East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.2024-006.060424

Vol. **14**, No. 3, pp. 490-506
August 2024

A Hybrid Neural-Network and MAC Scheme for Stokes Interface Problems

Che-Chia Chang[1,2], Chen-Yang Dai[1], Wei-Fan Hu[3,4], Te-Sheng Lin[1,4] and Ming-Chih Lai[1,*]

[1]*Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan.*
[2]*Institute of Artificial Intelligence Innovation, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan.*
[3]*Department of Mathematics, National Central University, Taoyuan 32001, Taiwan.*
[4]*National Center for Theoretical Sciences, National Taiwan University, Taipei 10617, Taiwan.*

Received 11 January 2024; Accepted (in revised version) 6 April 2024.

Abstract. In this paper, we present a hybrid neural-network and MAC (Marker-And-Cell) scheme for solving Stokes equations with singular forces on an embedded interface in regular domains. As known, the solution variables (the pressure and velocity) exhibit non-smooth behaviors across the interface so extra discretization efforts must be paid near the interface in order to have small order of local truncation errors in finite difference schemes. The present hybrid approach avoids such additional difficulty. It combines the expressive power of neural networks with the convergence of finite difference schemes to ease the code implementation and to achieve good accuracy at the same time. The key idea is to decompose the solution into singular and regular parts. The neural network learning machinery incorporating the given jump conditions finds the singular part solution, while the standard MAC scheme is used to obtain the regular part solution with associated boundary conditions. The two- and three-dimensional numerical results show that the present hybrid method converges with second-order accuracy for the velocity and first-order accuracy for the pressure, and it is comparable with the traditional immersed interface method in literature.

AMS subject classifications: 35J25, 65N06, 68T07

Key words: Stokes interface problems, neural networks, MAC scheme, hybrid method.

1. Introduction

In this paper, we consider d-dimensional ($d = 2$ or 3) Stokes equations in a regular domain $\Omega \subseteq \mathbb{R}^d$, in which an embedded interface Γ with codimension $d-1$ (assumed to be smooth and closed) separates the domain into Ω^- and Ω^+. Denoting the interface

[*]Corresponding author. *Email address:* `mclai@math.nctu.edu.tw` (M.C. Lai)

position by $\mathbf{X}$, an interfacial force $\mathbf{F}(\mathbf{X})$ defined only along the interface Γ is exerted to the surrounding fluid and affects the flow behavior accordingly. Assuming that the viscosity of both fluid subdomains are identical, one can write a single Stokes fluid system with an external body force $\mathbf{g}$, viz.

$$\begin{aligned} &-\nabla p(\mathbf{x}) + \mu\Delta\mathbf{u}(\mathbf{x}) + \int_\Gamma \mathbf{F}(\mathbf{X})\delta^d(\mathbf{x}-\mathbf{X})\,\mathrm{d}\mathbf{X} + \mathbf{g}(\mathbf{x}) = \mathbf{0}, && \mathbf{x}\in\Omega, \\ &\nabla\cdot\mathbf{u}(\mathbf{x}) = 0, && \mathbf{x}\in\Omega, \\ &\mathbf{u}(\mathbf{x}) = \mathbf{u}_b(\mathbf{x}), && \mathbf{x}\in\partial\Omega, \end{aligned} \tag{1.1}$$

where $\mathbf{u}(\mathbf{x})$ and $p(\mathbf{x})$ are the velocity and the pressure, respectively, μ is the constant viscosity, and $\mathbf{u}_b(\mathbf{x})$ is the velocity boundary condition. Notice that, the force term appeared in the first equation is singular and expressed in the Immersed Boundary (IB) formulation [17] in which the integral involves a d-dimensional Dirac delta function δ^d over a $(d-1)$-dimensional surface resulting in one-dimensional delta function singularity. The above system (1.1) can be solved efficiently by the IB method [17, 21]. That is, the integral involving the delta function δ^d (line integral for $d=2$ and surface integral for $d=3$) can be regularized via a discrete delta function (a regularized form of the Dirac delta function) so the interfacial force $\mathbf{F}$ can be spread into the fluid grid points near the interface. However, this singular force spreading process results in first-order accuracy for the velocity [16] and has $\mathcal{O}(1)$ error for the pressure [2].

Due to the delta function singularity in Eq. (1.1), the pressure and velocity are no longer smooth across the interface so that the problem can be reformulated as the immersed interface formulation [13]

$$-\nabla p(\mathbf{x}) + \mu\Delta\mathbf{u}(\mathbf{x}) + \mathbf{g}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{1.2}$$

$$\nabla\cdot\mathbf{u}(\mathbf{x}) = 0, \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{1.3}$$

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_b(\mathbf{x}), \quad \mathbf{x}\in\partial\Omega, \tag{1.4}$$

where the pressure and velocity fields are subjected to the following jump conditions (see the derivation in [11]):

$$[\![p(\mathbf{X})]\!] = F_n(\mathbf{X}), \quad \mathbf{X}\in\Gamma, \tag{1.5}$$

$$[\![\mathbf{u}(\mathbf{X})]\!] = \mathbf{0}, \quad \mu\left[\!\!\left[\frac{\partial\mathbf{u}}{\partial\mathbf{n}}(\mathbf{X})\right]\!\!\right] = -\big(\mathbf{F}(\mathbf{X}) - F_n(\mathbf{X})\mathbf{n}(\mathbf{X})\big), \quad \mathbf{X}\in\Gamma. \tag{1.6}$$

Here, $F_n(\mathbf{X}) = \mathbf{F}(\mathbf{X})\cdot\mathbf{n}(\mathbf{X})$ denotes the normal component of the interfacial force with $\mathbf{n}(\mathbf{X})$ being the unit outward normal vector at $\mathbf{X}\in\Gamma$. We use the double bracket $[\![\cdot]\!]$ to denote the jump of a quantity evaluated by the quantity from the Ω^+ side minus the one from the Ω^- side. From the jump condition (1.5), one can see that the pressure is discontinuous across Γ when $F_n \neq 0$. Also, from the jump condition (1.6), the velocity is continuous across the interface while its normal derivative is discontinuous and determined by the tangential part of $\mathbf{F}$. As a result, the velocity has a cusp behavior across the interface Γ.

As mentioned above, the pressure and velocity exhibit non-smooth behaviors across the interface, so extra discretization efforts must be paid near the interface in order to have small order of local truncation errors in finite difference schemes. A commonly used method is the Immersed Interface Method (IIM) [13] that incorporates the jump conditions into finite difference discretization on a uniform Cartesian grid and hereby achieves second-order convergence. Various versions of IIM can be found in [4,12,14,24] and references therein. However, the implementation of jump incorporations could be tedious when dealing with complex interface geometry, especially in three-dimensional problems.

