 0, 0 \leq x < 1, \\
  u = 1, & x = 1, 0 \leq y < 1 \text{ and } y = 1, 0 \leq x \leq 1,
\end{cases}
\] (4.6)

We use central difference method to discretize (4.6) on a uniform mesh with spatial size \( h \) in both \( x \) and \( y \) directions which yields a non-symmetric stencil

\[
\frac{1}{h^2} \begin{bmatrix}
  -h/2 - \varepsilon & h/2 - \varepsilon \\
  -4\varepsilon & h/2 - \varepsilon \\
  -h/2 - \varepsilon & -h/2 - \varepsilon
\end{bmatrix}.
\] (4.7)

For the requirement of stability, the central difference scheme needs to satisfy the Peclet condition

\[
Pe := \frac{h}{\varepsilon} \max(|a|, |b|) \leq 2,
\] (4.8)

which means that the central difference method cannot approximate the PDE solution when \( \varepsilon \) 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 FNS. In the next experiments, we will explore the diffusion- and convection-dominant cases, respectively.

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

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

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

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 Jacobi, CG, and MG (Jacobi) methods might diverge. Figure 8 shows that convergence factor of weighted Jacobi (\( \omega = 4/5 \)) for solving the system when \( \varepsilon = 10^{-2}, 10^{-3}, 10^{-6} \). It can be seen that when \( \varepsilon \) is small, the convergence factor of weighted Jacobi method for most frequency modes is bigger than 1, which causes this iterative method to diverge and be unsuitable as a smoother.
Fourier Neural Solver for large sparse linear algebraic systems

(a) Convergence factor of one times weighted Jacobi ($\omega = \frac{4}{5}$)

(b) Two times

(c) Three times

(d) Four times

(e) Five times

(f) Learned $\hat{e}$

Figure 6: The 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) $\varepsilon = 0.5$

(b) $\varepsilon = 0.1$

(c) $\varepsilon = 0.05$

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

(a) $\varepsilon = 10^{-2}$

(b) $\varepsilon = 10^{-3}$

(c) $\varepsilon = 10^{-6}$

Figure 8: The 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 $\Phi$ in Eq. (2.3), where $\mathcal{B}$ is a two-layer linear CNN with channels $1 \rightarrow 8 \rightarrow 1$, and the kernel size is $3 \times 3$. The $\Phi$ is trained together with $\mathcal{H}$, and the training hyperparameters are listed in Appendix 6.2. Figure 9(a) illustrates how this learned FNS is able to solve the linear system when $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. It is visible that FNS converges soon. Figure 9(b) shows the change of relative residual for FNS, GMRES, and BiCGSTAB($\ell$) ($\ell = 15$) with $\varepsilon = 10^{-6}$. It is clear that FNS has the fastest convergence rate.

![Figure 9](image)

Figure 9: (a): The change of relative residual with FNS iteration steps for different $\varepsilon$. (b): Comparison of FNS and GMRES, BiCGSTAB($\ell$) when $\varepsilon = 10^{-6}$.

4.3 Helmholtz equation

The Helmholtz equation we consider here is

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

where $\Omega = (0, 1)^2$, $\kappa$ is the wavenumber. We currently only take into account the zero Dirichlet boundary condition. We use the second order FDM to discretize (4.9) on a uniform mesh with spatial size $h$. The corresponding stencil reads

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

(4.10)

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 $\kappa = 125$. Take Krylov in [18] as $\Phi$, and $g(x) = 1$, the training hyperparameters are listed in Appendix 6.3. Figure 10 depicts how the relative residual decreased with different solvers. For $\kappa = 25$, FNS performs best for the first 300 steps, but BiCGSTAB performs better at the end. For $\kappa = 125$, FNS outperforms BiCGSTAB. And the GMRES results were too subpar to display in this case.

![Figure 10](image)

Figure 10: The change of relative residual for 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 the stationary method and frequency correction, which are used to eliminate errors in different Fourier modes. Numerical experiments show that FNS is more effective and robust than other solvers in solving anisotropy equation, convection-diffusion equation and Helmholtz equation. The core concepts discussed here can be available to a broad range of systems.

There is still a great deal of work to do. Firstly, we only considered uniform mesh in this paper. We will generalize FNS to non-uniform grids, by exploiting tools from geometric deep learning, such as graph neural network and graph Fourier transform. Secondly, as we discussed previously, the stationary method converges slowly or diverges in some situations, which prompts researchers to approximate solutions in other transform space. This is true for almost advanced iterative methods, including MG, Krylov subspace methods and 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 even learnable transforms based on data.

6 Training hyperparameters

6.1 Anisotropy equation

Table 3: Training hyperparameters for anisotropic equation.

