Neural Operators Meet Conjugate Gradients: The FCG-NO Method for Efficient PDE Solving

Alexander Rudikov [1 2] **Vladimir Fanaskov** [2] **Ekaterina Muravleva** [2 3] **Yuri M. Laevsky** [4] **Ivan Oseledets** [1 2]

Abstract

Deep learning solvers for partial differential equations typically have limited accuracy. We propose to overcome this problem by using them as preconditioners. More specifically, we apply discretization-invariant neural operators to learn preconditioners for the flexible conjugate gradient method (FCG). Architecture paired with novel loss function and training scheme allows for learning efficient preconditioners that can be used across different resolutions. On the theoretical side, FCG theory allows us to safely use nonlinear preconditioners that can be applied in $O(N)$ operations without constraining the form of the preconditioners matrix. To justify learning scheme components (the loss function and the way training data is collected) we perform several ablation studies. Numerical results indicate that our approach favorably compares with classical preconditioners and allows to reuse of preconditioners learned for lower resolution to the higher resolution data.

1. Introduction

The recent surge of interest in learning solution operators and surrogates for partial differential equations (PDEs) leads to several new approaches and architectures (Azizadenesheli et al., 2023), (Karniadakis et al., 2021), (Kovachki et al., 2021). Most notable, combination of functional methods with deep learning (spectral convolutions (Rippel et al., 2015), FNO (Li et al., 2020)); operator-valued kernels (Kadri et al., 2016), (Batlle et al., 2024); random feature model in Banach space; (Nelsen & Stuart, 2021) architectures based on universal approximation for operators (DeepONet (Lu et al., 2019) and its variants).

According to the large-scale benchmarks, typical accuracy of such methods is about $0.1\% - 1\%$ relative L_2 norm (Takamoto et al., 2022), (Lu et al., 2022), (de Hoop et al., 2022). On the other hand, classical numerical methods for PDEs are usually consistent, so they can reach arbitrary accuracy on a sufficiently fine grid.

Is it possible to combine the consistency with the efficiency of deep learning methods? One way around this is to build a hybrid system, that is, to replace some parts of the classical method with deep learning components (e.g., (Bar-Sinai et al., 2019), (Kochkov et al., 2021), (Greenfeld et al., 2019)). In the present contribution we focus on elliptic boundary value problems and show how to utilize neural operator (Li et al., 2020) (NO) with flexible conjugate gradient method (Notay, 2000) (FCG). As one can see on Figure 1, our approach indeed allows one to retain consistency which leads to much better error on grids with higher resolutions.

We are not the first ones, who propose to learn preconditioners. The previous contributions include at least (Hsieh et al., 2019), (Zhang et al., 2022), (Cui et al., 2022), (Häusner et al., 2023), (Li et al., 2023). However, our approach is the first one with three unique properties: (i) – we learn nonlinear operator with asymptotic complexity $O(N)$, which allows us not to prescribe a structure of the preconditioners matrix to have an efficient matrix-vector product, (ii) – our nonlinear operator is discretization-invariant so it can be applied at different resolutions, (iii) – we have convergence guarantees based on FCG theory built in (Notay, 2000). We provide more details on comparison with other approaches in Section 4.

To summarize, our contributions are:

1. We propose to train a neural operator as a nonlinear preconditioner for the flexible conjugate gradient method. We demonstrate that neural operators can be trained on lower resolutions and serve as an efficient preconditioner for higher resolutions.

2. We provide a novel learning scheme that involves drawing random vectors from the Krylov subspace. Ablation study shows that abandoning Krylov subspace

[1]Artificial Intelligence Research Institute [2]Skolkovo Institute of Science and Technology [3]Sberbank PJSC [4]Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Correspondence to: Alexander Rudikov <A.Rudikov@skoltech.ru>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

and using random right-hand sides[1] seriously impair training.

3. We put forward a novel loss based on energy norm that comes with convergence guarantees and significantly outperforms L_2 loss usually used for training.

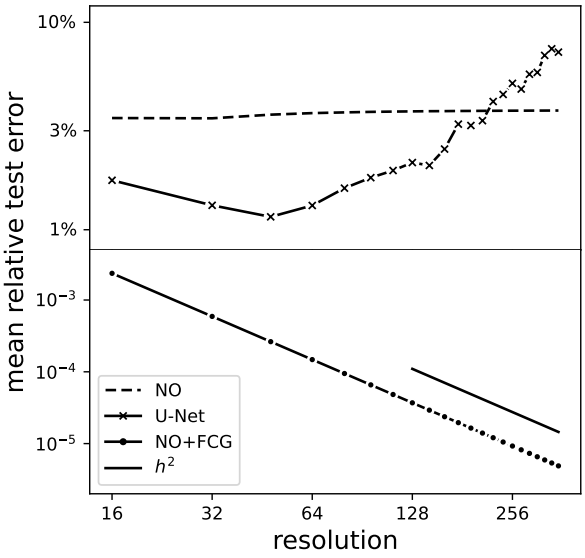

Figure 1. Comparison of accuracy for three approaches: U-Net – classical deep learning architecture, NO – neural operator, NO+FCG – hybrid approach advocated in the present article. All approaches are applied to the dataset for the Poisson equation (see Section 3.2). Architectures (number of layers, parameters, etc) are the same for all resolutions. Due to the prominent difference between accuracies, two distinct scales are used in the y axis. One can observe that owing to the finite receptive field the performance of U-Net deteriorates with the increase of resolution. The neural operator provides the same accuracy with resolution increase – this is a highly-praised "discretization invariance." For NO+FCG, the error decreases with the increase of resolution in the same way as for classical numerical methods.

Code and datasets are available on https://github.com/arudikov/FCG.

2. Neural Operator as a Preconditioner

Here, we introduce the PDE that we solve and provide a brief overview of discretization, iterative schemes, and linear and nonlinear preconditioning. After that, we outline the construction of the loss function, neural operator, and training strategy.

[1]In our case, we sample random right-hand sides as random trigonometric polynomials $\mathcal{P}(N_1, N_2, \alpha) = \Big\{ f(x) = \mathcal{R}\big(g(x)\big) : \mathcal{R}(c) \simeq \mathcal{N}(0, I) \Big\}$, where $g(x) = \sum_{m=0}^{N_1} \sum_{n=0}^{N_2} \dfrac{c_{mn} \exp\left(2\pi i (m x_1 + n x_2)\right)}{(1+m+n)^{\alpha}}$.

2.1. Elliptic Equations

A family of PDEs we consider in this article has a form

$$-\sum_{ij=1}^{2} \frac{\partial}{\partial x_i} \left(a(x) \frac{\partial u(x)}{\partial x_j} \right) = f(x) \\ x \in \Gamma \equiv (0,\ 1)^2,\ u(x)\big|_{x \in \partial\Gamma} = 0, \tag{1}$$

where $\partial\Gamma$ is a boundary of the unit hypercube Γ, and $a(x) \geq \epsilon > 0$. Applying the finite element method or finite difference method (FDM) to (1) we can obtain a large sparse linear algebraic system with a symmetric positive definite matrix:

$$Au = f;\ u,\ f \in \mathbb{R}^n,\ A \in \mathbb{R}^{n \times n}, A \succ 0. \tag{2}$$

The system can be solved in $O\left(n^3\right)$ operations by Gauss elimination, but for large n^3 one often resorts to iterative techniques that can better leverage the special sparsity stricture of matrix A.

2.2. Preconditioning of Linear System

One of the commonly used iterative methods for solving (2) is conjugate gradient (CG). However, in practice, the CG often converges slowly, and the convergence also depends on the distribution of the eigenvalues and the initial residual. One way to enhance convergence is to precondition the system to improve its spectral properties. The modified linear system becomes

$$B^{-1}Au = B^{-1}f, \tag{3}$$

where $B \in \mathbb{R}^{n \times n}$ is called the preconditioner.[2] To be successful, preconditioner B should have several properties: (i) it should improve spectral properties of A, e.g., the condition number of $B^{-1}A$ is much smaller than the condition number of A; (ii) B easily invertible, i.e., $Bg = r$ is cheap to solve; (iii) for preconditioner conjugate gradient (PCG) (Axelsson, 1996) one need to ensure B is symmetric positive definite. All these requirements greatly complicate the construction of preconditioners. Fortunately, with a slight decrease in numerical efficiency, one can switch to a less restrictive set of nonlinear preconditioners.