Until recent years, a new approach called physics-informed neural networks (PINNs) [18], was proposed to solve partial differential equations using machine learning methodology. The idea of PINN is to find a neural network approximate solution that minimizes the mean squared error loss consisting of the residuals of underlying differential equations along with boundary and initial conditions. Inspired by PINN, the authors have developed a series of structure-preserving neural network methods to solve elliptic interface problems [8,23] and Stokes interface problems [22] that are able to capture the solution and derivative jump discontinuities sharply. These neural network solvers are mesh-free, so the correction of local truncation errors near the interface can be completely avoided. The results (in terms of accuracy) are also comparable with the ones obtained from traditional IIM mentioned above. Other related works on the elliptic interface problem include the deep unfitted Nitsche method proposed by Guo *et al.* [6] and the convergence of PINNs method shown by Wu *et al.* [25]. Nevertheless, so far all the neural network solvers do not show a clean rate of convergence observed in traditional finite difference and finite element methods. So in [9], the authors have developed a novel hybrid method that inherits both advantages of neural network approach and traditional finite difference method to solve Poisson interface problems. The entire computation only comprises a supervised learning task for function approximation and a fast direct solver for the Poisson equation, which can be easily and directly implemented regardless of the interface geometry. More importantly, this hybrid method shows second-order convergence in the numerical results. In this paper, we aim to extend this hybrid methodology for Poisson interface problem to the Stokes interface problem (1.2)-(1.4) with the jump conditions (1.5)-(1.6). Again, the whole computation of the proposed method simply consists of a supervised learning task and a regular Stokes solver. We should emphasize that we do not attempt to make a rigorous competition or comparison with existing traditional methods. Instead, we provide an alternative easy-to-implement way to solve Stokes interface problems by leveraging the strengths of machine learning approach and finite difference method.

The rest of the paper is organized as follows. In Section 2, we present the hybrid methodology with detailed implementations. Numerical results for Stokes interface problems are given in Sections 3, followed by some concluding remarks and future work in Section 4.

2. Hybrid Methodology for Stokes Interface Problems

From the jump conditions (1.5)-(1.6), it is clear to see that both the pressure and velocity of the Stokes equations are piecewise-smooth due to the presence of the interface Γ. The key idea of the present hybrid approach is to decompose the solution variables into two distinct parts; namely, the singular and regular parts as

$$p(\mathbf{x}) = p_r(\mathbf{x}) + p_s(\mathbf{x}), \tag{2.1}$$

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_r(\mathbf{x}) + \mathbf{u}_s(\mathbf{x}), \tag{2.2}$$

where the subscript r and s denote the regular (smooth) and singular (non-smooth) component, respectively. More precisely, we hope that the regular part solutions p_r and $\mathbf{u}_r$ are fairly smooth over the entire fluid domain Ω so no jumps occur across the interface. That is, we need to impose $[\![p_r]\!] = 0$ and $[\![\mathbf{u}_r]\!] = \mu[\![\partial\mathbf{u}_r/\partial\mathbf{n}]\!] = \mathbf{0}$ across the interface. On the other hand, the singular part solution variables p_s and $\mathbf{u}_s$ are responsible for having all discontinuous contributions. Thus we can easily derive from the jump conditions (1.5)-(1.6) and the decompositions (2.1)-(2.2) that

$$[\![p_s(\mathbf{X})]\!] = F_n(\mathbf{X}), \qquad \mathbf{X} \in \Gamma, \tag{2.3}$$

$$[\![\mathbf{u}_s(\mathbf{X})]\!] = \mathbf{0}, \quad \mu\left[\!\!\left[\frac{\partial \mathbf{u}_s}{\partial \mathbf{n}}(\mathbf{X})\right]\!\!\right] = -\big(\mathbf{F}(\mathbf{X}) - F_n(\mathbf{X})\mathbf{n}(\mathbf{X})\big) \equiv -\mathbf{F}_\tau(\mathbf{X}), \quad \mathbf{X} \in \Gamma. \tag{2.4}$$

For the sake of brevity, we use the term $\mathbf{F}_\tau$ to represent the tangential force. In the following sections, we shall explain how to obtain singular and regular solutions in a sequential manner using our proposed hybrid methodology.

2.1. Singular part solution

The singular part solution is found by a supervised learning for neural network function approximations as follow. As mentioned earlier, both p_s and the derivatives of $\mathbf{u}_s$ should be discontinuous across the interface. To this end, we formally define the singular part solution in a piecewise manner as

$$p_s(\mathbf{x}) = \begin{cases} \mathscr{P}(\mathbf{x}), & \text{if} \quad \mathbf{x} \in \Omega^-, \\ 0, & \text{if} \quad \mathbf{x} \in \Omega^+, \end{cases} \quad \text{and} \quad \mathbf{u}_s(\mathbf{x}) = \begin{cases} \mathscr{U}(\mathbf{x}), & \text{if} \quad \mathbf{x} \in \Omega^-, \\ \mathbf{0}, & \text{if} \quad \mathbf{x} \in \Omega^+, \end{cases} \tag{2.5}$$

where $\mathscr{P}$ and $\mathscr{U}$ are neural network functions to be found. Plugging the above expression into the jump conditions (2.3)-(2.4), we immediately derive that the unknown function $\mathscr{P}$ and $\mathscr{U}$ must fulfill the following constraints along the interface:

$$\mathscr{P}(\mathbf{X}) = -F_n(\mathbf{X}), \qquad \mathbf{X} \in \Gamma, \tag{2.6}$$

$$\mathscr{U}(\mathbf{X}) = \mathbf{0}, \quad \mu\frac{\partial \mathscr{U}(\mathbf{X})}{\partial \mathbf{n}} = \mathbf{F}_\tau(\mathbf{X}), \quad -\nabla\mathscr{P}(\mathbf{X}) + \mu\Delta\mathscr{U}(\mathbf{X}) - [\![\mathbf{g}(\mathbf{X})]\!] = \mathbf{0}, \quad \mathbf{X} \in \Gamma. \tag{2.7}$$

Note that, the third constraint of Eq. (2.7) is requested to have sufficient smoothness for the regular part solution (see Eq. (2.10)). We shall examine this additional condition later in the next subsection.

In this paper, we utilize the standard supervised learning methodology to find the above unknown functions $\mathscr{P}$ and $\mathscr{U}$ representing by the corresponding neural networks. We must point out that, it is not necessary to train a network model consisting of a $(\mathscr{P}, \mathscr{U})$ pair to fulfill Eqs. (2.6) and (2.7) simultaneously, although it seems that both of them are coupled together due to the third constraint of Eq. (2.7). In fact, such $\mathscr{P}$ and $\mathscr{U}$ are not unique in the sense that there exists infinitely many functions defined in the whole domain Ω that satisfy the restrictions (2.6) and (2.7) along Γ. This observation provides us some kind of freedom to determine $\mathscr{P}$ and $\mathscr{U}$ separately. Thus, we can first learn $\mathscr{P}$ to fulfill Eq. (2.6), and then use the obtained $\mathscr{P}$ to learn $\mathscr{U}$ through Eq. (2.7) accordingly.

