Journal of Computational Physics 426 (2021) 109951

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations

Xiaowei Jin [a,b,1], Shengze Cai [c,1], Hui Li [a,b,*], George Em Karniadakis [c,**]

[a] Key Lab of Smart Prevention and Mitigation of Civil Engineering Disasters of the Ministry of Industry and Information Technology, Harbin Institute of Technology, Harbin 150090, China
[b] Key Lab of Structures Dynamic Behavior and Control of the Ministry of Education, Harbin Institute of Technology, Harbin 150090, China
[c] Division of Applied Mathematics, Brown University, Providence, RI 02912, USA

ARTICLE INFO

Article history:
Received 13 March 2020
Received in revised form 12 September 2020
Accepted 23 October 2020
Available online 1 November 2020

Keywords:
PINNs
Turbulence
Velocity-pressure formulation
Vorticity-velocity formulation
Ill-posed problems
Transfer learning

ABSTRACT

In the last 50 years there has been a tremendous progress in solving numerically the Navier-Stokes equations using finite differences, finite elements, spectral, and even meshless methods. Yet, in many real cases, we still cannot incorporate seamlessly (multi-fidelity) data into existing algorithms, and for industrial-complexity applications the mesh generation is time consuming and still an art. Moreover, solving ill-posed problems (e.g., lacking boundary conditions) or inverse problems is often prohibitively expensive and requires different formulations and new computer codes. Here, we employ physics-informed neural networks (PINNs), encoding the governing equations directly into the deep neural network via automatic differentiation, to overcome some of the aforementioned limitations for simulating incompressible laminar and turbulent flows. We develop the Navier-Stokes flow nets (NSFnets) by considering two different mathematical formulations of the Navier-Stokes equations: the velocity-pressure (VP) formulation and the vorticity-velocity (VV) formulation. Since this is a new approach, we first select some standard benchmark problems to assess the accuracy, convergence rate, computational cost and flexibility of NSFnets; analytical solutions and direct numerical simulation (DNS) databases provide proper initial and boundary conditions for the NSFnet simulations. The spatial and temporal coordinates are the inputs of the NSFnets, while the instantaneous velocity and pressure fields are the outputs for the VP-NSFnet, and the instantaneous velocity and vorticity fields are the outputs for the VV-NSFnet. This is unsupervised learning and, hence, no labeled data are required beyond boundary and initial conditions and the fluid properties. The residuals of the VP or VV governing equations, together with the initial and boundary conditions, are embedded into the loss function of the NSFnets. No data is provided for the pressure to the VP-NSFnet, which is a hidden state and is obtained via the incompressibility constraint without extra computational cost. Unlike the traditional numerical methods, NSFnets inherit the properties of neural networks (NNs), hence the total error is composed of the approximation, the optimization, and the generalization errors. Here, we empirically attempt to quantify these errors by varying the sampling ("residual") points, the iterative solvers, and the size of the NN architecture. For the laminar flow solutions, we show that both the VP and the VV formulations are comparable in accuracy but their best performance corresponds to different NN architectures. The

* Corresponding author at: Key Lab of Smart Prevention and Mitigation of Civil Engineering Disasters of the Ministry of Industry and Information Technology, Harbin Institute of Technology, Harbin 150090, China.
** Corresponding author at: Division of Applied Mathematics, Brown University, Providence, RI 02912, USA.
E-mail addresses: lihui@hit.edu.cn (H. Li), george_karniadakis@brown.edu (G.E. Karniadakis).
[1] The first two authors contributed equally to this work.

https://doi.org/10.1016/j.jcp.2020.109951

initial convergence rate is fast but the error eventually saturates to a plateau due to the dominance of the optimization error. For the turbulent channel flow, we show that NSFnets can sustain turbulence at $\mathrm{Re}_\tau \sim 1,000$, but due to expensive training we only consider part of the channel domain and enforce velocity boundary conditions on the subdomain boundaries provided by the DNS data base. We also perform a systematic study on the weights used in the loss function for balancing the data and physics components, and investigate a new way of computing the weights dynamically to accelerate training and enhance accuracy. In the last part, we demonstrate how NSFnets should be used in practice, namely for ill-posed problems with incomplete or noisy boundary conditions as well as for inverse problems. We obtain reasonably accurate solutions for such cases as well without the need to change the NSFnets and at the same computational cost as in the forward well-posed problems. We also present a simple example of transfer learning that will aid in accelerating the training of NSFnets for different parameter settings.

1. Introduction

There exist many different computational fluid dynamics (CFD) methods developed over the span of more than five decades, which work very effectively if the governing equations are precisely known or if all the scales are accurately resolved. However, in many real world applications, either the physics is not totally known, e.g., in reactive transport, or the large spatio-temporal spectrum of scales may be prohibitively expensive to resolve, hence resorting to subgrid scale closures. In the last five years, there have been several efforts to integrate neural networks (NNs) in the solution of the incompressible Navier-Stokes equations following different approaches. For turbulent flows, the most common approach is to derive data-driven turbulence closure models. For example, Ling et al. [1] proposed a data-driven Reynolds-averaged Navier–Stokes (RANS) turbulence closure model by embedding the Galilean invariance into deep neural networks and demonstrated better accuracy for predicting the Reynolds stresses. Similarly, Wang et al. [2] used random forest regression to predict the discrepancies of the baseline RANS-predicted Reynolds stresses compared to those from the DNS data, hence predicting the Reynolds stresses with high accuracy. Jiang et al. [3] developed a novel RANS stress closure with machine-learning-assisted parameterization and nonlocal effects, aiming at reducing both structural and parametric inaccuracies and achieving a more appropriate description for Reynolds stress anistropy. For large-eddy simulation (LES) of isotropic turbulence, Zhou et al. [4] developed a data-driven subgrid scale model by using NNs with only one hidden layer. In addition, some reduced order models (ROMs) or fast prediction models in fluid mechanics have also been investigated. For example, convolutional neural networks (CNNs) were used to construct the prediction model of cylinder wake in [5], and a temporal CNN was used to establish a data-driven model for predicting the coefficients of proper orthogonal decomposition (POD) modes of cylinder wake in [6]. The bidirectional recurrent neural networks were employed to predict the POD coefficients of cylinder wake based on a few velocity measurements [7], obtaining more accurate results than the extended POD approach [8,9]. Moreover, deep learning techniques were also applied to particle image velocimetry (PIV) for analyzing laboratory data of turbulent boundary layer [10]. Comprehensive summaries of progress in fluid mechanics due to the introduction of various machine learning techniques can be found in [11,12].

We have followed a different path by exploiting the universal approximation property of NNs, which together with automatic differentiation enables us to develop Navier-Stokes "solvers" that do not require mesh generation. They are easy to implement, and can be particularly effective for multiphysics and inverse fluid mechanics problems especially when some (multi-fidelity) data are available that can play the role of closure for the missing physics. In particular, Raissi et al. [13,14,15] first introduced the concept of physics-informed neural networks (PINNs) to solve forward and inverse problems involving several different types of PDEs. This approach has also been used to simulate vortex induced vibrations in [16] and also to tackle ill-posed inverse fluid mechanics problems, a framework called "hidden fluid mechanics" presented in [17]. The flows considered in the aforementioned works are laminar flows at relatively low Reynolds numbers, described by the incompressible Navier-Stokes equations in velocity-pressure (VP) form. A fundamental question is if PINNs can simulate turbulence directly, similarly to direct numerical simulation (DNS) using high-order discretization [18,19]. Another important question is if there is another formulation of the Navier-Stokes equations, e.g., in vorticity-velocity (VV) form, that may achieve higher accuracy or may be amenable to a more efficient training. Ultimately, our aim for these PINN-based Navier-Stokes solvers (the NSFnets) is to employ them for simulating flow problems, where the classical CFD solvers are limited, either due to the geometric complexity (e.g., moving and highly distorted domains), noisy, gappy or even missing boundary conditions, or unknown fluid properties.

In the current study, we first address the question of accuracy and convergence of NSFnets, bearing in mind that in addition to the approximation error encountered in the classical numerical methods, in NN there are additional errors associated with optimization since the solution method is typically based on iterative stochastic gradient descent searching for minima of a high-dimensional non-convex surface. To this end, we use analytical solutions for two-dimensional and three-dimensional flows and also comparisons with DNS of turbulent channel flow available at [20–22] are implemented. In particular, we perform NSFnet simulations by considering two forms of the governing Navier-Stokes equations: the VP

form and the VV form. For the VP-NSFnet, the inputs are the spatial and temporal coordinates while the outputs are the instantaneous velocity and pressure fields. For the VV-NSFnet, the inputs again are the spatial and temporal coordinates while the outputs are the instantaneous velocity and vorticity fields. We use automatic differentiation (AD) [23] to deal with the differential operators in the Navier-Stokes equations, which leads to very high computational efficiency compared to numerical differentiation. Numerical diffusion and dispersion errors are coming from the discretization of time and space and using the solutions of the discretized differential equations to approximate the original partial differential equations. However, the discretization of time and space is not used in NSFnets, thus avoiding the classical artificial dispersion and diffusion errors. Furthermore, with AD we differentiate the NN rather than the data directly and hence we can deal with noisy inputs or solutions with limited regularity. There are also distinct advantages in employing both the VP and the VV formulations using NSFnets. For example, to infer the pressure equation we do not use an additional Poisson pressure equation as is usually done with the splitting methods [24] and no data is required for the pressure as boundary or initial conditions for VP-NSFnet; the pressure is a hidden state and is obtained via the incompressibility constraint. The CFD *coupled* solvers avoid the extra equation but are subject to other stability conditions that complicate the solution spaces and correspondingly the grids required. In the VV-NSFnet, it is easy to incorporate vorticity boundary conditions, which come in the form of constraints, directly into the loss function.

In the first part, in order to assess accuracy and convergence we simulate several laminar flows, including two-dimensional steady Kovasznay flow, two-dimensional unsteady cylinder wake and three-dimensional unsteady Beltrami flow, using the two types of NSFnets. We perform a systematic convergence study for the Kovasznay flow with respect to sampling points, which we call "residual" points in this paper, as well as with respect to the architecture size and for different optimizers. Moreover, we demonstrate how we can perform *a posteriori* adaptive refinement based on the residuals of the VP and VV governing equations. In order to accelerate training and enhance accuracy, we also investigate the use of dynamic weights in the loss function for the various components following the work of [25]. Finally, in this first part we report the first results on directly simulating turbulence using NSFnets. To this end, we consider turbulent channel flow at $\mathrm{Re}_\tau \sim 1{,}000$ using primarily VP-NSFnet as the available data bases are derived based on VP type formulations. We perform NSFnet simulations by considering different subdomains with different sizes at various locations in the channel and for different time intervals. In addition, we investigate the influence of weights in the loss function on the accuracy of VP-NSFnet. In the second part, we demonstrate how to use NSFnets beyond classical CFD applications, e.g., in solving ill-posed (noisy or missing boundary conditions) or inverse problems (unknown fluid properties) that would be unsolvable or very expensive with the standard methods. We also make an attempt in demonstrating how we can alleviate the relatively high computational cost of NSFnets by giving a simple example of transfer learning.

The paper is organized as follows. We first introduce the NSFnets in section 2, and present the problem setup and NSFnet simulation results for laminar flows in section 3. We then present VP-NSFnet results for turbulent channel flow in section 4. We set up and solve some problems beyond the ability of traditional CFD solvers in section 5. We summarize our findings in section 6. In the Appendix, we provide more details on systematic studies of convergence with respect to the number of sampling points, number of boundary points, size of the neural network as well as the weighting coefficients in the loss function.

2. Methodology

We introduce two formulations of the unsteady incompressible three-dimensional Navier-Stokes equations: the velocity-pressure (VP) form and the vorticity-velocity (VV) form, as well as their corresponding physics-informed neural networks (PINNs), shown in Fig. 1.

The VP form of the incompressible Navier-Stokes equations is:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{u} \qquad \text{in } \Omega, \tag{1a}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{in } \Omega, \tag{1b}$$

$$\mathbf{u} = \mathbf{u}_\Gamma \qquad \text{on } \Gamma_D, \tag{1c}$$

$$\frac{\partial \mathbf{u}}{\partial n} = 0 \qquad \text{on } \Gamma_N, \tag{1d}$$

where t is the non-dimensional time, $\mathbf{u}(\mathbf{x},t) = [u, v, w]^T$ is the non-dimensional velocity vector, p is the non-dimensional pressure, and $\mathrm{Re} = U_{ref} D_{ref}/\nu$ is the Reynolds number defined by a characteristic length D_{ref}, reference velocity U_{ref} and kinematic viscosity ν. The initial and boundary conditions are required in order to solve Eq. (1). Here, Γ_D and Γ_N denote the Dirichlet and Neumann boundaries, respectively. We note that only Dirichlet boundary conditions, provided by reference solutions (analytical solutions or DNS data), will be considered in this study; any other boundary conditions can be easily treated in NSFnets. In this study, instead of using conventional computational fluid dynamics (CFD) methods, we investigate the possibility of using neural networks (NNs) for solving the Navier-Stokes equations. In other words, the solutions of Navier-Stokes equations are approximated by a deep neural network, which takes spatial and temporal coordinates as inputs and predicts the corresponding velocity and pressure fields, i.e., $(t, x, y, z) \mapsto (u, v, w, p)$. A schematic illustration of the

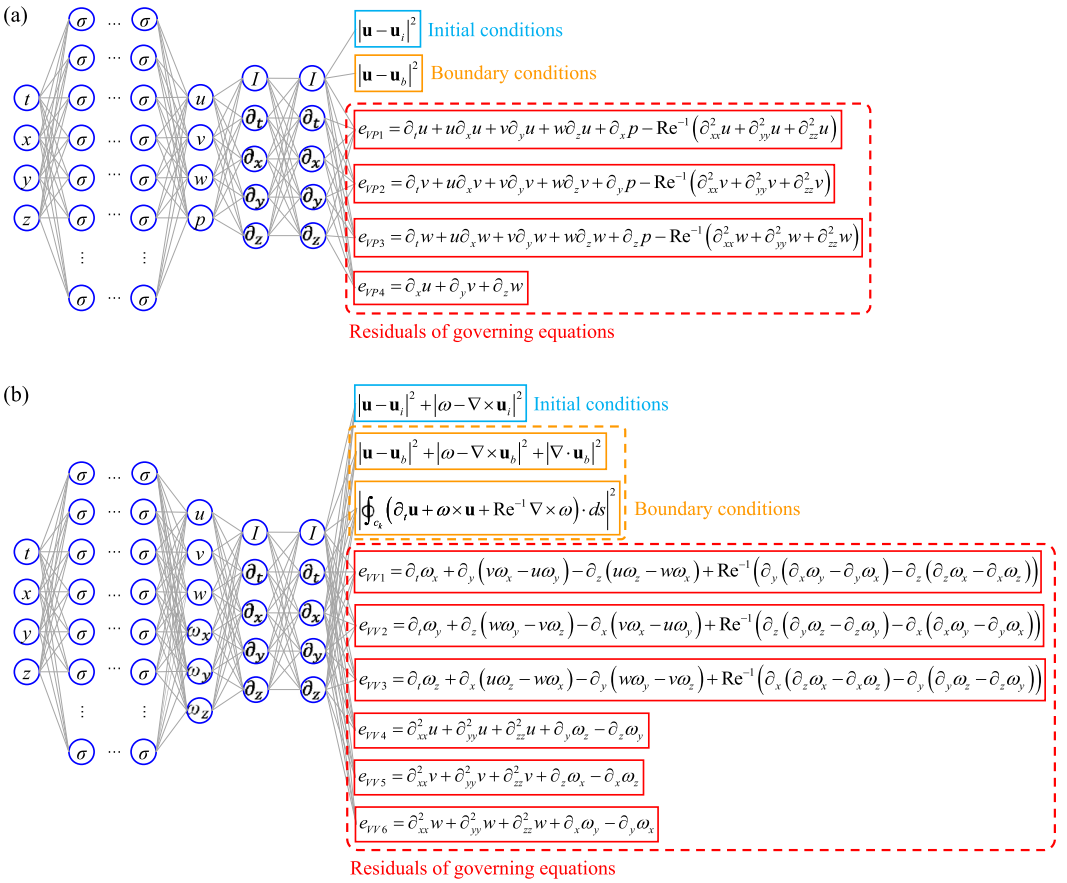

Fig. 1. A schematic of NSFnets: (a) the velocity-pressure (VP) form; (b) the vorticity-velocity (VV) form. The left part of the NN is an uninformed network, while the right part implements the VP and VV formulations using AD. We only show the operators in the right part as the NNs induced by AD of the VP and VV differential operators are too complicated and cannot be visualized even with specialized methods such as "TensorBoard" in TensorFlow.