2.3. Nonlinear Preconditioners

A projection iterative method suitable for more general preconditioners appeared in (Notay, 2000). The algorithm is called Flexible Conjugate Gradient (FCG) and is given in

[2]Note that formally for $A \succ 0$ one should use $B^{-\frac{1}{2}}AB^{-\frac{1}{2}}$ to preserve properties of the system. But since $B^{-\frac{1}{2}}AB^{-\frac{1}{2}}$ and $B^{-1}A$ are spectrally equivalent one can rewrite conjugate gradient algorithm to work directly with $B^{-1}A$ using B inner product. See Section 9.2 from (Saad, 2003) for more details.

Algorithm 1 Flexible Conjugate Gradients

Input: A, $\mathcal{B}$, f, $m_{\max} > 0$, iter.
Ensure: $u_{\text{iter}}, r_{\text{iter}}$.
Initialize $u_0 \leftarrow \mathcal{N}(0,\, 1) \in \mathbb{R}^n$, $r_0 \leftarrow f - Au_0 \in \mathbb{R}^n$.
for $i = 0$ **to** iter $- 1$ **do**
  $w_i \leftarrow \mathcal{B}(r_i)$
  $m_i \leftarrow \min(i,\, \max(1,\ \bmod\ (i,\, m_{\max} + 1)))$
  $p_i \leftarrow w_i - \sum_{k=i-m_i}^{i-1} \dfrac{(w_i, s_k)}{(p_k, s_k)}\, p_k$
  $s_i \leftarrow Ap_i$
  $u_{i+1} \leftarrow u_i + \dfrac{(p_i, r_i)}{(p_i, s_i)}\, p_i$
  $r_{i+1} \leftarrow r_i - \dfrac{(p_i, r_i)}{(p_i, s_i)}\, s_i$
end for

Algorithm 1. It uses many vectors instead of one so it is less effective than CG from a computational perspective. But in return, one can consider B in Equation (3) to be the nonlinear operator $\mathcal{B}$. The only technical requirement is given in the following result from (Notay, 2000):

Theorem 2.1. *Let A, $B \in \mathbb{R}^{n\times n}$ be symmetric positive definite matrices and $\mathcal{B} : \mathbb{R}^n \to \mathbb{R}^n$ be a nonlinear operator. Let f, u_0 be the vectors of $\mathbb{R}^n$, and let $\{r_i\}_{i=0,1,\ldots}$, $\{p_i\}_{i=0,1,\ldots}$, $\{u_i\}_{i=1,2,\ldots}$ be the sequences of vectors generated by applying Algorithm 1 to A, $\mathcal{B}$, f, and u_0 with some given sequences of non-negative integer parameters $\{m_i\}_{i=0,1,\ldots}$.*

If, for any i,

$$\frac{\left\|\mathcal{B}(r_i) - B^{-1}r_i\right\|_B}{\left\|B^{-1}r_i\right\|_B} \leqslant \varepsilon_i < 1, \tag{4}$$

then

$$\frac{\left\|u - u_{i+1}\right\|_A}{\left\|u - u_i\right\|_A} \leqslant \frac{\kappa\big(B^{-1}A\big)\cdot\gamma_i - 1}{\kappa\big(B^{-1}A\big)\cdot\gamma_i + 1},$$

where $\gamma_i = \dfrac{1+\varepsilon_i}{1-\varepsilon_i}\cdot\dfrac{\left(1+\varepsilon_i^2\right)^2}{\left(1-\varepsilon_i^2\right)}$, *and* $\left\|u\right\|_A = \sqrt{(u, Au)}$.

In plain English, Theorem 2.1 states that as long as nonlinear preconditioner $\mathcal{B}$ is close enough to the linear preconditioner B (in a sense of equation (4)), FCG converges with roughly the same speed as CG with B taken as a preconditioner. This powerful result allows us to use a nonlinear neural network as a preconditioner and simultaneously provide a loss, suitable for training this neural network. Both these points are explained in more detail in the next section.

2.4. Learning Scheme

2.4.1. Loss Functions

The principal idea of the article is to select a family of neural networks with weights θ as nonlinear preconditioner $\mathcal{B}(r;\theta)$ for FCG described in Algorithm 1. Theorem 2.1 can be interpreted in terms of optimization target. To increase the rate of convergence that depends on $\kappa(B^{-1}A)$ we need to use $B^{-1} = A^{-1}$. Therefore, we can find parameters from the loss

$$L_{\text{opt}}(\theta) = \max_r \frac{\left\|\mathcal{B}(r;\theta) - A^{-1}r\right\|_A}{\left\|A^{-1}r\right\|_A},\ \text{s.t.}\ \|r\|_2 = 1. \tag{5}$$

If we were able to minimize Equation (5) with respect to θ we would find a neural network that provides a nonlinear preconditioner as close to A^{-1} as possible. Unfortunately (5) is a hard minimax problem so we relax it to

$$L_{\text{mean}}(\theta) = \mathbb{E}_r \frac{\left\|\mathcal{B}(r;\theta) - A^{-1}r\right\|_A}{\left\|A^{-1}r\right\|_A}. \tag{6}$$

To tie this loss function to the usual operator learning framework described in (Li et al., 2020) we suppose that parameters of PDE (1) are drawn from some distributions in Banach space $a \sim p_a$ and $f \sim p_f$ after that the final loss, which we call Notay loss after the author of (Notay, 2000), becomes

$$L_{\text{Notay}}(\theta) = \mathbb{E}_{r,a,f} \frac{\left\|\mathcal{B}(r;\theta) - A^{-1}r\right\|_A}{\left\|A^{-1}r\right\|_A}, \tag{7}$$

where A depends on a via discretization, r depends on f and the distribution of initial guess $u_0 \sim \mathcal{N}(0, I)$. Even though r is not independent from other variables we still have freedom in the choice of distribution for r.

The simplest case is to compute the residual from u_0 and f. However, one expects that distribution for r will drift from $r = f - Au_0$, $f \sim p_f$, $u_0 \sim \mathcal{N}(0, I)$ since both CG and FCG seek a solution in the Krylov subspace

$$\mathcal{K}_m(A, r_0) = \text{Span}\Big\{r_0, Ar_0, \ldots, A^{m-1}r_0\Big\}. \tag{8}$$

So it is natural to consider a more general distribution for r drawing vectors from Krylov subspaces using distributions for u_0 and f

$$r \sim p_{\mathcal{K}_m}(r) \Leftrightarrow r \in \mathcal{K}_m(A, r_0),\ f \sim p_f,\ u_0 \sim \mathcal{N}(0, I). \tag{9}$$

In other words, to draw residuals from $p_{\mathcal{K}_m}(r)$ we draw u_0 and f and run CG for m iterations.

Of course a valid alternative to Equation (7) is an ordinary L_2 loss function

$$L_2(\theta) = \mathbb{E}_{r,a,f} \frac{\left\|\mathcal{B}(r;\theta) - A^{-1}r\right\|_2}{\left\|A^{-1}r\right\|_2}. \tag{10}$$

Since L_2 loss (10) is commonly used to learn the solution operator for the elliptic equation, we will also try to use it to learn the preconditioner.

2.4.2. NEURAL OPERATOR

In principle, losses (7) and (10) are suitable for arbitrary neural networks. However, it is desirable to learn a preconditioner for one low-resolution discretization and apply it to discretizations with higher resolutions. Models uniquely suited for that task are neural operators (NOs) that are known to be consistent under the change of resolution. Following (Li et al., 2020), (Fanaskov & Oseledets, 2022) we consider spectral neural operators (SNO) as the primary model. Additionally, we check how the proposed method works using Fourier neural operator (FNO) (Li et al., 2020) and classical machine learning architectures, including DilResNet (Stachenfeld et al., 2021) and U-Net (Ronneberger et al., 2015) as preconditioners in the FCG algorithm. A brief description of the architectures of all four models is presented in the Appendix A.

3. Experiments

Here, we provide evidence of the effectiveness of the proposed approach. For that purpose, we design several experiments in $D = 2$. We start with the description of the training details.

