East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.2022-328.230423

Vol. **13**, No. 3, pp. 740-758
August 2023

Linearized Learning with Multiscale Deep Neural Networks for Stationary Navier-Stokes Equations with Oscillatory Solutions

Lizuo Liu[1], Bo Wang[2] and Wei Cai[1,*]

[1]*Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA.*
[2]*LCSM(MOE), School of Mathematics and Statistics, Hunan Normal University, Changsha, Hunan 410081, P.R. China.*

Received 22 November 2022; Accepted (in revised version) 23 April 2023.

Congratulations on the 60th birthday of Professor Tao Tang with friendship and admiration for his many achievements!

Abstract. In this paper, we present linearized learning methods to accelerate the convergence of training for stationary nonlinear Navier-Stokes (NS) equations. To solve the stationary nonlinear NS equation with deep neural networks, we integrate linearizations of the nonlinear convection term in the NS equations into the training process of multi-scale deep neural network (DNN) approximations of the NS solution. Four forms of linearizations are considered. We solve highly oscillating stationary flows in complex domains utilizing the proposed linearized learning with multiscale neural networks. The results show that multiscale deep neural network combined with the linearized schemes can be trained much faster and accurately than regular fully connected DNNs.

AMS subject classifications: 35Q68, 65N99, 68T07

Key words: Linearized learning, stationary Navier-Stokes equation, multi-scale deep neural network.

1. Introduction

Deep neural network (DNN) machine learning methods have been researched as alternative numerical methods for solving partial differential equations arising from many practical engineering problems. The deep learning framework for solving those kinds of problems uses the given partial differential equations as regularization in the loss function during training, where the auto-differentiation can be applied to the inputs of the neural network. Since auto differentiation with respect to the inputs of neural network are built-in, thus there is no need for any pre-generated meshes in the solution domain. Therefore,

*Corresponding author. *Email address:* cai@smu.edu (W. Cai)

such a framework has the potential of being a flexible meshless method to solve governing equations from fluid and solid mechanics in complex geometries, as an alternative method to traditional finite element method. Moreover, these methods have shown much power in solving high dimensional parabolic PDEs [6, 12, 18].

Fluid mechanics, on the other hand, has also been one of the active research fields for the applications of neural network with physical information as regularizations. In the work of [2, 13], the authors proposed a method that combines the Navier-Stokes (NS) equation with visualization data to predict the velocity field and pressure field, with synthetic data in [13] and real experimental imaging data in [2], respectively. In [5], a physical-informed neural network is used for solving the Reynolds-averaged Navier-Stokes equations with Reynolds-stress components ($\overline{u^2}, \overline{uv}$, and $\overline{v^2}$) as extra outputs of the neural networks. Rao *et al.* [14] proposed a mixed-variable scheme with Cauchy stress tensor to eliminate the intractability of the complex form of naive Navier-Stokes equation and its high-order derivatives (e.g., ∇^2) and this scheme was applied to learn the steady flow and the transient flow passing a cylinder respectively. Furthermore, Oldenburg *et al.* [11] proposed the Geometry Aware Physics Informed Neural Network to handle the Navier-Stokes equations with irregular geometry where they utilize the shape encoding network, i.e., an encoder, to reduce the geometry dimensions to a size-fixed latent vector k and k will be the input of two additional neural networks, one to handle the boundary constraints and one to handle physical information, i.e., the governing PDEs. In the meantime, the error estimations for neural networks to approximate the Navier-Stokes equations has been studied in [4].

Recent studies on DNNs have shown that they have a frequency dependence performance in learning solution of PDEs and fitting functions. Namely, the lower frequency components of the solution are learned first and quickly compared with the higher frequency components [17]. Several attempts have been made to remove such a frequency bias for the DNNs. The main idea is to convert the higher frequency content of the solution to a lower frequency range so the conventional DNNs can learn the solution in acceptable training epochs. One way to achieve this goal is to use phase shifts [3] while the other is to introduce a multiscale structure into the DNNs [10] where in which sub-neural networks with different scales will target different ranges of the frequency in the solutions. The PhaseDNN has been shown to be very effective for high frequency wave propagation while the MscaleDNN [10] has been used to learn highly oscillatory Stokes flow solutions in complex domains [16] as well as high dimensional PDEs [18].

Most of the previous works are focusing on Linear PDEs. The learning of the solution of linear PDEs via least squared residuals of the PDEs is in some sense equivalent to a fitting problem in the frequency domain in view of the Parseval's identity of Fourier transforms. So it is natural the performance improvements of multiscale DNN also holds for learning the solution of linear PDEs.

Additional difficulties arise when there are nonlinearities introduced in the PDEs. Based on the results from Jin *et al.* [8], it is found that it could take $\mathcal{O}(10^4)$ epochs to solve a simple domain problem, thus ineffective and impractical especially when highly oscillating problems are to be considered. Also, the MscaleDNN applied directly to the nonlinear Navier-Stokes equation did not produce the same large improvement over conventional DNNs as

in the case of the linear Stokes equations [16]. To handle such issue, we developed a linearized learning procedure for the Navier-Stokes equation by integrating linearizations of the Navier-Stokes equation in the loss function and dynamically updating the linearization as the learning is being carried out. Numerical results demonstrated the fast convergence of this approach in producing highly accurate approximation to oscillatory solutions of the Navier-Stokes equations.

The rest of the paper will be organized as follows. Section 2.1 will review several iterative schemes for solving Navier-Stokes equations commonly used by the finite element methods that inspires the linearized learning scheme of this paper. Section 3 will introduce the multiscale DNN structure for learning oscillatory solutions with wide range of frequencies, and then four linearized learning schemes for the Navier-Stokes equation will be proposed in Section 3.2. Numerical tests of the linearized learning schemes will be conducted for 2-D oscillatory flows in a domain containing one or multiple random cylinder(s) in Section 4. Finally, conclusion and future work will be discussed in Section 5.

2. Iterative Method for Stationary Navier–Stokes Equations

2.1. Stationary Navier-Stokes equations

The problem considered in this paper is the following stationary Navier-Stokes equations:

$$\begin{aligned} &(\mathbf{u}\cdot\nabla)\mathbf{u}-\nu\Delta\mathbf{u}+\nabla p=\mathbf{f} && \text{in } \Omega,\\ &\nabla\cdot\mathbf{u}=0 && \text{in } \Omega,\\ &\mathbf{u}=\mathbf{g} && \text{on } \partial\Omega, \end{aligned} \tag{2.1}$$

where Ω is an open bounded domain in $\mathbb{R}^d, d=2,3$. $\mathbf{u}$ are the stationary flow velocity, p is the pressure, ν is the kinematic viscosity, $\mathbf{f}$ is the external source term. In this paper, we consider the incompressible flow, thus the constraints $\nabla\cdot\mathbf{u}=0$ are considered. The boundary conditions are Dirichlet boundary conditions.