PINNs for solving Eq. (1) is shown in Fig. 1(a), which consists of a fully-connected network and the residual networks. Here, the nonlinear activation function σ is the hyper tangent function *tanh*. In NSFnets, the derivatives in the PDE are approximated by the derivatives of the output with respect to the input of the PINNs. Therefore, the activation function needs to be differentiable for NSFnets. However, the commonly used activation functions such as ReLU fail to satisfy the continuous second-order derivatives, hence we employ a sigmoidal activation function with continuous derivatives in this study. In particular, the hyperbolic tangent activation function *tanh* is similar to the identity function near 0, so typically it performs better than the logistic sigmoid [26]. Therefore, *tanh* is chosen as the activation function in this study. For the VP form, the residuals include the errors of the momentum equations and the divergence-free constraint. In order to compute the residuals of the Navier-Stokes equations e_{VP1} to e_{VP4}, the partial differential operators are computed by using automatic differentiation (AD), which can be directly formulated in the deep learning framework, e.g., using "`tf.gradients()`" in TensorFlow. In AD, the derivatives in the governing equations are approximated by the derivatives of the output with respect to the input of the PINNs.

The loss function for training the parameters of VP-NSFnet to obtain the solutions of Eq. (1) is defined as follows:

$$L = L_e + \alpha L_b + \beta L_i, \tag{2a}$$

$$L_e = \frac{1}{N_e}\sum_{i=1}^{4}\sum_{n=1}^{N_e}|e_{VPi}^n|^2, \tag{2b}$$

$$L_b = \frac{1}{N_b}\sum_{n=1}^{N_b}|\mathbf{u}^n - \mathbf{u}_b^n|^2, \tag{2c}$$

$$L_i = \frac{1}{N_i}\sum_{n=1}^{N_i}|\mathbf{u}^n - \mathbf{u}_i^n|^2, \tag{2d}$$

where L_e, L_b and L_i represent loss function components corresponding to the residual of the Navier-Stokes equations, the boundary conditions, and the initial conditions, respectively; N_b, N_i and N_e denote the number of training data for different terms; $\mathbf{u}_b^n = [u_b^n, v_b^n, w_b^n]^T$ and $\mathbf{u}_i^n = [u_n^i, v_n^i, w_n^i]^T$ are the given velocities for the nth data point on the boundaries and at the initial time, respectively; e_{vpi}^n represents the residual of the ith equation at the nth data point. The weighting coefficients α and β are used to balance different terms of the loss function and accelerate convergence in the training process. We consider the initial and boundary conditions as supervised data-driven parts, and the residual of the Navier-Stokes equations as the unsupervised physics-informed part in the loss function. We note that no data is provided for the pressure as boundary or initial conditions, which means that p is a hidden state and is obtained via the incompressibility constraint without splitting the Navier-Stokes equations, hence, resembling the coupled solvers for CFD but without the need for special grids or compatible function spaces. The VP-NSFnet does not require any boundary conditions for pressure, however, pressure boundary conditions are usually carefully treated in the traditional coupled solvers for CFD [27–29]. An adaptive optimization algorithm, *Adam* [30], is used to minimize the loss function in Eq. (2). The parameters of the neural networks are randomly initialized using the Xavier scheme [31]. The solutions are obtained when the training of the NSFnet converges, i.e., the total loss function reaches a very small value.

We also propose NSFnets for the VV formulation of the Navier-Stokes equations, which is an alternative to the VP form in simulating incompressible flows; the equivalence of VP and VV formulations was proved in [32,33]. The rotational form of the VV formation of the Navier-Stokes equations is:

$$\frac{\partial \omega}{\partial t} + \nabla \times (\omega \times \mathbf{u}) = -\frac{1}{\mathrm{Re}} \nabla \times \nabla \times \omega \qquad \text{in } \Omega, \tag{3a}$$

$$\nabla^2 \mathbf{u} = -\nabla \times \omega \qquad \text{in } \Omega, \tag{3b}$$

$$\omega = \nabla \times \mathbf{u} \qquad \text{on } \Gamma, \tag{3c}$$

$$\oint_{c_k} \left(\frac{\partial \mathbf{u}}{\partial t} + \omega \times \mathbf{u} + \frac{1}{\mathrm{Re}} \nabla \times \omega \right) \cdot ds = 0, \qquad k = 1, ..., q, \tag{3d}$$

$$\mathbf{u} = \mathbf{u}_\Gamma \qquad \text{on } \Gamma_D, \tag{3e}$$

$$\frac{\partial \mathbf{u}}{\partial n} = 0 \qquad \text{on } \Gamma_N, \tag{3f}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{at one point on } \Gamma, \tag{3g}$$

$$\omega = \nabla \times \mathbf{u} \quad \text{at } t = 0 \qquad \text{in } \Omega, \tag{3h}$$

where $\omega = [\omega_x, \omega_y, \omega_z]^T$ is the vorticity with three components; the domain is q-multiply connected and c_k's are the q independent contours. The boundary conditions are defined by Eqs. (3c) to (3g) and the initial condition is constrained by Eq. (3h). Similarly, we assume that the solutions of the VV form (3) are approximated by a neural network, whose function can be written as $(t, x, y, z) \mapsto (u, v, w, \omega_x, \omega_y, \omega_z)$. We note again that no data is provided for the pressure to the VP-NSFnet, which is a hidden state and is obtained via the incompressibility constraint without splitting the equations; however, this inferred variable may suffer from lower accuracy. In contrast, in the VV-NSFnet, the pressure term is eliminated, and the incompressibility constraint is satisfied exactly. The architecture of the VV-NSFnet for solving Eq. (3) is shown in Fig. 1(b), where e_{vv1} to e_{vv6} represent the residuals of the VV formulation of the Navier-Stokes Eqs. (3a) and (3b). The corresponding loss function of the VV-NSFnet is defined as follows:

$$L = L_e + \alpha L_b + \beta L_i \tag{4a}$$

$$L_e = \frac{1}{N_e} \sum_{i=1}^{6} \sum_{n=1}^{N_e} |e_{vvi}^n|^2 \tag{4b}$$

$$L_b = \frac{1}{N_b} \sum_{n=1}^{N_b} \left(|\mathbf{u}^n - \mathbf{u}_b^n|^2 + |\omega^n - \nabla \times \mathbf{u}_b^n|^2 + |\nabla \cdot \mathbf{u}_b^n|^2 \right) \tag{4c}$$

$$L_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \left(|\mathbf{u}^n - \mathbf{u}_i^n|^2 + |\omega^n - \nabla \times \mathbf{u}_i^n|^2 \right), \tag{4d}$$

where $\omega^n = [\omega_x^n, \omega_y^n, \omega_z^n]^T$ denotes the vorticity for the nth data point by NSFnet; e_{vvi}^n represents the residual of the ith equation at the nth data point. Note that only boundary and initial values of velocity are provided in the loss function. For the vorticity term, the boundary and initial conditions are embedded in the losses (4c) and (4d) as constraints. The parameters of the neural network are also learned by using the *Adam* optimizer. Because higher-order derivatives and more equations are involved in the VV-NSFnet, its computational cost is higher than the cost of VP-NSFnet.

We note that each dimension of the inputs of NSFnets are normalized to $[-1, 1]$, and the corresponding governing equations are also normalized by the same factor. Note that the weighting coefficients in the loss functions (2) and (4) play a very important role in the training process. However, choosing appropriate weights for NSFnets is generally very tedious. On the one hand, the optimal values of α and β are problem-dependent and we cannot fix them for different flows. On the other hand, tuning the weights arbitrarily requires a trial and error procedure which is quite tedious and time-consuming. To tackle this problem, we apply the strategy of dynamic weights [25] for choosing α and β in NSFnet simulations. The idea of dynamic weights is to adaptively update the coefficients by utilizing the back-propagated gradient statistics during network training. For a general gradient descent algorithm, the iterative formulation of the parameters of NSFnets can be expressed as:

$$\theta^{(k+1)} = \theta^{(k)} - \eta\nabla_\theta L_e - \eta\alpha\nabla_\theta L_b - \eta\beta\nabla_\theta L_i, \tag{5}$$

where θ denotes the parameters of the neural network, namely the weights of all the fully-connected layers, k is the iteration step, and η is the learning rate. In order to balance the contributions of different terms in Eq. (5), Wang et al. [25] proposed to use a dynamic weight strategy during network training. At each training step, e.g., $(k+1)$th iteration, the estimates of α and β can be computed by:

$$\hat{\alpha}^{(k+1)} = \frac{\max_\theta\{|\nabla_\theta L_e|\}}{\overline{|\nabla_\theta \alpha^{(k)} L_b|}}, \quad \hat{\beta}^{(k+1)} = \frac{\max_\theta\{|\nabla_\theta L_e|\}}{\overline{|\nabla_\theta \beta^{(k)} L_i|}}, \tag{6}$$

where $\max_\theta\{|\nabla_\theta L_e|\}$ is the maximum value attained by $|\nabla_\theta L_e|$; $\overline{|\nabla_\theta \alpha^{(k)} L_b|}$ and $\overline{|\nabla_\theta \beta^{(k)} L_i|}$ denote the means of $|\nabla_\theta \alpha^{(k)} L_b|$ and $|\nabla_\theta \beta^{(k)} L_i|$, respectively. We note that the initial values of dynamic weights should be provided. As an alternative, we also propose the following way to estimate α and β:

$$\hat{\alpha}^{(k+1)} = \frac{\overline{|\nabla_\theta L_e|}}{\overline{|\nabla_\theta L_b|}}, \quad \hat{\beta}^{(k+1)} = \frac{\overline{|\nabla_\theta L_e|}}{\overline{|\nabla_\theta L_i|}}. \tag{7}$$

The gradients with respect to parameters of the neural network can be easily computed by AD in the deep learning framework. Consequently, the weighting coefficients for the next iteration are updated using a moving average form:

$$\alpha^{(k+1)} = (1-\lambda)\alpha^{(k)} + \lambda\hat{\alpha}^{(k+1)}, \quad \beta^{(k+1)} = (1-\lambda)\beta^{(k)} + \lambda\hat{\beta}^{(k+1)}. \tag{8}$$

The coefficient λ in Eq. (8) is a hyperparameter determining how fast the contributions of previous dynamic weights decay. Smaller λ makes the previous values contribute more to the current dynamic weights, which will ensure the adjustment stable during training. Therefore, $\lambda = 0.1$ is selected. The strategy of dynamic weights will be applied to most of the NSFnet simulations later.

Usually the residual points are randomly sampled in the computational domain. To enhance the solution accuracy of PINNs with respect to residual points distribution, we also employ residual-based adaptive refinement (RAR) [34]: first randomly sample the initial residual points to train the NSFnet; then randomly collect $\mathcal{N}_e$ points to evaluate the absolute physical loss $\mathcal{L}_e = \frac{1}{\mathcal{N}_e}\sum_i \sum_{n=1}^{\mathcal{N}_e} |e_i^n|$ (here, e_i^n represents the residual of the ith equation at the nth data point), and repeat adding residual points to the locations where the magnitude of $\mathcal{L}_e^n = \sum_i |e_i^n|$ is large to train the NSFnet until $\mathcal{L}_e$ is smaller than a prescribed threshold $\mathcal{E}_0$.

We have introduced two different formulations of NSFnets which correspond to the VP form and the VV form. We carry out several numerical experiments with NSFnets of different sizes. However, further accuracy enhancement may be possible for each case presented below using optimization in the size of architecture, the learning rate and even the optimizer. Such parametric and sensitivity studies are presented in the Appendix but future works should rely on meta-learning and the automation of this process.

3. Simulations of laminar flows

In this section, we apply the proposed NSFnets to simulate different incompressible Navier-Stokes flows, including 2D steady Kovasznay flow, 2D unsteady cylinder wake and 3D unsteady Beltrami flow. We present comparisons between the VV and VP-NSFnets and investigate the influence of dynamic weights and RAR on the accuracy of the solution. Other enhancements can include the use of adaptive activation function to accelerate training [35,36], but we did not pursue this in the current work. To evaluate the performance of the NSFnet simulations, we define the relative L_2 error at each time step as

$$\epsilon_V = \| \hat{V} - V \|_2 / \| V \|_2, \tag{9}$$

where V denotes the velocity components (u, v, w) or the pressure p, and the hat represents the values inferred by NSFnets. We also consider the error convergence with the number of residual points and boundary points using the relative L_2 error. The reference velocity and pressure are given by analytical solutions or high-fidelity DNS results. We note that to evaluate the accuracy of NSFnet solutions, we apply a shift for the NSFnet simulation results to bring the means of the pressure for reference DNS results and NSFnet results to the same value.

Table 1
Kovasznay flow: relative L_2 errors of velocity and pressure solutions for NSFnets with different sizes. The number of residual points is 2,601. The number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weight is $\alpha = 100$.

NN size	VP-NSFnet			VV-NSFnet	
	ϵ_u(%)	ϵ_v(%)	ϵ_p(%)	ϵ_u(%)	ϵ_v(%)
4×50	0.0072±0.0010	0.0584±0.0081	0.0271±0.0041	0.0223±0.0046	0.0635±0.0122
7×50	0.0054±0.0012	0.0431±0.0037	0.0141±0.0035	0.0069±0.0010	0.0351±0.0041
7×100	**0.0026±0.0005**	**0.0260±0.0066**	**0.0094±0.0023**	0.0032±0.0008	0.0169±0.0033
10×100	0.0027±0.0016	0.0260±0.0052	0.0111±0.0050	**0.0024±0.0003**	**0.0144±0.0022**
10×250	0.0059±0.0009	0.0506±0.0111	0.0307±0.0135	0.0040±0.0005	0.0185±0.0018

3.1. Kovasznay flow

We use the Kovasznay flow as the first test case to demonstrate the performance of NSFnets. This 2D steady Navier-Stokes flow has the following analytical solution:

$$\begin{aligned} u(x, y) &= 1 - e^{\zeta x}\cos(2\pi y), \\ v(x, y) &= \frac{\zeta}{2\pi} e^{\zeta x}\sin(2\pi y), \\ p(x, y) &= \frac{1}{2}(1 - e^{2\zeta x}), \end{aligned} \tag{10}$$

where

$$\zeta = \frac{1}{2\nu} - \sqrt{\frac{1}{4\nu^2} + 4\pi^2}, \quad \nu = \frac{1}{\mathrm{Re}} = \frac{1}{40}.$$

We consider a computational domain of $[-0.5, 1.0] \times [-0.5, 1.5]$. We first determine the optimization strategy. There are 101 points with fixed spatial coordinate on each boundary, i.e., $N_b = 4 \times 101$. For computing the equation loss of NSFnets, 2,601 points are randomly selected inside the domain. There is no initial condition for this steady flow. We use the *Adam* optimizer to provide a better set of initial neural network learnable variables. Then, L-BFGS-B is used to finetune the neural networks to obtain higher accuracy. The training process of L-BFGS-B is terminated automatically based on the increment tolerance. In this section, we use 3×10^4 *Adam* iterations with learning rate 10^{-3} before the L-BFGS-B training. The influence of the number of *Adam* iterations is discussed in Fig. A.1 in Appendix A, where we also investigate the performance of NSFnets with respect to the number of sampling points and boundary points.

We then investigate the influence of different NSFnet architectures. The fixed weighting coefficient $\alpha = 100$ for boundary conditions is chosen for training these NSFnets. We note that all the loss components for labeled data are in L_b, and for the unsupervised part are in L_e when solving Kovasznay flow. The results are summarized in Table 1 for the number of residual points 2,601. We note that five independent runs with independent initial network parameters are implemented. As shown in the table, both formulations of NSFnets with proper neural network architecture are able to attain the solutions with high accuracy. The relative L_2 errors can even reach the order of 10^{-5}. We can also observe that there exists overfitting when increasing the neural network size. Note that while here the number of residual points is 2,601, we also tried a smaller number of residual points for different network sizes, namely 144 and 400 points. The results are summarized in Appendix A in Tables A.1, A.2, respectively. With smaller number of residual points, the NSFnets are more prone to overfitting. For instance, the best neural network architecture considered for VP-NSFnet with 144 residual points is 4×50, while the best neural network architecture for VP-NSFnet with 2,601 residual points is 7×100. Moreover, the VP-NSFnet is more prone to overfitting than VV-NSFnet. Also, the highest accuracy for VV-NSFnet with the same number of residual points is higher than that of VP-NSFnet, i.e., the VV-NSFnet outperforms VP-NSFnet when using large networks.

