Neural Processing Letters (2024) 56:233
https://doi.org/10.1007/s11063-024-11620-1

A Neural Network-Based Poisson Solver for Fluid Simulation

Zichao Jiang[1] · Zhuolin Wang[1] · Qinghe Yao[1] · Gengchao Yang[1] · Yi Zhang[2] · Junyang Jiang[1]

Accepted: 6 April 2024

Abstract
The pressure Poisson equation is usually the most time-consuming problem in fluid simulation. To accelerate its solving process, we propose a deep neural network-based numerical method, termed Deep Residual Iteration Method (DRIM), in this paper. Firstly, the global equation is decomposed into multiple independent tridiagonal sub-equations, and DRIM is capable of solving all the sub-equations simultaneously. Moreover, we employed Residual Network and a correction iteration method to improve the precision of the solution achieved by the neural network in DRIM. The numerical results, including the Poiseuille flow, the backwards-facing step flow, and driven cavity flow, have proven that the numerical precision of DRIM is comparable to that of classic solvers. In these numerical cases, the DRIM-based algorithm is about 2–10 times faster than the conventional method, which indicates that DRIM has promising applications in large-scale problems.

Keywords Pressure Poisson equation · Deep neural network · Deep residual iteration method · Fluid simulation

1 Introduction

In a simulation of an incompressible fluid, the velocity and pressure are coupled and cause an inevitable implicit process. In well-known algorithms, e.g. SIMPLE, PISO [1], the coupled pair of velocity and pressure are solved via the "prediction-correction process": first, the velocity is predicted based on the non-pressure terms or initial pressure; secondly, the pressure is achieved via solving the Poisson equation and then the velocity is corrected by the pressure gradient. In this routine, the pressure Poisson equation (PPE) is usually the most time-consuming and challenging problem of some numerical methods for the Naiver Stokes equations [2, 3].

In recent years, neural networks (NN), especially deep neural networks (DNN), have been widely employed in natural language processing [4], computer vision [5], and other

✉ Qinghe Yao
yaoqh@sysu.edu.cn

1 School of Aeronautics and Astronautics, Sun Yat-Sen University, Guangzhou 510275, China

2 School of Mathematics and Computing Science, Guilin University of Electronic Technology, Guilin 541004, China

Published online: 10 September 2024

fields with the growth of computing power [6]. Specifically, NNs are readily trainable universal function approximators [7] that satisfy high-dimensional and complex problems. In the fields of scientific computation, the universal approximation properties of NNs provide an alternative approach for solving partial differential equations (PDEs) [8–15].

Currently, most of the NN-based solvers focus on formulating PDEs as optimization problems, and their critical component is to design a suitable loss function [8]. For instance, the deep Galerkin method (DGM) [13], the deep Ritz method (DRM) [14], and physics-informed neural network (PINN) [15] have unique loss functions based on the least-squares, the Ritz method, the collocation methods, respectively. These methods can achieve a decent accuracy; for example, an improved version of PINN, hp-VPINN [8], can reach an error (L^2-norm) of 10^{-4}–10^{-5}. Moreover, all these methods are valid for high-dimensional equations, avoiding the curse of dimensionality. In recent years, a number of other updated versions of PINN have also been published and successfully utilized in a variety of fields [16–18]. Although these methods have the remarkable superiority mentioned above, they all require repeated training for different equations, which is inappropriate for solving the PPE.

This manuscript presents an NN-based method to solve the PPE without repetitive training to accelerate the SIMPLE algorithm. Although the training of the network can be computationally intensive and time-consuming, it is done offline [19] and only once for a given category of problems. Specifically, after determining the equation size and non-zero element distribution, we use random equations for training. For equations with the same structure, additional repetitive training is unnecessary. This characteristic is particularly effective for simulations involving time iterations, as the equations to be solved often do not change in size over time. Hence, the trained model can be directly applied to solve all types of equations of the same size. The basic idea of our method can be divided into three steps: (i) transforming the discrete Poisson equation into a series of multi-diagonal equations, as frequently employed by multiple large-scale numerical solvers [3, 20–22]; (ii) transforming the multi-diagonal equations into a set of vectors as the input of our DNN model to predict the solution; (iii) improving the numerical accuracy of the achieved solution.

Specifically, we employed the discrete sine transformation (DST) [23] to convert the 2nd-order central difference scheme of the Poisson equation into a series of independent tridiagonal equations. An arbitrary tridiagonal equation can be represented wholly and naturally by four vectors, and thus, can be directly served as the input of the NN model. Actually, by adjusting the architecture of the input layers, the NN model can solve all the equations with a similar structure, which includes all multi-diagonal matrices. This implies that the NN-based method has potential advantages for higher-order schemes and complex geometries.

In recent years, several published works follow the above path; for instance, a solver based on convolutional neural networks (CNN) was proposed by Xiao et al. [2]. This CNN-based solver reduces the dimension of the input through principal component analysis (PCA), enabling the two-dimensional Poisson equation to be passed directly into the CNN model. Despite its high numerical error (10^{-2}–10^{-1}), it still demonstrates that the NN-based Poisson solver is a promising path.

For high-efficiency transient simulations of incompressible flow, we mainly focused on the accuracy of the pressure. In this research, we constructed a DNN model that solves the discrete Poisson equation and employed a series of techniques to enhance its accuracy. This method we proposed is called deep residual iteration method (abbreviated as DRIM) and the basic components of DRIM can be divided into two parts:

(1) Residual connection [24], a widely used network structure that resolves gradient vanishing and network degradation;
(2) Correction iterations, an algorithm that reduces the error iteratively by feeding the residual of each predicted solution back into the NN model.

This paper is organized as follows. The main methods, including a DST-based SIMPLE algorithm and DRIM, are introduced in Sect. 2. Section 3 demonstrates the high efficiency and accuracy reliability of the proposed method in this paper by applying it in a real fluid simulation. Specifically, we achieve a long-term transient simulation of the flow field in the opposing cavity based on the proposed method in this paper.

2 Methodology

In this section, we introduce our proposed DRIM-based fluid simulation method. In the algorithm, we employ the SIMPLE algorithm as a framework and transform the global PPE into a series of independent sub-equations by DST. Since each sub-equation is a tridiagonal linear system of equations, we directly utilize DRIM to solve all the sub-equations simultaneously and assemble the solutions into the global pressure field. In addition, in DRIM, to avoid network degradation and to improve the precision of the DNN model, we adopt the Residual Network (Res-Net) architecture and the correction iteration process.

2.1 The DST-Based Transformation of the Poisson Equation

Incompressible Naiver-Stokes equation is the governing equation in the algorithm:

$$\begin{cases} \nabla \cdot \boldsymbol{u} = 0, \\ \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)u = -\nabla p + \nu \nabla^2 \boldsymbol{u}, \end{cases} \tag{1}$$

where $\boldsymbol{u}$ is the flow velocity, p is the fluid pressure, and ν is the kinematic viscosity. In (1), the density of the fluid is assumed to be 1 and the mass force is assumed to be 0.

To maintain the incompressibility in (1), i.e., the divergence of the velocity is identically equal to zero, the calculation of the velocity in the SIMPLE algorithm is decomposed into the following three steps:

(a) Predict the velocity: the intermediate velocity

$\boldsymbol{u^*}$ is calculated by

$$\boldsymbol{u}^* = \boldsymbol{u}^n - \Delta t \nabla \boldsymbol{u}^n \cdot \boldsymbol{u}^n + \nu \Delta t \nabla^2 \boldsymbol{u}^n, \tag{2}$$

where Δt is the time step, and the superscript n indicates the current time step.

(b) Calculate the pressure: the pressure is calculated by PPE, as

$$\nabla^2 p^{n+1} = -\tfrac{1}{\Delta t} \nabla \cdot \boldsymbol{u}^*. \tag{3}$$

(c) Correct the velocity: the velocity is corrected by the pressure term, as

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t \nabla p^{n+1}. \tag{4}$$

In this work, we focus on the fast solving algorithm of the PPE shown in (3). As an elliptic equation, the PPE has to be solved simultaneously over the entire field and have $(N_x \times N_y)$ degrees of freedom (where N_x, N_y are the number of nodes in each dimension) for a 2-dimensional problem. Therefore, it is impractical to input all the node information into a DNN model, and we employed a DST-based decoupling method as shown in Algorithm 1.