To solve the stationary Navier-Stokes equation in a least squared minimal residual approach, intuitively, the PDEs (2.1) are written in a system of first order equations as in [1] by introducing an extra velocity-gradient term, $\underline{\mathbf{U}}$, where $\underline{\mathbf{U}}_{ij}=(\partial u_i/\partial x_j)$, $i,j=1,\ldots,d$ for $d=2,3$ such that $\nabla\cdot\underline{\mathbf{U}}=\Delta\mathbf{u}$. In this paper, we consider the velocity-pressure formulation for the Navier Stokes equations as our benchmark loss, following the results of [16],

$$-\nu\nabla\cdot\underline{\mathbf{U}}+\underline{\mathbf{U}}\cdot\mathbf{u}+\nabla p=\mathbf{f} \quad \text{in } \Omega, \tag{2.2a}$$

$$\underline{\mathbf{U}}-(\nabla\mathbf{u})^T=0 \quad \text{in } \Omega, \tag{2.2b}$$

$$\nabla\cdot\mathbf{u}=0 \quad \text{in } \Omega. \tag{2.2c}$$

Similarly, to obtain an equation for the pressure p, we take divergence on both sides of Eq. (2.2a) and apply the Eq. (2.2c) to arrive at

$$\Delta p+2(-u_x v_y+u_y v_x)=\nabla\cdot\mathbf{f}, \tag{2.3}$$

where the subscripts $\cdot_x, \cdot_y$ means the derivative with respect to x, y, respectively. Then, a loss function for the velocity gradient (Vg) and a velocity-pressure (VP) formulation of the NS equations can be defined as

$$\begin{aligned} L_{VgVP}(\theta_u, \theta_p, \theta_{\underline{\mathbf{U}}}) := & \| \nu \nabla \cdot \underline{\mathbf{U}} - \underline{\mathbf{U}} \cdot \mathbf{u} - \nabla p + \mathbf{f} \|_{\Omega}^2 + \alpha \| \mathbf{u} - \mathbf{g} \|_{\partial\Omega}^2 \\ & + \beta \| \Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f} \|_{\Omega}^2 \\ & + \gamma \| \nabla \cdot \mathbf{u} \|_{\Omega}^2 + \left\| \underline{\mathbf{U}} - (\nabla \mathbf{u})^T \right\|_{\Omega}^2, \end{aligned} \tag{2.4}$$

where α and β are penalty terms to enforce the Poisson equation for the pressure p and the Dirichlet boundary conditions of the velocity $\mathbf{u}$, respectively. $\| \cdot \|_{\Omega}$ is the L^2 norm on Ω and $\| \cdot \|_{\partial\Omega}$ is the L^2 norm on $\partial\Omega$. Later, we will show in Section 4.1 the training of the network based on the formulation (2.2), using the nonlinear first order system and the Poisson equation (2.3), converges slowly (even with the MscaleDNNs).

2.2. Iterative methods to solve stationary Navier-Stokes equations

Three iterative methods were introduced for solving the stationary Navier-Stokes equations in [7]. We will give a short review for those methods in this section.

Let X, Y, M be the Hilbert spaces $X = H_0^1(\Omega)^d, Y = L^2(\Omega)^d, M = L_0^2(\Omega)$. The superscript d at the end of $H_0^1(\Omega)^d$ and $L^2(\Omega)^d$ means the dimension. For the case we are interested in, $d = 2$. Then assume $\mathbf{u}_h^n \in X, p_h^n \in M$ are the solutions at n-th iteration of velocity $\mathbf{u}$ and pressure p, the three iterative schemes are given as follows:

Iterative Method I.

$$\begin{aligned} & a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) \\ & = (\mathbf{f}, \mathbf{v}_h), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1. \end{aligned} \tag{2.5}$$

Iterative Method II.

$$\begin{aligned} & a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^n, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^n, \mathbf{v}_h\right) \\ & = b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) + (\mathbf{f}, \mathbf{v}_h), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1. \end{aligned} \tag{2.6}$$

Iterative Method III.

$$\begin{aligned} & a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^n, \mathbf{v}_h\right) \\ & = (\mathbf{f}, \mathbf{v}_h), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1, \end{aligned} \tag{2.7}$$

where

$$\begin{aligned} & a(\mathbf{u}, \mathbf{v}) = \nu(\nabla \mathbf{u}, \nabla \mathbf{v}), && \mathbf{u}, \mathbf{v} \in X, \\ & d(\mathbf{v}, q) = (q, \operatorname{div} \mathbf{v}), && \mathbf{v} \in X, \quad q \in M, \\ & b(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \left((\mathbf{u} \cdot \nabla)\mathbf{v} + \frac{1}{2} \operatorname{div} \mathbf{u}\mathbf{v}\mathbf{w} \right), && \mathbf{u}, \mathbf{v}, \mathbf{w} \in X, \end{aligned}$$

$(\cdot, \cdot)$ is the L^2-scalar inner product.

DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17

Assume $a_1(\mathbf{u},\mathbf{v},\mathbf{w}) = ((\mathbf{u}\cdot\nabla)\mathbf{v},\mathbf{w})$, then

$$b(\mathbf{u},\mathbf{v},\mathbf{w}) = \frac{1}{2}a_1(\mathbf{u},\mathbf{v},\mathbf{w}) - \frac{1}{2}a_1(\mathbf{u},\mathbf{w},\mathbf{v}).$$

The bilinear term $a(\cdot,\cdot)$ is continuous and coercive on $X\times X$; the bilinear $d(\cdot,\cdot)$ satisfies that for all $q\in M$

$$\sup_{\mathbf{v}\in X}\frac{|d(\mathbf{v},q)|}{\|\nabla\mathbf{v}\|_{2,X}} \geq \beta_0\|q\|_{2,M},$$

where $\beta_0>0$, $\|\cdot\|_{2,X}$ and $\|\cdot\|_{2,M}$ are the corresponding L^2 norm in X and M respectively.

The trilinear form $a_1(\cdot,\cdot,\cdot)$ satisfies

$$|a_1(\mathbf{u},\mathbf{v},\mathbf{w})| \leq N\|\nabla\mathbf{u}\|_{2,X}\|\nabla\mathbf{v}\|_{2,X}\|\nabla\mathbf{w}\|_{2,X},$$

where $N>0$.

Under stability conditions

$$\frac{4N\|\mathbf{f}\|_{-1}}{\nu^2}<1, \quad \frac{25N\|\mathbf{f}\|_{-1}}{3\nu^2}<1,$$

where $\|\mathbf{f}\|_{-1} = \|\nabla\mathbf{f}\|_{2,X}$ and the uniqueness condition

$$\frac{N\|\mathbf{f}\|_{-1}}{\nu^2}<1,$$

the following error estimates for the three schemes can be obtained.

Given a mesh size h for a finite element method and the number of iterative steps m, for Iterative Methods I (2.5) and III (2.7), it has been shown [7] that

$$\begin{aligned}
&\nu\left\|\mathbf{u}-\mathbf{u}_h^n\right\|_{2,X} \leqslant C_1h^2 + C_2\nu\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},\\
&\nu\left\|\nabla\left(\mathbf{u}-\mathbf{u}_h^n\right)\right\|_{2,X} + \left\|p-p_h^n\right\|_{2,M} \leqslant C_3h + C_4\nu\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},
\end{aligned}$$

where $C_1,\ldots,C_4$ are constants.

For the Iterative Method II (2.6), we have

$$\begin{aligned}
&\nu\left\|\mathbf{u}-\mathbf{u}_h^n\right\|_{2,X} \leqslant C_5h^2 + C_6|\log h|^{1/2}\left\|\nabla\left(\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right)\right\|_{2,X}\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},\\
&\nu\left\|\nabla\left(\mathbf{u}-\mathbf{u}_h^n\right)\right\|_{2,X} + \left\|p-p_h^n\right\|_{2,X} \leqslant C_7h + C_8|\log h|^{1/2}\left\|\nabla\left(\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right)\right\|_{2,X}\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},
\end{aligned}$$

where $C_5,\ldots C_8$ are constants.