Let us illustrate the full training procedure in details. Firstly, we determine $\mathscr{P}$ to fulfill the restriction (2.6) via a mean squared error (MSE) loss model. That is, given a dataset with M training points $\{\mathbf{X}^i \in \Gamma\}_{i=1}^{M}$, the corresponding loss function is defined by the MSE of the condition (2.6) as

$$\text{Loss}_{\mathscr{P}}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M}\left(\mathscr{P}(\mathbf{X}^i;\boldsymbol{\theta}) + F_n(\mathbf{X}^i)\right)^2, \tag{2.8}$$

where $\boldsymbol{\theta}$ collects all trainable parameters (weights and biases) of $\mathscr{P}$. Once $\mathscr{P}$ is trained, we learn the vector function $\mathscr{U}$ that satisfies the constraints in Eq. (2.7). (Notice that $\mathscr{U} = (\mathscr{U}_1, \mathscr{U}_2)$ for $d = 2$ and $\mathscr{U} = (\mathscr{U}_1, \mathscr{U}_2, \mathscr{U}_3)$ for $d = 3$ case.) One could build a multi-output network to represent $\mathscr{U}$, however, training the network is challenging because more terms are used in the whole loss function. Instead, here, we construct independent single-output networks to learn each component of $\mathscr{U}$. This step allows us to obtain all $\mathscr{U}_j$ using parallel computations since all of them are decoupled (see Eq. (2.7)). Using the same training data points, we train each $\mathscr{U}_j$ again by the MSE of Eq. (2.7) as

$$\begin{aligned}\text{Loss}_{\mathscr{U}_j}(\boldsymbol{\theta}_j) = \frac{1}{M}\sum_{i=1}^{M}\Bigg[&\left(\mathscr{U}_j(\mathbf{X}^i;\boldsymbol{\theta}_j)\right)^2 + \left(\mu\frac{\partial \mathscr{U}_j}{\partial \mathbf{n}}(\mathbf{X}^i;\boldsymbol{\theta}_j) - F_{\tau_j}(\mathbf{X}^i)\right)^2 \\ &+\left(-\frac{\partial \mathscr{P}}{\partial x_j}(\mathbf{X}^i;\theta) + \mu\Delta\mathscr{U}_j(\mathbf{X}^i;\theta_j) - [\![g_j(\mathbf{X}^i)]\!]\right)^2\Bigg],\end{aligned} \tag{2.9}$$

where $\boldsymbol{\theta}_j$ denotes the trainable parameters of $\mathscr{U}_j$, F_{τ_j} and g_j represent the j-th element of $\mathbf{F}_\tau$ and $\mathbf{g}$, respectively. One should note that, although we train the above network functions using only training data points along Γ, the trained functions are defined in the whole domain thanks to the expressive power of the neural networks. The spatial partial derivatives of the network functions appearing in the above loss models can be directly evaluated using automatic differentiation. We employ the Levenberg-Marquardt method [15], a particularly designed efficient optimizer for nonlinear least-squares problems, to minimize the loss models (2.8) and (2.9). With both $\mathscr{P}$ and $\mathscr{U}$ (thus the singular part solution p_s and $\mathbf{u}_s$) in hand, we are ready to find the regular part solution in the following subsection.

2.2. Regular part solution

Once we obtained the singular part solution via the neural network learning machinery, our next step is to solve the regular parts for p_r and $\mathbf{u}_r$. Substituting the decompositions

(2.1) and (2.2) into the Stokes equations (1.2)-(1.4), we immediately find that the regular part solution must satisfy the following Stokes-like system:

$$\begin{aligned}
&-\nabla p_r(\mathbf{x})+\mu\Delta\mathbf{u}_r(\mathbf{x})=\nabla p_s(\mathbf{x})-\mu\Delta\mathbf{u}_s(\mathbf{x})-\mathbf{g}(\mathbf{x}), && \mathbf{x}\in\Omega, && (2.10)\\
&\nabla\cdot\mathbf{u}_r(\mathbf{x})=-\nabla\cdot\mathbf{u}_s(\mathbf{x}), && \mathbf{x}\in\Omega, && (2.11)\\
&\mathbf{u}_r(\mathbf{x})=\mathbf{u}_b(\mathbf{x}), && \mathbf{x}\in\partial\Omega, && (2.12)
\end{aligned}$$

where the boundary condition for $\mathbf{u}_r$ (2.12) is straightforwardly derived using Eq. (1.4) and the definition of $\mathbf{u}_s$ in Eq. (2.5). Based on the above system, it is apparent to see that the solution pair $(\mathbf{u}_r,p_r)$ satisfies the Stokes-like equations but the regular velocity is not divergence-free. Since we do not require the singular part solution to be divergence-free, the regular part solution must be compromised so that the resulting velocity is divergence-free. In order to obtain smooth pressure and velocity for the regular part, the right-hand side of Eq. (2.10) must exhibit zero jump discontinuity across the interface which can be easily seen from the third constraint in Eq. (2.7). Meanwhile, the right-hand side of Eq. (2.11) also has zero jump across the interface since $\nabla\cdot\mathcal{U}=0$ on Γ. To see this, one can write the divergence of $\mathcal{U}$ as

$$\nabla\cdot\mathcal{U}=\nabla_s\cdot\mathcal{U}+\frac{\partial\mathcal{U}}{\partial\mathbf{n}}\cdot\mathbf{n},$$

where the notation ∇_s represents the surface gradient operator involving the derivatives along the surface Γ. From the first constraint of $\mathcal{U}=0$ on Γ in Eq. (2.7), we can conclude that all surface derivatives are zero leading to $\nabla_s\cdot\mathcal{U}=0$. And, from the second constraint in Eq. (2.7), the normal derivative of $\mathcal{U}$ has only tangential components leading to $(\partial\mathcal{U}/\partial\mathbf{n})\cdot\mathbf{n}=0$ as well. Therefore, we have $\nabla\cdot\mathcal{U}=0$ on the interface.

Now, the regular system (2.10)-(2.12) can be discretized by the well-known MAC scheme [7] and solved by the Uzawa-type algorithm [20]. More precisely, the resultant linear system is a saddle point problem as

$$\begin{bmatrix} L & G\\ G^\top & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{u}_r\\ p_r\end{bmatrix}=\begin{bmatrix}\nabla p_s-\mu\Delta\mathbf{u}_s-\mathbf{g}\\ -\nabla\cdot\mathbf{u}_s\end{bmatrix}+\begin{bmatrix}\mathbf{f}(\mathbf{u}_b)\\ h(\mathbf{u}_b)\end{bmatrix}, \tag{2.13}$$

where L denotes the usual five-point discrete Laplacian operator $\mu\Delta_h$ applying to $\mathbf{u}_r$, G is the central difference gradient operator $-\nabla_h$ to p_r, and $G^\top$ is the central difference divergence operator $\nabla_h\cdot$ to $\mathbf{u}_r$. Note that, those difference operators are all performed on the MAC grid. In the right-hand side of the matrix equation (2.13), the first term involving the computation of derivatives of the singular part solution at those MAC grid (only inside the interface needed due to the definition (2.5)) can be done by automatic differentiation [5]. For instance, the term $\Delta\mathbf{u}_s$ can be computed as follows. Once $\mathcal{U}$ is learned, the singular part solution $\mathbf{u}_s$ is obtained. By the definition of $\mathbf{u}_s$, we thus have $\Delta\mathbf{u}_s(\mathbf{x})=\mathbf{0}$, $\mathbf{x}\in\Omega^+$, and $\Delta\mathbf{u}_s(\mathbf{x})=\Delta\mathcal{U}(\mathbf{x})$, $\mathbf{x}\in\Omega^-$, where $\Delta\mathcal{U}$ can be calculated directly by automatic differentiation. The second term is associated with boundary condition of $\mathbf{u}_r$ induced by the discretization operators L and $G^\top$. Here, we use the linear interpolation to approximate the velocity $\mathbf{u}_r$ at the ghost points near the boundary. Notice that, the MAC scheme has the advantage that no boundary condition for the regular pressure p_r is needed.