Algorithm 1 A DST-based SIMPLE method

```
Set the initial condition
Calculate computational parameters
Calculate transformed matrix  Â_i
for  n = 1, 2, …, MaxTimeStep
    Calculate the predicted velocity  u*
    Calculate the right-hand side vector  f_i = −∇ · u*/Δt
    Calculate the transformed vector  f̂_i
    Solve the sub-equations  Â_i p̂_i = f̂_i
    Calculate the pressure  p_i
    Calculate the corrected velocity  u^{n+1}
end
Output results
```

It is worth mentioning that the proposed method in this paper is based on the finite difference method (FDM) for discretization. In the algorithm shown in Algorithm 1, the PPE is discretized via the five-point difference scheme with 2nd-order precision, that

$$\frac{1}{\Delta x^2}\left(p_{i-1,j} - 2p_{i,j} + p_{i+1,j}\right) + \frac{1}{\Delta y^2}\left(p_{i,j-1} - 2p_{i,j} + p_{i,j+1}\right) = f_{i,j}, \tag{5}$$

where Δx and Δy are the spatial steps in x and y directions, respectively. The source term $f_{i,j}$ is the right-hand side of (5) and the linear equation of the non-boundary nodes can be denoted by $A\boldsymbol{p} = \boldsymbol{f}$, where $\boldsymbol{f}$ is the source term, A is a block-tridiagonal matrix,

$$A = \begin{bmatrix} T & I & & \\ I & T & \ddots & \\ & \ddots & \ddots & I \\ & & I & T \end{bmatrix}. \tag{6}$$

It is necessary to note that the subscripts of the elements in the vector $\boldsymbol{p}$ are in the sequence of $(p_{1,1}, p_{1,2}, \cdots, p_{1,m}, p_{2,1}, \cdots, p_{2,m}, \cdots, p_{m,m})$, the same for $\boldsymbol{f}$. In (6), T is a tridiagonal matrix, while I is an identity matrix. The specific form of the matrix T depends on the boundary conditions, for instance, the matrix T corresponding to the Dirichlet boundary can be calculated by

$$T = \begin{bmatrix} -2(c+1) & c & & \\ c & -2(c+1) & \ddots & \\ & \ddots & \ddots & c \\ & & c & -2(c+1) \end{bmatrix}, \tag{7}$$

where the constant $c = \Delta y^2/\Delta x^2$. The matrix T in (7) has the eigen decomposition in the form of

$$QTQ^{\mathrm{T}} = \Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_m), \tag{8}$$

where the matrix Q consisting of the eigenvectors of the matrix, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$, denotes the diagonal matrix filled by the eigenvalues. It has been proven that the matrix Q consisting of the eigenvectors of the matrix T can be represented by the coefficient matrix of DST, that

$$Q = \left[Q_{ij}\right] = \sqrt{2/(m+1)} \sin \tfrac{(i+1)(j+1)\pi}{m+1} (i, j = 1, 2, \ldots, m). \tag{9}$$

Therefore, after we define the block-diagonal matrix consisting of matrix Q as the global transformation matrix $Q_A = \mathrm{diag}(Q, Q, \ldots, Q)$, we can transform the original equation into a tridiagonal equation shown by

$$\hat{A}\hat{\boldsymbol{p}} = \hat{\boldsymbol{s}}, \tag{10}$$

where $\widehat{\boldsymbol{p}} = Q_A \boldsymbol{p}, \widehat{\boldsymbol{s}} = Q_A \boldsymbol{s}$, and the block-tridiagonal matrix $\widehat{A}$ is

$$\hat{A} = Q_A A Q_A^{\mathrm{T}} = \begin{bmatrix} \Lambda & I & & \\ I & \Lambda & \ddots & \\ & \ddots & \ddots & I \\ & & I & \Lambda \end{bmatrix}. \tag{11}$$

In (10), if the sequence of the elements in $\widehat{\boldsymbol{p}}$ and $\widehat{\boldsymbol{s}}$ transform into $\widehat{\boldsymbol{p}}' = \left(\widehat{p}_{1,1}, \ldots, \widehat{p}_{m,1}, \widehat{p}_{1,2}, \ldots, \widehat{p}_{m,2}, \ldots, \widehat{p}_{m,m}\right)$ and $\widehat{\boldsymbol{s}}' = \left(\widehat{s}_{1,1}, \ldots, \widehat{s}_{m,1}, \widehat{s}_{1,2}, \ldots, \widehat{s}_{m,2}, \ldots, \widehat{s}_{m,m}\right)$, the left-hand matrix form as

$$\hat{A}\prime = diag\left(\hat{A}_1\prime, \hat{A}_2\prime, \ldots, \hat{A}_m\prime\right), \tag{12}$$

where the sub-matrixes are

$$\hat{A}_j\prime = \begin{bmatrix} \lambda_j & 1 & & \\ 1 & \lambda_j & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & \lambda_j \end{bmatrix} (1 \le j \le m). \tag{13}$$

So far, we have transformed the global difference equation into a series of independent tridiagonal sub-equations in the form of $\widehat{A}_j\prime\widehat{\boldsymbol{p}}'_j = \widehat{\boldsymbol{s}}'_j$, where $\widehat{\boldsymbol{p}}'_j$ and $\widehat{\boldsymbol{s}}'_j$ are $\left(\widehat{p}_{1,j}, \ldots, \widehat{p}_{m,j}\right)$ and $\left(\widehat{s}_{1,j}, \ldots, \widehat{s}_{m,j}\right)$, respectively. By DST, $\widehat{\boldsymbol{p}}$ can be transformed into the pressure in the original space, which has the discrete form shown as

$$p = Q_A^{\mathrm{T}} \hat{\boldsymbol{p}}. \tag{14}$$

The DST-based method for solving the Poisson equation introduced in this section decouples a large global equation into multiple independent sub-equations, which decomposes the large-scale problem into multiple smaller problems to facilitate parallel computation. Furthermore, DST can be accelerated by a fast Fourier transformation (FFT) algorithm bringing only minor computational work.

2.2 The DNN Model

To solve the linear equations with the left-hand side matrix as shown in (13), first, we consider general matrices of the following form,

$$A = C((r_1, r_2, \ldots, r_{n-1}), (l_1, l_2, \ldots, l_{n-1})) = \begin{bmatrix} 1 & r_1 & & \\ l_1 & 1 & r_2 & \\ & l_2 & \ddots & r_{(n-1)} \\ & & l_{(n-1)} & 1 \end{bmatrix}, \tag{15}$$

where n is the size of the matrix. The mapping $\mathcal{C} : \mathbb{R}^{n-1} \times \mathbb{R}^{n-1} \to \mathbb{R}^{n\times n}$ in (15) represents the construction of a tridiagonal matrix with diagonal elements equal to 1 matrix using off-diagonal elements.

The set of vectors $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{g})$ composed by the off-diagonal elements of A, where $\boldsymbol{l} = \left(l_1, l_2, \ldots, l_{(n-1)}\right)$, $\boldsymbol{r} = \left(r_1, r_2, \ldots, r_{(n-1)}\right)$, and the right-hand vector $\boldsymbol{g}$, is a complete representation of an arbitrary tridiagonal equation $A\boldsymbol{x} = \boldsymbol{g}$. Therefore, we select these vectors as the input to the DNN model.