The strong forms of these three iterative methods, which will be used in defining the loss functions of linearized learning schemes, are given below.

Iterative Method I.

$$-\nu\Delta\mathbf{u}^n + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^{n-1} + \nabla p = \mathbf{f}. \tag{2.8}$$

Iterative Method II.

$$-\nu\Delta\mathbf{u}^n + \left[(\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^n + (\mathbf{u}^n\cdot\nabla)\mathbf{u}^{n-1}\right] + \nabla p = \mathbf{f} + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^{n-1}. \tag{2.9}$$

Iterative Method III.

$$-\nu\Delta\mathbf{u}^n + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^n + \nabla p = \mathbf{f}. \tag{2.10}$$

3. Linearized Learning Algorithm with Multiscale Deep Neural Network

3.1. Multiscale deep neural network (MscaleDNN)

In order to improve the capability of the DNN to represent functions with multiple scales, we developed the MscaleDNN [10], which consists of a series of parallel fully connected sub-neural networks, for solving partial differential equations. Each of the sub-networks will receive a scaled input with different scales. The final output of the MscaleDNN is a linear combination of the outputs of the parallel fully connected neural networks (refer to Fig. 1). The individual sub-network in the MscaleDNN with a scaled input is designed to approximate a segment of frequency content of the targeted function and the scaling is to convert a specific high frequency segment to a lower frequency domain, thus leading to frequency uniform convergence of approximations for highly oscillating functions. Furthermore, due to the radial scaling used in the MscaleDNN as shown in [10], it could be very powerful once we consider to approximate solutions of high dimensional PDEs [18].

Fig. 1 shows the schematics of a typical MscaleDNN consisting of n parallel sub-networks. Each sub-network are with L hidden layers and can be expressed as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]}\sigma \circ \left(\cdots(\boldsymbol{W}^{[1]}\sigma \circ (\boldsymbol{W}^{[0]}(\boldsymbol{x}) + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]})\cdots\right) + \boldsymbol{b}^{[L-1]},$$

where $W^{[1]}, \cdots, W^{[L-1]}$ are trainable weights and $b^{[1]}, \cdots, b^{[L-1]}$ are bias to be optimized via the training, $\sigma(x)$ is the activation function. Mathematically, a MscaleDNN solution $f(\boldsymbol{x})$ is represented by the following weighted sum of sub-networks $f_{\theta^{n_i}}$ with network pa-

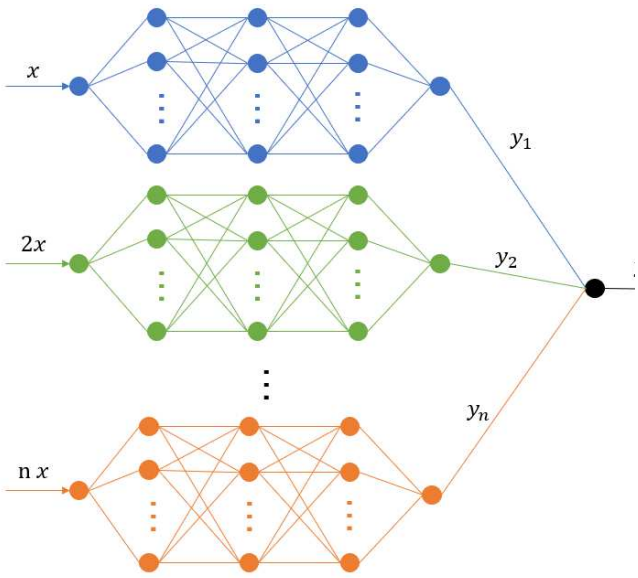


Figure 1: Schematics of MscaleDNN: The MscaleDNN shown has n scales $1, 2, \ldots, n$. The outputs of MscaleDNN y are linear combinations of $y_1, y_2, \ldots, y_n$.

DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17

rameters denoted by θ^{n_i}:

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} \omega_i f_{\theta^{n_i}}(\alpha_i \boldsymbol{x}),$$

where α_i is the chosen scale for the i-th sub-network as shown in Fig. 1. For more details on the design and discussion about the MscaleDNN, we refer to the original paper [10].

In this paper, the following plane wave activation function will be used due to its localized frequency property [15, 16],

$$\sigma(x) = \sin(x).$$

For the input scales, we consider the scale to be 2^{i-1} for the i-th sub-network.

3.2. Linearization schemes for neural network training

To speed up the convergence of the training of the DNN solutions of the NS equations, in this section, we will propose an iterative training procedure for the stationary Navier-Stokes equation based on the iterative scheme introduced in Section 2.2 so that the residual of the training procedure are linearized. Note the term linearize is specific for the non-linear term $(\mathbf{u}\cdot\nabla)\mathbf{u}$. In a nutshell, by fixing either $\mathbf{u}$ or $\nabla\mathbf{u}$ in $(\mathbf{u}\cdot\nabla)\mathbf{u}$, the stationary Navier-Stokes equation turns to be a linear equation, thus we coin the term linearized learning.

Assume the learned velocities up to current epoch are $\mathbf{u}_\theta^* = (u^*, v^*)$ and the velocities to be learned at current epoch are denoted as $\mathbf{u}_\theta = (u, v)$ with p_θ as the pressure to be learned at current epoch. Thus four schemes with different linearization are listed below.

Scheme 1 (**GradFixed**). Gradients of velocities in the nonlinear term will be fixed during training

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta\cdot\nabla)\mathbf{u}_\theta^* + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla\cdot\mathbf{f}. \end{aligned} \tag{3.1}$$

Scheme 2 (**VFixed**). Velocities in the nonlinear term are fixed during training, inspired by Iterative Method III (2.10)

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta^*\cdot\nabla)\mathbf{u}_\theta + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla\cdot\mathbf{f}. \end{aligned} \tag{3.2}$$

Scheme 3 (**VFixed1**). This is the strong form of the Iterative Method II (2.9), which can be seen as a modification of Scheme 2 (VFixed) (3.2)

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta^*\cdot\nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta\cdot\nabla)\mathbf{u}_\theta^* + \nabla p_\theta &= \mathbf{f} + (\mathbf{u}_\theta^*\cdot\nabla)\mathbf{u}_\theta^*, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla\cdot\mathbf{f}. \end{aligned} \tag{3.3}$$

Scheme 4 (**Hybrid**). The Navier-Stokes equation in this scheme is represented as the average of Scheme 1 (GradFixed) (3.1) and Scheme 2 (VFixed) (3.2)

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + \frac{1}{2}\left[(\mathbf{u}_\theta^*\cdot\nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta\cdot\nabla)\mathbf{u}_\theta^*\right] + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla\cdot\mathbf{f}. \end{aligned} \tag{3.4}$$

Based on Eqs. (3.1)-(3.4), we can design the following loss functions:

Loss function for Scheme 1.

$$\begin{aligned}
&\mathrm{L}_{\nabla} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}},\\
&\mathrm{R}_{\mathbf{u}} = \| - \nu \Delta \mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta - \mathbf{f}\|_{2,X}^2,\\
&\mathrm{R}_p = \|\Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f}\|_{2,M}^2,\\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 dS, \quad \mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 dx.
\end{aligned} \tag{3.5}$$