The weighting coefficient α for the boundary constraint, together with the error convergence with the number of residual points, is also investigated here. In addition to letting $\alpha = 100$, we also apply $\alpha = 1$ and implement dynamic weights (i.e., Eqs. (6) and (7)) for comparisons. In this assessment, we employ a small neural network with 4 hidden layers and 50 neurons per layer, and the number of boundary points is 4×101. We update the dynamic weights every 100 training epochs. The learning rate and optimization method are the same as mentioned above. This strategy is consistent with the use of dynamic weights in [25]. The dynamic weights with different number of residual points during the training process are displayed in Fig. 2. Here α for the data term in loss functions is initialized by 100 for the dynamic weighting strategy. We see that the evolution of dynamic weights during the training process does not vary noticeably when the number of residual points is changed. The resulting limit values of α are in the order of 20 for VP-NSFnet given by Eq. (6) and 10 for VP-NSFnet given by Eq. (7), while they are in the order of 100 for VV-NSFnet given by Eq. (6) and 400 for VV-NSFnet given by Eq. (7). The corresponding loss functions obtained by VP-NSFnet with $\alpha = 1$, $\alpha = 100$ and dynamic weighting strategy are presented in Figs. 3(a), (b) and (c), and for the VV-NSFnet in Figs. 3(d), (e) and (f). From the curves of the training loss, we find that the *Adam* optimizer is robust for the VP-NSFnet while it does not perform consistently for the VV-NSFnet.

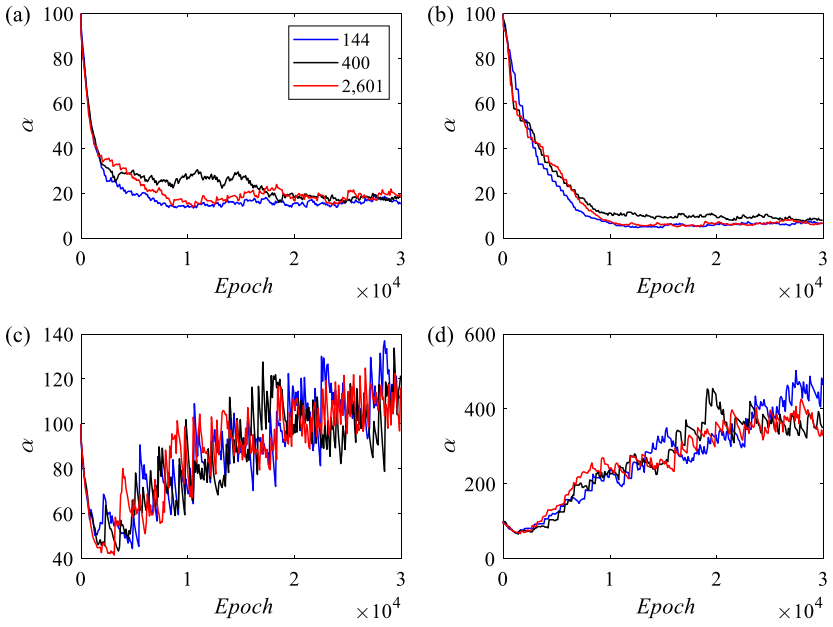

Fig. 2. Kovasznay flow: dynamic weights for (a) VP-NSFnet, given by Eq. (6); (b) VP-NSFnet, given by Eq. (7); (c) VV-NSFnet, given by Eq. (6); (d) VV-NSFnet, given by Eq. (7). The NN size is 4×50, the number of residual points is 144, 400, 2,601, and the number of boundary points is 4×101. To clearly show the evolution of dynamic weights, only one from five independent runs is selected.

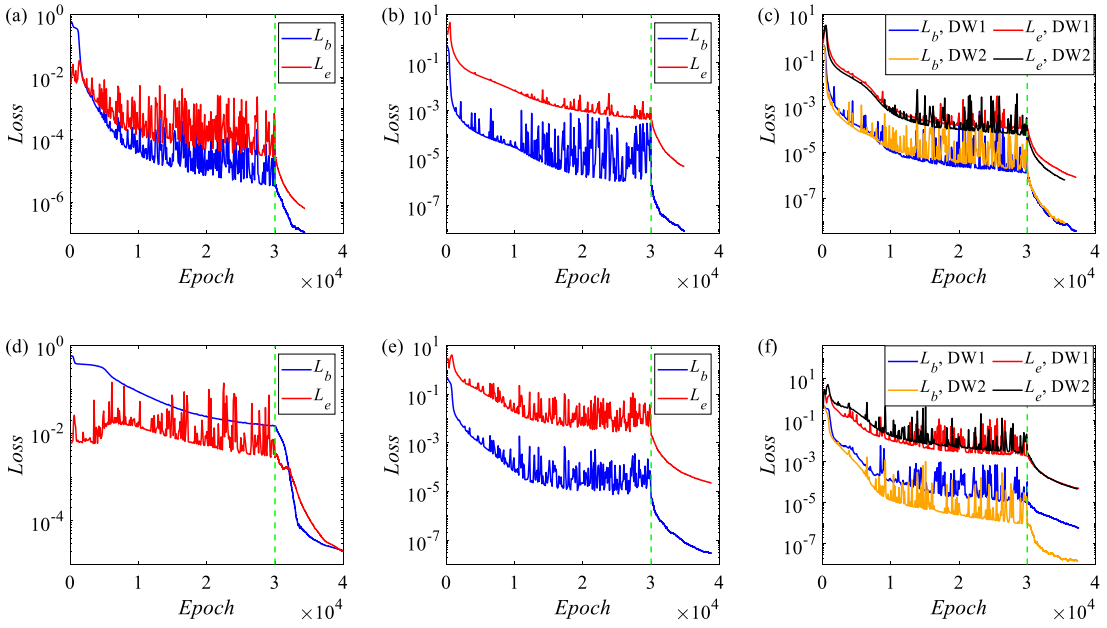

Fig. 3. Kovasznay flow: loss functions (physics loss L_e and boundary loss L_b) obtained by (a) VP-NSFnet, fixed weight $\alpha = 1$; (b) VP-NSFnet, fixed weight $\alpha = 100$; (c) VP-NSFnet, dynamic weight; (d) VV-NSFnet, fixed weight $\alpha = 1$; (e) VV-NSFnet, fixed weight $\alpha = 100$; (f) VV-NSFnet, dynamic weight. "DW1" denotes dynamic weights given by Eq. (6), and "DW2" denotes dynamic weights given by Eq. (7). The *Adam* optimizer is used before the vertical dashed green line, and the L-BFGS-B optimizer is used after that. The NN size is 4×50. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Applying a two-stage optimization strategy can obtain more consistent results. The error convergence with the number of residual points for the NSFnet simulations with different weights is discussed in Appendix A.

In order to demonstrate the effectiveness of dynamic weights, we analyze the gradients of the loss function with respect to the parameters of the NSFnets. The histograms of the back-propagated gradients ($\nabla_\theta L_e$ and $\nabla_\theta(\alpha L_b)$) after 10,000 iterations are shown in Fig. 4. Our goal is to balance the distributions of $\nabla_\theta L_e$ and $\nabla_\theta(\alpha L_b)$, thus these two terms can contribute equally to the parameter updating (i.e., Eq. (5)). As shown in Figs. 4(a) and 4(e), the gradients of two different terms are un-

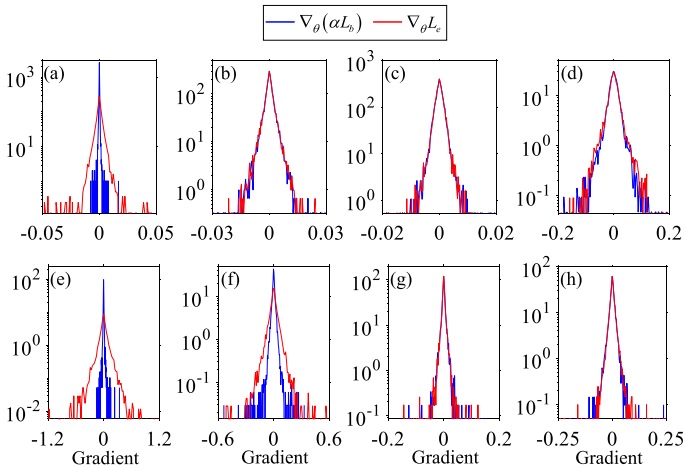

Fig. 4. Kovasznay flow: histograms of the gradients ($\nabla_\theta L_e$ and $\nabla_\theta(\alpha L_b)$) after 10,000 iterations during training the NSFnets, (a) VP-NSFnet, fixed weight $\alpha = 1$; (b) VP-NSFnet, fixed weight $\alpha = 100$; (c) VP-NSFnet, dynamic weight (Eq. (6)); (d) VP-NSFnet, dynamic weight (Eq. (7)); (e) VV-NSFnet, fixed weight $\alpha = 1$; (f) VV-NSFnet, fixed weight $\alpha = 100$; (g) VV-NSFnet, dynamic weight (Eq. (6)); (h) VV-NSFnet, dynamic weight (Eq. (7)). The NN size is 4×50, the number of residual points is 2,601, and the number of boundary points is 4×101. To clearly show the results, only one from five independent runs is selected.

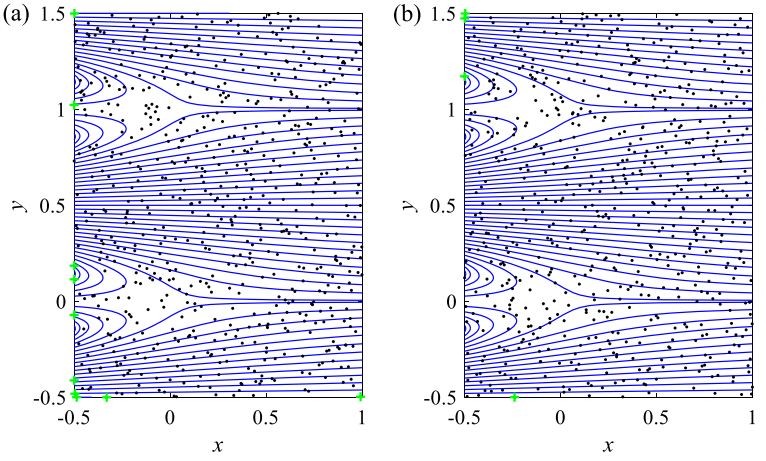

Fig. 5. Kovasznay flow: two examples for the residual points added via RAR for (a) VP-NSFnet and (b) VV-NSFnet, respectively. Black dots: initial residual points; green plus: added residual points; 10 and 4 residual points are added, in (a) and (b), respectively. The blue lines are the streamlines of flow field. The NN size is 4×50, the number of boundary points is 4×101, and the initial number of residual points is 625.

balanced when there is no weighting coefficient for the boundary constraints (i.e., $\alpha = 1$). When applying dynamic weights, the histograms of $\nabla_\theta L_e$ and $\nabla_\theta(\alpha L_b)$ are more consistent with each other, which leads to higher accuracy.

We then explore the potential of RAR to enhance the Kovasznay flow solution accuracy of NSFnets with respect to the distribution of residual points. The NN size is 4×50, the number of boundary points is 4×101, and the initial number of residual points is 625. The weight is fixed at $\alpha = 100$. The optimization strategy is as follows [34]: we first randomly sample 625 initial residual points to train the NSFnet with *Adam* and L-BFGS-B; then randomly collect $\mathcal{N}_e = 100,000$ points to evaluate the absolute physical loss $\mathcal{L}_e$, and repeat adding one residual point to the locations where $\mathcal{L}_e^n$ is large to train the NSFnets until $\mathcal{L}_e$ is smaller than a prescribed threshold $\mathcal{E}_0 = 2 \times 10^{-3}$ for VP-NSFnet and $\mathcal{E}_0 = 10^{-2}$ for VV-NSFnet. Fig. 5 illustrates two examples for the residual points added via RAR for NSFnets. Finally, the total number of residual points is 635 for VP-NSFnet and 629 for VV-NSFnet, respectively. We can find that the added points are mostly close to the inlet, which indicates the significance in the region near the boundary. We also implement the random sampling strategy for VP-NSFnet with 635 residual points and VV-NSFnet with 629 residual points, and the results are shown in Table 2. The results for random sampling represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The solution accuracy is obviously improved by about 50% via RAR.

Table 2
Kovasznay flow: relative L_2 errors of velocity and pressure solutions for NSFnets with residual points added via RAR. The NN size is 4×50, and the number of boundary points is 4×101. The total number of residual points is 635 for VP-NSFnet and 629 for VV-NSFnet, respectively.

Sampling	VP-NSFnet			VV-NSFnet	
	$\epsilon_u(\%)$	$\epsilon_v(\%)$	$\epsilon_p(\%)$	$\epsilon_u(\%)$	$\epsilon_v(\%)$
Random	0.0095±0.0026	0.0823±0.0167	0.0401±0.0061	0.0254±0.0058	0.0704±0.0160
RAR	0.0056	0.0369	0.0315	0.0083	0.0280

3.2. Two-dimensional cylinder wake

Here we use NSFnets to simulate the 2D vortex shedding behind a circular cylinder at $\mathrm{Re} = 100$. The cylinder is placed at $(x, y) = (0, 0)$ with diameter $D = 1$. High-fidelity DNS data from [15] is used as a reference and for providing boundary and initial data for NSFnet training. We consider a domain defined by $[1, 8] \times [-2, 2]$ and the time interval is $[0, 7]$ (more than one shedding period) with time step $\Delta t = 0.1$. As for the training data, we place 100 points along the x-direction boundary and 50 points along the y-direction boundary to enforce the boundary conditions and use 140,000 spatio-temporal scattered points inside the domain to compute the residuals. The NSFnets contain 10 hidden layers and 100 neurons per layer. In addition to the default models with $\alpha = \beta = 1$ and $\alpha = \beta = 100$, we again implement the dynamic weighting strategy for NSFnets. The training procedure is a bit different from the one we used for Kovasznay flow. All the NSFnets are trained using 5 000, 5 000, 50 000 and 50 000 *Adam* iterations with learning rates of 1×10^{-3}, 1×10^{-4}, 1×10^{-5} and 1×10^{-6}, respectively; finally, L-BFGS-B is used for finetuning.

A snapshot of the vorticity contours at $t = 4.0$ is shown in Fig. 6, demonstrating qualitative agreement of NSFnet inference with the DNS result. In Fig. 7 we present the dynamic weights of both VP- and VV-NSFnets. In this case, the weights are both initialized by the value 1. The variations of the weights correspond to the changes of learning rates. We can observe that both α and β are in the order of 10 and the weights for initial conditions are larger than the weights for boundary conditions in this case. The separated terms of the weighted loss function during training are given in Fig. 8. We employ a two-stage training to ensure the convergence for all the NSFnets. The relative L_2 errors of NSFnet simulations versus time are given in Fig. 9. We see that the VV-NSFnet performs better than the VP-NSFnet, and also that applying the dynamic weights can improve the simulation accuracy for both formulations.

3.3. Three-dimensional Beltrami flow

The analytical solutions of the unsteady three-dimensional Beltrami flow developed by Ethier and Steinman [37] are:

$$\begin{aligned}
u(x, y, z, t) &= -a\left[e^{ax}\sin(ay + dz) + e^{az}\cos(ax + dy)\right]e^{-d^2t},\\
v(x, y, z, t) &= -a\left[e^{ay}\sin(az + dx) + e^{ax}\cos(ay + dz)\right]e^{-d^2t},\\
w(x, y, z, t) &= -a\left[e^{az}\sin(ax + dy) + e^{ay}\cos(az + dx)\right]e^{-d^2t},\\
p(x, y, z, t) &= -\frac{1}{2}a^2\Big[e^{2ax} + e^{2ay} + e^{2az} + 2\sin(ax + dy)\cos(az + dx)e^{a(y+z)}\\
&\quad + 2\sin(ay + dz)\cos(ax + dy)e^{a(z+x)}\\
&\quad + 2\sin(az + dx)\cos(ay + dz)e^{a(x+y)}\Big]e^{-2d^2t},
\end{aligned} \tag{11}$$

where $a = d = 1$ is used. In NSFnet simulations, the computational domain is defined by $[-1, 1] \times [-1, 1] \times [-1, 1]$ and the time interval is $[0, 1]$. For the NSFnet training data, 31×31 points on each face are used for boundary conditions while a batch of 10,000 points in the spatio-temporal domain is used for the equations. The weighting coefficients of the loss function are both fixed in this case: $\alpha = \beta = 100$. We note that all the loss components for labeled data are in L_b and L_i, and for the unsupervised part are in L_e when solving Beltrami flow.