The basic idea of DRIM is to treat the DNN model as a high-dimensional approximation of the mapping $\boldsymbol{y} = \mathcal{S}(A, \boldsymbol{g}) = A^{-1}\boldsymbol{g}$. Specifically, the DNN model $\mathcal{S}_{\mathrm{NN}}$ in DRIM is comprised of N_l hidden layers with $N_i (i \le N_l)$ neurons in each layer that takes the following form

$$\begin{cases} R(\boldsymbol{r}) = \mathcal{T}_{1,N_{\mathcal{R}}} \circ \cdots \circ \mathcal{T}_{1,2} \circ \mathcal{T}_{1,1}(\boldsymbol{r}), \\ L(\boldsymbol{l}) = \mathcal{T}_{2,N_{\mathcal{L}}} \circ \cdots \circ \mathcal{T}_{2,2} \circ \mathcal{T}_{2,1}(\boldsymbol{l}), \\ G(\boldsymbol{g}) = \mathcal{T}_{3,N_{\mathcal{G}}} \circ \cdots \circ \mathcal{T}_{3,2} \circ \mathcal{T}_{3,1}(\boldsymbol{g}), \\ \mathcal{S}_{\mathrm{NN}}(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{g}; W, \boldsymbol{b}) = M \circ \mathcal{T}_{4,N_{\mathcal{S}}} \circ \cdots \circ \mathcal{T}_{4,2} \circ \mathcal{T}_{4,1}([\mathcal{R}(\boldsymbol{r}), \mathcal{L}(\boldsymbol{l}), \mathcal{G}(\boldsymbol{g})]; W, \boldsymbol{b}). \end{cases} \tag{16}$$

As shown in (16), the hidden layers can be divided into four partitions $N_l = N_{\mathcal{L}} + N_{\mathcal{R}} + N_{\mathcal{G}} + N_{\mathcal{S}}$. In each hidden layer, the nonlinear mapping $\mathcal{T}_i(\boldsymbol{x}; W, \boldsymbol{b}) : \mathbb{R}^{M_{N_{i-1}}} \to \mathbb{R}^{M_{N_i}}$ has the form as

$$\mathcal{T}_i(\boldsymbol{x}; W, \boldsymbol{b}) = ReLU(W\boldsymbol{x} + \boldsymbol{b}) + x, \tag{17}$$

where W is the weight matrix and $\boldsymbol{b}$ is the bias vector. Considering that the activation function is one of the most frequently implemented computations, we chose the ReLU function $\mathrm{ReLU}(x) = (x + |x|)/2$ because it is one of the least computationally complex activation functions and is well adapted to regression problems, making it more suitable for large-scale numerical computations.

In the output layer, the linear mapping $\mathcal{M} : \mathbb{R}^{M_{N_l}\times n} \to \mathbb{R}^n$ is define as

$$\mathcal{M}(\boldsymbol{x}, \boldsymbol{g}; W_{\mathcal{M}}) = W_{\mathcal{M}}x + g. \tag{18}$$

Compared with the conventional neural network model, this hidden layer combines a residual connection that makes the network avoid the vanishing gradient and the network degradation [24]. The mapping $\mathcal{M}$ is a short connection different from the residual connection, which can be construed as a transformation of the approximated vector into $\left(I - A^{-1}\right)\boldsymbol{g}$. This makes the output vector have zero mean, which is more beneficial for the training of networks with the ReLU activation function.

As mentioned in the introduction, we constructed the DNN model shown in (15) to derive the predicted solution by network inference. Therefore, it is necessary to pre-solve a batch of tridiagonal equations as the training set of the DNN model, which takes the form of

$$P_n = \left\{ \{ \boldsymbol{r}_i, \boldsymbol{l}_i, \boldsymbol{g}_i, \boldsymbol{y}_i \} : \boldsymbol{r}_i \in \mathbb{R}^{n-1}; \boldsymbol{l}_i \in \mathbb{R}^{n-1}; \boldsymbol{g}_i \in \mathbb{R}^n; \boldsymbol{y}_i = \mathcal{C}(\boldsymbol{r}_i, \boldsymbol{l}_i)^{-1} \boldsymbol{g}_i; i \le N_t \right\} (n \in \mathbb{N}^+). \quad (19)$$

The uniformity of the training set significantly affects the training effect; thus, we used random matrixes with each element independent and identically distributed as the training set to construct the training set in the model. Specifically, each element of $\boldsymbol{r}_i$, $\boldsymbol{l}_i$ and $\boldsymbol{g}_i$ is Gaussian distributed, that

$$\begin{cases} r_{ij}, l_{ij} \sim N\left(0, \rho_{\mathrm{v}}^2\right) (j < n), \\ g_{ik} \sim N\left(0, \rho_{\mathrm{g}}^2\right) (k \le n), \end{cases} \quad (20)$$

where ρ_{v} and ρ_{g} are the standard deviation of the training set and treated as given parameters in establishing the training set. Thus, the set P_n can be denoted by $P_n\left(\rho_{\mathrm{v}}, \rho_{\mathrm{g}}\right)$. To obtain a DNN model with a broader range of applications, the actual training set we use $\overline{P_n}$ is constructed from a set of random equations constructed with multiple sets of parameters, as

$$\begin{cases} \overline{P_n} = \bigcup\limits_{\rho_v \in V, \rho_g \in G} P_n\left(\rho_v, \rho_g\right), \\ V = \{0.01, 0.05, 0.1, 0.5, 1, 2, 4\}, \\ G = \{0.01, 0.05, 0.1, 0.5, 1, 2, 4\}. \end{cases} \quad (21)$$

In the DNN model, the loss function is specified as.

$$L = \frac{1}{N_t} \sum_{i=1}^{N_t} \left(\mathcal{S}_{\mathrm{NN}}\left(\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{g}_i\right) - \boldsymbol{y}_i \right)^2, \quad (22)$$

which is the commonly used Mean Squared Error (MSE) loss function in neural networks, especially for regression tasks.

The structure diagram of the DNN model introduced above is shown in Fig. 1.

It can be observed that, the model has a few hyper-parameters, including the number of the layers and neurons, and the training parameters. In Fig. 1, the number of layers in each part of the network $N_{\mathcal{L}} = N_{\mathcal{R}} = N_{\mathcal{G}} = N_{\mathcal{S}} = 3$, i.e., the total number of hidden layers N_l is 12, which is the choice to balance accuracy and computational consumption. For $i \le 3$, the number of the neurons in each hidden layer is $M_{i,j} = 20n$, while $M_{4,j} = 60n$. The number of neurons cannot be increased arbitrarily to balance the computational efficiency, and the above values are an optimal option that we have obtained through practical tests.

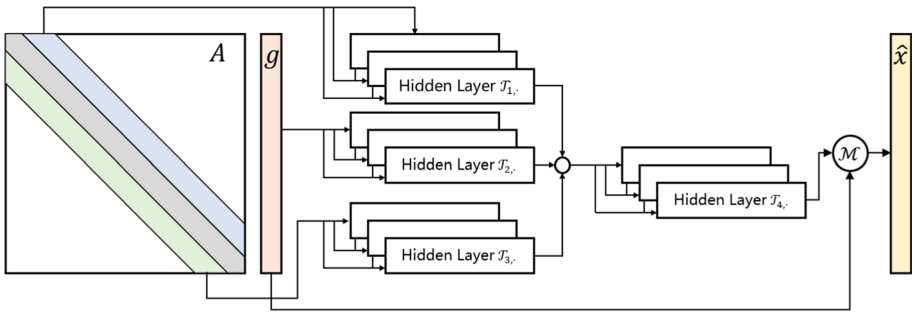


Fig. 1 Schematic of the DNN model in DRIM

2.3 The Iteration Method

In addition to predicting the solution of linear equations by neural networks, we adopt an iterative algorithm to improve the numerical accuracy in DRIM, as shown in Algorithm 2.

Algorithm 2 Iterative correction method

Multiply the right-hand side vector $\boldsymbol{g}$ by the amplification factor α
Solve the equation by the NN model and achieve a solution $\widehat{\boldsymbol{x}}^1$
Calculate the residual $\boldsymbol{r}^1 = \boldsymbol{g} - A\widehat{\boldsymbol{x}}^1/\alpha$
for $k = 1, 2, \ldots,$ MaxIterationStep
    if $\|\boldsymbol{r}^k\| \geq \tau\|\boldsymbol{r}^{k-1}\|$, **end**
    Calculate the right-hand side vector $\boldsymbol{r}_c = \alpha\boldsymbol{r}^k$
    Solve the equation with $\boldsymbol{r}_c$ as the right-hand side vector by the NN model and achieve a solution $\boldsymbol{c}$
    Update the solution $\widehat{\boldsymbol{x}}^{k+1} = \widehat{\boldsymbol{x}}^k + \boldsymbol{c}/\alpha$
    Update the residual $\boldsymbol{r}^k = \boldsymbol{g} - A\widehat{\boldsymbol{x}}^k/\alpha$
end
Output the solution $\widehat{\boldsymbol{x}} = \widehat{\boldsymbol{x}}^k$

The basic idea of the algorithm illustrated in Algorithm 2 is that, first, the DNN model yields a predicted solution $\widehat{\boldsymbol{x}}_0$ based on the equation, and $\widehat{\boldsymbol{x}}_0$ will also be adopted as the initial solution for the following iterations. In each iteration, the residual $\boldsymbol{r}^k$ of the current solution is fed into the DNN model instead of $\boldsymbol{g}$, and then the DNN model outputs the correction $\boldsymbol{c}$ of the current solution. This process is repeated until the convergence condition is reached. In this algorithm, the convergence condition is that the norm of the residual $\|\boldsymbol{r}^k\|$ stop decreasing at a certain step, i.e., $\|\boldsymbol{r}^k\| \geq \tau\|\boldsymbol{r}^{k-1}\|$, where τ is a constant. (We preset that $\|\boldsymbol{r}^0\|$ is a sufficiently large number.)