Loss function for Scheme 2.

$$\begin{aligned}
&\mathrm{L}_{\mathbf{u}} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}},\\
&\mathrm{R}_{\mathbf{u}} = \| - \nu \Delta \mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + \nabla p_\theta - \mathbf{f}\|_{2,X}^2,\\
&\mathrm{R}_p = \|\Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f}\|_{2,M}^2,\\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 dS, \quad \mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 dx.
\end{aligned} \tag{3.6}$$

Loss function for Scheme 3.

$$\begin{aligned}
&\mathrm{L}_{\mathbf{u}1} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}},\\
&\mathrm{R}_{\mathbf{u}} = \| - \nu \Delta \mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta - \mathbf{f} - (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta^*\|_{2,X}^2,\\
&\mathrm{R}_p = \|\Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f}\|_{2,M}^2,\\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 dS, \quad \mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 dx.
\end{aligned} \tag{3.7}$$

Loss function for Scheme 4.

$$\begin{aligned}
&\mathrm{L}_{\mathrm{H}} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}},\\
&\mathrm{R}_{\mathbf{u}} = \left\| -\nu \Delta \mathbf{u}_\theta + \frac{1}{2}\left[(\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^*\right] + \nabla p_\theta - \mathbf{f} \right\|_{2,X}^2,\\
&\mathrm{R}_p = \|\Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f}\|_{2,M}^2,\\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 \, dS, \quad \mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 \, dx.
\end{aligned} \tag{3.8}$$

Note $\|\cdot\|_{2,X}$ are the L^2 norm of space X, $\|\cdot\|_{2,M}$ are the L^2 norm of space M, and α, β, γ are the penalty terms. The Poisson equation for pressure p is also considered as a further regularization for training p_θ.

3.3. Linearized learning algorithms

Our linearized learning algorithm for the Navier-Stokes equation is implemented through the following steps illustrated in Algorithm 3.1. In the implementation, $\mathbf{u}_\theta^*$ and $\mathbf{u}_\theta$ are two different neural networks with same number of hidden layers and the same number of hidden neurons at each layer. Once the loss (3.5)-(3.8) decreases a specific ratio, then the parameters of $\mathbf{u}_\theta$ are copied to $\mathbf{u}_\theta^*$, and the parameters of $\mathbf{u}_\theta^*$ will be frozen up to c epochs until the loss decreases the specific ratio again. The algorithm will terminate once the training epoch is up to the maximum training epoch N. The optimizer considered is the popular Adam Optimizer [9].

Algorithm 3.1 Linearized Learning Algorithm.

Once the loss L is smaller than or equal to $\gamma\tau$, the parameters of $\mathbf{u}_\theta^i$ will be copied to $\mathbf{u}_\theta^*$ and note that the parameters of $\mathbf{u}_\theta^*$ will never be updated by Adam algorithm. It should be noted that the number of epochs before updating the fixed (linearized) term c will be a hyperparameter to be adjusted carefully.

procedure LINEARIZEDLEARNING($\mathbf{u}_\theta^0, \mathbf{u}_\theta^*, p_\theta$)
  $\gamma \leftarrow 0.9$ ▷ The ratio to make sure the loss is strictly less then the threshold τ when updating the network $\mathbf{u}_\theta^*$.
  $\tau \leftarrow 10^{12}$ ▷ The threshold.
  for $i \leftarrow 0, \ldots, N$ **do**
    for $j \leftarrow 1, \ldots, c$ **do** ▷ c is a variable to determine the epochs to train the new network $\mathbf{u}_\theta^i$.
      $L \leftarrow \text{Loss}(\mathbf{u}_\theta^*, \mathbf{u}_\theta^i, p_\theta)$ ▷ The Loss is one of (3.5)-(3.8).
      Update $\mathbf{u}_\theta^i$ by Adam with L.
      Update p_θ by Adam with L.
    end for
    if $L \le \gamma\tau$ **then**
      $\tau = L$;
      $\mathbf{u}_\theta^* \leftarrow \mathbf{u}_\theta^i$;
      $\mathbf{u}_\theta^{i+1} \leftarrow \mathbf{u}_\theta^i$;
    end if
  end for
  return $\mathbf{u}_\theta^*, p_\theta$ ▷ The outputs.
end procedure

4. Numerical Results

4.1. A benchmark: A non-oscillatory problem - effect of linearized learning

We first consider a non-oscillatory problem in a rectangle domain $\Omega = [0,2] \times [0,1]$ with one cylinder hole centered at $(0.7, 0.5)$ whose radius is 0.2 as shown in Fig. 2, the

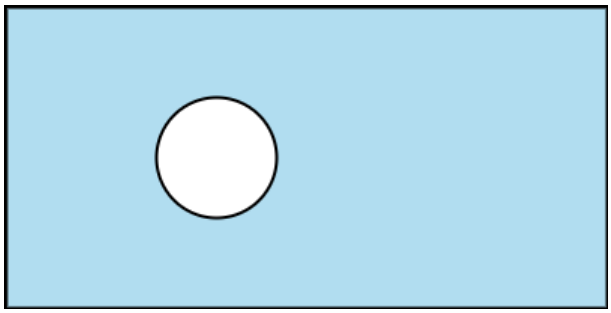


Figure 2: A simple domain with one hole: A rectangle domain $\Omega = [0,2] \times [0,1]$ with one cylinder hole centered at $(0.7, 0.5)$ whose radius is 0.2.

analytical solutions of the incompressible Navier-Stokes equations is given as follows:

$$\begin{aligned} u &= 1 - e^{\lambda x} \cos(2m\pi x + 2n\pi y), \\ v &= \frac{\lambda}{2n\pi} e^{\lambda x} \sin(2m\pi x + 2n\pi y) + \frac{m}{n} e^{\lambda x} \cos(2m\pi x + 2n\pi y), \\ p &= \frac{1}{2}(1 - e^{2\lambda x}), \quad \lambda = \frac{\mathrm{Re}}{2} - \sqrt{\frac{\mathrm{Re}^2}{4} + 4\pi^2}, \quad \mathrm{Re} = \frac{1}{\nu}. \end{aligned} \tag{4.1}$$

In this non-oscillatory case, we consider the frequency $m = 1, n = 2$ and the viscosity $\nu = 0.05$. The source term $\mathbf{f}$ is obtained by substituting the exact solution (4.1) into the Navier-Stokes equation (2.1). The boundary conditions are Dirichlet boundary conditions which are obtained by computing the analytical solutions (4.1) on boundaries, including the four edges of the rectangle and the circle inside. The penalty terms α, β, γ are set to be $10^4, 1, 1$ for this case. We will show the linearization could speed up the learning procedure for a neural network to learn the solution of the stationary Navier-Stokes equation in this section.