Rewriting the above linear system as

$$\begin{bmatrix} L & G \\ G^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_r \\ p_r \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

we can find its solution by the Schur complement technique [10]. Using the block LU decomposition, one can further rewrite the system as

$$\begin{bmatrix} L & \mathbf{0} \\ G^\top & -G^\top L^{-1} G \end{bmatrix} \begin{bmatrix} I & L^{-1}G \\ \mathbf{0} & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_r \\ p_r \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{2.14}$$

where I is the identity matrix. Then, we can solve the above linear system (2.14) via the following fractional steps by first introducing the intermediate velocity $\mathbf{u}^* = \mathbf{u}_r + (L^{-1}G)p_r$:

1. Solve $L\mathbf{u}^* = b_1$ to obtain the intermediate velocity field $\mathbf{u}^*$. Notice that, this comprises a fully decoupled linear system of Poisson equation for each component of $\mathbf{u}^*$, so we need to apply two ($d = 2$) or three ($d = 3$) fast Poisson solvers that can be efficiently done using the public software package such as Fishpack [1] or Fast Fourier Transform (FFT).

2. Solve $(-G^\top L^{-1}G)p_r = \tilde{b}_2 - G^\top \mathbf{u}^*$ using the conjugate gradient method. In each iteration step, a matrix-vector multiplication $(-G^\top L^{-1}G)\psi$ for some vector ψ is required; the inversion of L^{-1} again can be performed efficiently by the fast Poisson solvers mentioned in Step 1. We must emphasize that, since the kernel of the underlying matrix $-G^\top L^{-1}G$ is obviously spanned by the constant vector $\mathbf{1} = [1, 1, \cdots, 1]^\top$, here we use the modified right-hand side vector $\tilde{b}_2 = b_2 - \mathbf{1}^\top b_2/\|\mathbf{1}\|^2$ instead of the original b_2. That is, we project b_2 into the solution space to guarantee the existence of the solution p_r. The iteration stops whenever the system residual is less than 10^{-12}.

3. Compute the regular part velocity $\mathbf{u}_r = \mathbf{u}^* - (L^{-1}G)p_r$. Again, this involves the usage of the fast Poisson solvers.

As a conclusion for the computational complexity, the overall cost in the above three steps can be counted in terms of the number of fast Poisson solvers being applied.

2.3. Summary of the hybrid method

Let us summarize the step by step procedure of the proposed hybrid method.

1. Use neural network learning machinery to construct $\mathscr{P}$ and $\mathscr{U}$, and form the singular part solution p_s and $\mathbf{u}_s$.

2. Apply the MAC scheme to discretize the Stokes-like system and use the Uzawa-type algorithm to find the regular part solution for the velocity $\mathbf{u}_r$ and pressure p_r.

3. Add the singular and regular parts of solution to obtain the desired velocity $\mathbf{u}$ and pressure p to the Stokes interface problem.

Let us conclude this section by remarking several features of the present method in the following. The proposed method builds upon the hybrid method introduced in [9] for Poisson interface problem and extends to solve the Stokes interface problem. In contrast to traditional sharp interface methods such as in [13], the present method avoids the effort of careful constructing additional correction terms near the interface grid points to achieve desired accuracy. We utilize a supervised learning technique for the singular part solution and traditional finite difference scheme for the regular part solution. Once the network solution $\mathscr{P}$ and $\mathscr{U}$ have been trained, they can be utilized to solve the problem for various grid sizes. This allows adapting the method to different resolutions flexibly. Another notable advantage is the ability to leverage well-established fast Poisson solvers, which facilitates the efficient solution of linear systems arising in this method. This efficiency is crucial for practical implementation and enables handling larger and more complex problems. Furthermore, it is straightforward to extend the proposed method to multiple-interface problems. We shall also point out that the numerical error of the present method is induced by the neural network approximations (optimization and approximation errors) and finite difference discretizations (truncation error). Despite that, we can observe the numerical convergence behavior in our results shown in the next section which is hardly seen in complete neural network approaches.

3. Numerical Results

In this section, we perform a series of numerical tests to validate the efficiency and convergence of the proposed method. For all examples, we fix the constant viscosity by $\mu = 1$. We use the sigmoid function as the activation function for the neural networks to approximate $\mathscr{P}$ and $\mathscr{U}$, and stop the network training once the loss value reaches the threshold 10^{-10}. All the problems considered here are defined on regular domains with a uniform layout of N^d grid points. The numerical experiments were conducted on a PC equipped with an Intel Core i7-13700 CPU and 128GB RAM which gives sufficient computational resources for all the tests.

Example 3.1. The first example, referred from the 2D IIM Stokes solver in [3], aims to validate the convergence property of the present hybrid method. The domain is set as $\Omega = [-2,2]^2$ with a unit circular interface, $\Gamma = \{\mathbf{X}(\theta) = (\cos\theta, \sin\theta) \,|\, \theta \in [0,2\pi)\}$ immersed in the domain. The interfacial force consists of both tangential and normal components as $\mathbf{F}(\mathbf{X}(\theta)) = 2\sin(3\theta)\boldsymbol{\tau}(\theta) - \cos^3(\theta)\mathbf{n}(\theta)$ in which the tangent vector can be calculated by $\boldsymbol{\tau}(\theta) = \mathbf{X}'(\theta) = (-\sin\theta, \cos\theta)$ and the normal vector thus is $\mathbf{n}(\theta) = (\cos\theta, \sin\theta)$.