3.1. Training Details

According to (9), the residuals should be derived from m iterations of CG. Thus, we propose the following training scheme Figure 2. Following this scheme, we start with the generation of train $\mathcal{D}_{\text{train}}$ and test $\mathcal{D}_{\text{test}}$ datasets that consist of sparse matrices A and the right-hand sides f of (2). For train dataset $\mathcal{D}_{\text{train}}$, we generate N_{train} samples of u_{exact} and A. Therefore, we calculate right-hand sides as $f = Au_{\text{exact}}$. Then, we use m iterations of the CG (not necessarily until complete convergence) on the training dataset. The CG output: u_{iter}, r_{iter} which is in the Krylov subspace, is used for the training of the neural operator (NO). The trained NO is used as the preconditioner for the FCG algorithm applied on the test dataset $\mathcal{D}_{\text{test}}$.

To accelerate the training of NO, we eliminate the need to invert the matrix A in the loss function (7). To accomplish this, we use already generated u_{exact} for the train dataset and the fact that

$$A^{-1}r_i = A^{-1}(Au_{\text{exact}} - Au_i) = u_{\text{exact}} - u_i = e_i,$$

where e_i is an error at iteration i. After that, we can rewrite Notay loss (7) in the form

$$L_{\text{Notay}}(\theta) = \mathbb{E}_{r,a,f}\frac{\left\|\mathcal{B}(r;\theta) - e\right\|_A}{\left\|e\right\|_A}, \tag{11}$$

and L_2-loss in the form

$$L(\theta) = \mathbb{E}_{r,a,f}\frac{\left\|\mathcal{B}(r;\theta) - e\right\|_2}{\left\|e\right\|_2}. \tag{12}$$

3.2. Datasets

To verify the proposed approach, we use two versions of the elliptic equation (1) in the domain $\Gamma \equiv (0,1)^D$.

First, we define random trigonometric polynomials

$$\mathcal{P}\big(N_1, N_2, \alpha\big) = \Big\{f(x) = \mathcal{R}\left(g(x)\right) : \mathcal{R}(c) \simeq \mathcal{N}(0, I)\Big\},$$

where

$$g(x) = \sum_{m=0}^{N_1}\sum_{n=0}^{N_2}\frac{c_{mn}\exp\left(2\pi i(mx_1 + nx_2)\right)}{(1+m+n)^\alpha}.$$

For the first dataset, called Poisson, we use random trigonometric polynomials for f:

$$a(x) = I;\ f(x) \simeq \mathcal{P}(5,5,2). \tag{13}$$

For the second one, called Diffusion, we use trigonometric polynomials for both a and f:

$$a(x) \simeq \mathcal{P}(5,5,2) + 10;\ f(x) \simeq \mathcal{P}(5,5,2). \tag{14}$$

Both above mentioned datasets are smooth, and one may be interested whether proposed approach is applicable to non-smooth datasets. Therefore we provide additional results for the experiments with non-smooth diffusion coefficients in Appendix C.

3.3. Ablation Study

In this article, we have proposed three new components:

1. The nonlinear preconditioner $\mathcal{B}$ in the form of a NO in the Algorithm 1;
2. The loss function derived from the Theorem 2.1 for training this NO;
3. The novel sampling strategy of a train dataset from the Krylov subspace.

Next, we present the results of the ablation study for all these components using SNO as the preconditioner.

3.3.1. FCG VS. CLASSICAL TECHNIQUES

We perform several experiments to compare learned preconditioners with classical approaches. More specifically, we use Jacobi, symmetric Gauss-Seidel (Saad, 2003) and incomplete LU (ILU) preconditioner implemented in SuperLU library (Demmel, 1999).

Recall, that Jacobi iteration reads

$$x^{n+1} = x^n + D(A)^{-1}(f - Ax^n), \tag{15}$$

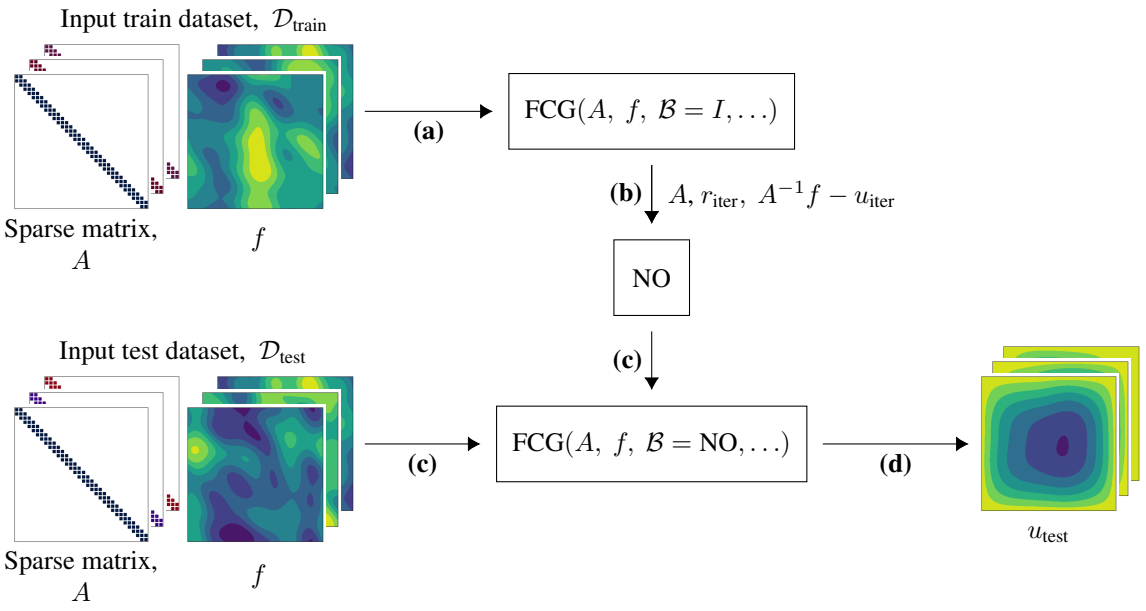

Figure 2. The full scheme of the proposed approach: starts from the input train dataset, $\mathcal{D}_{\text{train}} = (A, f)$, where $f = Au_{\text{exact}}$. **(a)** Submit $\mathcal{D}_{\text{train}}$ to the CG (FCG with $\mathcal{B} = I$) with 100 iterations to generate residuals r_{iter} from the Krylov subspaces. **(b)** Train the NO on the FCG output r_{iter} with the use of A, u_{iter} for the calculation of L_{Notay}. **(c)** Apply the FCG with $\mathcal{B} = \text{NO}$ with the test dataset, $\mathcal{D}_{\text{test}}$. **(d)** Output u_{test}.

Dataset	grid	NO+FCG	CG	Jacobi(4)	GS(1)	GS(4)	ILU(1)	ILU(8)
Poisson	32	**9**	74	30	27	13	69	27
	64	**14**	130	58	47	23	110	77
	128	**20**	216	112	78	37	185	128
Diffusion	32	**9**	75	32	27	13	69	27
	64	**14**	132	61	46	22	110	74
	128	**19**	215	115	78	36	177	128

Table 1. Comparison of FCG with classical preconditioning techniques. The table contains the first iteration number i such that $\|r_i\|_2 \,/\, \|r_0\|_2 \leq 10^{-6}$ for different resolutions.

where $D(A)$ is a diagonal part of matrix A. Symmetric Gauss-Seidel iteration reads

$$x^{n+1/2} = x^n + L(A)^{-1}(f - Ax^n),$$
$$x^{n+1} = x^{n+1/2} + U(A)^{-1}(f - Ax^{n+1/2}), \tag{16}$$

where $L(A)$ and $U(A)$ are lower and upper triangular parts of A.

When classical iteration are used to approximate $A^{-1}z$ they are used for k steps (e.g., Jacobi(4) means 4 iteration of the form (15)) with $x^0 = 0$ and $b = z$.

ILU is a LU decomposition of sparse matrix A with control of fill-in. For the details on the particular version used, we refer to (Demmel, 1999). In Table 1 ILU(k) number k corresponds to parameter *filling_factor* in documentation. It prescribes the number of nonzero elements in the preconditioner. For example, for 5-point stencil discretization of the Poisson equation, the preconditioner formed with ILU(1) and ILU(8) have $\simeq 1.5\times$ and $\simeq 6.7\times$ more nonzero elements that original matrix.