We compared the performance of the convergence of fully connected network (fcn) with different loss schemes, including the VgVP formulation (2.4) and three linearized schemes (3.5) or (3.6) or (3.8). The training data are generated by randomly sampling 160000 points inside Ω and 16000 points on $\partial\Omega$ during each epoch. In the learning process, the number of batches for each epoch are set to be 50. We choose the fully connected neural network with 4 hidden layer, 100 hidden neurons each layer for $\mathbf{u}_\theta, \mathbf{u}^*_\theta, p_\theta$ for all cases. The hyperparameters are the same for four cases. The losses during training for different cases by minimizing given loss function are compared in Fig. 3. The results show that the three linearized learning neural networks converge in 300 epochs for all schemes while learning using the loss function (2.4) for the nonlinear Navier-Stokes equations fails to (top line in Fig. 3). The comparisons of the x component of velocity and pressure along line $y = 0.7$ of different linearized schemes after 300 epoch training are shown in Figs. 4 and 5. For the current case, the Hybrid scheme (3.8) offers the best approximation, but the results of the GradFixed scheme (3.5) and the VFixed scheme (3.6) lose some accuracy, which corresponds to the loss Fig. 3.

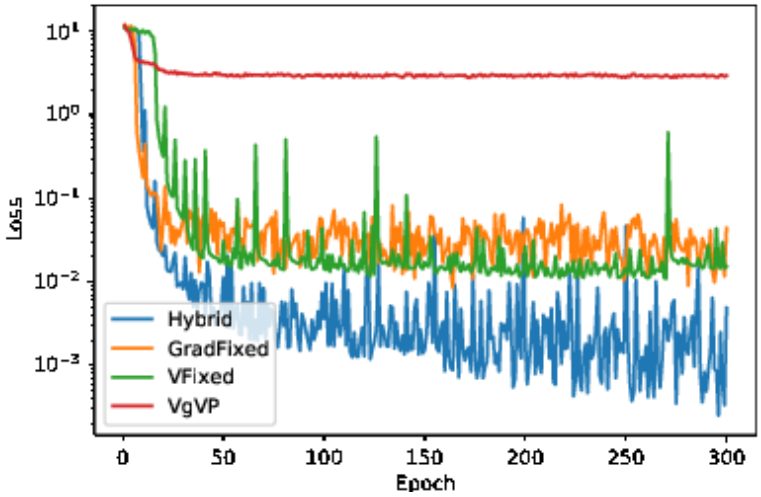
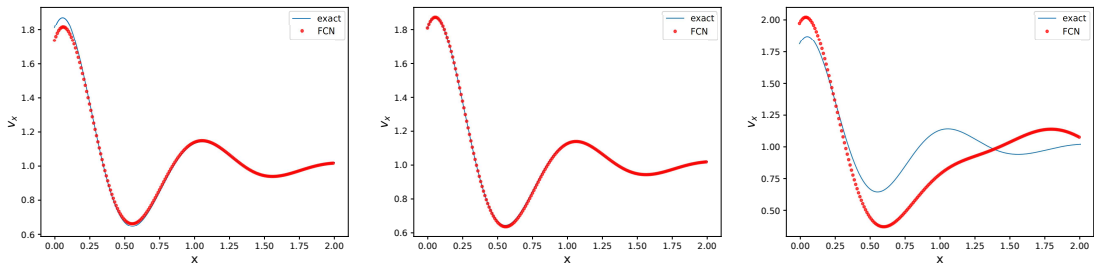
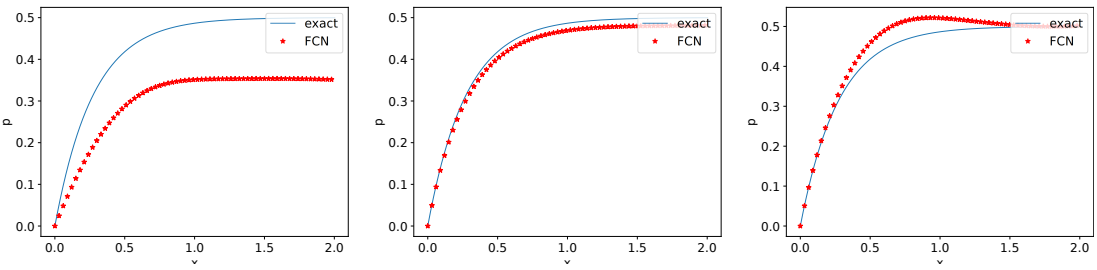


Figure 3: Losses (bottom 3 lines) of three linearized learning schemes (3.5), (3.6) or (3.8) and loss (top line) based on nonlinear Navier-Stokes equation (2.4). The results show that the neural networks with three linearized learning schemes (GradFixed (3.5), VFixed (3.6) and Hybrid (3.8)) converge fast comparing with the neural networks using the VgVp loss function (2.4).



a) VFixed scheme b) Hybrid scheme c) GradFixed scheme

Figure 4: Linearized learning of fully connected network (FCN): The x components of velocity after 300 epoch training for bench mark problem along line $y = 0.7$.



a) VFixed scheme b) Hybrid scheme c) GradFixed scheme

Figure 5: Linearized learning of fully connected network (FCN):The pressures after 300 epoch training for bench mark problem along line $y = 0.7$.

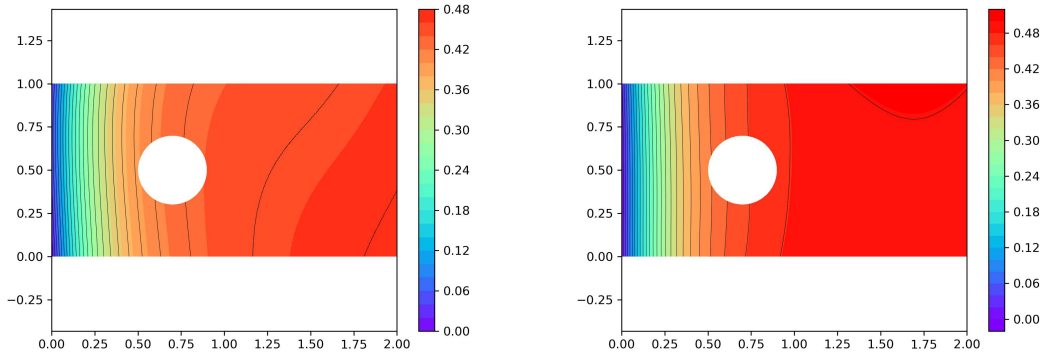
4.2. Performance: Oscillating flows learned by MscaleDNN with linearized learning

In our previous work [16], it has been shown that the MscaleDNN could improve the approximation performance dramatically when learning oscillatory solutions for the linear Stokes equations. Based on the previous experience, we consider MscaleDNN is the preferential framework for solving Navier-Stokes equation with oscillatory solutions, combined with the linearized learning scheme. The frequencies now are taken to be $m = 40, n = 35$, much higher than the benchmark problem. We also adjusted the learning rate during training by a decreasing of 5% every 50 epochs for the oscillating flow case to accelerate the convergence.

4.2.1. A simple domain - effect of MscaleDNN

In this section, we consider the same simple domain as in Fig. 2 and utilize Scheme 1 (GradFixed) (3.1) of the linearized learning algorithms where the previous velocity are used to linearize the convection term. The purpose of this section is to show that MscaleDNN combining with linearized learning scheme could offer extraordinary performance improvement for stationary Navier-Stokes equations with oscillating solutions.