In the correction algorithm, the amplification factor α is introduced to improve the accuracy of the algorithm. Inspired by the property that the exact mapping $\mathcal{S}$ must satisfy that $\mathcal{S}(A, k\boldsymbol{g}) = k\mathcal{S}(A, \boldsymbol{g})$ for an arbitrary constant k, we applied this constant as an amplification factor to DRIM. Specifically, we replace $\mathcal{S}_{\mathrm{NN}}(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{g})$ by $\mathcal{S}_{\mathrm{NN}}(\boldsymbol{l}, \boldsymbol{r}, \alpha\boldsymbol{g})/\alpha$ where the factor α is greater than 1. The factor theoretically amplifies the input and is more conducive to the prediction of the DNN model with the ReLU activation function.

We evaluated the influence of the factors ρ_{g}, ρ_v and α on the numerical accuracy of DRIM with a series of numerical tests. The equations for the test are constructed in the same method as the training set but are completely different matrices. In addition, we constructed multiple test sets with different ρ_{g} and ρ_v for separate testing and collecting the mean errors $\|r\|_2$ for comparison. $\|r\|_2$ is calculated by

$$\|r\|_2 = \frac{1}{N_t}\sum_{\mathrm{i}=1}^{N_t}\left\|A_i\hat{\boldsymbol{x}}_i - \boldsymbol{y}_i\right\|, \tag{23}$$

where $(A_i, \boldsymbol{y}_i)$ is a single element of the test set and N_t is the number of test sets, $N_t = 1000$ in this test. Moreover, we quantified the effect of the amplification factor α based on the test, and the numerical results are shown in Fig. 2.

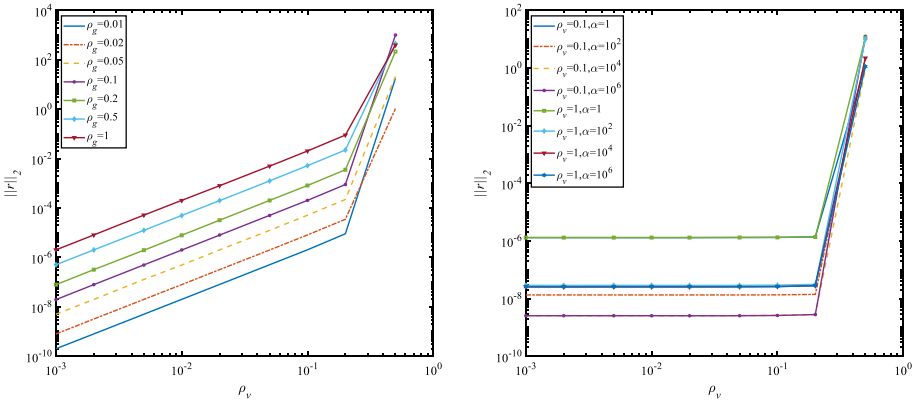


Fig. 2 The mean error norms $\|r\|_2$ of DRIM for different ρ_g and ρ_v of the test set. (Left: $\|r\|$ of the DNN model; right: $\|r\|_2$ of DRIM)

Figure 2a demonstrates that $\|r\|_2$ is positively correlated with both ρ_g and ρ_v. For smaller ρ_g, there is an approximate relationship that

$$\|r\|_2 \approx C_L \rho_g^2 \rho_v^2, \tag{24}$$

where C_L is a constant independent of ρ_g and ρ_v. Although having the same polynomial order in (24), ρ_g and ρ_v have essentially different effects on the numerical error: ρ_v affects the diagonal dominance of the equation, i.e., the regularity of the equation; ρ_g has a linear relationship with the solution of the equation, which directly determines the magnitude of the solution. If ρ_v approaches or exceeds 0.5, the regularity of the equation is so poor that the equation cannot be solved by the DNN model, and DRIM is consequently invalid.

Although ρ_v has a significant effect on the accuracy of the DNN model, Fig. 2b illustrates that the corrected iteration method in DRIM almost eliminates its effect. This implies that for an arbitrary equation, the error of DRIM will only be related to the magnitude of the solution, i.e., DRIM has a constant relative accuracy.

The effect of the matrix structure (the size of the matrix and the relative position of non-diagonal elements) on the accuracy of DRIM is evaluated as well. Numerical results indicate that the approximate relation (23) remains valid for the equations with a similar structure, thus we compared the constants C_L for the four typical types of matrices at different sizes, as shown in Fig. 3.

The four types of matrices involved in Fig. 4 represent a typical tridiagonal matrix (A), a tridiagonal matrix with an asymmetric structure (B), a tridiagonal matrix with an interval (C), and a five-diagonal matrix (D), respectively. In Fig. 3, the relative difference of constants C_L among different types of matrices does not exceed 15%, which demonstrates that DRIM is effective for a variety of types of equations. The DNN model illustrated in Fig. 1 is for tridiagonal equations, but it also has the potential to solve other equations of multiple diagonal forms, and it does not restrict the position of non-diagonal vectors. Therefore, when we adopt a difference scheme with higher-order precision or encounter more complicate boundary conditions, we only need to modify the architecture of the input layers and the training set of the DNN model.

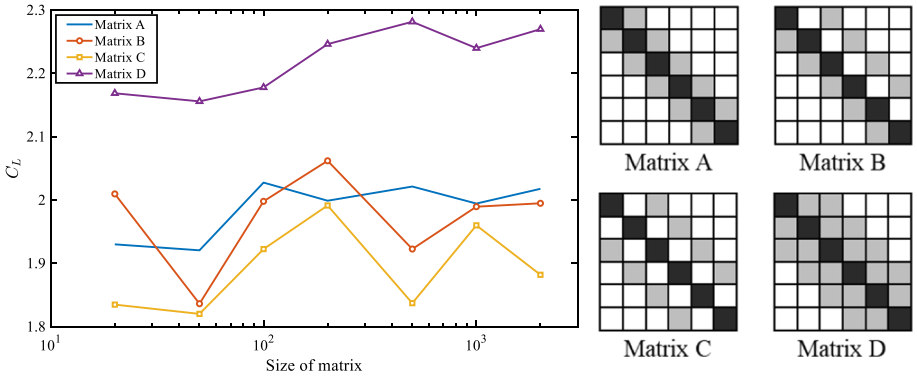
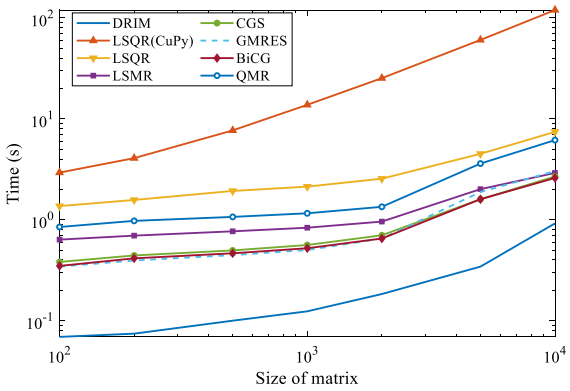


Fig. 3 The constant C_L of the error for different sizes and structures of matrices

Fig. 4 The computational time of different solving methods



3 Numerical Results and Validation

In this section, we examine the performance of DRIM by considering different numerical examples. We focus on the computational efficiency and numerical accuracy of DRIM in fluid simulation. Thus, we employed DRIM to solve well-known cases, e.g. Poiseuille flow, backward-facing step flow, and driven cavity flow. For comparison, we present the computational times and numerical solutions of the classic algorithms and the experimental data in the classic literature.