| Method          | 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 4: HyperNN architecture parameters for Anisotropic 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=4,k=3,s=2,p=1)+Relu()
ConvTranspose2d(i=4,o=2,k=3,s=2,p=2)
AdaptiveAvgPool2d(n)
```

6.2 Convection-diffusion equation

Table 5: Training hyperparameters for convection-diffusion equation.

| Method     | learning rate | batch size | $K$  | xavier init | grad clip |
|------------|---------------|------------|------|-------------|-----------|
| FNS(Jacobi)| $10^{-4}$     | 100        | 10   | $10^{-2}$   | false     |
| FNS(Conv)  | $10^{-4}$     | 100        | 1 ~ 100 | $10^{-2}$   | 1.0       |

6.3 Helmholtz equation

References

[1] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. Templates for the solution of linear systems: building blocks for iterative methods. SIAM, 1994.

[2] Yousef Saad. Iterative methods for sparse linear systems. SIAM, 2003.
Fourier Neural Solver for large sparse linear algebraic systems

Table 6: 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=4,k=3,s=2,p=1)+Relu() | ConvTranspose2d(i=4,o=2,k=3,s=2,p=2) |

Table 7: 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 |

[3] Magnus R Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving. *Journal of research of the National Bureau of Standards*, 49(6):409, 1952.

[4] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[5] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[6] William L Briggs, Van Emden Henson, and Steve F McCormick. A multigrid tutorial. SIAM, 2000.

[7] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. Multigrid. Elsevier, 2000.

[8] Robert D Falgout. An introduction to algebraic multigrid. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2006.

[9] Jinchao Xu and Ludmil Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.

[10] Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabell, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

[11] Kevin Luna, Katherine Klymko, and Johannes P Blaschke. Accelerating gmres with deep learning in real-time. *arXiv preprint arXiv:2103.10975*, 2021.

[12] Gabriel D Weymouth. Data-driven multi-grid solver for accelerated pressure projection. *arXiv preprint arXiv:2110.11029*, 2021.

[13] Claudio Tomasi and Rolf Krause. Construction of grid operators for multilevel solvers: a neural network approach. *arXiv preprint arXiv:2109.05873*, 2021.

[14] Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12129–12140, 2021.

[15] Ru Huang, Ruipeng Li, and Yuanzhe Xi. Learning optimal multigrid smoothers via neural networks. *arXiv preprint arXiv:2102.12071*, 2021.

[16] Fan Wang, Xiang Gu, Jian Sun, and Zongben Xu. Learning-based local weighted least squares for algebraic multigrid method. *Available at SSRN 4110904*.

[17] Vladimir Fanaskov. Neural multigrid architectures. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[18] Yuyan Chen, Bin Dong, and Jinchao Xu. Meta-mgnet: Meta multigrid networks for solving parameterized partial differential equations. *Journal of Computational Physics*, 455:110996, 2022.

[19] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. Black-box learning of multigrid parameters. *Journal of Computational and Applied Mathematics*, 368:112524, 2020.

[20] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pages 2415–2423. PMLR, 2019.

[21] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pages 6489–6499. PMLR, 2020.
Fourier Neural Solver for large sparse linear algebraic systems

Table 8: HyperNN architecture parameters for Helmholtz equation. Notations in ConvTranspose2d are: i: in channels; o: out channels; k: kernel size; s: stride; p: padding.

```plaintext
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)
```

[22] Paola F Antonietti, Matteo Caldana, and Luca Dede. Accelerating algebraic multigrid methods via artificial neural networks. *arXiv preprint arXiv:2111.01629*, 2021.

[23] Antonio Stanziola, Simon R Arridge, Ben T Cox, and Bradley E Treeby. A helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *Journal of Computational Physics*, 441:110430, 2021.

[24] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[25] Yael Azulay and Eran Treister. Multigrid-augmented deep learning preconditioners for the helmholtz equation. *arXiv preprint arXiv:2205.11025*, 2022.

[26] Yogi A Erlangga, Cornelis W Oosterlee, and Cornelis Vuik. A novel multigrid based preconditioner for heterogeneous helmholtz problems. *SIAM Journal on Scientific Computing*, 27(4):1471–1492, 2006.

[27] Henri Calandra, Serge Gratton, Julien Langou, Xavier Pinel, and Xavier Vasseur. Flexible variants of block restarted gmres methods with application to geophysics. *SIAM Journal on Scientific Computing*, 34(2):A714–A736, 2012.

[28] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

[29] Stefanos Nikolopoulos, Ioannis Kalogeris, Vissarion Papadopoulos, and George Stavroulakis. Ai-enhanced iterative solvers for accelerating the solution of large scale parametrized linear systems of equations. *arXiv preprint arXiv:2207.02543*, 2022.

[30] Rita Stanaityte et al. *ILU and Machine Learning Based Preconditioning For The Discretized Incompressible Navier-Stokes Equations*. PhD thesis, 2020.

[31] Ayano Kaneda, Osman Akar, Jingyu Chen, Victoria Kala, David Hyde, and Joseph Teran. A deep gradient correction method for iteratively solving linear systems. *arXiv preprint arXiv:2205.10763*, 2022.

[32] Nils Margenberg, Dirk Hartmann, Christian Lessig, and Thomas Richter. A neural network multigrid solver for the navier-stokes equations. *Journal of Computational Physics*, 460:110983, 2022.

[33] Nils Margenberg, Robert Jendersie, Thomas Richter, and Christian Lessig. Deep neural networks for geometric multigrid methods. *arXiv preprint arXiv:2106.07687*, 2021.

[34] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[35] Gerard LG Sleijpen and Diederik R Fokkema. Bicgstab (ell) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis.*, 1:11–32, 1993.

[36] Paul N Swarztrauber. The methods of cyclic reduction, fourier analysis and the facr algorithm for the discrete solution of poisson’s equation on a rectangle. *Siam Review*, 19(3):490–501, 1977.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
[39] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

[40] Gene H Golub and Richard S Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods. Numerische Mathematik, 3(1):157–168, 1961.

[41] Mark Adams, Marian Brezina, Jonathan Hu, and Ray Tuminaro. Parallel multigrid smoothing: polynomial versus gauss–seidel. Journal of Computational Physics, 188(2):593–610, 2003.

[42] RV Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsauflosung. ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 9(1):58–77, 1929.