The results appear in Table 1. As one can see, the learned preconditioner allows for faster convergence than standard approaches on all resolutions tested. GS(4) provides the second best result. Note, however, that unlike GS(4) (and ILU preconditioner), our approach does not require one to solve linear problems with large sparse triangular matrices, so it is much faster when parallel architectures are available.

In addition, we explore how the Notay loss (7) differs by iteration for two methods: CG and NO+FCG, see Figure 3.

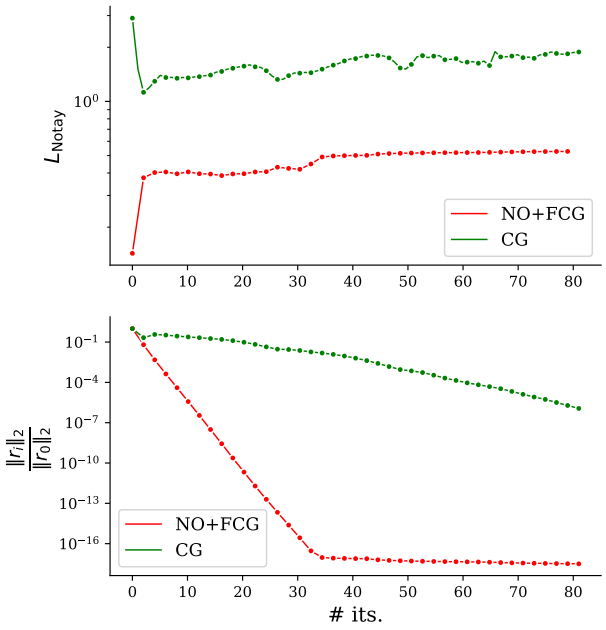

Figure 3. The behavior of L_{Notay} and the decline of residuals by iteration for Poisson equation with grid $= 32$ in cases of CG and NO+FCG.

Dataset	grid	$\|r_i\|_2/\|r_0\|_2$ 10^{-3}	10^{-6}	10^{-12}
Poisson	32	43%	34%	9%
	64	58%	31%	14%
	128	74%	40%	33%
Diffusion	32	22%	21%	5%
	64	32%	32%	11%
	128	66%	44%	42%

Table 2. The difference between wall-clock times needed to drop initial residual by three different factors for FCG and CG is assessed by the formula: $\left(1 - t_{\text{FCG}}/t_{\text{CG}}\right) \cdot 100\%$. In table, there are results for FCG with SNO as the preconditioner. In all cases, NO was trained on residuals from Krylov subspace, $r \sim p_{\mathcal{K}_m}(r)$ with Notay loss.

We can calculate the L_{Notay} for CG as it is FCG with $\mathcal{B} = I$. We can see that, for the proposed method, we have a fast convergence and $L_{\text{Notay}} < 1$. It is consistent with the Theorem 2.1. For the CG case, there are $L_{\text{Notay}} > 1$ and a slow convergence.

One may point out that applying FCG with NO as a preconditioner can be slower in terms of wall-clock time not iterations. Therefore, we measure the difference between times for convergence of classical CG and FCG with SNO as a preconditioner using formula:

$$\left(1 - t_{\text{FCG}}/t_{\text{CG}}\right) \cdot 100\%.$$

The results for SNO are presented in the Table 2. NO+FCG demonstrates enhanced convergence efficiency, resulting in substantial reductions in both the number of iterations and the processing time. Moreover, this time advantage becomes greater as the grid size increases. The results for other models can be found in Appendix A.6.

3.3.2. L_{NOTAY} Loss vs. L_2 Loss

For this experiment, we have generated training datasets $\mathcal{D}_{\text{train}}$ for Poisson (13) and Diffusion (14) equations with $N_{\text{train}} = \text{grid}$. We chose this value because the higher the resolution, the more samples are needed to train NO. To make a comparison of losses, we generated residuals from the Krylov subspace by utilizing 100 iterations of CG. In total, NO was trained on $100 \cdot N_{\text{train}}$ residuals and errors. The errors were calculated as $e_i^j = A_j^{-1} f_j - u_i^j$, where $i = 1, \ldots, 100$, $j = 1, \ldots, N_{\text{train}}$ and u_i^j is the solution on i-th iteration of CG for j-th sample. A detailed description of NO and training details are available in Appendix A.1. For NO training, we used Notay and L_2 losses in the forms (11) and (12) respectively.

The test datasets $\mathcal{D}_{\text{test}}$ consists of $N_{\text{test}} = 20$ samples. Then we run N_{iter} of FCG with trained NO as the preconditioner. We chose $N_{\text{iter}} = 2 \cdot \text{grid}$ because higher resolution requires more iterations to converge. The parameter m_{max} in Algorithm 1 was selected to be 20. The numbers of iterations needed to drop the initial residual by factors $10^3, 10^6, 10^{12}$ are illustrated in Table 3.

As can be seen from the Table 3, the number of iterations required to reduce residuals by any factor increased when we use L_2 loss instead of L_{Notay} loss. For factors $10^3, 10^6, 10^{12}$, this increase was more than 25%, 50%, 60%, respectively. For L_{Notay} loss, the number of iterations for factor 10^{12} increases less than 1.55 times, with a double increase in resolution. For L_2 loss, this number of iterations grows more than 1.55 times with double increase in resolution. Additionally, the missing results for grid $= 128$ means that there was no convergence for this case (see Figure 9 in Appendix D).

Furthermore, we explore how the values of (7) differ by iteration for NO trained with different losses (L_{Notay} and L_2), see Figure 4. L_{Notay} correlates with rate of convergence according to Theorem 2.1. We can see that, for both cases, we have $L_{\text{Notay}} < 1$. However, training a NO with the proposed loss (11) gives lower values of function (7) in all iterations of the FCG than training with a standard L_2-loss.

		L_{Notay}			L_2		
		$\|r_i\|_2/\|r_0\|_2$			$\|r_i\|_2/\|r_0\|_2$		
Dataset	grid	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}
Poisson	32	4	9	20	5	15	34
	64	5	14	31	7	22	53
	128	6	20	48	21	86	210
Diffusion	32	4	9	31	8	22	50
	64	5	14	36	11	36	80
	128	5	19	47	—	—	—

Table 3. The number of iterations for FCG with $\mathcal{B} = \text{NO}$ needed to drop initial residual by three different factors. In table, there are two cases: **(a)** NO trained with Notay loss in form (11); **(b)** NO trained with and with L_2-loss in form (12). In both cases, NO was trained on residuals from Krylov subspace, $r \sim p_{\mathcal{K}_m}(r)$.

		$r \sim p_{\mathcal{K}_m}(r)$			$r \sim p_{\mathcal{K}_0}(r)$		
		$\|r_i\|_2/\|r_0\|_2$			$\|r_i\|_2/\|r_0\|_2$		
Dataset	grid	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}
Poisson	32	4	9	20	4	10	21
	64	5	14	31	5	18	67
	128	6	20	48	7	49	153
Diffusion	32	4	9	31	5	23	56
	64	5	14	36	6	—	—
	128	5	19	47	10	—	—

Table 4. The number of iterations for FCG with $\mathcal{B} = \text{NO}$ needed to drop initial residual by three different factors. In table, there are two cases: **(a)** NO trained on residuals from Krylov subspace; **(b)** NO trained on residuals obtained from random right-hand sides. In both cases, NO was trained with Notay loss in the form (11).

3.3.3. FCG WITH NO TRAINED ON DIFFERENTLY GENERATED RESIDUALS

Train datasets $\mathcal{D}_{\text{train}}$ for both equations had $N_{\text{train}} = \text{grid}$. For residuals $r \sim p_{\mathcal{K}_m}(r)$, the training procedure is the same as in Section 3.3.2. For random residuals, we generated $100 \cdot N_{\text{train}}$ samples of $u_i^j \sim \mathcal{N}(0, I)$ and $f_i^j \sim p_f$, where $i = 1, \ldots, 100$, $j = 1, \ldots, N_{\text{train}}$. Therefore, the residuals and errors were calculated as follows: $r_i^j = f_i^j - A_j u_i^j$, $e_i^j = A_j^{-1} f_i^j - u_i^j$. We trained NO with Notay loss using the architecture and training setup described in Appendix A.