The exact velocity $\mathbf{u} = (u_1, u_2)$ written in polar coordinates is chosen by

$$u_1(r,\theta) = \begin{cases} \dfrac{1}{8}r^2\cos(2\theta) + \dfrac{1}{16}r^4\cos(4\theta) - \dfrac{1}{4}r^4\cos(2\theta), & \text{if} \quad r < 1, \\ -\dfrac{1}{8}r^{-2}\cos(2\theta) + \dfrac{5}{16}r^{-4}\cos(4\theta) - \dfrac{1}{4}r^{-2}\cos(4\theta), & \text{if} \quad r \geq 1, \end{cases}$$

$$u_2(r,\theta)=\begin{cases}-\dfrac{1}{8}r^2\sin(2\theta)+\dfrac{1}{16}r^4\sin(4\theta)+\dfrac{1}{4}r^4\sin(2\theta), & \text{if} \quad r<1,\\ \dfrac{1}{8}r^{-2}\sin(2\theta)+\dfrac{5}{16}r^{-4}\sin(4\theta)-\dfrac{1}{4}r^{-2}\sin(4\theta), & \text{if} \quad r\geq 1,\end{cases}$$

while the exact pressure written in Cartesian coordinates $(x=r\cos\theta, y=r\sin\theta)$ is given as

$$p(x,y)=\begin{cases}x^3+\cos(\pi x)\cos(\pi y), & \text{if} \quad r<1,\\ \cos(\pi x)\cos(\pi y), & \text{if} \quad r\geq 1.\end{cases}$$

With the above exact velocity and pressure, we can obtain the external force $\mathbf{g}=(g_1,g_2)$ in Cartesian coordinates accordingly as

$$g_1(x,y)=\begin{cases}-\pi\sin(\pi x)\cos(\pi y)+6x^2-3y^2, & \text{if} \quad r<1,\\ -\pi\sin(\pi x)\cos(\pi y)-\dfrac{3(x^4-6x^2y^2+y^4)}{(x^2+y^2)^4}, & \text{if} \quad r\geq 1,\end{cases}$$

$$g_2(x,y)=\begin{cases}-\pi\sin(\pi x)\cos(\pi y)-6xy, & \text{if} \quad r<1,\\ -\pi\sin(\pi x)\cos(\pi y)-\dfrac{12(x^3y-xy^3)}{(x^2+y^2)^4}, & \text{if} \quad r\geq 1.\end{cases}$$

As discussed earlier, we first learn the neural network functions $\mathscr{P}$ and $\mathscr{U}$ using the losses of Eq. (2.8) and (2.9), respectively. Then we use the learned functions to obtain the singular part solution for all various grid sizes. Theoretically, one might want to choose the number of training points on the interface M to be linearly scaled with the grid number N accordingly so that mesh resolutions on the interface and the domain can be consistent. However, for the sake of efficiency, we simply choose the number M to have a successful training (the loss value reaches the threshold 10^{-10}). Here, we set the number $M=400$ based on the grid size $N=256$ so that mesh resolutions on the interface and the domain are almost the same. We train three independent shallow (one-hidden-layer) neural networks (one for $\mathscr{P}$ and the other two for $\mathscr{U}=(\mathscr{U}_1,\mathscr{U}_2)$) with 50 neurons in the hidden layer and $M=400$ randomly sampled training points along the interface Γ. The training results are shown in Table 1. Within a few hundred epochs, the training losses decrease to below 10^{-10}, and the training time take less than three seconds.

In Table 2, we report the L^∞-errors as well as the corresponding rates of convergence of the numerical velocity $\mathbf{u}=\mathbf{u}_r+\mathbf{u}_s=(u_1,u_2)$ and pressure $p=p_r+p_s$. As seen clearly, all the fluid variables achieve approximately second-order convergence, and the magnitudes

Table 1: Elapsed time for training the singular part solution in Example 3.1.

	training time (s)	final training loss	epochs
$\mathscr{P}$	0.13	8.949e-11	54
$\mathscr{U}_1$	2.63	9.878e-11	328
$\mathscr{U}_2$	2.94	3.920e-11	353

Table 2: Mesh refinement results of the 2D Stokes interface problem on N^2 grid points in Example 3.1.

N	$e_\infty(u_1)$	rate	$e_\infty(u_2)$	rate	$e_\infty(p)$	rate	$e_\infty(\nabla\cdot\mathbf{u})$
32	2.714e-02	-	2.062e-02	-	9.804e-02	-	3.742e-05
64	6.108e-03	2.15	4.685e-03	2.14	3.488e-02	1.49	1.209e-06
128	1.553e-03	1.98	1.256e-03	1.90	8.477e-03	2.04	3.580e-07
256	3.578e-04	2.12	2.870e-04	2.13	2.418e-03	1.81	5.664e-07
512	8.952e-05	2.00	7.571e-05	1.92	6.015e-04	2.01	5.866e-07
1024	2.116e-05	2.08	2.140e-05	1.82	1.547e-04	1.96	5.899e-07

of error are comparable with the ones obtained by IIM [12]. Although this particular choice of number $M=400$ is based on the grid size used $N=256$, we further run the tests for the finer cases $N=512,1024$, and the results keep the second-order convergence. It seems that the network solution for the singular part is sufficiently accurate so it does not pollute the overall accuracy. In other words, the rate of convergence is completely determined by the computation of the regular part solution in this example.

Furthermore, we also check the divergence-free condition by computing the L^∞-error of $\nabla\cdot\mathbf{u}$, denoting by $e_\infty(\nabla\cdot\mathbf{u}) = \|\nabla_h\cdot\mathbf{u}_r + \nabla\cdot\mathbf{u}_s\|_\infty$. Here, the regular part divergence is calculated by applying the usual finite difference scheme to $\mathbf{u}_r$ on MAC grid while the singular part divergence is calculated directly by applying the automatic differentiation to $\mathbf{u}_s$ at the same positions as the regular part. (In fact, we only need to compute the singular part divergence $\nabla\cdot\mathbf{u}_s$ in the sub-domain Ω^- since it is zero in Ω^+ by definition.) One can see that the L^∞-error of $\nabla\cdot\mathbf{u}$ can be reached up to the order of magnitude 10^{-7} after the grid point $N=128$ is used. Thus, the present methodology preserves the divergence-free condition quite well.

Example 3.2. This example will showcase the reliability of our hybrid method in the scenario when the exact solution is unavailable. In this test, an elliptical interface $\Gamma = \{\mathbf{X}(\theta) = (0.5\cos\theta, 0.3\sin\theta)\,|\,\theta\in[0,2\pi)\}$ is immersed in the square domain $\Omega=[-1,1]^2$ while the interfacial force comprises a curvature normal force and tangential force as $\mathbf{F}(\mathbf{X}(\theta)) = 0.1\kappa(\theta)\mathbf{n}(\theta) - 0.1\tau(\theta)$, where $\kappa(\theta)$ denotes the local curvature at $\mathbf{X}(\theta)$. The fluid boundary condition $\mathbf{u}_b$ is obtained by using the spectrally accurate boundary integral method [19] with free-space Stokeslets kernel — i.e. $\mathbf{u}(\mathbf{x})\to\mathbf{0}$ as $\|\mathbf{x}\|\to\infty$.

We follow the same setup for the network training procedure as in Example 3.1; namely, we train three independent shallow neural networks with 50 neurons in the hidden layer and 400 randomly sampled training points along the interface Γ. The training results are

Table 3: Elapsed time for training the singular part solution in Example 3.2.

	training time(s)	final training loss	epochs
$\mathscr{P}$	0.28	4.870e-11	150
$\mathscr{U}_1$	7.64	8.742e-11	878
$\mathscr{U}_2$	7.86	8.118e-11	906

shown in Table 3. In fewer than 1000 epochs, the training losses decrease to below 10^{-10}, and the training process takes less than eight seconds.