The two-stage optimization (*Adam* with learning rate 10^{-3}, 30,000 epochs and L-BFGS-B) is implemented to train the neural networks, and we considered two NN sizes 4×50 and 7×50. The temporal mean relative L_2 errors of velocity and pressure solutions for NSFnets with two NN sizes are given in Table 3. The temporal relative L_2 errors of velocity and pressure solutions for NSFnets with NN size 7×50 are shown in Table 4. We can observe that, unlike the Kovasznay flow, as we increase the NN size we obtain higher accuracy, with the VV-NSFnet outperforming the VP-NSFnet. A snapshot of the velocity fields together with the absolute errors at $t = 1.00$ and on the slice $z = 0$ is displayed in Fig. 10. The absolute errors of the VV-NSFnet are almost an order lower than those of the VP-NSFnet. The areas of low accuracy for VP-NSFnet are concentrated in the center while the error distribution for VV-NSFnet is more uniform than for VP-NSFnet.

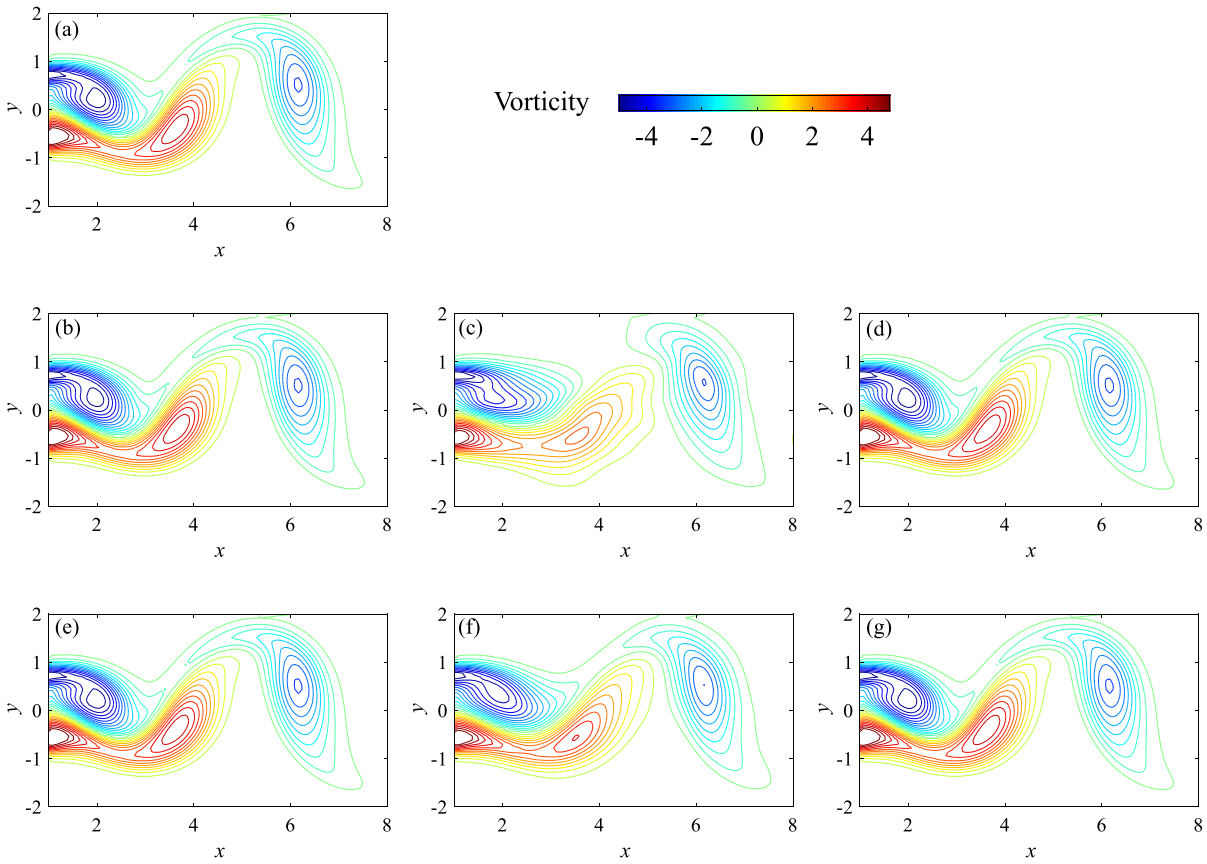

Fig. 6. Flow past a circular cylinder: contours of the vorticity on the same contour levels at $t = 4.0$: (a) reference DNS solution from [15]; (b) VP-NSFnet, fixed weights $\alpha = \beta = 1$; (c) VP-NSFnet, fixed weights $\alpha = \beta = 100$; (d)VP-NSFnet, dynamic weights; (e) VV-NSFnet, fixed weights $\alpha = \beta = 1$; (f) VV-NSFnet, fixed weights $\alpha = \beta = 100$; (g) VV-NSFnet, dynamic weights. The dynamic weights here are given by Eq. (6).

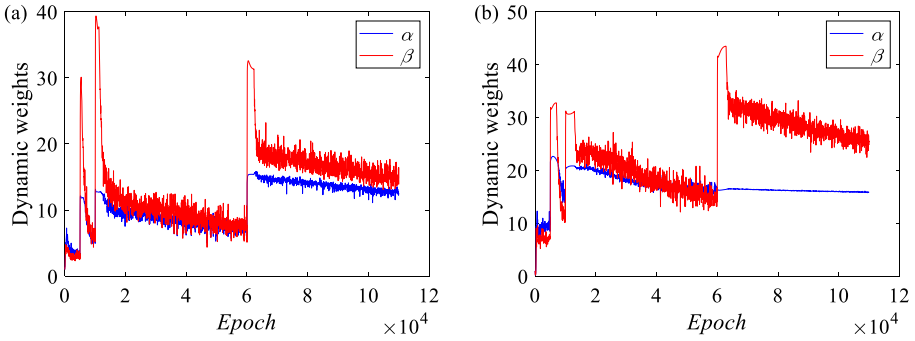

Fig. 7. Flow past a circular cylinder: dynamic weights given by Eq. (6) for (a) VP-NSFnet and (b) VV-NSFnet. The oscillations are due to the changes of learning rate in this case.

4. Simulations of turbulent channel flow

4.1. Problem setup

We simulate turbulent channel flow at $\mathrm{Re}_\tau = 9.9935 \times 10^2$ systematically by using VP-NSFnets. We use the turbulent channel flow database [20–22] at http://turbulence.pha.jhu.edu as the reference DNS solution. The database provides both the reference and some initial or boundary conditions for the VP-NSFnet. The DNS domain for the channel flow in the database is $[0, 8\pi] \times [-1, 1] \times [0, 3\pi]$; the mean pressure gradient is $dP/dx = 0.0025$. The non-dimensional time step for DNS is 0.0013 while the online database time step is 0.0065 (five times of that for DNS). So, the time step of 0.0065 is also used for evaluating the residuals in NSFnets, i.e., five times of that for DNS. We perform NSFnet simulations by considering

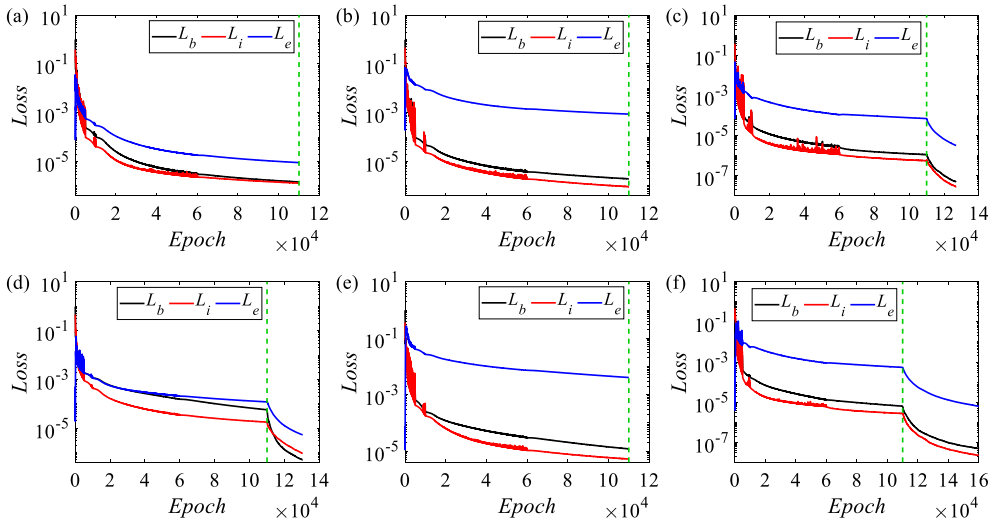

Fig. 8. Flow past a circular cylinder: loss functions (physics loss L_e, boundary loss L_b and initial loss L_i) obtained by (a) VP-NSFnet, fixed weights $\alpha = \beta = 1$; (b) VP-NSFnet, fixed weights $\alpha = \beta = 100$; (c) VP-NSFnet, dynamic weights given by Eq. (6); (d) VV-NSFnet, fixed weights $\alpha = \beta = 1$; (e) VV-NSFnet, fixed weights $\alpha = \beta = 100$; (f) VV-NSFnet, dynamic weights given by Eq. (6). The *Adam* optimizer is used before the dashed green line, and the L-BFGS-B optimizer is used after the dashed green line. The NN size is 10×100.

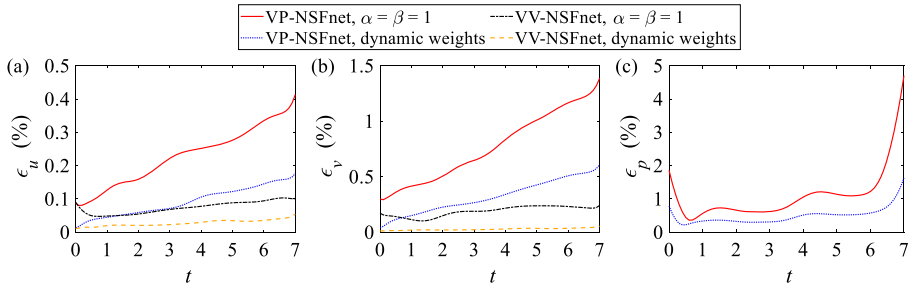

Fig. 9. Flow past a circular cylinder: relative L_2 errors of NSFnets simulations for (a) the streamwise velocity, (b) the crossflow velocity and (c) pressure. The dynamic weights here are given by Eq. (6).

Table 3
Beltrami flow: temporal mean relative L_2 errors of velocity and pressure solutions for NSFnets with different sizes (4×50 and 7×50). The number of residual points is 10,000. The number of boundary points is $6 \times 31 \times 31$. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weights are $\alpha = \beta = 100$.

	$\overline{\epsilon}$	4×50	7×50
VP	$\overline{\epsilon}_u$(%)	0.0856±0.0095	0.0731±0.0149
	$\overline{\epsilon}_v$(%)	0.1497±0.0556	0.1016±0.0191
	$\overline{\epsilon}_w$(%)	0.1530±0.0476	0.0968±0.0226
	$\overline{\epsilon}_p$(%)	13.6513±1.0952	8.9123±1.7447
VV	$\overline{\epsilon}_u$(%)	0.0254±0.0033	0.0238±0.0033
	$\overline{\epsilon}_v$(%)	0.0240±0.0017	0.0238±0.0020
	$\overline{\epsilon}_w$(%)	0.0260±0.0042	0.0241±0.0016

different subdomains with different sizes at various locations in the channel. In the first example, we place a box with size ∼200 in wall-units covering a long time period. Then, we test the NSFnet simulation at a larger domain covering half the channel height. Finally, we check the influence of weighting coefficients to the NSFnet simulation accuracy. According to the dataset description, the viscous sublayer covers the range of $y^+ < 30$. Therefore, the simulation over a large domain in section 4.3 covers the whole sublayer, while the simulation in section 4.4 covers the range of $y^+ < 24$; there is no viscous sublayer involved in the simulations of section 4.2. We note that the simulations in the subdomains begin with fully developed turbulence as initial conditions. We employ mini-batch to train NSFnets in this study. There are three parts in the input data corresponding to the initial conditions, the boundary conditions and the residuals of equations, respectively.

Table 4
Beltrami flow:relative L_2 errors of velocity and pressure solutions for NSFnets with NN size 7×50. The number of residual points is 10,000. The number of boundary points is $6 \times 31 \times 31$. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weights are $\alpha = \beta = 100$.

	ϵ	$t = 0$	$t = 0.25$	$t = 0.50$	$t = 0.75$	$t = 1.00$
VP	$\epsilon_u(\%)$	0.0110±0.0015	0.0408±0.0085	0.0594±0.0112	0.0908±0.0178	0.1634±0.0418
	$\epsilon_v(\%)$	0.0119±0.0021	0.0548±0.0126	0.0917±0.0181	0.1314±0.0226	0.2185±0.0530
	$\epsilon_w(\%)$	0.0114±0.0015	0.0636±0.0134	0.0987±0.0370	0.1320±0.0463	0.1783±0.0300
	$\epsilon_p(\%)$	9.4342±1.4554	8.9781±1.6447	8.6570±1.8246	8.5588±2.0414	8.9335±2.4350
VV	$\epsilon_u(\%)$	0.0074±0.0014	0.0167±0.0038	0.0223±0.0050	0.0294±0.0041	0.0435±0.0061
	$\epsilon_v(\%)$	0.0071±0.0007	0.0153±0.0030	0.0210±0.0034	0.0302±0.0036	0.0452±0.0056
	$\epsilon_w(\%)$	0.0074±0.0007	0.0185±0.0033	0.0243±0.0021	0.0291±0.0030	0.0409±0.0028

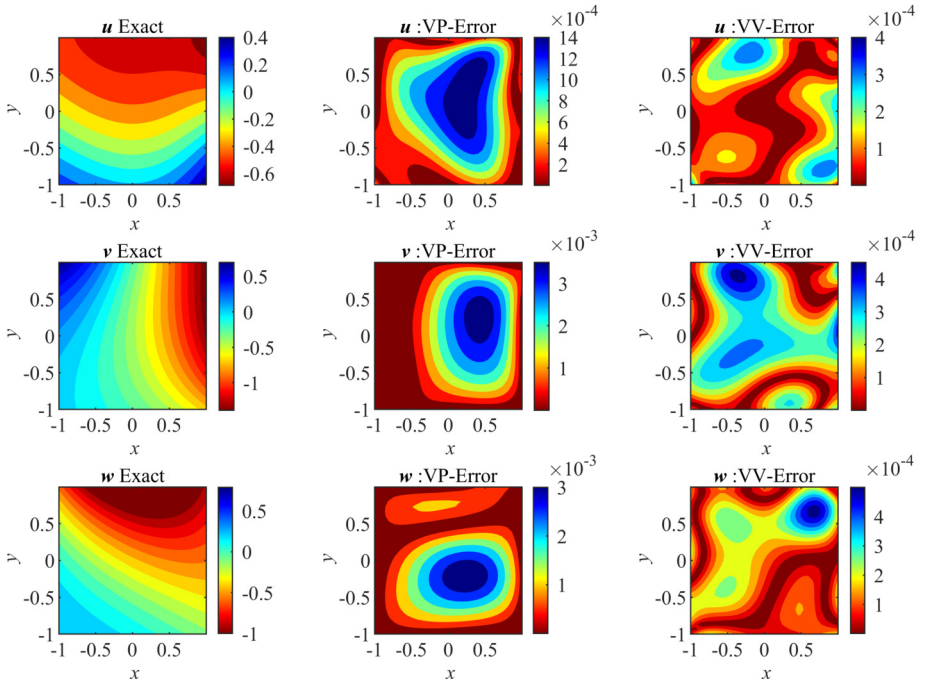

Fig. 10. Beltrami flow: velocity fields and error distribution at $t = 1.00$ on the plane $z = 0$.