The test datasets $\mathcal{D}_{\text{test}}$ contain $N_{\text{test}} = 20$ samples. The FCG algorithm has the same number of iterations and the same parameter $m_{\max}$ as in Section 3.3.2. The results are shown in Table 4.

It can be seen from the data in Table 4 that for the simplest case (Poisson equation, grid $= 32$), we got almost the same results for both types of residuals. However, with increasing resolution, there was a significant deterioration in convergence for residuals not from the Krylov subspace. For the Poisson equation, the number of iterations for factor 10^{12} increased more than 2 times when the resolution doubled. For the Diffusion equation, the algorithm stopped converging when the grid is larger than 32 (see Figure 10, Figure 11 in Appendix D).

3.4. Different Grid Approach

Training a neural operator on data with a finer grid takes longer. This motivated our decision to utilize a coarse grid-trained preconditioner in the FCG algorithm for fine grid applications. SNO is suitable for this task as it can be easily applied to fine resolution after training on coarse one. The testing process becomes more complex when conducted on a higher resolution than the one used during training. Therefore, we trained NO on more samples than in previ-

Dataset	Train grid	Test grid	$\|r_i\|_2/\|r_0\|_2$ 10^{-3}	10^{-6}	10^{-12}
Poisson	32	64	9	18	39
	64	128	12	29	68
	32	128	24	61	141
	64	256	26	67	150
	128	256	16	45	103
Diffusion	32	64	8	18	42
	64	128	9	24	59
	32	128	17	57	148
	64	256	24	87	233
	128	256	16	46	112

Table 5. Number of iterations for FCG with $\mathcal{B} = \text{NO}$ needed to drop initial residual by three different factors. The table consists of results for different grids on training and testing.

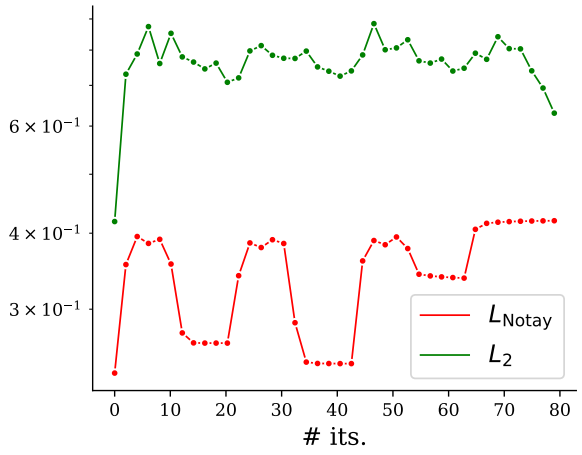

Figure 4. The behavior of L_{Notay} by iteration for Diffusion equation with grid $= 32$ in cases of NO trained with two different losses (L_{Notay} and L_2).

ous experiments. There were $N_{\text{train}} = 100$ samples in the training datasets $\mathcal{D}_{\text{train}}$. For generation residuals and errors, we used 100 iterations in CG. The details of the architecture and the training process are presented in Appendix A.

We tested FCG with NO on $N_{\text{test}} = 20$ samples. The resolutions of the test datasets $\mathcal{D}_{\text{test}}$ were twice or four times higher than the resolutions of the train datasets $\mathcal{D}_{\text{train}}$. Thus, FCG needed more iterations to converge. We selected $N_{\text{iter}} = 300$ for all cases. The results are demonstrated in Table 5.

The results from Table 5 demonstrate that the operator can serve as an effective preconditioner even if it is trained at a lower resolution.

3.5. Results for Other Models

We evaluated the number of iterations needed to reach a specific reduction in residuals (e.g., $\|r\|_2/\|r_0\|_2 \simeq 10^{-3}$) for FCG when using different models as preconditioners. The results for FNO, DilResNet and U-Net are presented in the Table 6. All models gave faster convergence of the algorithm as a preconditioner than SNO. However, SNO has the least number of parameters, therefore its application inside the algorithm takes less time.

4. Related Research

Literature on constructing preconditioners is vast, so we do not attempt a systematic review. In place of that, we will mention several contributions that allow us to frame our research properly.

As a first class of work, we would like to mention methods that construct preconditioners by solving auxiliary optimization problems (e.g., loss (7) that we used). Typically, such works restrict somehow the form of matrix M used as a preconditioner and optimize $\left\|I - M^{-1}A\right\|$ or $\|M - A\|$. For example, in (Tyrtyshnikov, 1992) author showed how to efficiently construct general circulant preconditioners, and in (Grote & Huckle, 1997) authors exploited sparsity constraints.

A similar strategy is widely used with models based on neural networks. For example, in (Cui et al., 2022) authors train a neural network to output eigenvalues of preconditioner in Fourier basis in effect reproducing the circulant preconditioner. Similarly, authors of (Häusner et al., 2023) utilize a graph neural network with sparsity constraints on Cholesky

		FNO			DilResNet			U-Net		
		$\|r_i\|_2/\|r_0\|_2$			$\|r_i\|_2/\|r_0\|_2$			$\|r_i\|_2/\|r_0\|_2$		
Dataset	grid	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}
Poisson	32	2	7	15	2	3	6	2	5	11
	64	4	10	26	2	3	7	2	7	19
	128	6	17	40	2	6	13	3	8	30
Diffusion	32	2	8	19	2	5	11	2	6	12
	64	3	9	29	2	5	11	2	7	16
	128	5	14	36	2	6	20	3	11	38

Table 6. The number of iterations for FCG with $\mathcal{B} = \text{NO}$ needed to drop initial residual by three different factors. In table, there are the results for three different models trained with Notay loss in form (11). In all cases, NOs was trained on residuals from Krylov subspace, $r \sim p_{\mathcal{K}_m}(r)$.

factors to train linear preconditioner for CG.

The second class of works is learning generalized preconditioners, i.e., matrix M used to transform the linear equation $MAx = Mb$ to improve the convergence of base iterations (Richardson, Jacobi, multigrid). As an example of such approaches, we can mention (Hsieh et al., 2019) and (Zhang et al., 2022). In the first contribution authors train the linear U-Net model as a preconditioner for Richardson iterations, in the second contribution authors pursue a similar idea with DeepONet architecture but with Jacobi and multigrid.

Finally, one recent contribution that is closest in spirit to the present research is (Kopaničáková & Karniadakis, 2024). In this paper, authors used flexible GMRES with DeepONet preconditioner. Unfortunately, DeepONet is less suitable for processing grids with different resolutions, so authors were forced to invent several intricated algebraic techniques to make the network work for the general grid.

Compared with the mentioned works, our approach is more direct since FCG, which we use as a base iterative method, is not restricted to working with linear preconditioners. Besides, we directly learn nonlinear operators, and with that, we avoid the need to consider structured matrices that guarantee cheap matrix-vector products. Finally, the proposed loss function is markedly different from what is used in other contributions.

5. Conclusion

In the present research, we have suggested using a trained neural operator as a nonlinear preconditioner for the flexible conjugate gradient method. The results of our study demonstrated the superiority of the proposed approach over utilizing different classical preconditioners. In addition, we introduced a novel loss function derived from the energy norm that guarantees convergence and achieves superior results compared to the L_2 loss function used in training. Moreover, we have implemented a novel learning scheme incorporating random vectors derived from the Krylov subspace. Based on the results of the ablation study, the utilization of classical random right-hand sides instead of the Krylov subspace severely diminished the effectiveness of training. Our findings also demonstrated that neural operators can be trained at lower resolutions and effectively act as a cost-effective preconditioner for higher resolutions.

The limitation of the proposed method is that the matrix A should be symmetric positive definite as we use CG. Our potential for utilizing our approach on higher resolutions is limited by GPU memory.

Future research directions include a comprehensive comparison of our proposed method with alternative approaches for constructing trainable preconditioners. Preliminary test results of comparison with approach proposed in (Li et al., 2023) are described in Appendix B. It is also planned to explore the proposed approach with non-uniform mesh. By utilizing different polynomials in SNO, we can extend our approach to non-uniform grids. Moreover, we plan to explore other groups of PDEs for which the large sparse linear algebraic system has a symmetric positive definite matrix.