To measure the convergence of the solutions, since the explicit solution is not available, we compute the error in a successive manner with $\|\phi_N - \phi_{2N}\|_\infty$, where ϕ_N denotes some fluid variable solution with the grid number N. It is worth mentioning that the interpolation on the MAC grid layout is not straightforward since the solutions at different resolutions would not coincide at the same location. In the present method, we simply apply the linear interpolation to the regular part solution and use neural network approximation for the singular part solution at target grid points.

The mesh refinement results are shown in Table 4. When increasing the grid number, all numerical solutions converge with roughly second-order accuracy. In addition, we compute the interfacial velocity (u_1^Γ, u_2^Γ) at the discrete marker points $\{(0.5\cos(\theta_k), 0.3\sin(\theta_k)) | \theta_k = 2k\pi/64, k = 1,2,\cdots,64\}$. Again, this is done by the interpolation technique aforementioned. Referring the exact solution obtained by the boundary integral method with spectral accuracy [19], Table 5 indicates that the numerical interfacial velocity obtained by the present method also attains second-order convergence.

Lastly, to illustrate the ability of our method in capturing the derivative discontinuities for the velocity field and the jump discontinuity for the pressure, we show the cross-sectional views of the numerical velocity field and pressure in Fig. 1. It can be clearly seen that the cusp (derivative discontinuity) for the velocity and jump for the pressure occur at the interface.

Table 4: Mesh refinement results of the 2D Stokes interface problem on N^2 grid points in Example 3.2.

N	$e_\infty(u_1)$	rate	$e_\infty(u_2)$	rate	$e_\infty(p)$	rate
32	1.638e-03	-	1.002e-03	-	2.267e-02	-
64	4.465e-04	1.87	2.579e-04	1.96	6.514e-03	1.80
128	1.071e-04	2.06	6.371e-05	2.02	1.635e-03	1.99
256	2.698e-05	1.99	1.604e-05	1.99	5.843e-04	1.48

Table 5: Mesh refinement results of the interfacial velocity (u_1^Γ, u_2^Γ) in Example 3.2. We use the results of boundary integral method [19] as the exact solution and compute the errors.

N	$e_\infty(u_1^\Gamma)$	rate	$e_\infty(u_2^\Gamma)$	rate
32	4.507e-04	-	6.773e-04	-
64	9.893e-05	2.19	1.352e-04	2.33
128	2.666e-05	1.89	2.117e-05	2.67
256	6.565e-06	2.02	4.518e-06	2.23

Example 3.3. Now, we proceed to solve the test example referred from [24] for three-dimensional Stokes interface problem in the cubic domain $\Omega = [-2,2]^3$. The interface Γ is simply described by the unit sphere $\Gamma = \{\mathbf{X} = (x,y,z) \mid x^2 + y^2 + z^2 = 1\}$. Here, we choose the exact velocity $\mathbf{u} = (u_1, u_2, u_3)$ as

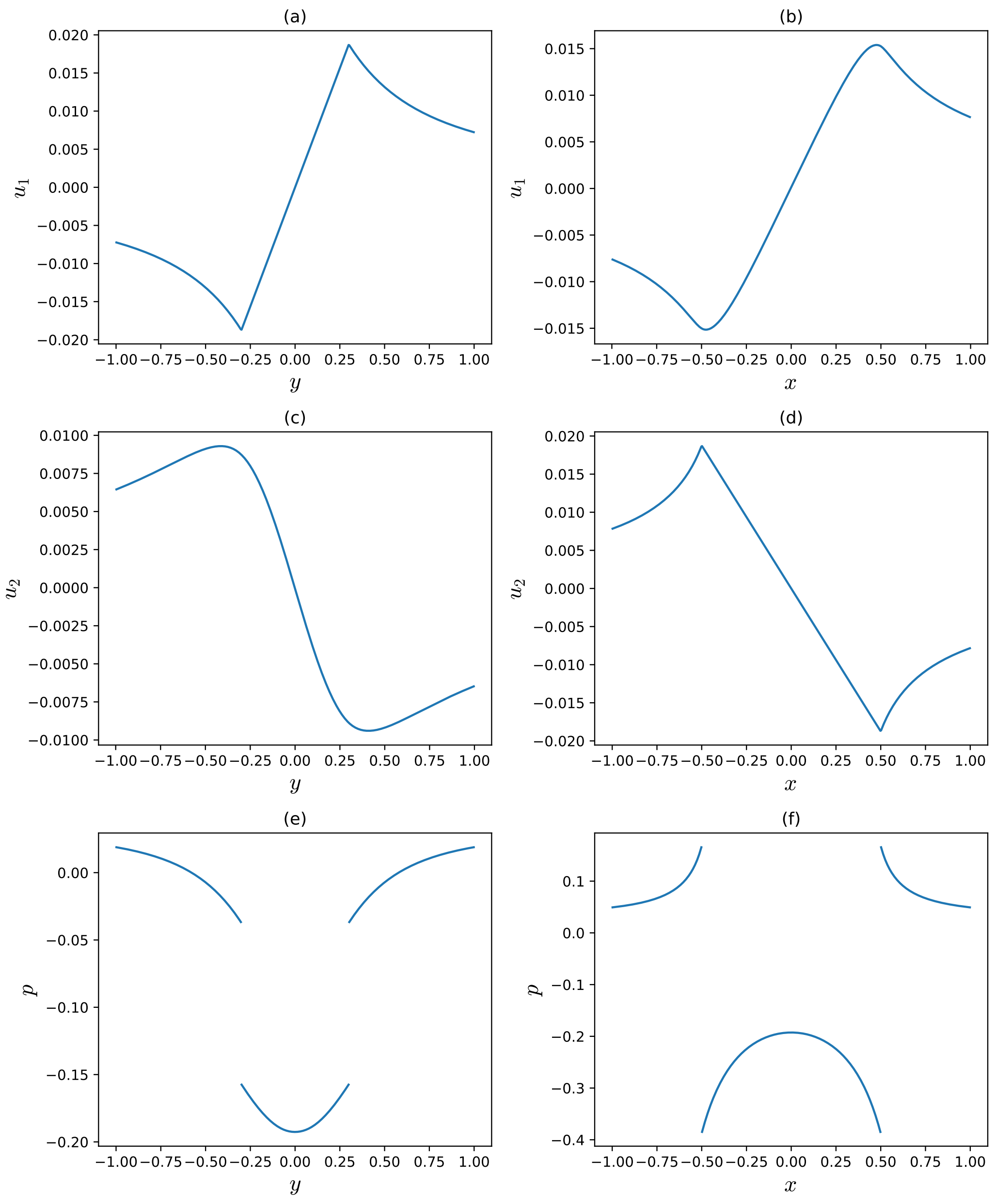

Figure 1: The cross-sectional plots of the velocity field and pressure along the grid lines in Example 3.2. The mesh width is $h=2/N$ with $N=512$. (a)u_1 along the grid line $x=0$; (b) u_1 along the grid line $y=h/2$; (c) u_2 along the grid line $y=0$; (d) u_2 along the grid line $x=h/2$; (e) p along the grid line $x=h/2$; (f) p along the grid line $y=h/2$.