Therefore, we specify the total number of iterations n_{it} in one training epoch, and data in each part are evenly divided into n_{it} small mini-batches. The total data in an entire mini-batch include data from each small mini-batch. We note that we use all the DNS data from the first snapshot for the initial conditions and all the DNS data at the NSFnet computational domain boundaries for the boundary conditions of NSFnets. However, to reduce the computational time, the number of residual points at each time step is less than the number of points for the initial conditions, but we still obtain accurate NSFnet solutions. In fact, the number and distribution of residual points can be further refined by RAR [34], which is beyond the scope of this study. As for the NN size, we increased it from a small size NN (4×50) and found that the present size did not exhibit overfitting, similar to the implementation for Beltrami flow. The simulation over a long time interval has been repeated twice with different initial NSFnet parameters, but not much deviation has been observed; thus, we report only one case here. However, the primary goal of the study in this section is to investigate if the VP-NSFnet can sustain turbulence, and our preliminary results are indeed encouraging.

4.2. Simulation results over a long time interval

We first investigate if the VP-NSFnet can sustain turbulence, so we carry out simulations covering a relatively long time interval. In this test, a subdomain at $[12.47, 12.66] \times [-0.90, -0.70] \times [4.61, 4.82]$ ($190 \times 200 \times 210$ in wall-units) is considered as the simulation domain of VP-NSFnet. We perform two different simulations covering the non-dimensional time domain [0, 0.52] (81 time steps, 25.97 in wall-units) and [0, 0.832] (129 time steps, 41.55 in wall-units). Here, we define the local convective time unit of the simulation region as $T_c^+ = L_x^+ / U(y)_{min} = 12.0$. ($L_x^+$ is the size of the domain in streamwise direction.) Therefore, 25.97 and 41.55 cover more than two convective time units, i.e., $2.2T_c^+$ and $3.5T_c^+$ respectively, for this example. We use 20,000 points inside the domain, 6,644 points on the boundary sampled at each

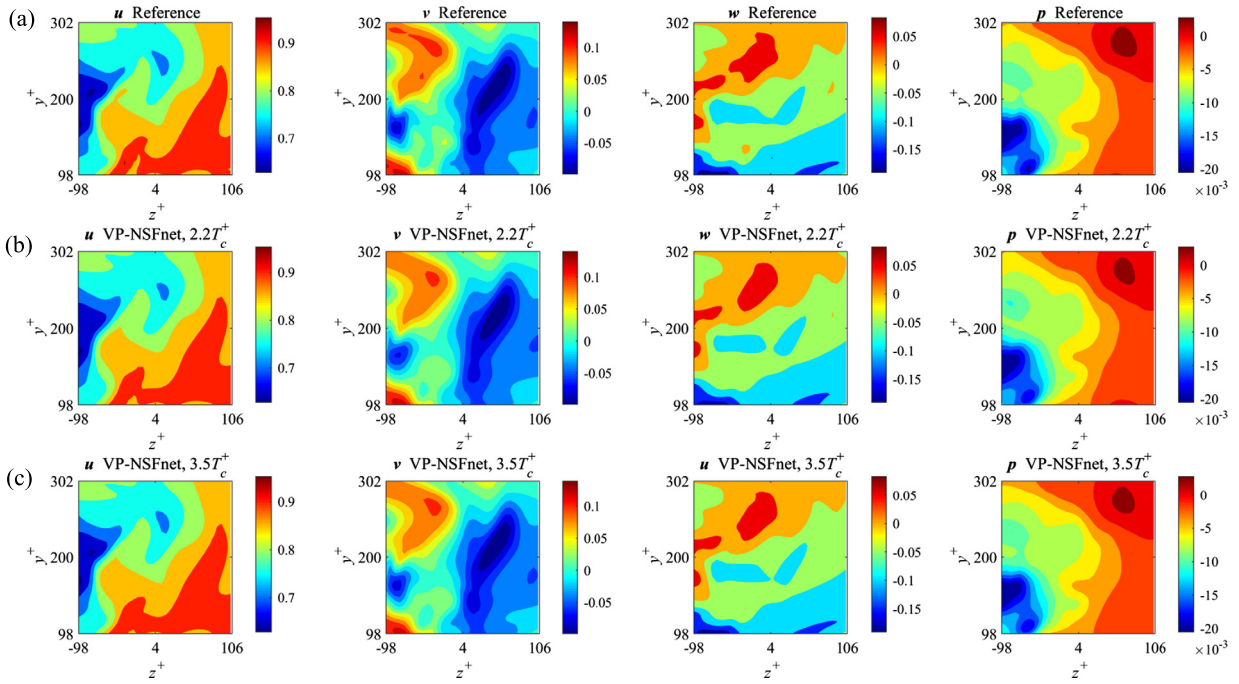

Fig. 11. Long time interval: comparisons of instantaneous $z-y$ plane flow fields between reference DNS and VP-NSFnet at $t^+ = 24.67$: (a) reference solutions; (b) VP-NSFnet, simulation covers $2.2T_c^+$; (c) VP-NSFnet, simulation covers $3.5T_c^+$.

time step, together with 33,524 points at the initial time step to compute the loss function. We set the total number of iterations $n_{it} = 150$ in one training epoch. There are 10 hidden layers in the VP-NSFnet with 300 neurons per layer. The initial learning rate for *Adam* decays from 10^{-3} (1,000 training epochs) to 10^{-4} (4,000 training epochs), 10^{-5} (1,000 training epochs) and 10^{-6} (500 training epochs) in the training process. The weights in Eq. (2a) are $\alpha = 100$, $\beta = 100$. The comparisons of instantaneous flow fields between reference DNS and VP-NSFnet at $t^+ = 24.67$ are given in Fig. 11. The convergence of loss functions is shown in Fig. 12. The comparisons for accuracy of VP-NSFnet solutions are shown in Fig. 13. All the simulation errors of velocity components are less than 10%, but the relative L_2 error of pressure can reach 15% or 19%. As mentioned above, no data is provided for the pressure to the VP-NSFnet, which is a hidden state and is obtained via the incompressibility constraint without splitting the equations. This inferred variable may suffer from lower accuracy on the edge of the time domain (initial and end). This is because more boundary data (from Dirichlet boundary conditions) are used in the middle of the time domain. However, if longer time is simulated, boundary data and residual data will be used for all the time domain, so no divergence will happen if the NSFnets are trained adequately. Overall, a good VP-NSFnet simulation accuracy is obtained. These results suggest that VP-NSFnet can sustain turbulence for a long time period.

4.3. Simulation results over a large domain

We consider a larger domain covering half channel height in this test. We consider a subdomain at $[12.47, 12.66] \times [-1, -0.0031] \times [4.61, 4.82]$ (about $190 \times 997 \times 210$ in wall-units) as the VP-NSFnet simulation domain; and the non-dimensional time domain is set as $[0, 0.104]$ (17 time steps, 5.19 in wall-units). We place 100,000 points inside the domain and 26,048 points on the boundary sampled at each time step, and 147,968 points at the initial time step to determine the loss function. The total number of iterations n_{it} in one training epoch is taken as 150. There are 10 hidden layers in the VP-NSFnet, with 300 neurons per layer. The initial learning rate for *Adam* decays from 10^{-3} (250 training epochs) to 10^{-4} (4,250 training epochs), 10^{-5} (500 training epochs) and 10^{-6} (500 training epochs) in this numerical example. The weights in Eq. (2a) are $\alpha = 100$, $\beta = 100$. Comparisons of instantaneous flow fields between reference DNS and VP-NSFnet at the final simulation time step are shown in Fig. 14. The convergence and accuracy of VP-NSFnet solutions are shown in Fig. 15. All the simulation errors of velocity components are less than 10%, but the relative L_2 error of pressure can reach 17%. In such a large domain, complex interactions between different scales of eddies occur, and this domain covers the whole range including the law of the wall, the viscous sub-layer, the buffer layer, the log-law region and the outer layer [38]. However, VP-NSFnet can still get very accurate solutions. The results also indicate that the relative L_2 errors of the wall-normal and spanwise velocities are much higher than that of streamwise velocity, i.e., nearly an order higher. This is caused by larger amplitude of the streamwise velocity than the other two velocity components, also nearly an order higher, as shown in Fig. 14. A proper normalization with careful tuning of anisotropic weights may result in a more balanced accuracy.

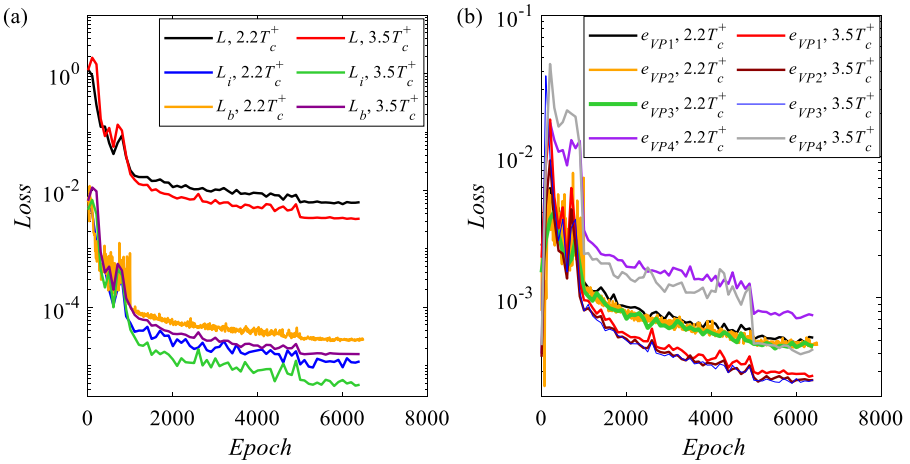

Fig. 12. Long time interval: convergence of the VP-NSFnet simulation: (a) convergence of total loss functions, initial loss functions and boundary loss functions; (b) convergence of residuals of governing equations.

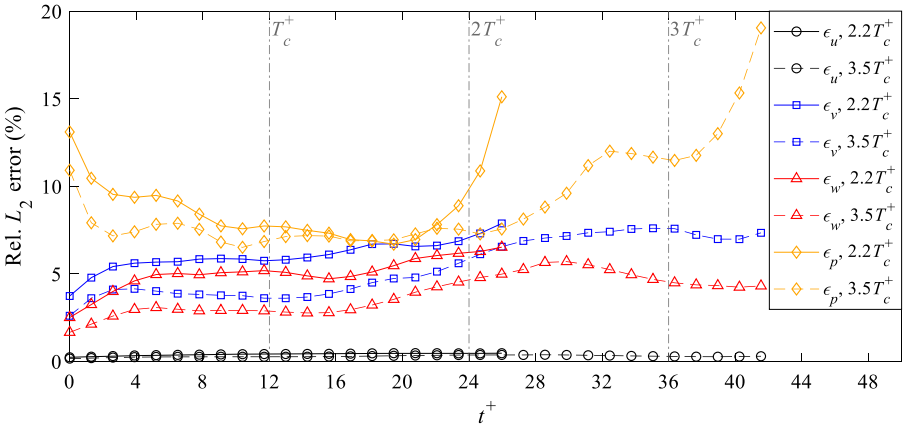

Fig. 13. Long time interval: accuracy of the VP-NSFnet simulation: the solid lines and dashed lines denote simulations covering $2.2T_c^+$ and $3.5T_c^+$, respectively.

4.4. Influence of weights

In the above numerical experiments for turbulent channel flow, all the weights are manually tuned to obtain satisfying results. In this section, we investigate the influence of weights, especially the dynamic weights, to the VP-NSFnet simulation accuracy. The formulation of Eq. (6) is applied to the loss function of VP-NSFnet. However, different from Eq. (6), a normalization factor γ is taken into account in this section. Hence, the dynamic weights for the turbulence simulation can be expressed as:

$$\hat{\alpha}^{(k+1)} = \frac{\max_\theta\{|\nabla_\theta L_e|\}}{\gamma\overline{|\nabla_\theta \alpha^{(k)} L_b|}}. \tag{12}$$

We consider a subdomain of [12.53, 12.59] × [−1, −0.9762] × [4.69, 4.75] (about 60 × 24 × 60 in wall-units) as the VP-NSFnet simulation domain; the non-dimensional time domain is set as [0, 0.104] (17 time steps, 5.19 in wall-units). For this small domain, the initial velocity values are not used in the loss function, i.e., $\beta = 0$, but instead we can learn them. There are 2,000 points inside the domain and 1,100 points on the boundary sampled at each time step to determine the loss function. We take the total number of iterations n_{it} of one training epoch as 10. There are 5 hidden layers in the VP-NSFnet, with 200 neurons per layer. In all the examples, the initial learning rate for *Adam* decays from 10^{-3} (5,000 training epochs) to 10^{-4} (5,000 training epochs), 10^{-5} (25,000 training epochs) and 5×10^{-6} (25,000 training epochs). We set five different strategies to take different weights for the boundary data in this experiment. In the first two strategies, we use fixed weights of $\alpha = 1$ and $\alpha = 100$. Then, we use the dynamic weights given by Eq. (12) with different normalization factors, i.e., $\gamma = 1$, $\gamma = 5$ and $\gamma = 10$. The comparisons of instantaneous flow fields ($z - y$ plane) between reference DNS and VP-NSFnet simulations at $t^+ = 5.19$ and $x^+ = -12.27$ with various weights are shown in Fig. 16. The evolution of

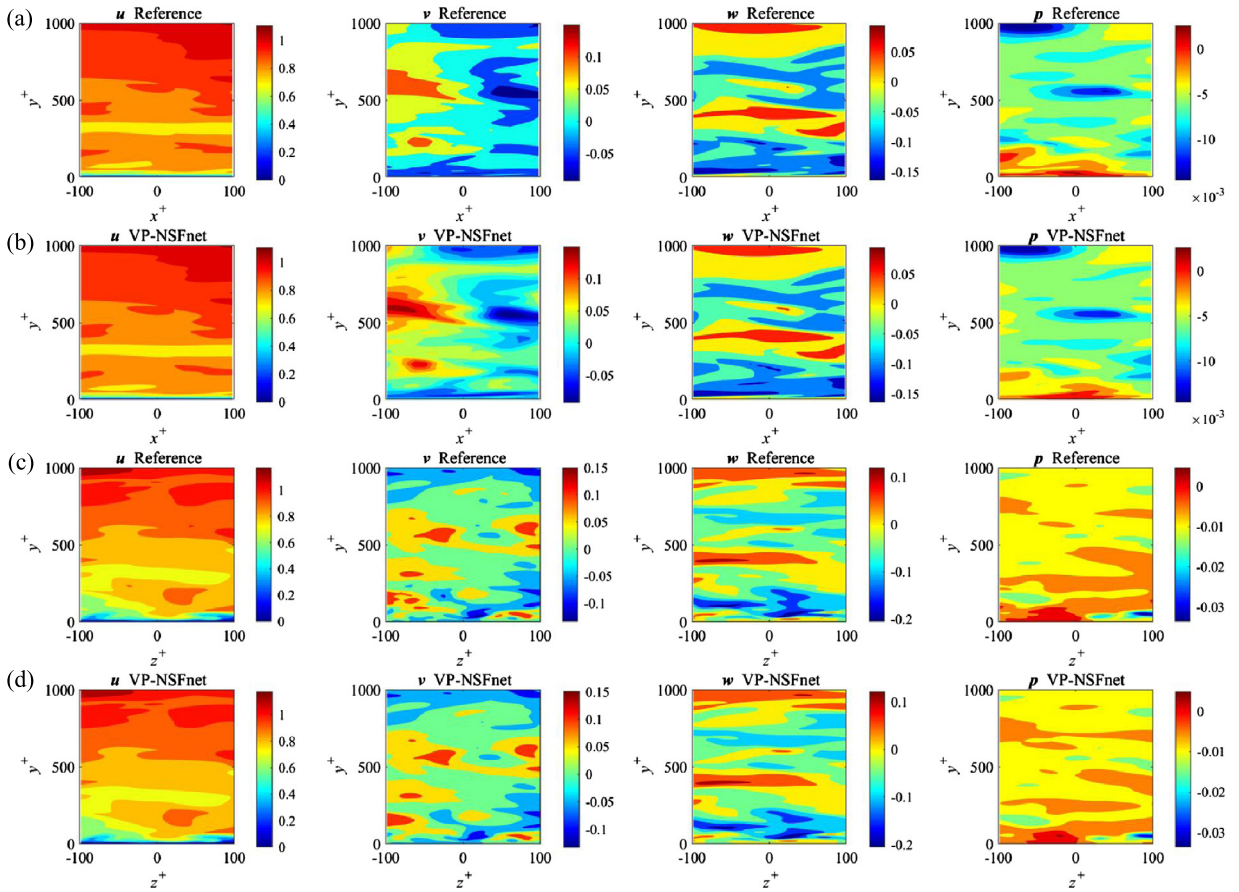

Fig. 14. Large domain: comparisons of instantaneous flow fields between reference DNS and VP-NSFnet at $t^+ = 5.19$ covering half channel height: (a) reference solutions, $x-y$ plane, $z^+ = 0$; (b) VP-NSFnet, $x-y$ plane, $z^+ = 0$; (c) reference solutions, $z-y$ plane, $x^+ = 0$; (d) VP-NSFnet, $z-y$ plane, $x^+ = 0$.