3.1 The Results of Computational Efficiency

The acceleration effect of DRIM on the linear equation solving process is our most important result. To evaluate the computational efficiency of DRIM for equations of different sizes, we construct random equations of sizes from 100 to 100,000. For each equation size, we solved 1000 equations separately with the classic algorithms and DRIM and calculated the total computational time of the solving processes, shown in Fig. 4.

As a NN-based algorithm, the efficient implementation of DRIM must rely on GPU devices, but most classic algorithms or libraries do not have natural GPU compatibility.

Table 1 Parameters of the test platform

Hardware parameters		Software environment	
CPU model	Intel Xeon-W3175X	Library	Version
CPU cores	28	SciPy	1.5.2
CPU frequency	3.1 GHz	TensorFlow	2.2.0
GPU model	Nvidia RTX2080Ti	CuPy	8.3.0
RAM	256 GB	CUDA	10.2

To rigorously evaluate the efficiency advantages brought by DRIM, we choose the classic algorithms on CPU and GPU at the same time (Table 1).

In this test, both the DRIM-based solver and classic solvers are implemented in Python [25] and the DNN model is based on TensorFlow platform [26], while the classic algorithm is provided by SciPy library [27] (for CPU implementation) and CuPy (for GPU implementation). Specifically, the classic algorithms in the test consist of BiCG [28], CGS [29], GMRES [30], QMR [31, 32], and LSMR [33], which are iteration-based algorithms adapted to large-scale computation [34]. Furthermore, to ensure that the classic algorithms served as references can operate at optimal efficiency, the SciPy-based solvers in the test all employ compressed sparse row (CSR) format for the storage and computation of the equations.

To make this test closer to the practical simulations, the computational time results in Fig. 4 contain the time of both the solving process and the matrix assembly process. This causes the algorithms operating on the GPU, including the CuPy-based LSQR algorithm and DRIM, to be constrained by communication bottlenecks. The detailed test results is listed in Table 2.

In the results, DRIM offers a 2–10 times speed-up relative to the most efficient classic algorithm for solving equations of all sizes. Nonetheless, for larger equations (approaching a size of 10^4), the computational time increases more rapidly due to the limitation of available GPU memory (as tested with only 11GB of GPU memory). Therefore, for large-scale simulations, it is advisable to utilize GPUs with larger memory capacities or to implement optimization strategies specifically for memory constraints.

In addition to the size of the matrix, the number of equations has a certain impact on the calculation time. Different from the classic methods, DRIM, as a DNN-based method, is

Table 2 Computational time of different methods with different sizes of equations (unit: second)

Size of matrix	Methods							
	DRIM	CuPy-LSQR	CGS	LSQR	LSMR	GMRES	BiCG	QMR
10^2	0.069	2.935	0.382	1.366	0.636	0.343	0.349	0.848
2×10^2	0.074	4.079	0.443	1.571	0.696	0.394	0.414	0.975
5×10^2	0.100	7.683	0.497	1.929	0.768	0.446	0.465	1.067
10^3	0.124	13.778	0.562	2.130	0.834	0.501	0.522	1.159
2×10^3	0.184	25.323	0.702	2.552	0.959	0.652	0.652	1.344
5×10^3	0.343	60.555	1.597	4.502	2.010	1.882	1.597	3.605
10^4	0.920	119.849	2.681	7.435	2.911	3.033	2.594	6.153

capable to input multiple equations at one time and solve them simultaneously. The character is of positive significance to reduce CPU–GPU communication and improve GPU utilization efficiency. This feature has positive implications for reducing CPU–GPU communication and increasing the efficiency of GPU utilization. To demonstrate this advantage, we conducted a comparison of matrices of different sizes, as shown in Table 3.

It can be observed from Table 3 that DRIM is not more efficient than other methods in solving 1 or 10 equations. However, as the number of equations becomes larger, the time consumption of the classic methods tends to grow linearly with the number of equations, while the growth rate of DRIM is significantly lower. To further demonstrate this property, we show the ratio of the solving time for multiple equations to that for a single equation in Fig. 5.

In Fig. 5, the computational time of the classic algorithm on the CPU (LSQR in Fig. 5) is approximately linear with respect to the number of equations. The same algorithm on the GPU has a significant acceleration for the small matrices (100×100), but is also approximately linear for larger matrices. This indicates that the algorithm on GPU is limited by communication bottlenecks and explains why the CuPy-based algorithm in Fig. 4 does not achieve the expected high efficiency. For all sizes of matrices in Fig. 5, the computational time of DRIM increases more smoothly as the number of equations increases. This property is derived from the transformation of matrices into vectors by neural networks and enables DRIM well suited to be combined with the DST-based decomposition methods. It is worth mentioning that in a 3D simulation, the number of sub-equations is the square of that in a 2D simulation, thus DRIM will have a higher efficiency advantage for 3D problems.

3.2 The Poiseuille Flow

The Poiseuille flow is a well-known problem with analytic solutions; therefore, it is a suitable case for verifying the accuracy of DRIM. We construct a model of Poiseuille flow shown in Fig. 6. The boundary conditions of the model are constant velocity $u = (\overline{u}, 0)$ at the inlet, fully developed flow ($\partial_x u_x = 0, u_y = 0$) at the outlet, and solid boundaries at the other boundaries. The parameters of the geometry and the boundary conditions are listed in Table 4.

In the analytical solution as well as in the numerical results, the velocity distribution at the outlet boundary gradually stabilizes and finally shows the form of a quadratic function. The simulation results of the fully developed flow are shown in Fig. 7a and b illustrates the velocity distribution at the outlet boundary.

In Fig. 7b, in terms of the distribution of outlet velocity, the computational results of DRIM coincide with the analytical solution, which proves that DRIM has good numerical accuracy in this model. In the following sections, we will apply DRIM to some complex flow models to further verify its reliability and numerical accuracy.

3.3 The Backwards-Facing Step Flow

A backwards-facing step flow can be regarded as having the simplest geometry while retaining rich flow physics manifested by flow separation, flow reattachment, and multiple recirculating bubbles in the channel depending on the Reynolds number and the geometrical parameters such as the step height and the channel height. The geometric model and boundary conditions of the backwards-facing flow are similar to those of the Poiseuille flow. However, part of the inlet is the rigid boundary, as shown in Fig. 8.

Table 3 Computational time of different methods with different numbers of equations (unit: second)

Number of equations	Size of matrix								
	100×100			1000×1000			$10{,}000 \times 10{,}000$		
	DRIM	LSQR	CuPy-LSQR	DRIM	LSQR	CuPy-LSQR	DRIM	LSQR	CuPy-LSQR
1	0.0374	0.0016	0.0226	0.0349	0.0025	0.0191	0.0348	0.0077	0.1579
10	0.0358	0.0149	0.0294	0.0376	0.0243	0.1775	0.0416	0.0795	1.2526
10^2	0.0460	0.1543	0.3421	0.0459	0.2375	1.4441	0.1028	0.7994	12.041
10^3	0.0704	1.3949	2.9347	0.1261	2.1394	13.771	0.9256	7.4888	120.10
10^4	0.3986	13.817	29.062	1.2090	21.413	136.69	10.171	75.330	1211.2

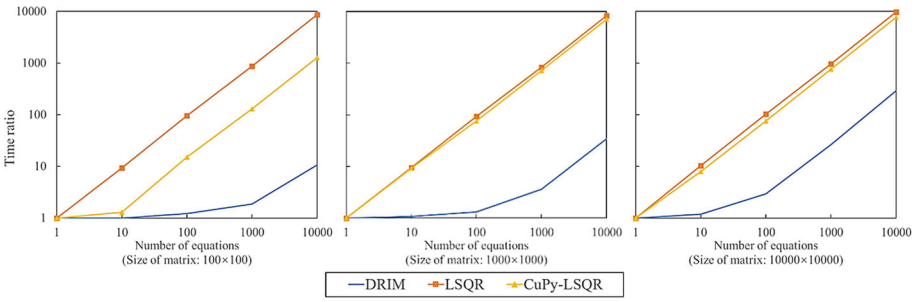


Fig. 5 Ratio of the computational time of multiple equations to the computational time of single equation