Acknowledgements

We would like to thank Vladislav Trifonov for providing the results of the PCG method proposed in (Li et al., 2023) for comparison with the proposed approach in this work.

The work was supported by the Analytical center under the RF Government (subsidy agreement 000000D730321P5Q0002, Grant No. 70-2021-00145 02.11.2021)

Impact Statement

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

References

Axelsson, O. *Iterative solution methods*. Cambridge university press, 1996.

Azzizadenesheli, K., Kovachki, N., Li, Z., Liu-Schiaffini, M., Kossaifi, J., and Anandkumar, A. Neural Operators for Accelerating Scientific Simulations and Design. *arXiv preprint arXiv:2309.15325*, 2023.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Batlle, P., Darcy, M., Hosseini, B., and Owhadi, H. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024.

Cui, C., Jiang, K., Liu, Y., and Shu, S. Fourier Neural Solver for large sparse linear algebraic systems. *Mathematics*, 10(21):4014, 2022.

de Hoop, M. V., Huang, D. Z., Qian, E., and Stuart, A. M. The cost-accuracy trade-off in operator learning with neural networks. *arXiv preprint arXiv:2203.13181*, 2022.

Demmel, J. W. Superlu users' guide. 1999.

Fanaskov, V. and Oseledets, I. Spectral neural operators. *arXiv preprint arXiv:2205.10573*, 2022.

Gautschi, W. *Orthogonal polynomials: computation and approximation*. OUP Oxford, 2004.

Golub, G. H. and Welsch, J. H. Calculation of Gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

Greenfeld, D., Galun, M., Basri, R., Yavneh, I., and Kimmel, R. Learning to optimize multigrid PDE solvers. In *International Conference on Machine Learning*, pp. 2415–2423. PMLR, 2019.

Grote, M. J. and Huckle, T. Parallel preconditioning with sparse. *SIAM Journal on Scientific Computing*, 18(3): 838–853, 1997.

Häusner, P., Öktem, O., and Sjölund, J. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. *arXiv preprint arXiv:2305.16368*, 2023.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning neural PDE solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Kadri, H., Duflos, E., Preux, P., Canu, S., Rakotomamonjy, A., and Audiffren, J. Operator-valued kernels for learning from functional response data. 2016.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Kopaničáková, A. and Karniadakis, G. E. Deeponet based preconditioning strategies for solving parametric linear systems of equations. *arXiv preprint arXiv:2401.02016*, 2024.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Li, Y., Chen, P. Y., Matusik, W., et al. Learning Preconditioner for Conjugate Gradient PDE Solvers. *arXiv preprint arXiv:2305.16432*, 2023.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Nelsen, N. H. and Stuart, A. M. The random feature model for input-output maps between banach spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, 2021.

Notay, Y. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22(4):1444–1460, 2000.

Rippel, O., Snoek, J., and Adams, R. P. Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Saad, Y. *Iterative methods for sparse linear systems*. SIAM, 2003.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35: 1596–1611, 2022.

Trefethen, L. N. Is Gauss quadrature better than Clenshaw–Curtis? *SIAM review*, 50(1):67–87, 2008.

Trefethen, L. N. and Bau, D. *Numerical linear algebra*. SIAM, 2022.

Tripura, T. and Chakraborty, S. Wavelet neural operator: a neural operator for parametric partial differential equations. *arXiv preprint arXiv:2205.02191*, 2022.

Tyrtyshnikov, E. E. Optimal and superoptimal circulant preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 13(2):459–473, 1992.

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

A. Models' Details

In addition to the primary model (SNO), this research employs three alternative models as preconditioners. As a rule, neural PDE solvers are either Neural Operators or classical architectures used for image processing. Since our approach is architecture-agnostic, we include results for both types of neural networks. Next, we describe architectures of all models and a part of the results of using them as a preconditioner in the FCG algorithm.

A.1. SNO

We consider spectral neural operators (SNO) (Fanaskov & Oseledets, 2022) with linear integral kernels

$$u \leftarrow \int dx A_{ij} p_j(x) \left(p_i, u\right),$$

where $p_j(x)$ are orthogonal or trigonometric polynomials.

Layers with linear integral kernels can be described in terms of analysis and synthesis matrices. To describe these matrices consider a finite set of orthogonal polynomials (with weight function w) $p_n(x),\ n = 0, \dots, N-1$. The synthesis operator is

$$f(x) = \sum_{i=0}^{N-1} p_i(x) c_i \iff f = \mathcal{S}c,$$

where c is a vector with expansion coefficients $c_i,\ i = 0, \dots, N-1$ defined by

$$c_i = \frac{1}{h_i} \int p_i(x) f(x) w dx,\ h_i = \int p_i^2(x) w dx.$$

When the Gauss quadrature (Golub & Welsch, 1969; Gautschi, 2004; Trefethen, 2008) is used for the integral evaluation, it is easy to show that the analysis operator has a form

$$c = \mathcal{A}f = D^{-1} \mathcal{S}^\top \big(f(x) \odot w\big),$$

where

$$D = \operatorname{diag}\Big(p_0^\top \big(p_0 \odot w\big), \dots, p_{N-1}^\top \big(p_{N-1} \odot w\big)\Big).$$

So, the whole integral kernel has a form $u \leftarrow \mathcal{S}L\mathcal{A}u$ where L is a linear operator on $\mathbb{R}^N$. For example, Fourier neural operator (Li et al., 2020) employees FFT as analysis and inverse FFT as synthesis; neural operators from (Fanaskov & Oseledets, 2022) utilizes DCT-II to work with Chebyshev polynomials; wavelet neural operator (Tripura & Chakraborty, 2022) applies FWT and inverse FWT.

We use SNO in Fourier basis (see (Fanaskov & Oseledets, 2022)) with encoder-processor-decoder architecture. We change number of features according to grid, see Table 7. Number of SNO layers is 4 and number of orthogonal polynomials is 20. We utilize GeLU as an activation function.

Dataset	grid	SNO	FNO	DilResNet	U-Net
Poisson	32	32	32	24	10
	64	32	32	24	10
	128	80	32	32	16
Diffusion	32	48	48	24	10
	64	64	64	24	10
	128	80	80	32	16

Table 7. Number of features for all models and datasets.

A.2. FNO

The Fourier Neural Operator (FNO), first introduced in (Li et al., 2020), is characterized by an architecture that includes an encoder, multiple Fourier layers, and a decoder. The $(i+1)$-th Fourier layer can be expressed as

$$z_{i+1} = \sigma\left(\mathcal{F}^{-1}\left(R_i \cdot \mathcal{F}\left(z_i\right)\right) + \text{conv}(z_i)_{W_i} + b_i\right),$$

where σ is an activation function, $\mathcal{F}$ and $\mathcal{F}^{-1}$ are Fast Fourier and inverse Fourier transforms, R_i and W_i are weight matrices, b_i is a bias vector and conv stands for convolution with kernel size 1.

In order to use the FNO as a preconditioner, we used the following parameters: 4 Fourier layers, 16 modes and GELU as an activation function. The number of features in each layer depends on dataset and grid (see Table 7).

A.3. DilResNet

The usual form of dilated residual network was proposed in (Stachenfeld et al., 2021). In this work, DilResNet employs a configuration consisting of 4 blocks. The number of features in each layer is determined by the dataset and grid, as detailed in Table 7. Each block comprises a series of convolutions with strides of $[1, 2, 4, 8, 4, 2, 1]$ and a kernel size of 3. In addition, skip connections are applied after each block and the activation function ReLU is utilized.

A.4. U-Net

We used the usual form of U-Net proposed in (Ronneberger et al., 2015). The U-Net architecture is characterized by a series of levels, where each level has roughly half of resolution as the previous one, and the feature count doubles. We applied a sequence of 3 convolutions on each level, followed by max pooling, and then used transposed convolution for upsampling. After upsampling, 3 additional convolutions were applied on each level. The U-Net used here consists of 4 layers and utilizes ReLU activation function. The number of features in each layer varies based on the dataset and the grid, as outlined in Table 7.