$$u_1(x,y,z)=\begin{cases}\dfrac{1}{4}yz, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ \dfrac{1}{4}yz(x^2+y^2+z^2), & \text{if} \quad \mathbf{x}\in\Omega^+,\end{cases}$$

$$u_2(x,y,z)=\begin{cases}\dfrac{1}{4}xz, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ \dfrac{1}{4}xz(x^2+y^2+z^2), & \text{if} \quad \mathbf{x}\in\Omega^+,\end{cases}$$

$$u_3(x,y,z)=\begin{cases}-\dfrac{1}{2}xy(1-x^2-y^2), & \text{if} \quad \mathbf{x}\in\Omega^-,\\ -\dfrac{1}{2}xyz^2, & \text{if} \quad \mathbf{x}\in\Omega^+,\end{cases}$$

and the exact pressure p

$$p(x,y,z)=\begin{cases}\left(-\dfrac{3}{4}x^3+\dfrac{3}{8}x\right)yz, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ 0, & \text{if} \quad \mathbf{x}\in\Omega^+.\end{cases}$$

One can accordingly obtain the external force field $\mathbf{g}=(g_1,g_2,g_3)$ given by

$$g_1(x,y,z)=\begin{cases}\left(-\dfrac{9}{4}x^2+\dfrac{3}{8}\right)yz, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ -\dfrac{7}{2}yz, & \text{if} \quad \mathbf{x}\in\Omega^+,\end{cases}$$

$$g_2(x,y,z)=\begin{cases}\left(-\dfrac{3}{4}x^3+\dfrac{3}{8}x\right)z, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ -\dfrac{7}{2}xz, & \text{if} \quad \mathbf{x}\in\Omega^+,\end{cases}$$

$$g_3(x,y,z)=\begin{cases}\left(-\dfrac{3}{4}x^3-\dfrac{45}{8}x\right)y, & \text{if} \quad \mathbf{x}\in\Omega^-,\\ xy, & \text{if} \quad \mathbf{x}\in\Omega^+.\end{cases}$$

The interfacial force can be derived from the exact velocity and pressure given above. Also, the boundary conditions are given by the exact velocity.

We use a one-hidden-layer neural network with 50 neurons for the pressure and one-hidden-layer neural network with 100 neurons for each component of the velocity to train the networks. The training data is generated by choosing 1000 random points on the sphere. The training results are shown in Table 6. Again, the pressure only takes less than one second to train, while each velocity component takes about 10-20 seconds.

In Table 7, we show the L^∞-errors and rates of convergence of the numerical solutions. The velocity shows clear second-order convergence, while the pressure is reduced to first-order only. Although not shown here, the largest numerical errors for the computed pressure occur at the corners of the cube. We believe that employing the quadratic interpolation to approximate the velocity $\mathbf{u}_r$ at the ghost points near the boundary could help reduce this error.

Table 6: Elapsed time for training the singular part solution in Example 3.3.

	training time(s)	final training loss	epochs
$\mathscr{P}$	0.43	7.347e-11	181
$\mathscr{U}_1$	13.02	8.118e-11	803
$\mathscr{U}_2$	12.07	6.640e-11	764
$\mathscr{U}_3$	19.06	8.268e-11	1246

Table 7: Mesh refinement results of the 3D Stokes interface problem in Example 3.3.

N	$e_\infty(u_1)$	rate	$e_\infty(u_2)$	rate	$e_\infty(u_3)$	rate	$e_\infty(p)$	rate
16	3.626e-02	-	3.626e-02	-	2.258e-02	-	1.283e+00	-
32	1.041e-02	1.80	1.041e-02	1.80	6.534e-03	1.79	7.553e-01	0.76
64	2.768e-03	1.91	2.768e-03	1.91	1.773e-03	1.88	4.137e-01	0.87
128	7.123e-04	1.96	7.123e-04	1.96	4.653e-04	1.93	2.177e-01	0.93

Example 3.4. In this example, we solve a three-dimensional Stokes interface problem with an oblate spheroid as the interface. The domain is a cube $\Omega = [-1,1]^3$. The interface is given by

$$\Gamma = \{\mathbf{X} = (x,y,z) \mid x^2/0.5^2 + y^2/0.5^2 + z^2/0.3^2 = 1\}$$

that can be parameterized by $\mathbf{X}(\theta,\varphi) = (0.5\cos\varphi\cos\theta, 0.5\cos\varphi\sin\theta, 0.3\sin\varphi)$, where $\theta \in [0,2\pi)$ and $\varphi \in [-\pi/2,\pi/2]$. Here, we simply take the interfacial force as the surface tension force (with the constant surface tension to be one) as $\mathbf{F} = -2H\mathbf{n}$, where H is the mean curvature and $\mathbf{n}$ is the unit normal vector. The no-slip boundary condition $\mathbf{u}_b(\mathbf{x}) = \mathbf{0}$ is applied along the boundary domain $\partial\Omega$ and the zero external force field $\mathbf{g} = \mathbf{0}$ is assumed.

We use a one-hidden-layer neural network with 100 neurons for the pressure and two-hidden-layer neural network with 30 neurons for each component of the velocity to train the networks. The training points are generated by choosing 1000 random points on the interface. The training results are shown in Table 8.

As in Example 3.2, the explicit formulas for the solution are not available so we compute the errors in the same manner as there. In Table 9, we see that the numerical results show second-order convergence in the velocity, while first-order convergence in the pressure. We further depict the cross-sectional view of the flow quiver (u_1,u_3) along the plane $y = 2/h$ in Fig. 2. One can see that, the flow tends to reduce the absolute magnitudes of the

Table 8: Elapsed time for training the singular part solution in Example 3.4.

	training time(s)	final training loss	epochs
$\mathscr{P}$	10.097	1.850e-16	3000
$\mathscr{U}_1$	121.154	8.999e-13	2519
$\mathscr{U}_2$	142.336	2.346e-12	3000
$\mathscr{U}_3$	144.442	1.399e-12	3000

Table 9: Mesh refinement results of the 3D Stokes interface problem in Example 3.4.

N	$e_\infty(u_1)$	rate	$e_\infty(u_2)$	rate	$e_\infty(u_3)$	rate	$e_\infty(p)$	rate
32	7.694e-03	-	5.531e-03	-	1.131e-02	-	1.057e+00	-
64	2.096e-03	1.88	1.418e-03	1.96	3.169e-03	1.84	5.731e-01	0.88
128	5.429e-04	1.95	4.326e-04	1.71	8.681e-04	1.87	2.945e-01	0.96