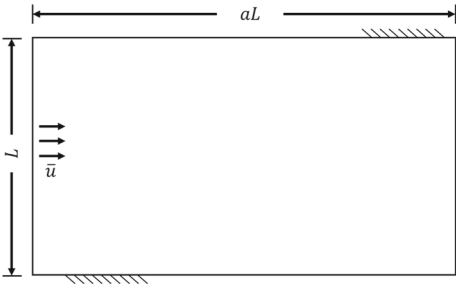


Fig. 6 The geometry and boundary condition of the case of Poiseuille flow

Table 4 The parameters settings in the case of Poiseuille flow

Parameter	Value
Viscosity $\boldsymbol{v}$	0.01
Time step $\boldsymbol{\Delta t}$	0.01
Spatial step on x-axis $\boldsymbol{\Delta x}$	0.02
Spatial step on y-axis $\boldsymbol{\Delta y}$	0.02
Distance between walls $\boldsymbol{L}$	1
Aspect ratio $\boldsymbol{a}$	10
Inlet velocity $\overline{\boldsymbol{u}}$	1

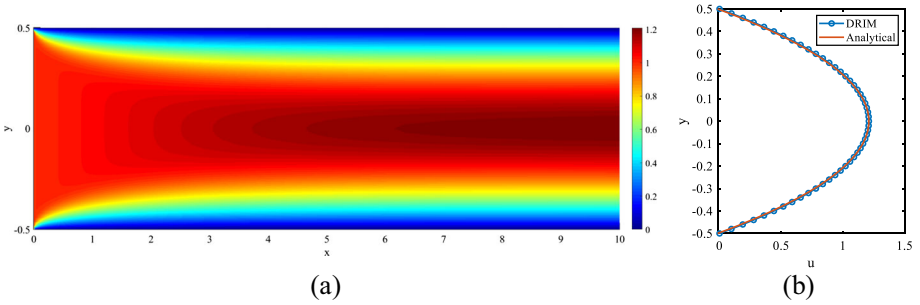


Fig. 7 The velocity distribution in the case of Poiseuille flow. **a**: x-axis velocity; **b**: the velocity comparison at the outlet boundary

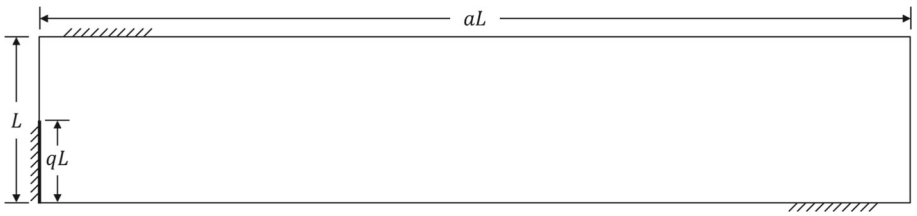


Fig. 8 The boundary condition of the case of backward-facing step flow

The parameter settings of the case of backward-facing step flow are similar to those of the Poiseuille flow, where the spatial and time steps, the inlet velocity, and the viscosity are the same. However, the backwards-facing step flow contains more complex flow phenomena and a longer spatial scale is necessary to ensure its full development, thus we set the geometric parameters as in Table 5

We selected several representative time steps from the simulation results to demonstrate the flow development process, as shown in Fig. 9. We focus on whether the proposed algorithm accurately calculates the structure of the vortices in the flow, and therefore, Fig. 9 presents the distribution of vorticity at each time point.

In Fig. 9, the separation and reattachment of the flow can already be clearly observed, and two main eddies (the lower wall eddy and upper wall eddy) are evident in the flow field. Moreover, we further evaluated the numerical accuracy of DRIM by comparing the results with those in the literature. Figure 10 illustrates three substantial characteristic lengths of the backwards-facing step flow in the form of contour plots of the vorticity, including the length of the lower eddy (x_1) and the position of the flow separation (x_2) and reattachment (x_3). We

Table 5 The geometric parameters in the case of backward-facing step flow

Parameter	Value
Distance between walls $\boldsymbol{L}$	1
Aspect ratio $\boldsymbol{\alpha}$	30
Length of step $\boldsymbol{q}$	0.5

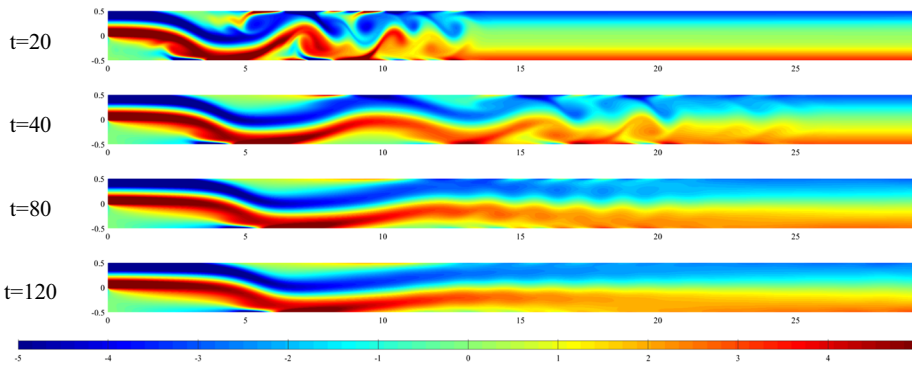


Fig. 9 Numerical results of backwards-facing step flow with Re = 1000 (vorticity distribution)

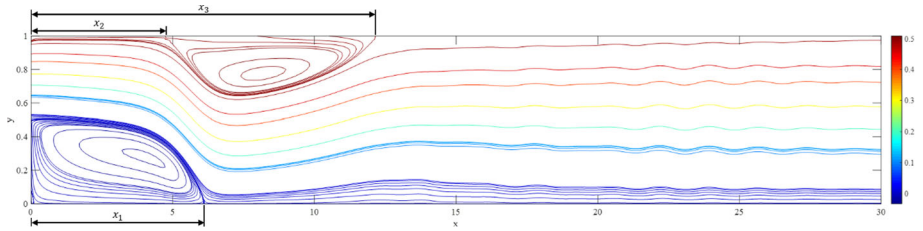


Fig. 10 Characteristic lengths in the case of backward-facing step flow

Table 6 Characteristic lengths in the case of backward-facing step flow

	x_1/L	x_2/L	x_3/L
DRIM	6.06	4.8	12.16
Ref. [35]	6.55	5.17	12.67
Ref. [36]	6.5605	5.237	12.441

compared these characteristic lengths in the simulation results with those in References [35] and [36], as shown in Table 6.

In the comparison in Table 6, the computational results of DRIM agree well with those of References in terms of the characteristic values, which further demonstrates the good excellent numerical accuracy of DRIM in fluid simulation.

3.4 The Driven Cavity Flow

In addition to the above two cases, we utilized DRIM for the simulation of driven cavity flow, a well-known benchmark problem. Compared with the previous two examples, the driven cavity flow contains more flow properties and is more sensitive to the Reynolds number. Therefore, we set up the models with Reynolds numbers of 1000 and 10,000 for numerical validation.

The geometry and boundary conditions of the model are set as shown in Fig. 11. This model has simple geometric parameters; thus, we set the Reynolds number by viscosity. We set the time step $\Delta t = 0.01$ and the spatial step $\Delta x = 0.025$, and the total size of the grid is 400×400.

Figures 12 and 13 present the vorticity distribution at each stage of the driven cavity flow with Reynolds numbers of 1000 and 10,000, respectively.

Because of the complexity of this model, we used two methods to verify the numerical accuracy of the computational results. The first is the comparison of the velocity distribution on the test line (shown in Fig. 11) with the reference [37], shown in Fig. 14.

In Fig. 14, the results for both Reynolds numbers are very close to those of reference [37]. In the driven cavity flow with a Reynolds number of 10,000, there are regions with sharp velocity variations in both x and y directions, and DRIM has high accuracy in these regions as well.