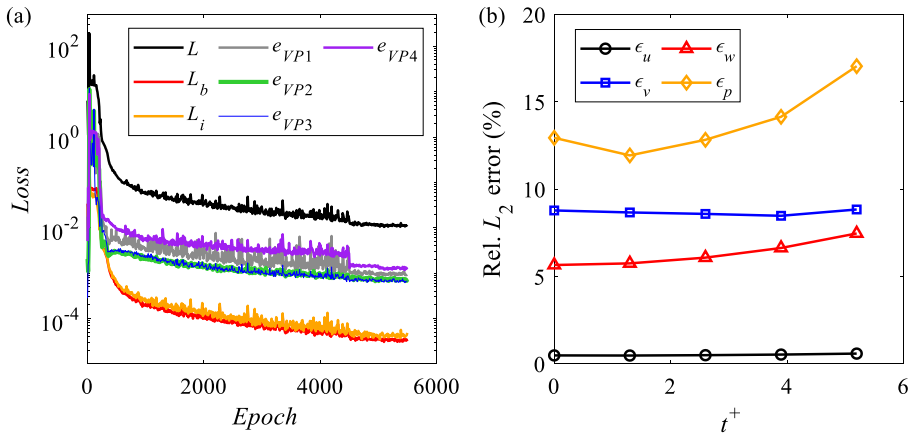

Fig. 15. Large domain: convergence and accuracy of VP-NSFnet covering half channel height: (a) convergence of loss function; (b) relative L_2 error.

dynamic weights in the training process is shown in Fig. 17. The convergence of the VP-NSFnet simulation with various hyperparameters is shown in Fig. 18. The accuracy of the VP-NSFnet simulation with various hyperparameters is shown in Fig. 19. It is noted from Fig. 16 that there exists large discrepancy between the reference solution and the VP-NSFnet solution with fixed weight $\alpha = 1$, thereby its accuracy is of low level. Therefore, the VP-NSFnet simulation accuracy with fixed weight $\alpha = 1$ is not shown in Fig. 19. Overall, the best result for this turbulence simulation is obtained when applying dynamic weights with $\gamma = 5$. A dynamic variation of the weights can improve the performance of VP-NSFnet.

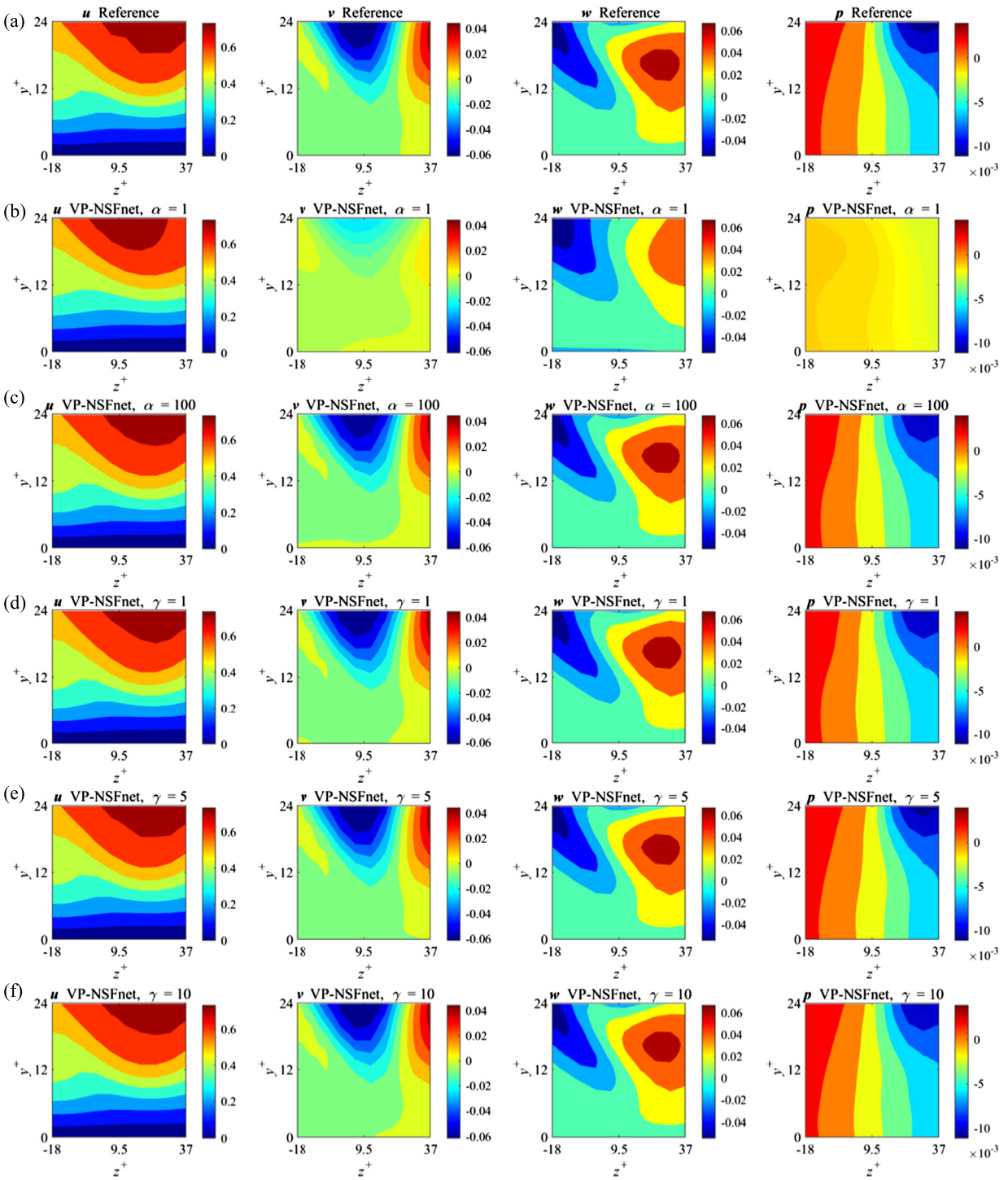

Fig. 16. Influence of weights: comparisons of instantaneous flow fields ($z-y$ plane) between reference DNS and VP-NSFnet at $t^+ = 5.19$ and $x^+ = -12.27$: (a) reference solutions; (b) VP-NSFnet, fixed weight $\alpha = 1$; (c) VP-NSFnet, fixed weight $\alpha = 100$; (d) VP-NSFnet, dynamic weight with normalization factor $\gamma = 1$; (e) VP-NSFnet, dynamic weight with normalization factor $\gamma = 5$; (f) VP-NSFnet, dynamic weight with normalization factor $\gamma = 10$. All the dynamic weights here are given by Eq. (12).

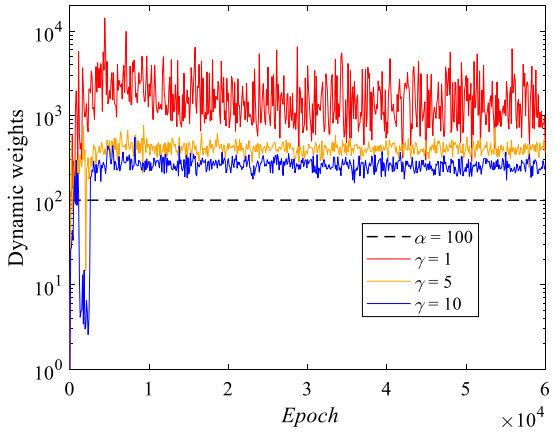

Fig. 17. Influence of weights: dynamical weights with different normalization factors versus training epochs. All the dynamic weights here are given by Eq. (12).

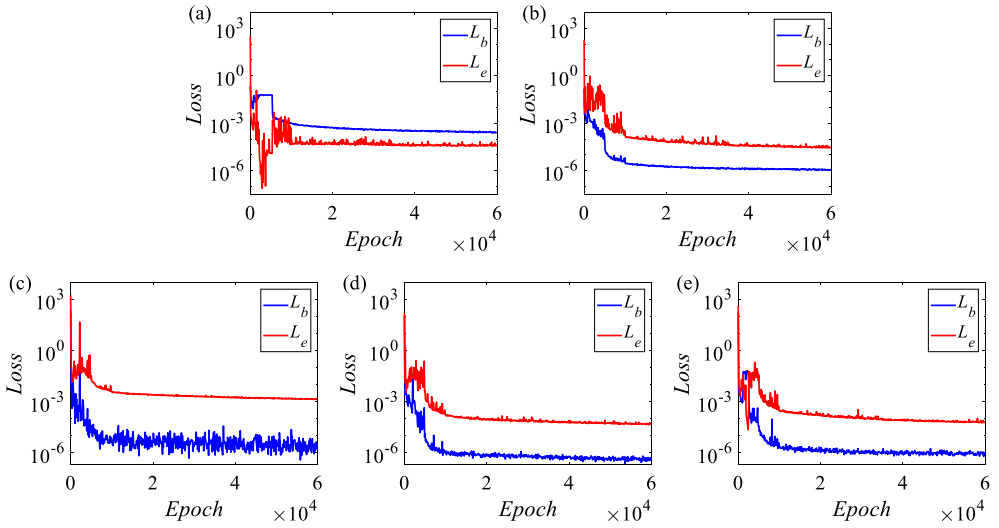

Fig. 18. Influence of weights: convergence of the VP-NSFnet simulation: (a) VP-NSFnet, fixed weight $\alpha = 1$; (b) VP-NSFnet, fixed weight $\alpha = 100$; (c) VP-NSFnet, dynamic weight with normalization factor $\gamma = 1$; (d) VP-NSFnet, dynamic weight with normalization factor $\gamma = 5$; (e) VP-NSFnet, dynamic weight with normalization factor $\gamma = 10$. All the dynamic weights here are given by Eq. (12).

5. NSFnets can do more than CFD solvers

Having demonstrated some of the attributes and limitations of NSFnets, in this section we provide some highlights of how NSFnets can be used best, especially for flow problems where traditional CFD solvers may fail or are computationally prohibitive. First, we consider problems with missing or noisy boundary conditions, then we consider unknown fluid properties, and finally we give a simple example of transfer learning.

5.1. Kovasznay flow with missing or noisy boundary conditions

We consider the Kovasznay flow as an example, and we use the same computational domain as in section 3. For Kovasznay flow, we solve six cases without the complete boundary conditions using NSFnets, which are obviously ill-posed problems for traditional CFD solvers. The results are summarized in Table 5. The number of residual points is 1,444. The NN size is 4×50. The red lines denote the known boundary conditions (BC), while the black dash lines denote the unknown BC. The weight for known boundary conditions is fixed at $\alpha = 100$. The NSFnets are trained using the same optimization method as in section 3. We expect that for Case 1 with the complete boundary conditions to obtain the highest accuracy. However, NSFnets also obtain good accuracy for Case 2 without the upper and lower boundary conditions, and for Case 3 without the outlet conditions. For Case 4 without upper and outlet conditions and for Case 5 without inlet conditions, the VV-NSFnet outperforms VP-NSFnet and obtains relative L_2 error even smaller than 15%. Clearly, both NSFnets fail in solving Case 5 with high accuracy because the inlet boundary conditions play an important role for this problem. We also solve

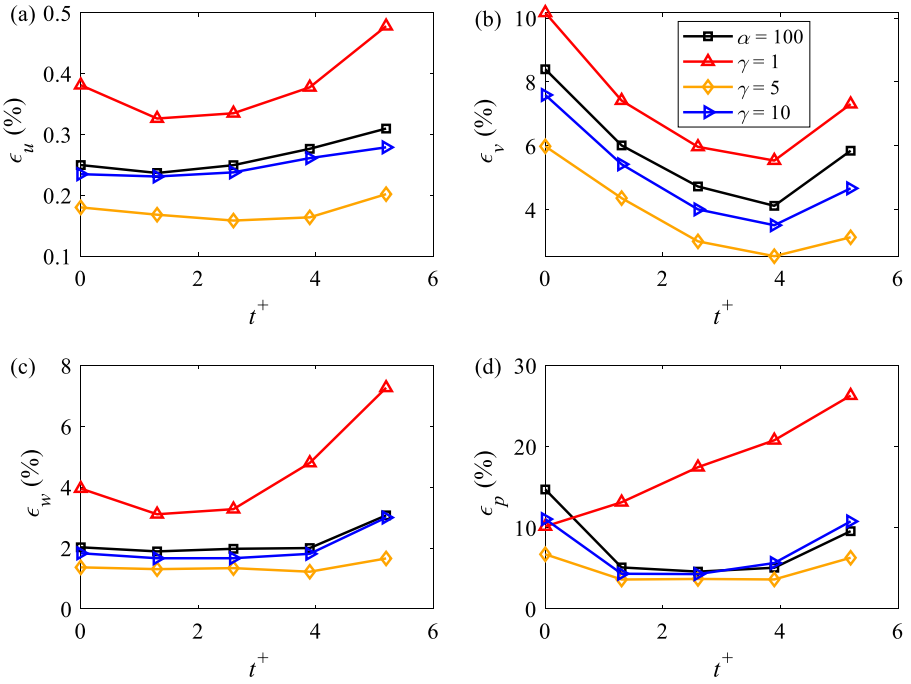

Fig. 19. Influence of weights: accuracy of VP-NSFnet for various weights: (a) to (d) relative L_2 errors of u, v, w and p.

Table 5
Kovasznay flow: relative L_2 errors of velocity and pressure solutions for NSFnets with ill-posed boundary conditions. The number of residual points is 1,444. The NN size is 4×50. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weight is $\alpha = 100$. The red lines denote the known boundary conditions, and black dash lines denote the unknown. The blue line in Case 7 represents the noisy boundary condition, where the noise amplitude here is 10% of the standard deviation of the data.