The multiscale deep neural networks are given 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$, whose subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. As a comparison, we also trained a 4-layer fully connected neural network with 1024 hidden neurons combining GradFixed scheme in 1000 epochs. Figs. 6 and 7 show the predictions of networks after 1000 epoch training. Figs. 8 and 9 give more details along line $y = 0.7$. The penalty terms α, β, γ are set to be $10^4, 1, 1$, respectively, as the same as in Section 4.1. Fig. 10 shows the relative errors of these 2 different neural network structures along line $y = 0.7$. We could conclude that the MscaleDNN improves the accuracy of both the pressure field and the velocity field compared with the fully connected neural network.

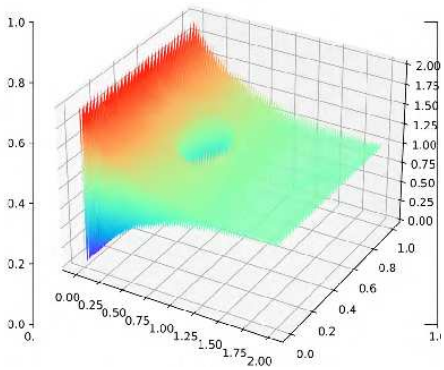


a) Contour of pressure of the oscillatory case after 1000 epoch training for linearized learning with fully-connected network (FCN)

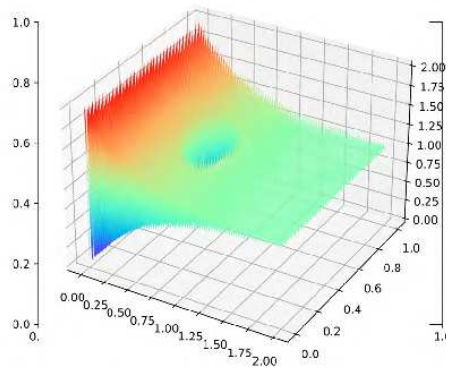
b) Contour of velocity of the first component after 1000 epoch training for linearized learning with MscaleDNN

Figure 6: Pressure of the oscillatory case for linearized learning with MscaleDNN and fully connected networks (FCN).

DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17

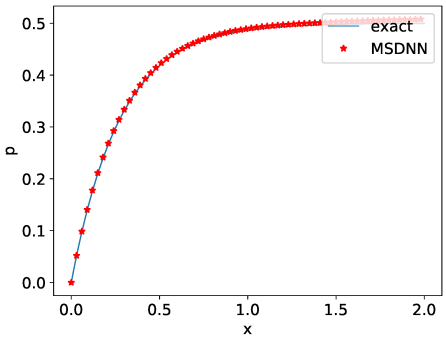


a) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for linearized learning with fully-connected network (FCN)

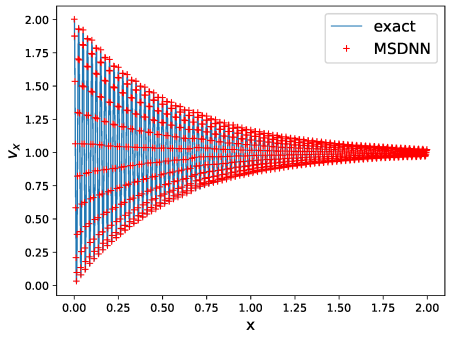


b) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for linearized learning with MscaleDNN

Figure 7: The first component of velocity of the oscillatory case for linearized learning with MscaleDNN and fully connected networks (FCN).

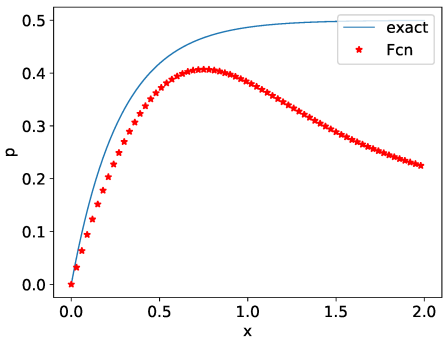


a) Pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$

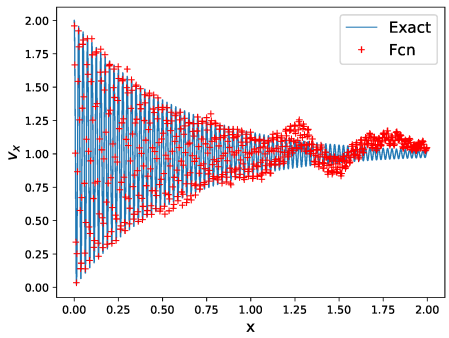


b) Velocity of the first component after 1000 epoch training along line $y = 0.7$

Figure 8: The results of the oscillatory case using linearized learning with multi-scale neural networks.



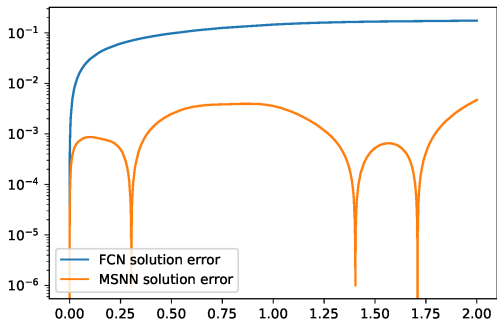
a) Pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$



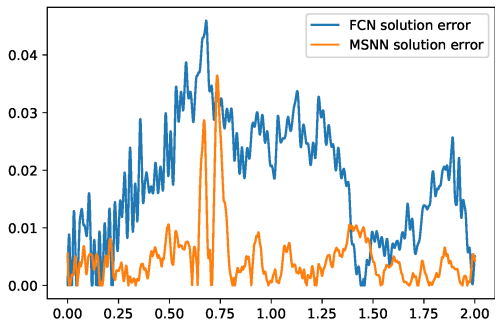
b) Velocity of the first component after 1000 epoch training along line $y = 0.7$

Figure 9: The results of the oscillatory case using linearized learning with fully connected networks (FCN).

DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17



a) Error of two different models w.r.t. pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$



b) Error of two different models w.r.t. velocity of the first component after 1000 epoch training along line $y = 0.7$

Figure 10: The errors of the oscillatory case for linearized learning with fully-connected network (FCN) and multiscale network (MSNN).

4.2.2. A complex domain

In this section, we consider an oscillating case in complex domain with more than one hole. The domain is shown in Fig. 11. In this case, we use the similar settings for the multiscale deep neural networks, adjustments of learning rates, and sampling strategies like what we choose in the oscillatory case with the scale of the simple domain with one hole in Section 4.2. The multiscale deep neural networks has 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$, whose subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. The frequency we select for this case are the same as what in Section 4.2.1. The initial learning rate for this case is $4e-3$. The penalty terms α, β, γ are set to be $10^4, 1, 1$, respectively, the same as in Section 4.1. Fig. 12 shows contours of the first component of velocity for different schemes. Figs. 13 and 14 show the relative errors of these 4 different linearization schemes along line $y = 0.7$. All schemes we propose converges more accurately to the exact solutions.

Fig. 15 displays the behavior of the velocity field's divergence under varying penalty values γ while the neural networks are trained by scheme (3.3). The results indicate that as γ increases, the divergence of velocity decreases. The divergence-free property thus is enforced through the extra regularization with large penalty.