A.5. Training details

Dataset	grid	ν	ν decay / epoch	weight decay	N_{epoch}	N_{batch}
Poisson	32, 64	$5 \cdot 10^{-4}$	$0.5/50$	10^{-2}	150	32
	128	$5 \cdot 10^{-4}$	$0.5/50$	10^{-2}	150	8
Diffusion	32, 64	$5 \cdot 10^{-4}$	$0.5/50$	10^{-2}	200	16
	128	$5 \cdot 10^{-4}$	$0.5/50$	10^{-2}	200	4

Table 8. Training details: ν — learning rate, ν decay / epoch — weight decay per epoch, N_{epoch} — number of epoch used for training, N_{batch} — batch size.

All models are trained using the same training procedure. The details of this procedure is described in Table 8.

A.6. Results

In addition to the results described in the main part of the article, we measure wall-clock time needed to reach particular accuracy with and without a preconditioner. This allows for a fair comparison that takes into account additional computations needed to apply preconditioner. The difference between times needed for the convergence of FCG and CG is assessed by the formula: $\left(1 - t_{\text{FCG}}/t_{\text{CG}}\right) \cdot 100\%$. The results are presented in the Table 9. It can be seen from the results that despite the fact that FNO gives faster convergence than SNO, in terms of iterations, SNO is faster in terms of time. Moreover, the results show that the maximum advantages of using neural networks as preconditioners is achieved with the highest data resolution (128×128).

		$\left(1 - t_{\text{FCG}}/t_{\text{CG}}\right) \cdot 100\%$											
		SNO			FNO			DilResNet			U-Net		
Dataset	grid	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}
Poisson	32	43%	34%	9%	53%	16%	—	44%	57%	46%	47%	32%	7%
	64	58%	31%	14%	46%	21%	—	70%	74%	66%	73%	45%	16%
	128	74%	40%	33%	64%	29%	21%	85%	69%	68%	80%	62%	34%
Diffusion	32	22%	21%	5%	24%	3%	—	35%	24%	6%	23%	26%	17%
	64	32%	32%	11%	35%	30%	—	51%	57%	52%	61%	51%	44%
	128	66%	44%	42%	51%	41%	37%	77%	70%	59%	72%	56%	38%

Table 9. The difference between wall-clock times needed to drop initial residual by three different factors for FCG and CG is assessed by the formula: $\left(1 - t_{\text{FCG}}/t_{\text{CG}}\right) \cdot 100\%$. In table, there are results for four models: SNO, FNO, DilResNet, U-Net. In all cases, NO was trained on residuals from Krylov subspace, $r \sim p_{\mathcal{K}_m}(r)$.

B. Another Preconditioning Method

For a symmetric positive definite linear system, one can form the preconditioner in the form of a Cholesky decomposition (Trefethen & Bau, 2022) $P = LL^\top$, to obtain a preconditioned linear system $P^{-1}Ax = P^{-1}f$. Conjugate gradient with preconditioner in such form called preconditioned conjugate gradient (PCG). In the research (Li et al., 2023), the approach of finding lower triangular matrix L using graph neural networks (GNNs) is proposed:

$$L(\theta) = \text{GNN}_\theta(A, f).$$

The GNN was trained using the following loss function:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \left\| L(\theta)L(\theta)^\top x_i - f_i \right\|_2^2$$

Thus, we compare the convergence rate of our approach with the approach proposed in (Li et al., 2023) (see Table 10). In this preliminary test, we apply both approaches only to the Poisson dataset. It can be seen that FCG with a non-linear preconditioner manages to converge faster than PCG with a trainable, yet linear, preconditioner (see Table 10 and Figure 5).

		FCG			PCG		
		$\|r_i\|_2/\|r_0\|_2$			$\|r_i\|_2/\|r_0\|_2$		
Dataset	grid	10^{-3}	10^{-6}	10^{-12}	10^{-3}	10^{-6}	10^{-12}
Poisson	32	4	9	20	8	19	39
	64	5	14	31	11	23	47
	128	6	20	48	20	39	75

Table 10. The number of iterations for FCG with $\mathcal{B} =$ SNO and for PCG needed to drop initial residual by three different factors.

C. More Complex Dataset

The results shown in the main text of the article were obtained on datasets with smooth diffusion coefficient and right-hand sides. To show the ability of the algorithm FCG with NO as a preconditioner to find a solution for a dataset with non-smooth coefficients, we decided to generate two datasets with non-smooth coefficients $a_1(x)$ and $a_2(x)$:

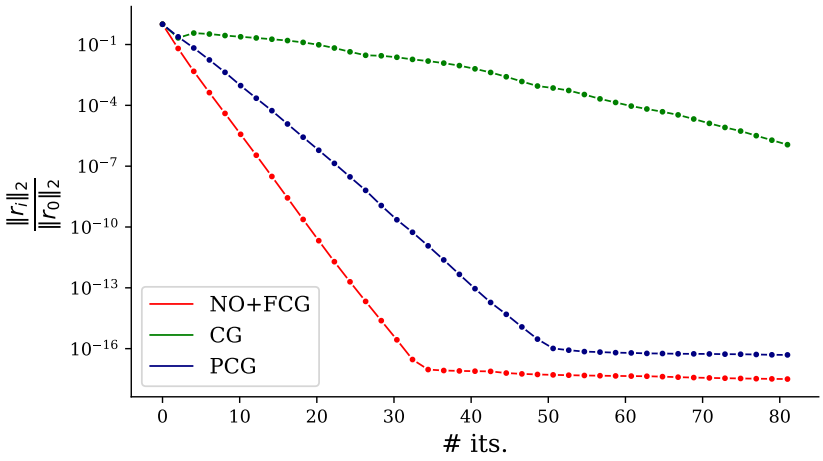

Figure 5. The decline of residuals by iteration for Poisson equation with grid $= 32$ for FCG with SNO, classical CG and PCG proposed in (Li et al., 2023).

$$a_1(x) = \begin{cases} 1, & x \geq 0.5; \\ \varepsilon, & x < 0.5, \end{cases} \tag{17}$$

$$a_2(x) = \begin{cases} 0.1 \cdot \mathcal{P}(5, 5, 2) + 1, & x \geq 0.5; \\ \varepsilon, & x < 0.5, \end{cases} \tag{18}$$

where $\mathcal{P}(5, 5, 2)$ is random trigonometric polynomials, and ε is a constant chosen randomly from uniform distribution ($\varepsilon \sim 0.1 \cdot \mathcal{U}(0, 1)$). The examples of such diffusion coefficients are presented in Figure 6.

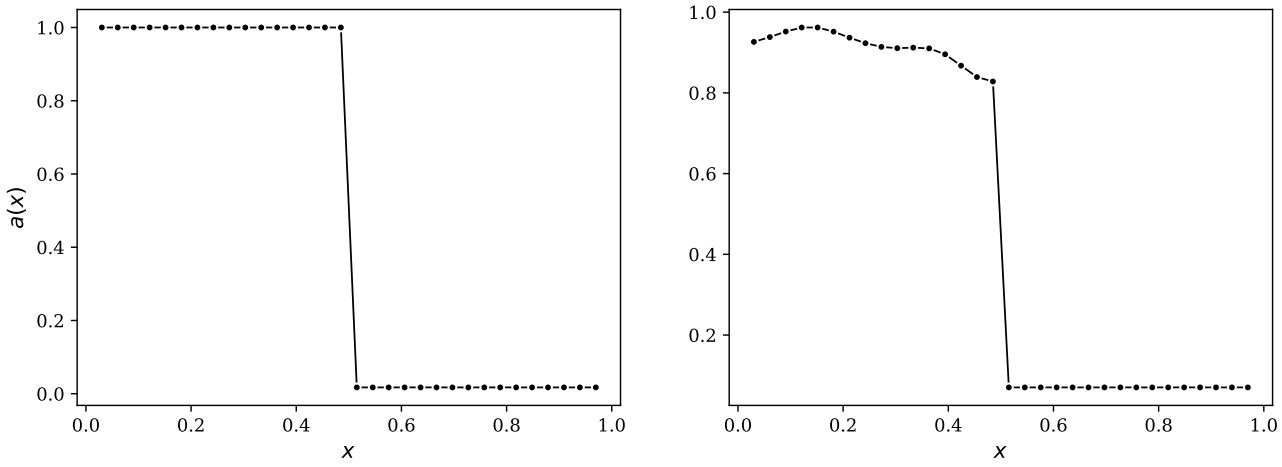

Figure 6. Two types of non-smooth diffusion coefficients: $a_1(x)$ (17) and $a_2(x)$ (18).