Case No.	VP-NSFnet			VV-NSFnet	
	ϵ_u(%)	ϵ_v(%)	ϵ_p(%)	ϵ_u(%)	ϵ_v(%)
1	0.0087±0.0031	0.0684±0.0096	0.0255±0.0028	0.0253±0.0040	0.0684±0.0108
2	0.0562±0.0174	0.3413±0.0521	0.1168±0.0251	0.1428±0.0105	0.7022±0.0689
3	0.0568±0.0203	0.5522±0.1609	0.1805±0.0662	0.1354±0.0299	0.7857±0.3315
4	3.1264±1.0624	21.6768±4.8140	2.6143±0.3520	1.4802±0.4273	5.3877±1.0976
5	4.1259±1.6747	22.6913±9.8166	8.4113±5.1687	2.7105±1.2792	14.2649±6.9280
6	0.4368±0.1232	2.8600±1.0120	1.1921±0.2445	0.3141±0.0377	1.1860±0.1919
7	1.6690±0.0588	5.5177±0.5896	2.3138±0.7574	1.3218±0.0703	4.1756±0.2656

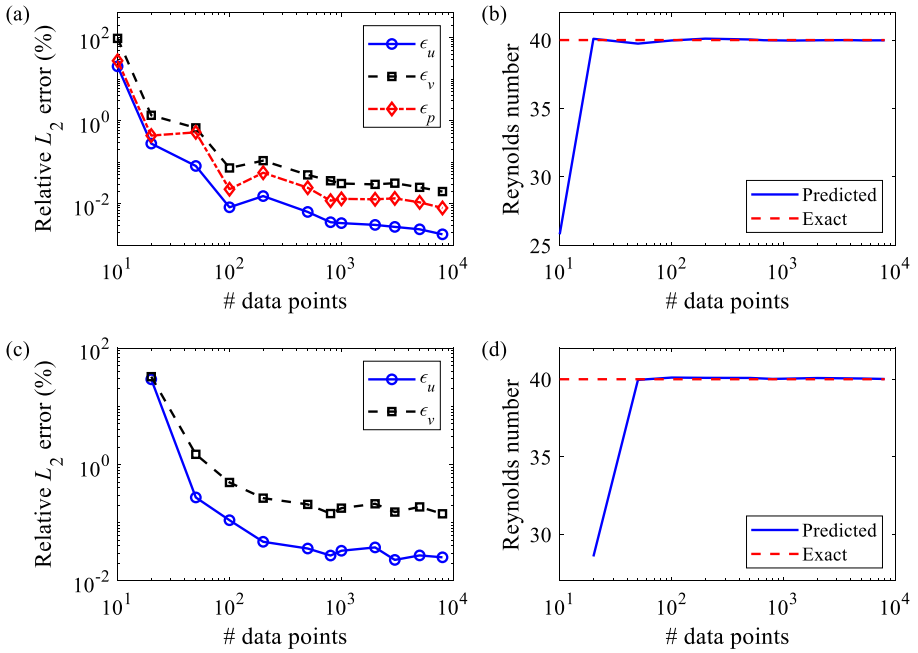

Fig. 20. Error convergence with the number of data points to learn the Reynolds number: (a) convergence of error for VP-NSFnet; (b) convergence of learned Reynolds number for VP-NSFnet; (c) convergence of error for VV-NSFnet; (d) convergence of learned Reynolds number for VV-NSFnet. The result is selected from five independent runs with independent initial network parameters with the highest accuracy. The NN size is 10×100, the number of residual points is 2,000, and the number of boundary points is 4×101.

Case 6 with only the inlet conditions and data on the middle vertical line with NSFnets, which is similar to standard data assimilation problems, and obtain good accuracy. Moreover, we demonstrate here that NSFnets can also deal with *noisy* boundary conditions. In Case 7 in Table 5, random noise is added to the inlet velocity, which is the most important boundary. The noise level is 10%, which denotes the ratio between the noise amplitude and the standard deviation of the original data. Note that the L-BFGS-B optimizer is not applied when dealing with noisy data. The results show that both NSFnets can attain relatively good accuracy for noisy boundary conditions.

5.2. Learning the unknown Reynolds number

We then consider another problem: given only some scattered velocity data, the quantities of interest are the Reynolds number and the entire flow field, which could be computationally very expensive with standard CFD solvers. The NN size is 10×100, the number of residual points is 2,000, and the number of boundary points is 4×101. In addition to the weights and biases of the NSFnets, the Reynolds number is also set as a learnable variable, and the NSFnets are trained using the same optimization method as in section 3. The error convergence with the number of data points to learn the Reynolds number is shown in Fig. 20. With increased data points, both NSFnets obtain the entire flow fields with high accuracy, and successfully learn the unknown Reynolds number.

5.3. Transfer learning of NSFnets

In general, we have to implement the training of NSFnets again when the flow conditions (e.g., Reynolds number, boundary conditions) change. This is similar to the traditional CFD solvers but the computational cost of NSFnets is much higher, e.g., 5-10 times higher. However, the training of NSFnets can be accelerated by using transfer learning. For instance, the parameters of the well-trained NSFnets for a small Reynolds number can be transferred to the NSFnets for a large Reynolds number, thus only a few iterations are required for the new NSFnets to obtain accurate solutions. Take Kovasznay flow as an example, let the NN size be 7×100. The number of residual points is 2,601, and the number of boundary points is 4×101. Assuming that we have already trained the NSFnets for $\mathrm{Re} = 40$ with good accuracy, now we want to solve the flow with $\mathrm{Re} = 60$. First, the network parameters are initialized by the pretrained NSFnets for $\mathrm{Re} = 40$; then, the NSFnet parameters are finetuned according to the boundary conditions and governing equations for $\mathrm{Re} = 60$. In the transfer learning procedure, only L-BFGS-B is used for training since the initialization is close to the solution now.

The relative L_2 errors of velocity and pressure solutions and computational cost for VP-NSFnet with L-BFGS-B training and transfer learning are shown in Table 6. In the L-BFGS-B training case, the network is initialized randomly. When the NSFnet parameters are transferred from $\mathrm{Re} = 40$ to $\mathrm{Re} = 60$, the computational efficiency is obviously improved compared

Table 6
Kovasznay flow (Re = 60): relative L_2 errors of velocity and pressure solutions and computational cost for VP-NSFnet with transfer learning. The number of residual points is 2,601. The NN size is 7 × 100, the number of residual points is 2,601, and the number of boundary points is 4 × 101. The results represent mean ± standard deviation from 10 independent runs with independent initial network parameters. "Finetuning" means the neural network is initialized by the pretrained NSFnets for Re = 40. Otherwise, the network is randomly initialized. The training is performed on a Quadro RTX 6000 GPU.

	ϵ_u (%)	ϵ_v (%)	ϵ_p (%)	Comput. cost (min)
L-BFGS-B	0.0945 ± 0.0724	0.8280 ± 0.5638	0.3779 ± 0.2429	11.4408 ± 5.7397
Finetuning	0.0061	0.0706	0.0232	2.4617

to L-BFGS-B training. The solution accuracy is also improved dramatically. This may be attributed to the similar flow field for Kovasznay flow with different Reynolds numbers, then better NSFnet parameters are initially given, leading to both improved solution speed and accuracy. Similar results can be found when using VV-NSFnet, which are presented in Table A.3 in Appendix A. We also perform transfer learning for higher Reynolds number (Re = 80) and obtain consistent results, which can be seen in Table A.4 and A.5.

6. Summary

In this study, we explored the effectiveness of PINNs to directly simulate incompressible flows, ranging from laminar flows to turbulent channel flow. We have formulated NSFnets based on two different forms of the governing Navier-Stokes equations: the velocity-pressure (VP) form and the velocity-vorticity (VV) form. The spatial and temporal coordinates are the inputs of the PINNs, and the instantaneous velocity and pressure fields are the outputs for the VP-NSFnet; similarly, the instantaneous velocity and vorticity fields are the outputs for the VV-NSFnet. We used automatic differentiation to represent all the differential operators in the Navier-Stokes equations; then the equations can be formulated by the neural networks. We regard the initial and boundary conditions as supervised data-driven parts, and the residual of the Navier-Stokes equations as the unsupervised physics-informed part in the loss function of PINNs. We note that no data was provided for the pressure as boundary or initial conditions for VP-NSFnet, which was a hidden state and was obtained indirectly via the incompressibility constraint without splitting the equations. Convergence of NSFnets was monitored using the total loss function as well as the individual loss functions. We simulated several laminar flows, including two-dimensional steady Kovasznay flow, two-dimensional cylinder wake and three-dimensional Beltrami flow using the two forms NSFnets. For Kovasznay flow, we systematically investigated the convergence of solution error with the number of residual points and boundary points. We also carried out a study on the influence of weights in the various contributions to the loss function. We found that for laminar cases, the VV-NSFnet achieves better accuracy than the VP-NSFnet, and that a dynamic variation of the weights can improve the performance of both NSFnets. We also explored the potential of transfer learning to accelerate the solution speed, and both the computational efficiency and solution accuracy are obviously improved. Then, we solved several ill-posed or inverse problems that cannot be directly solved by CFD solvers.

In addition, we explored the possibility of simulating turbulent channel flow at $\mathrm{Re}_\tau \sim 1{,}000$ using NSFnets. We performed NSFnet simulations by considering different subdomains with different sizes at various locations in the channel and for different time intervals. Established DNS databases provided the proper initial and boundary conditions for the NSFnet simulations. We obtained good agreement between the DNS results and the VP-NSFnet simulation results upon convergence of the loss function. The long time period simulation suggests that NSFnets can sustain turbulence with errors bounded at reasonable levels. We also investigated the use of dynamic versus fixed weights and demonstrated how dynamic weights could further enhance the accuracy of VP-NSFnet. Unlike the VP-form, attempts to train the VV-NSFnet failed to provide satisfactory convergence of the loss function for the governing equations. For $\alpha = 50,000$ we obtained reasonable accuracy, with the loss function for boundary conditions converging to small values but the residual for the governing equations remained very large. This may be related to the fact that the data in this case are obtained from DNS data bases with a VP formulation, and hence the boundary conditions derived may not be so accurate but rather inconsistent with the VV form of the governing equations. We plan to revisit this issue in future work.

The current study for modeling turbulence using NSFnets is the first attempt to evaluate the PINN performance, and while the first results are encouraging, the broader question is if PINNs can provide sufficient accuracy to sustain turbulence with non-stochastic boundary conditions but rather the simple zero Dirichlet and periodic boundary conditions employed for the entire domain in spectral type simulations of turbulence. To address this question expeditiously, the efficiency of PINNs has to be improved significantly, including the development of a multi-node GPU code that can accelerate significantly the training process. Such speed up can be further enhanced by using adaptive activation functions as demonstrated in [35,36]. Moreover, a further study is needed to derive the proper normalization procedure for the three different velocity components so that we can obtain uniform accuracy, as in the current study the streamwise component is inferred with an order of magnitude higher accuracy compared to the crossflow velocity components when the streamwise velocity is of an order higher than the crossflow velocity.

CRediT authorship contribution statement

Xiaowei Jin: Conceptualization, Methodology, Investigation, Coding, Validation, Writing - original draft preparation, Writing - reviewing and editing. **Shengze Cai:** Conceptualization, Methodology, Investigation, Coding, Validation, Writing - original draft preparation, Writing - reviewing and editing. **Hui Li:** Conceptualization, Methodology, Supervision, Writing - original draft preparation, Writing - reviewing and editing, Funding acquisition. **George Em Karniadakis:** Conceptualization, Methodology, Investigation, Supervision, Writing - original draft preparation, Writing - reviewing and editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

X. Jin and H. Li would like to acknowledge funding from the National Natural Science Foundation of China (NSFC) (Grant No. U1711265). S. Cai and G.E. Karniadakis would like to acknowledge funding from the DOE grant DE-AC05-76RL01830.

Appendix A. Systematic parametric study for Kovasznay flow

As shown in Fig. A.1, first we consider the influence of the number of initial *Adam* iterations before L-BFGS-B on the final accuracy with different learning rates. The shaded regions and error bars represent one standard-deviation from ten independent runs with independent initial network parameters. Higher accuracy can be obtained using 3×10^4 *Adam* iterations with learning rate 10^{-3}. Therefore, this optimization strategy is applied to Kovasznay flow in the paper.

Next, we investigate the error convergence with the number of residual points and boundary points for different NN sizes to simulate the Kovasznay flow. We employ different sizes of NSFnets by varying the number of hidden layers and the number of neurons per layer. We aim to obtain more accurate solutions by adding more points in the computational domain for each NN size. The fixed weighting coefficient $\alpha = 100$ for boundary conditions is chosen for training these NSFnets. The error convergence with the number of residual points is shown in Fig. A.2. The shaded regions represent one standard-deviation from five independent runs with independent initial network parameters. Here, the number of boundary points is fixed at 4×101. We give the empirical convergence rate for each NN size before convergence in Fig. A.2. When increasing the number of residual points, both formulations of NSFnets are able to reach converged solutions, and both NSFnets can attain solutions with high accuracy when the number of residual points is greater than 10^3. The converged relative errors are in the order of 10^{-5} to 10^{-3}. We can also observe that the NSFnets with small NN size converge faster than those with large NN size. The VP-NSFnet even exhibits exponential convergence rate when the NN size is 4×50 and 7×50. Thus, when the number of residual points is between 100 and 600, the solutions of VP-NSFnets with NN size 4×50 and 7×50 are more accurate than those with size 10×100. For the converged solutions, the performance of NSFnets is improved as the network size increases.

Similarly, we investigate the error convergence with respect to the number of boundary points in Fig. A.3. Here, the number of residual points is fixed at 1,444. Both formulations of NSFnets are able to reach converged solutions when

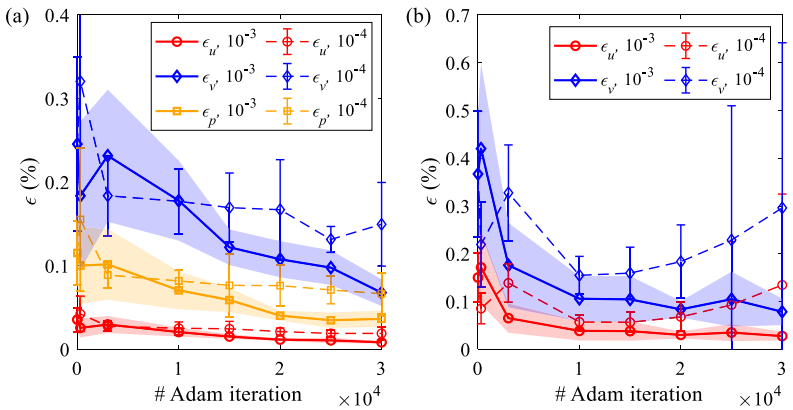

Fig. A.1. Kovasznay flow: influence of the number of initial *Adam* iterations before L-BFGS-B on the final accuracy with different learning rates (10^{-3} and 10^{-4}, see the legend) for (a) VP-NSFnet and (b) VV-NSFnet. The shaded regions and error bars represent one standard-deviation from ten independent runs with independent initial network parameters. The NN size is 4×50, the number of residual points is 2,601, and the number of boundary points is 4×101. (NN size is the number of hidden layers $\times$ the number of neurons per layer.)

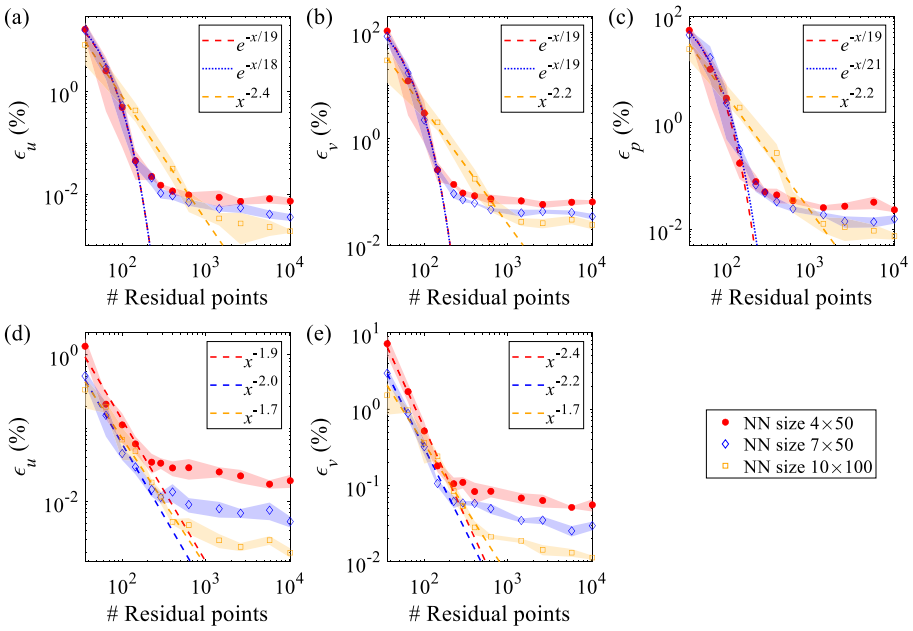

Fig. A.2. Kovasznay flow: error convergence with the number of residual points for different NN sizes: (a), (b) and (c) show convergence of errors for VP-NSFnet; (d) and (e) show convergence of errors for VV-NSFnet. The shaded regions represent one standard-deviation from five independent runs with independent initial network parameters. The number of boundary points is 4 × 101. Three network architectures are demonstrated.