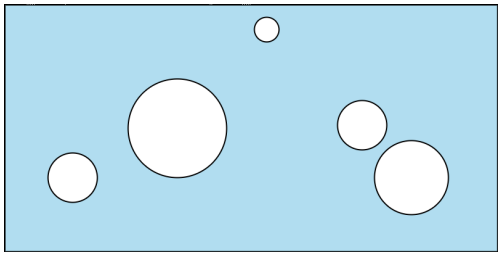
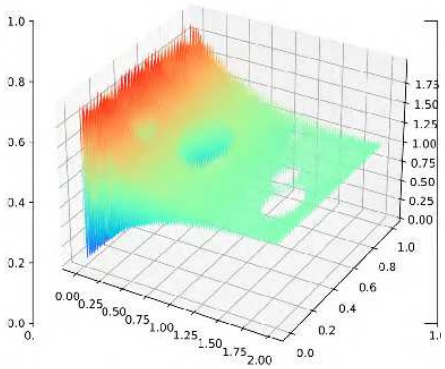
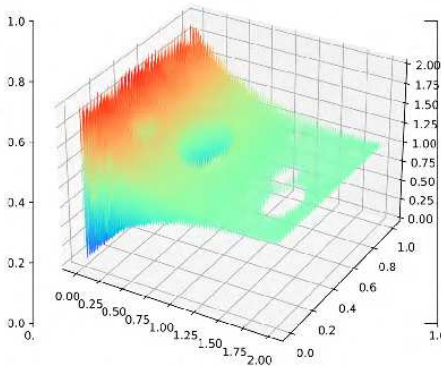


Figure 11: A more complex domain.

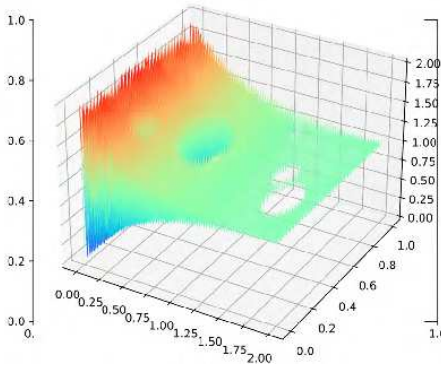
DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17



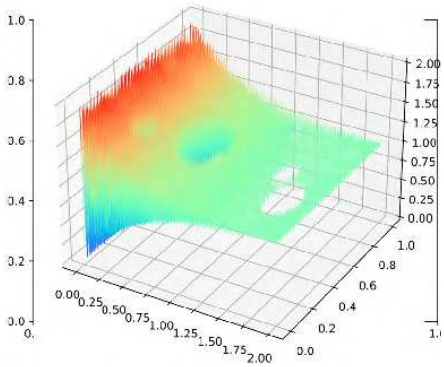
a) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme gradFixed (3.1)



b) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme vFixed (3.2)



c) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme vFixed1 (3.3)



d) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme Hybrid (3.4)

Figure 12: The results of the first component of velocity of the oscillatory case for four linearized learning schemes with MscaleDNN.

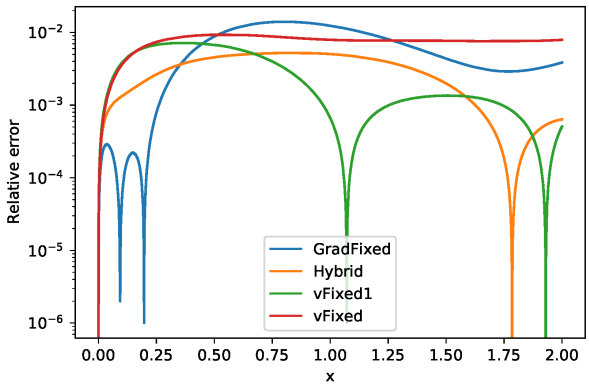
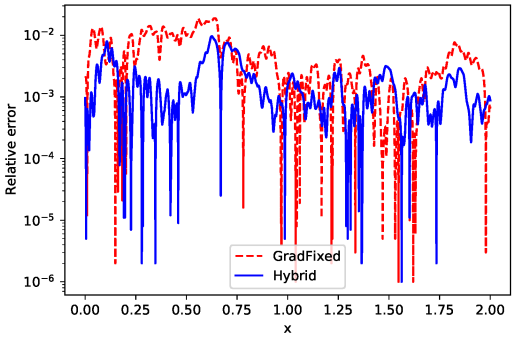
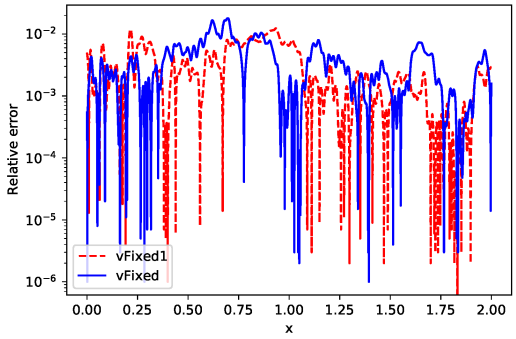


Figure 13: Relative errors at the line $y = 0.7$ of four linearized learning schemes with MscaleDNN for pressure of the complex domain case after 1000 epoch training.

DOI https://doi.org/10.4208/eajam.2022-328.230423 | Generated on 2024-12-19 01:07:17

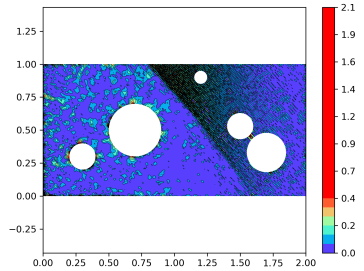


a) Relative errors alone line $y = 0.7$ of GradFixed linearized learning scheme (3.1) and Hybrid linearized learning scheme (3.4) for the first component of velocity of the complex domain case after 1000 epoch training

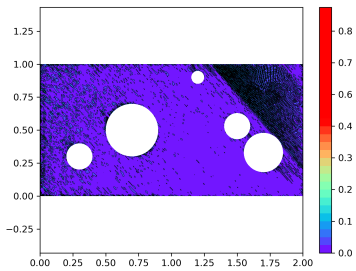


b) Relative errors along line $y = 0.7$ of vFixed linearized learning scheme (3.2) and vFixed1 linearized learning scheme (3.3) for the first component of velocity of the complex domain case after 1000 epoch training

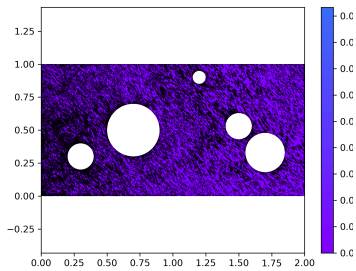
Figure 14: The relative errors of the complex domain case for four linearized learning schemes with MscaleDNN.



a) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^0$. The largest value is 2.16



b) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^3$. The largest value is 0.9

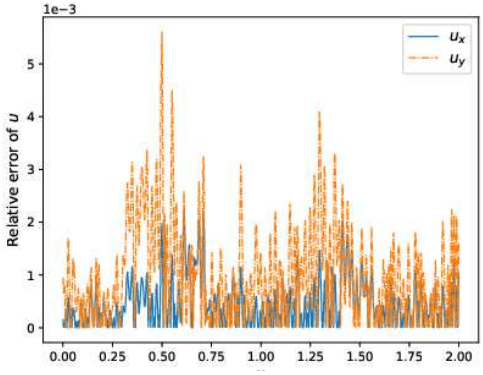


c) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^6$. The largest value is 0.07

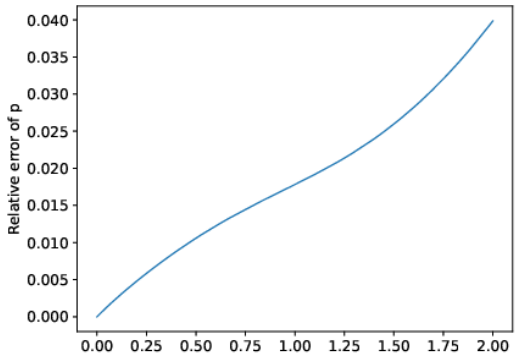
Figure 15: The divergence of velocity $\nabla \cdot \mathbf{u}$ where $\mathbf{u}$ is trained by linearized learning scheme (3.3) with 1000 epochs.