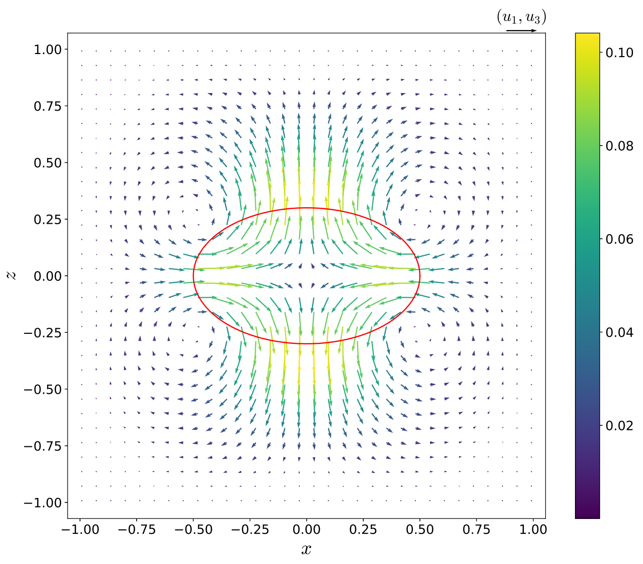

Figure 2: Quiver plot for (u_1,u_3) along the plane $y=h/2$ in Example 3.4. The mesh width is $h=2/N$ with $N=128$. The color indicates the absolute magnitude of the velocity field.

mean curvature along the interface and relaxes to a spherical shape. This instant flow tendency matches well with the simulations when the interface dynamics is considered. Meanwhile, since the fluid is incompressible, we can see that two vortex dipoles (or four counter-rotating vortices) occur near both sides. Therefore, our proposed hybrid method is indeed able to predict physically reasonable results.

4. Conclusion and Future Work

In this paper, we present a hybrid neural-network and finite-difference method for solving Stokes equations with singular forces on an interface in regular domains. Overall speaking, this hybrid method takes both advantages of neural network and finite difference scheme. Our approach is based on decomposing the solution into two parts: the singular part captures the non-smooth solution behavior while the regular part represents the smooth solution. To find the singular part, we make use of neural network function approximators to learn the solution variables that satisfy certain constraints along the interface. Once the singular part solution is obtained, the regular part can be solved via a Stokes-like system using traditional MAC scheme. The overall computational costs can be calculated in terms of the number of fast Poisson solvers used in the iterations. Through several numerical experiments, we demonstrate that the present hybrid method indeed obtains numerical results with traditional convergence property for both two- and three-dimensional Stokes interface problems. As a future work, we plan to apply the present method to solve time-dependent dynamic interface problems for flows in different applications, such as the vesicle hydrodynamics, electro-hydrodynamics, or phoretic systems.

Acknowledgements

W.-F. Hu, T.-S. Lin, and M.-C. Lai acknowledge the supports by National Science and Technology Council, Taiwan, under the research grants 111-2115-M-008-009-MY3, 111-2628-M-A49-008-MY4, and 110-2115-M-A49-011-MY3, respectively. W.-F. Hu and T.-S. Lin also acknowledge the supports by National Center for Theoretical Sciences, Taiwan.

References

[1] J. Adams, P. Swarztrauber and R. Sweet, *Fishpack – a package of fortran subprograms for the solution of separable elliptic partial differential equations* (1980), `https://ui.adsabs.harvard.edu/abs/2016ascl.soft09004A`

[2] K.-Y. Chen, K.-A. Feng, Y. Kim and M.-C. Lai, *A note on pressure accuracy in immersed boundary method for Stokes flow*, J. Comput. Phys. **230**, 4377–4383 (2011).

[3] R. Cortez, *The method of regularized Stokeslets*, SIAM J. Sci. Comput. **23**, 1204–1225 (2001).

[4] H. Dong, Z. Zhao, S. Li, W. Ying and J. Zhang, *Second order convergence of a modified MAC scheme for Stokes interface problems*, J. Sci. Comput. **96**, Paper No. 27 (2023).

[5] A. Griewank and A. Walther, *Evaluating derivatives: Principles and techniques of algorithmic differentiation*, SIAM (2008).

[6] H. Guo and X. Yang, *Deep unfitted Nitsche method for elliptic interface problems*, Commun. Comput. Phys. **31**, 1162–1179 (2022).

[7] F. Harlow and J. Welsh, *Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface*, Phys. Fluids **8**, 2181–2189 (1965).

[8] W.-F. Hu, T.-S. Lin and M.-C. Lai, *A discontinuity capturing shallow neural network for elliptic interface problems*, J. Comput. Phys. **469**, 111576 (2022).

[9] W.-F. Hu, T.-S. Lin, Y.-H. Tseng and M.-C. Lai, *An efficient neural-network and finite-difference hybrid method for elliptic interface problems with applications*, Commun. Comput. Phys. **33**, 1090–1105 (2023).

[10] M.-C. Lai, W.-F. Hu and W.-W. Lin, *A fractional step immersed boundary method for Stokes flow with an inextensible interface enclosing a solid particle*, SIAM J. Sci. Comput. **34**, B692–B710 (2012).

[11] M.-C. Lai and Z. Li, *A remark on jump conditions for the three-dimensional Navier-Stokes equations involving an immersed moving membrane*, Appl. Math. Lett. **14**, 149–154 (2001).

[12] M.-C. Lai and H.-C. Tseng, *A simple implementation of the immersed interface methods for Stokes flows with singular forces*, Comput. & Fluids **37**, 99–106 (2008).

[13] R. LeVeque and Z. Li, *Immersed interface method for Stokes flow with elastic boundaries or surface tension*, SIAM J. Sci. Comput. **18**, 709–735 (1997).

[14] Z. Li, K. Ito and M.-C. Lai, *An augmented approach for Stokes equations with discontinuous viscosity and singular forces*, Comput. & Fluids **36**, 622–635 (2007).

[15] D. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, SIAM J. Appl. Math. **11**, 431–441 (1963).

[16] Y. Mori, *Convergence proof of the velocity field for a Stokes flow immersed boundary method*, Commun. Pure Appl. Math. **LXI**, 1231–1263 (2008).

[17] C. Peskin, *The immersed boundary method*, Acta Numer. **11**, 479–517 (2002).

[18] M. Raissi, P. Perdikaris and 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**, 686–707 (2019).

[19] X. Sun and X. Li, *A spectrally accurate boundary integral method for interfacial velocities in two-dimensional Stokes flow*, Commun. Comput. Phys. **8**, 933–946 (2010).

[20] E.Y. Tau, *Numerical solution of the steady Stokes equations*, J. Comput. Phys. **99**, 190–195 (1992).

[21] J. Teran and C. Peskin, *Tether force constraints in Stokes flow by the immersed boundary method on a periodic domain*, SIAM J. Sci. Comput. **31**, 3404–3416 (2009).

[22] Y.-H. Tseng and M.-C. Lai, *A discontinuity and cusp capturing PINN for Stokes interface problems with discontinuous viscosity and singular forces*, Ann. Appl. Math. **39**, 385–403 (2023).

[23] Y.-H. Tseng, T.-S. Lin, W.-F. Hu and M.-C. Lai, *A cusp-capturing PINN for elliptic interface problems*, J. Comput. Phys. **491**, 112359 (2023).

[24] W. Wang and Z. Tan, *A simple augmented IIM for 3D incompressible two-phase Stokes flows with interfaces and singular forces*, Comput. Phys. Commun. **270**, 108154 (2022).

[25] S. Wu, A. Zhu, Y. Tang and B. Lu, *Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems*, Commun. Comput. Phys. **33**, 596–627 (2023).