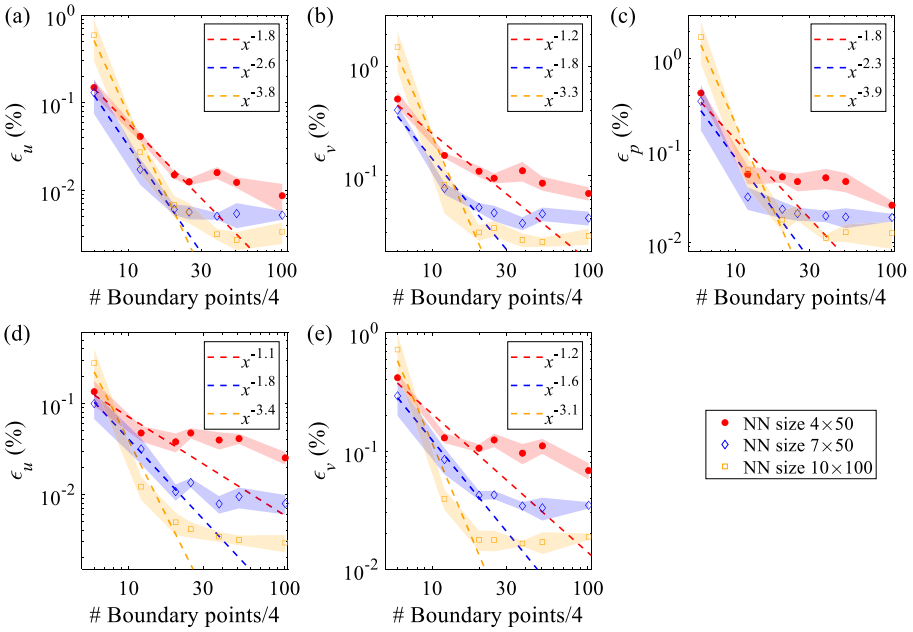

Fig. A.3. Kovasznay flow: error convergence with the number of boundary points for different NN sizes: (a), (b) and (c) are convergence of errors for VP-NSFnet; (d) and (e) are convergence of errors for VV-NSFnet. The shaded regions represent one standard-deviation from five independent runs with independent initial network parameters. The number of residual points is 1,444. Three network architectures are demonstrated.

increasing the number of boundary points. However, the NSFnets with large NN size converge faster than those with small NN size, which is different from the trend in the error convergence with the number of residual points (see Fig. A.2).

We also give the loss functions (physics loss L_e and boundary loss L_b) obtained by NSFnets with different number of residual points in Fig. A.4. We can see that the convergence of different parts of the loss function does not vary noticeably when the number of residual points is changed.

The plots of the error convergence with the number of residual points for the NSFnet simulations with different weights are given in Fig. A.5; the empirical convergence rates are also given in Fig. A.5. For fixed weights ($\alpha = 1$ and $\alpha = 100$), the

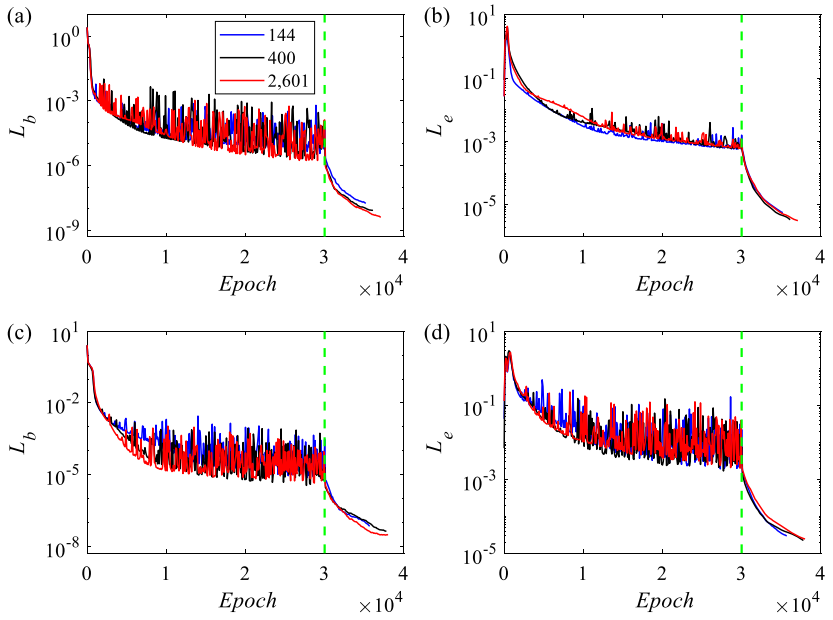

Fig. A.4. Kovasznay flow: loss functions (physics loss L_e and boundary loss L_b) for (a) L_b, VP-NSFnet; (b) L_e, VP-NSFnet; (c) L_b, VV-NSFnet; (d) L_e, VV-NSFnet. The *Adam* optimizer is used before the vertical dashed green line with fixed weight $\alpha = 100$, and the L-BFGS-B optimizer is used after that. The NN size is 4×50, the number of residual points is 144, 400, 2,601, and the number of boundary points is 4×101. To clearly show the evolution of the loss functions, only one from five independent runs is selected.

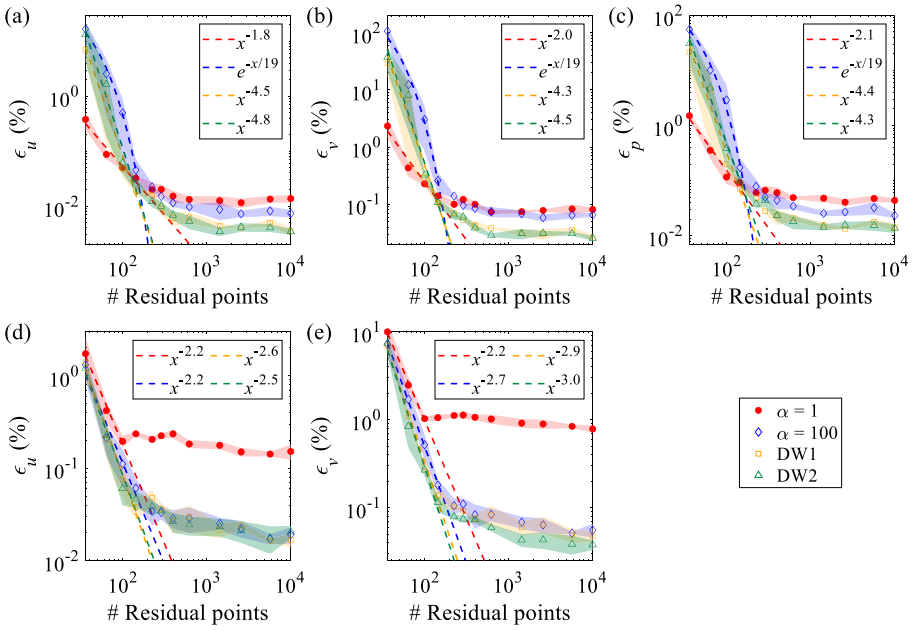

Fig. A.5. Kovasznay flow: error convergence with the number of residual points for different weights: (a), (b) and (c) show convergence of error for VP-NSFnet; (d) and (e) show convergence of error for VV-NSFnet. The shaded regions represent one standard-deviation from five independent runs with independent initial network parameters. "DW1" denotes dynamic weights given by Eq. (6), and "DW2" denotes dynamic weights given by Eq. (7). The NN size is 4×50, and the number of boundary points is 4×101.

VP-NSFnet outperforms the VV-NSFnet for simulating Kovasznay flow. The NSFnets with $\alpha = 100$ perform better than those with $\alpha = 1$ for both VP and VV formulations, especially the VV-NSFnet. When applying dynamic weights during network training, we can obtain more accurate solutions than the former two fixed-weight cases. The convergence rates for both dynamic weights are almost the same, and faster than those for fixed $\alpha = 1$.

Table A.1
Kovasznay flow: relative L_2 errors of velocity and pressure solutions for NSFnets with different sizes. The number of residual points is 144. The number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weight is $\alpha = 100$.

NN size	VP-NSFnet			VV-NSFnet	
	ϵ_u(%)	ϵ_v(%)	ϵ_p(%)	ϵ_u(%)	ϵ_v(%)
4×50	0.0452±0.0264	0.2610±0.1047	0.1739±0.0860	0.0609±0.0159	0.1804±0.0307
7×50	0.0430±0.0099	0.2614±0.0418	0.3121±0.0916	0.0296±0.0061	0.1062±0.0100
7×100	0.1845±0.0988	0.9972±0.5146	1.0449±0.5125	0.0272±0.0066	0.1443±0.0222
10×100	0.4335±0.1731	2.0383±0.6012	1.9404±0.7305	0.0484±0.0127	0.2410±0.0279
10×250	5.8514±3.8789	28.1177±19.8851	39.5849±40.5420	0.0315±0.0080	0.1312±0.0147

Table A.2
Kovasznay flow: relative L_2 errors of velocity and pressure solutions for NSFnets with different sizes. The number of residual points is 400. The number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from five independent runs with independent initial network parameters. The weight is $\alpha = 100$.

NN size	VP-NSFnet			VV-NSFnet	
	ϵ_u(%)	ϵ_v(%)	ϵ_p(%)	ϵ_u(%)	ϵ_v(%)
4×50	0.0115±0.0020	0.0851±0.0105	0.0441±0.0137	0.0286±0.0065	0.0832±0.0254
7×50	0.0093±0.0015	0.0622±0.0021	0.0330±0.0059	0.0135±0.0037	0.0582±0.0044
7×100	0.0129±0.0040	0.0685±0.0077	0.0437±0.0106	0.0078±0.0026	0.0347±0.0036
10×100	0.0316±0.0141	0.1784±0.0900	0.2723±0.1245	0.0053±0.0005	0.0282±0.0058
10×250	0.1355±0.0912	0.7287±0.4998	0.5517±0.4510	0.0105±0.0015	0.0354±0.0056

Table A.3
Kovasznay flow (Re = 60): relative L_2 errors of velocity and pressure solutions and computational cost for VV-NSFnet with transfer learning. The number of residual points is 2,601. The NN size is 7×100, the number of residual points is 2,601, and the number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from 10 independent runs with independent initial network parameters. "Finetuning" means the neural network is initialized by the pretrained NSFnets for Re = 40. Otherwise, the network is randomly initialized. The training is performed on a QUADRO RTX 6000 GPU.

	ϵ_u (%)	ϵ_v (%)	Comput. cost (min)
L-BFGS-B	0.1062 $\pm$ 0.0304	0.6729 $\pm$ 0.2018	17.9132 $\pm$ 8.6283
Finetuning	0.0099	0.0764	2.3782

Table A.4
Kovasznay flow (Re = 80): relative L_2 errors of velocity and pressure solutions and computational cost for VP-NSFnet with transfer learning. The number of residual points is 2,601. The NN size is 7×100, the number of residual points is 2,601, and the number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from 10 independent runs with independent initial network parameters. "Finetuning" means the neural network is initialized by the pretrained NSFnets for Re = 40. Otherwise, the network is randomly initialized. The training is performed on a Quadro RTX 6000 GPU.

	ϵ_u (%)	ϵ_v (%)	ϵ_p (%)	Comput. cost (min)
L-BFGS-B	0.1214±0.0740	1.1305±0.8148	0.4334±0.2310	9.4732±5.0144
Finetuning	0.0075	0.0891	0.0292	2.6910

Table A.5
Kovasznay flow (Re = 80): relative L_2 errors of velocity and pressure solutions and computational cost for VV-NSFnet with transfer learning. The number of residual points is 2,601. The NN size is 7×100, the number of residual points is 2,601, and the number of boundary points is 4×101. The results represent mean $\pm$ standard deviation from 10 independent runs with independent initial network parameters. "Finetuning" means the neural network is initialized by the pretrained NSFnets for Re = 40. Otherwise, the network is randomly initialized. The training is performed on a QUADRO RTX 6000 GPU.

	ϵ_u (%)	ϵ_v (%)	Comput. cost (min)
L-BFGS-B	0.0897±0.0235	0.7396±0.2006	16.2803±6.8725
Finetuning	0.0077	0.0892	3.1808

References

[1] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[2] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data, Phys. Rev. Fluids 2 (2017) 034603.
[3] C. Jiang, J. Mi, S. Laima, H. Li, A novel algebraic stress model with machine-learning-assisted parameterization, Energies 13 (2020) 258.
[4] Z. Zhou, G. He, S. Wang, G. Jin, Subgrid-scale model for large-eddy simulation of isotropic turbulent flows using an artificial neural network, Comput. Fluids 195 (2019) 104319.
[5] X. Jin, P. Cheng, W.-L. Chen, H. Li, Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder, Phys. Fluids 30 (2018) 047105.
[6] P. Wu, J. Sun, X. Chang, W. Zhang, R. Arcucci, Y. Guo, C.C. Pain, Data-driven reduced order model with temporal convolutional neural network, Comput. Methods Appl. Mech. Eng. 360 (2020) 112766.
[7] X. Jin, S. Laima, W.-L. Chen, H. Li, Time-resolved reconstruction of flow field around a circular cylinder by recurrent neural networks based on non-time-resolved particle image velocimetry measurements, Exp. Fluids 61 (2020) 114.
[8] Z. Hosseini, R.J. Martinuzzi, B.R. Noack, Sensor-based estimation of the velocity in the wake of a low-aspect-ratio pyramid, Exp. Fluids 56 (2015) 13.
[9] S. Discetti, M. Raiola, A. Ianiro, Estimation of time-resolved turbulent fields through correlation of non-time-resolved field measurements and time-resolved point measurements, Exp. Therm. Fluid Sci. 93 (2018) 119–130.
[10] S. Cai, S. Zhou, C. Xu, Q. Gao, Dense motion estimation of particle images via a convolutional neural network, Exp. Fluids 60 (2019) 73.
[11] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annu. Rev. Fluid Mech. 51 (2019) 357–377.
[12] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[13] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561, 2017.
[14] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561, 2017.
[15] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[16] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137.
[17] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030.
[18] J. Kim, P. Moin, R. Moser, Turbulence statistics in fully developed channel flow at low Reynolds number, J. Fluid Mech. 177 (1987) 133–166.
[19] G.E. Karniadakis, S. Sherwin, Spectral/hp Element Methods for Computational Fluid Dynamics, Oxford University Press, 2013.
[20] E. Perlman, R. Burns, Y. Li, C. Meneveau, Data exploration of turbulence simulations using a database cluster, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, ACM, 2007, p. 23.
[21] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, G. Eyink, A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence, J. Turbul. 9 (2008) N31.
[22] J. Graham, K. Kanov, X. Yang, M. Lee, N. Malaya, C. Lalescu, R. Burns, G. Eyink, A. Szalay, R. Moser, et al., A web services accessible database of turbulent channel flow and its use for testing a new integral wall model for les, J. Turbul. 17 (2016) 181–215.
[23] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018).
[24] G.E. Karniadakis, M. Israeli, S.A. Orszag, High-order splitting methods for the incompressible Navier-Stokes equations, J. Comput. Phys. 97 (1991) 414–443.
[25] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arXiv preprint, arXiv:2001.04536, 2020.
[26] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[27] M. Darwish, I. Sraj, F. Moukalled, A coupled finite volume solver for the solution of incompressible flows on unstructured grids, J. Comput. Phys. 228 (2009) 180–201.
[28] Z. Chen, A. Przekwas, A coupled pressure-based computational method for incompressible/compressible flows, J. Comput. Phys. 229 (2010) 9150–9165.
[29] C.-N. Xiao, F. Denner, B.G. van Wachem, Fully-coupled pressure-based finite-volume framework for the simulation of fluid flows at all speeds in complex geometries, J. Comput. Phys. 346 (2017) 91–130.
[30] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[31] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[32] J. Trujillo, G.E. Karniadakis, A penalty method for the vorticity-velocity formulation, J. Comput. Phys. 149 (1999) 32–58.
[33] H.L. Meitz, H.F. Fasel, A compact-difference scheme for the Navier–Stokes equations in vorticity-velocity formulation, J. Comput. Phys. 157 (2000) 371–403.
[34] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, arXiv preprint, arXiv:1907.04502, 2019.
[35] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[36] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A 476 (2020) 20200334.
[37] C.R. Ethier, D. Steinman, Exact fully 3D Navier-Stokes solutions for benchmarking, Int. J. Numer. Methods Fluids 19 (1994) 369–375.
[38] S.B. Pope, Turbulent Flows, Cambridge University Press, Cambridge, 2000, p. 276.