The second numerical verification is about the time evolution of the accuracy. We separately applied DRIM and LGMRES to the simulation of the model with Reynolds numbers

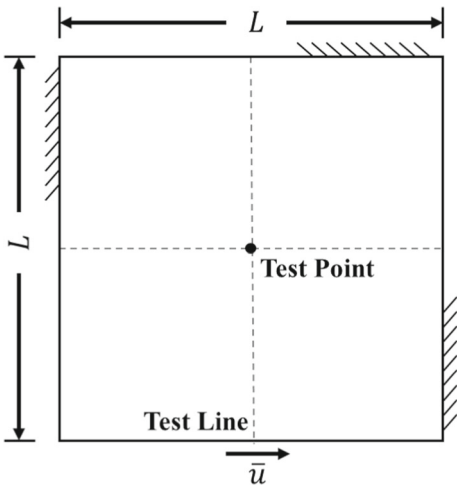


Fig. 11 The boundary condition of the case of driven cavity flow

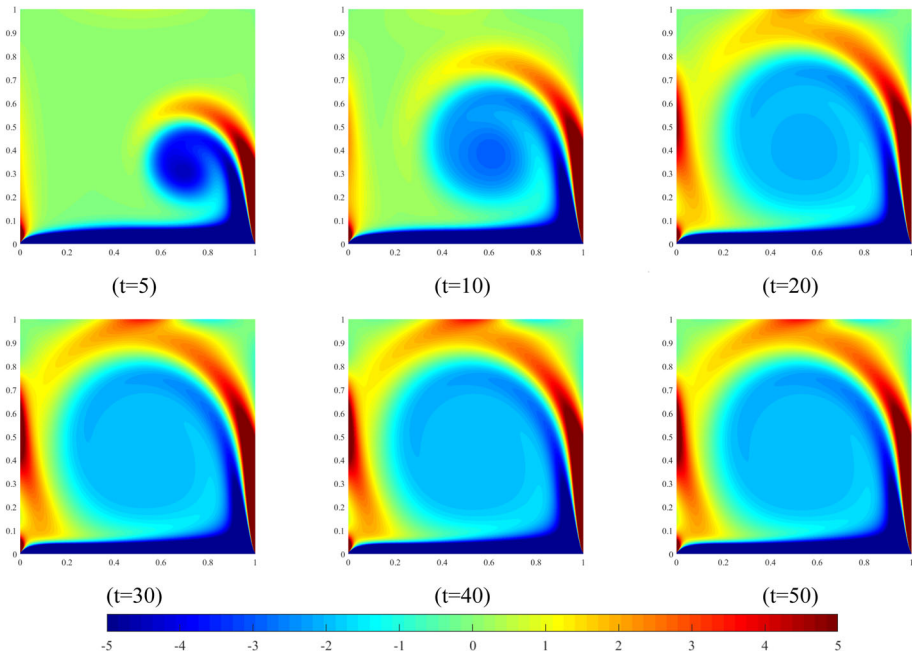


Fig. 12 Numerical results of cavity flow with Re = 1000 (vorticity distribution)

of 1000 and 10,000, and the pressure development with time at the test point in the center of the model (as shown in Fig. 11) is shown in Fig. 15.

In Fig. 15, the pressure difference at the test point is kept around 10^{-5}, thus indicating that DRIM has high numerical accuracy; moreover, the numerical error is stable and does not accumulate to cause a divergence.

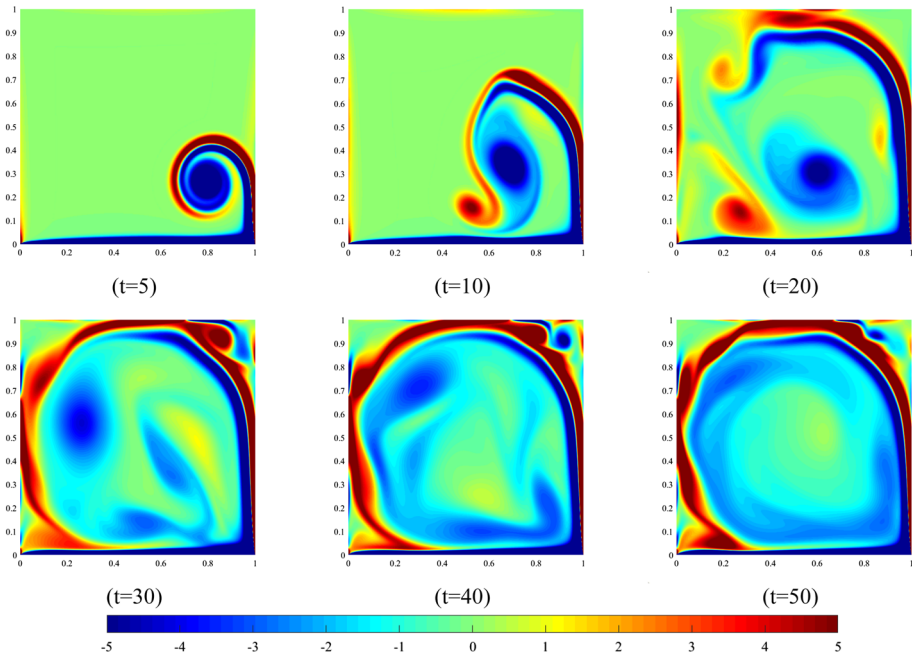


Fig. 13 Numerical results of cavity flow with Re = 10,000 (vorticity distribution)

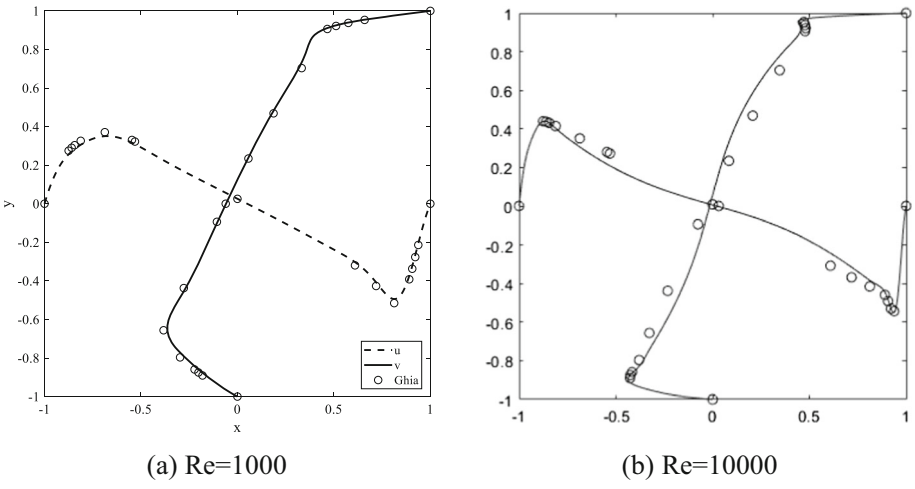


Fig. 14 The comparison of the velocity distribution on the test line

In addition to numerical accuracy, we are also concerned about the effect of DRIM on the acceleration of time advance in fluid simulations. Figure 16 presents the average time of a single time step of the LGMRES-based solver and DRIM-based solver when the size of the grid is N × N.

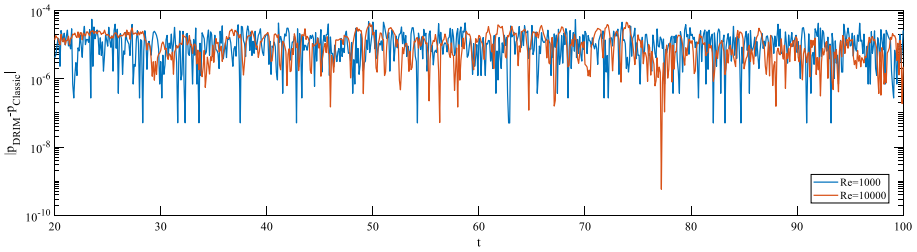
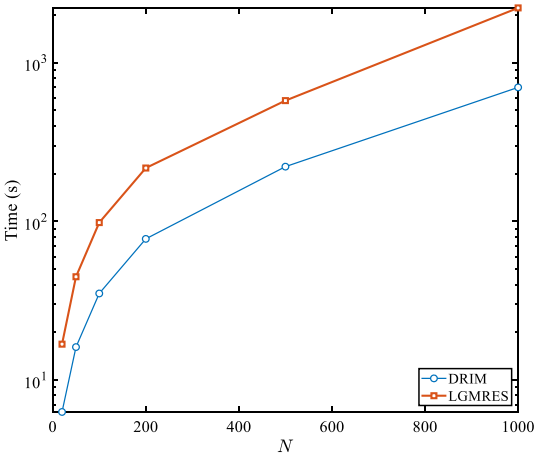


Fig. 15 The error of the pressure at the test point

Fig. 16 The efficiency test for the case of driven cavity flow



Due to the other non-optimized subroutines and the limited size of the problem, the DRIM-accelerated SIMPLE algorithm is difficult to achieve the efficiency improvement shown in Fig. 4, but DRIM still achieves a 2–4 times speed-up in the solving the PPE. Therefore, it can be proved that DRIM is a promising and efficient algorithm for the simulation of incompressible flow and other large-scale problems.