4.2.3. Smaller viscosity coefficient

In this section, we consider the same oscillating case in the complex domain but with smaller viscosity coefficient, $\nu = 0.001$. The multiscale deep neural networks are the same as in the oscillatory case with 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$ and the corresponding subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. The learning rates are multiplied by 0.1 at the 100th, 300th, 600th epoch. The batch size is 8092 in the domain and 512 on the boundary of domain, respectively. The penalties considered in the case are 10^4 for α, 1 for β and 10^3 for γ, respectively. The frequencies selected for this case are the same as what in Section 4.2.1. The initial learning rate is $4e-3$. The scheme is (3.3) as it gives the best results for the complex domain case in Section 4.2.2. Figs. 16 and 17 show the results of linearized learning scheme (3.3). The divergence of velocity is even better comparing with the larger viscosity coefficient case.



a) Relative errors for velocities in both x direction and y direction alone line $y = 0.7$



b) Relative errors for pressure p alone line $y = 0.7$

Figure 16: The relative errors of velocity $\mathbf{u}$ and pressure p along line $y = 0.7$ where neural networks are trained by linearized learning scheme (3.3) with 1000 epochs as $\nu = 0.001$.

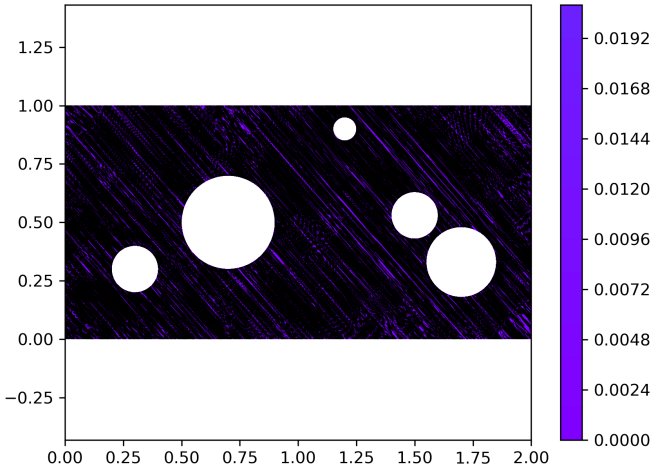


Figure 17: The divergence of velocity $\nabla \cdot \mathbf{u}$ where neural networks are trained by linearized learning scheme (3.3) with 1000 epochs as $\nu = 0.001$. The largest value is 0.023.

5. Conclusion and Future Work

In this paper we proposed four linearized learning schemes to solve the stationary highly oscillatory Navier-Stokes flows with multiscale deep neural networks and showed the acceleration of convergence of the schemes are substantial, which demonstrate the capability of the multiscale deep neural networks and the effectiveness of the linearized schemes to solve the nonlinear Navier-Stokes equations. These schemes shed some lights on the practical applications of neural network machine learning algorithms to the nonlinear equations, which are time-consuming using traditional finite element methods. The deep neural network based methods offer an alternative that does not require meshes and has no need to solve large-scale linear systems, as in traditional numerical methods.

There are more works to be done for these linearized learning methods, among them the most important is to understand the convergence property of these schemes. The applications of these schemes to other nonlinear PDEs will also be considered. Another challenging work is to consider the time dependent Navier-Stokes equation, which will be explored in a future work. Additionally, exploring the extension of the presented method to the 3D Navier-Stokes equation will be a topic for future research.

Acknowledgements

The work of W. Cai was supported by the U.S. National Science Foundation (Grant No. DMS-2207449), and the research of B. Wang was partially supported by the NSFC (Grant Nos. 11771137, 12022104).

References

[1] P.B. Bochev and M.D. Gunzburger, *Least-Squares Finite Element Methods*, Applied Mathematical Sciences, Vol. 166, Springer (2009).
[2] S.Z. Cai, Z.C. Wang, F. Fuest, Y.J. Jeon, C. Gray and G.E. Karniadakis, *Flow over an espresso cup: Inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks*, J. Fluid Mech. **915**, A102 (2021).
[3] W. Cai, X.G. Li and L.Z. Liu, *A phase shift deep neural network for high frequency approximation and wave problems*, SIAM J. Sci. Comput. **42**, A3285–A3312 (2020).
[4] T. De Ryck, A.D. Jagtap and S. Mishra, *Error estimates for physics informed neural networks approximating the Navier-Stokes equations*, IMA J. Numer. Anal. drac085 (2023).
[5] H. Eivazi, M. Tahani, P. Schlatter and R. Vinuesa, *Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations*, Phys. Fluids. **34**, 075117 (2022).
[6] J.Q. Han, A. Jentzen and W.N. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA. **115**, 8505–8510 (2018).
[7] Y.N. He and J. Li, *Convergence of three iterative methods based on the finite element discretization for the stationary Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg. **198**, 1351–1359 (2009).
[8] X.W. Jin, S.Z. Cai, H. Li and G.E. Karniadakis, *NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations*, J. Comput. Phys. **426**, 109951 (2021).
[9] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv:1412.6980 (2014).
[10] Z.Q. Liu, W. Cai and Z.Q.J. Xu, *Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains*, Commun. Comput. Phys. **28**, 1970–2001 (2020).
[11] J. Oldenburg, F. Borowski, A. Öner, K.P. Schmitz and M. Stiehm, *Geometry aware physics informed neural network surrogate for solving Navier-Stokes equation (GAPINN)*, Adv. Model. Simul. Eng. Sci. **9**, 8 (2022).
[12] M. Raissi, *Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations*, arXiv:1804.07010 (2018).
[13] M. Raissi, A. Yazdani and G.E. Karniadakis, *Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations*, Science **367**, 1026–1030 (2020).
[14] C.P. Rao, H. Sun and Y. Liu, *Physics-informed deep learning for incompressible laminar flows*, Theor. App. Mech. Lett. **10**, 207–212 (2020).

[15] V. Sitzmann, J.N.P. Martel, A.W. Bergman, D.B. Lindell and G. Wetzstein, *Implicit neural representations with periodic activation functions*, in: *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Neural Inf. Process. Ser. (2020).
[16] B. Wang, W.Z. Zhang and W. Cai, *Multi-scale deep neural network (MscaleDNN) methods for oscillatory stokes flows in complex domains*, Commun. Comput. Phys. **28**, 2139–2157 (2020).
[17] Z.Q.J. Xu, *Frequency principle: Fourier analysis sheds light on deep neural networks*, Commun. Comput. Phys. **28**, 1746–1767 (2020).
[18] W.Z. Zhang and W. Cai, *FBSDE based neural network algorithms for high-dimensional quasilinear parabolic PDEs*, J. Comput. Phys. **470**, 111557 (2022).