We tested our approach on 32×32 and 64×64 grids and compared it with the CG algorithm. The convergence process of both algorithms applied to non-smooth datasets with a grid of 32 is shown in Figure 7, with a grid of 64 in Figure 8. The results indicate that even for non-smooth datasets, using NO as a preconditioner significantly increases the convergence rate.

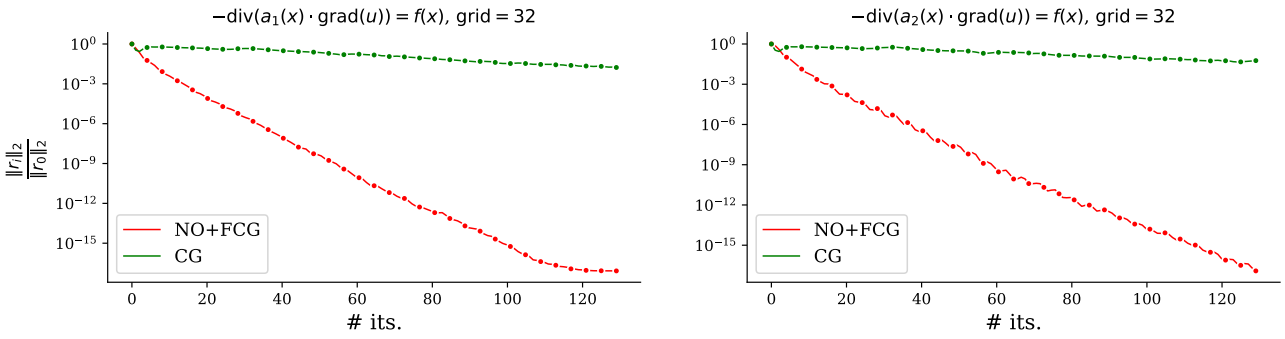

Figure 7. The decline of residuals by iteration for Diffusion equation with grid $= 32$ in cases of non-smooth coefficients.

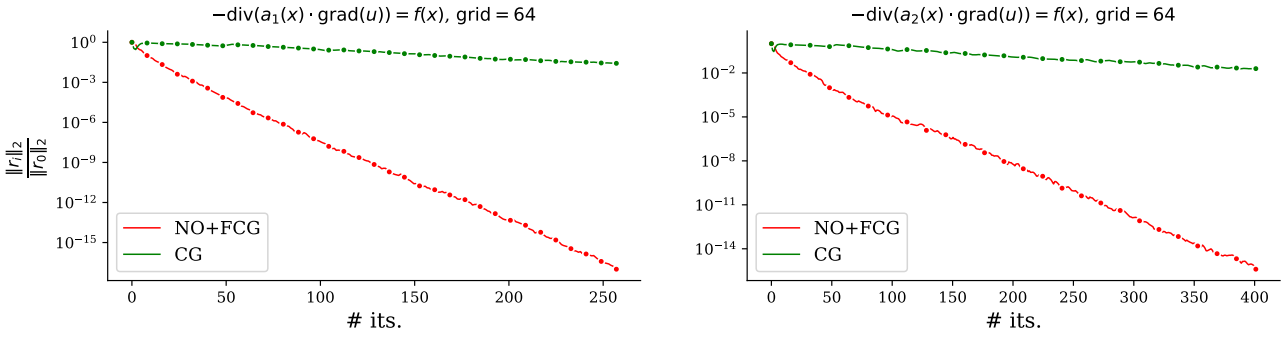

Figure 8. The decline of residuals by iteration for Diffusion equation with grid $= 64$ in cases of non-smooth coefficients.

D. FCG Convergence for Experiments with Missing Values in Results

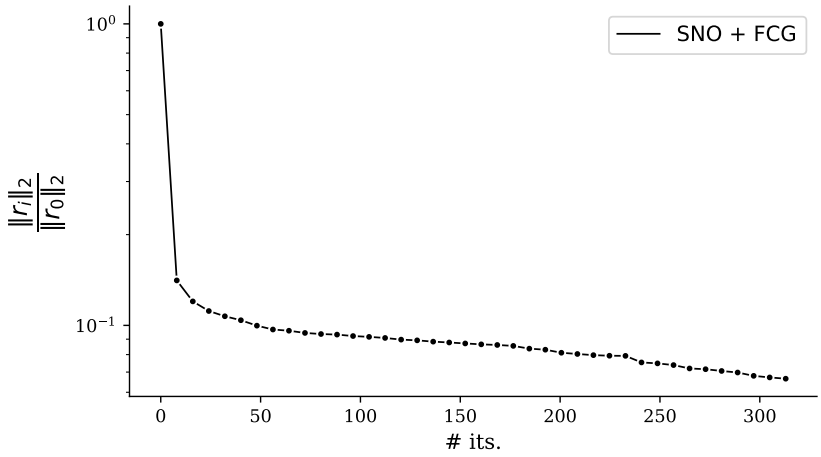

Figure 9. The decline of residuals by iteration for Diffusion equation with grid $= 128$ in L_2, $r \sim p_{\mathcal{K}_m}(r)$ case.

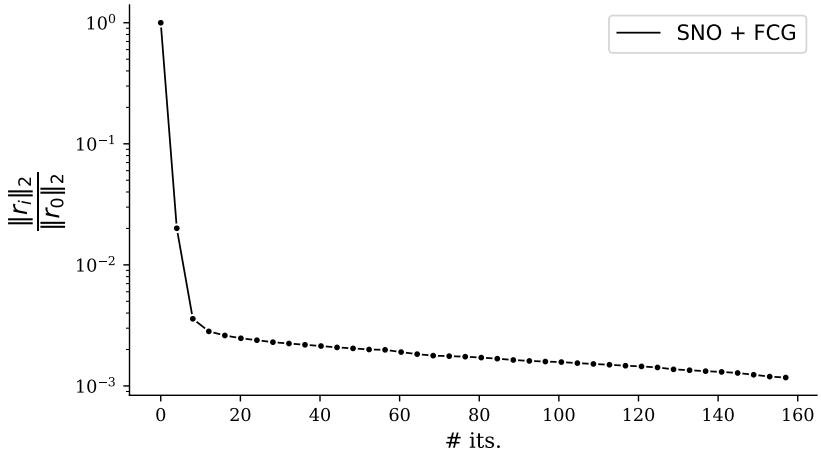

Figure 10. The decline of residuals by iteration for Diffusion equation with grid $= 64$ in L_{Notay}, $r \sim p_{\mathcal{K}_0}(r)$ case.

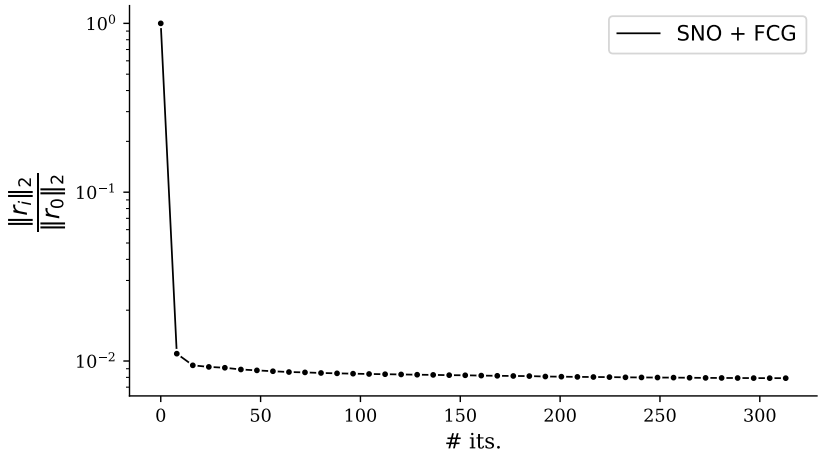

Figure 11. The decline of residuals by iteration for Diffusion equation with grid $= 128$ in L_{Notay}, $r \sim p_{\mathcal{K}_0}(r)$ case.