4 Conclusion

In this paper, we introduced DRIM, an innovative DNN-based method to solve PPE, which accelerates the SIMPLE algorithm in the simulation of incompressible flow. DRIM combines the advantages of DNN and correction iteration methods to yield high accuracy and computational efficiency on a heterogeneous architecture platform.

(1) Based on the DNN algorithm, DRIM has high computation efficiency and native hardware compatibility on GPU and other heterogeneous platforms. Compared with the classic solving methods, DRIM can solve numerous linear equations with low computation complexity simultaneously.
(2) We utilized a DST-based method to transform the original PPE into multiple independent sub-equations. These sub-equations are all tridiagonal equations. Therefore, we employ a vector-set representation to enable them to be input into the DNN model.

(3) To balance computational efficiency and accuracy, we construct a 12-layer DNN model and train it with random equations. Res-Net architecture is combined in the model to avoid the network degradation and vanishing gradient problem.
(4) To achieve acceptable precision, we employed the correction iteration to iteratively reduce the error. The numerical results have proven that the iteration method almost eliminates the influence of the diagonal dominance of the equation on the accuracy.
(5) We applied DRIM to the numerical simulations of the Poiseuille flow, the backwards-facing step flow, and the driven cavity flow, respectively. The numerical results have proven that the numerical precision of DRIM is comparable to that of classic solvers and the DRIM-based algorithm is about 2–10 times faster than the conventional method.

Like the other algorithms based on NN, the performance of DRIM is also significantly affected by the factors of input variables. We investigate some essential factors, e.g. the diagonal dominance and the norm of the righthand-side vector and propose several ideas that might optimize the performance of DRIM.

Acknowledgements This research was funded by the national key R&D program for international collaboration, under grant 2020YFA0712502. The Natural Science Foundation of China (NSFC), grant 11972384, and Guangdong Science and Technology Fund, grant 2021B1515310001, also supported this work.

Author contributions Zichao Jiang and Qinghe Yao wrote the main manuscript text. Zichao Jiang, Zhuolin Wang and Junyang Jiang developed the solver proposed in this manuscript. Gengchao Yang and Zhang Yi prepared the figures and revised the section 1 and 3 of the manuscript. All authors reviewed the manuscript.

Declarations

Conflict of interest The authors declare no conflict of interests.

References

1. Issa RI, Gosman AD, Watkins AP (1986) The computation of compressible and incompressible recirculating flows by a non-iterative implicit scheme. J Comput Phys 62(1):66–82
2. Xiao X, Zhou Y, Wang H, Yang X (2020) A novel CNN-based poisson solver for fluid simulation. IEEE Trans Visual Comput Graphics 26(3):1454–1465
3. Costa P (2018) A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. Comput Math Appl 76(8):1853–1862
4. Alshemali B, Kalita J (2020) Improving the reliability of deep neural networks in NLP: a review. Knowl-Based Syst 191:105210. https://doi.org/10.1016/j.knosys.2019.105210
5. Zhang XY, Zou JH, He KM, Sun J (2016) Accelerating very deep convolutional networks for classification and detection. IEEE Trans Pattern Anal Mach Intell 38(10):1943–1955
6. Han J, Nica M, Stinchcombe AR (2020) A derivative-free method for solving elliptic partial differential equations with deep neural networks. J Comput Phys 419:109672
7. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366. https://doi.org/10.1016/0893-6080(89)90020-8

8. Kharazmi E, Zhang Z, Karniadakis GE (2021) hp-VPINNs: variational physics-informed neural networks with domain decomposition. Comput Methods Appl Mech Eng 374:113547
9. Berg J, Nystrom K (2018) A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317:28–41
10. Chaudhari P, Oberman A, Osher S, Soatto S, Carlier G (2017) Partial differential equations for training deep neural networks. In 2017 Fifty-First Asilomar Conference on Signals, Systems, and Computers. Conference Record of the Asilomar Conference on Signals Systems and Computers. pp 1627–1631
11. Weinan E, Han J, Jentzen A (2017) Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun Math Stat 5(4):349–380. https://doi.org/10.1007/s40304-017-0117-6
12. Rudd K, Di Muro G, Ferrari S (2014) A constrained backpropagation approach for the adaptive solution of partial differential equations. IEEE Trans Neural Netw Learn Syst 25(3):571–584
13. Sirignano J, Spiliopoulos K (2018) DGM: a deep learning algorithm for solving partial differential equations. J Comput Phys 375:1339–1364
14. Weinan E, Bing Yu (2018) The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun Math Stat 6(1):1–12. https://doi.org/10.1007/s40304-018-0127-z
15. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
16. Jagtap AD, Karniadakis GE (2020) Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Commun Comput Phys 28(5):2002–2041
17. Papadopoulos L, Bakalakos S, Nikolopoulos S, Kalogeris I, Papadopoulos V (2023) A computational framework for the indirect estimation of interface thermal resistance of composite materials using XPINNs. Int J Heat Mass Transf 200:123420. https://doi.org/10.1016/j.ijheatmasstransfer.2022.123420
18. Shukla K, Jagtap AD, Karniadakis GE (2021) Parallel physics-informed neural networks via domain decomposition. J Comput Phys 447:110683. https://doi.org/10.1016/j.jcp.2021.110683
19. Ray D, Hesthaven JS (2018) An artificial neural network as a troubled-cell indicator. J Comput Phys 367:166–191. https://doi.org/10.1016/j.jcp.2018.04.029
20. Xie JB, He JC, Bao Y, Chen X (2021) A low-communication-overhead parallel DNS method for the 3D incompressible wall turbulence. Int J Comput Fluid Dyn 35(6):413–432
21. Xue W, Wang Y, Chen Z, Liu H (2023) An integrated model with stable numerical methods for fractured underground gas storage. J Clean Prod 393:136268
22. Xue W, Wang Y, Liang Y, Wang T, Ren B (2024) Efficient hydraulic and thermal simulation model of the multi-phase natural gas production system with variable speed compressors. Appl Therm Eng 242:122411
23. Ahmed N, Natarajan T, Rao KR (1974) Discrete cosine transform. IEEE Trans Comput C–23(1):90–93. https://doi.org/10.1109/T-C.1974.223784
24. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. arXiv e-prints, p arXiv:1512.03385
25. Van Rossum GD, Fred L (2009) Python 3 reference manual. CreateSpace
26. Abadi M et al. (2016) Tensorflow: large-scale machine learning on heterogeneous distributed systems
27. Pauli V et al (2020) SciPy 1.0: fundamental algorithms for scientific computing in python. Nat Methods 17:261–272
28. Van Der Vorst HA (1992) Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J Sci Stat Comput 13(2):631–644
29. Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems. SIAM J Sci Stat Comput 10(1):36–52
30. Saad Y, Schultz MH (1986) GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput 7(3):856–869
31. Freund RW, Nachtigal NM (1991) QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numerische Mathematik 60(1):315–339. https://doi.org/10.1007/BF01385726
32. Freund RW, Nachtigal NM (1994) An implementation of the QMR method based on coupled two-term recurrences. SIAM J Sci Comput 15(2):313–337
33. Fong DCL, Saunders M (2011) LSMR: an iterative algorithm for sparse least-squares problems. SIAM J Sci Comput 33(5):2950–2971
34. Benzi M (2002) Preconditioning techniques for large linear systems: a survey. J Comput Phys 182(2):418–477
35. Erturk E (2008) Numerical solutions of 2-D steady incompressible flow over a backward-facing step, part I: high Reynolds number solutions. Comput Fluids 37(6):633–655

36. Ramšak M, Škerget L (2004) A subdomain boundary element method for high-Reynolds laminar flow using stream function-vorticity formulation. Int J Numer Methods Fluids 46(8):815–847. https://doi.org/10.1002/fld.776
37. Ghia U, Ghia KN, Shin CT (1982) High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. J Comput Phys 48(3):387–411