Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2022-0284

Vol. **33**, No. 4, pp. 1090-1105
April 2023

An Efficient Neural-Network and Finite-Difference Hybrid Method for Elliptic Interface Problems with Applications

Wei-Fan Hu[1,4], Te-Sheng Lin[2,4,*], Yu-Hau Tseng[3] and Ming-Chih Lai[2]

[1] *Department of Mathematics, National Central University, Taoyuan 32001, Taiwan.*
[2] *Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan.*
[3] *Department of Applied Mathematics, National University of Kaohsiung, Kaohsiung 81148, Taiwan.*
[4] *National Center for Theoretical Sciences, National Taiwan University, Taipei 10617, Taiwan.*

Received 6 November 2022; Accepted (in revised version) 4 March 2023

Abstract. A new and efficient neural-network and finite-difference hybrid method is developed for solving Poisson equation in a regular domain with jump discontinuities on embedded irregular interfaces. Since the solution has low regularity across the interface, when applying finite difference discretization to this problem, an additional treatment accounting for the jump discontinuities must be employed. Here, we aim to elevate such an extra effort to ease our implementation by machine learning methodology. 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 solution, while the standard five-point Laplacian discretization is used to obtain the regular solution with associated boundary conditions. Regardless of the interface geometry, these two tasks only require supervised learning for function approximation and a fast direct solver for Poisson equation, making the hybrid method easy to implement and efficient. The two- and three-dimensional numerical results show that the present hybrid method preserves second-order accuracy for the solution and its derivatives, and it is comparable with the traditional immersed interface method in the literature. As an application, we solve the Stokes equations with singular forces to demonstrate the robustness of the present method.

AMS subject classifications: 65N06, 65N99, 35J25

Key words: Neural networks, sharp interface method, fast direct solver, elliptic interface problem, Stokes equations.

*Corresponding author. *Email addresses:* wfhu@math.ncu.edu.tw (W.-F. Hu), tslin@math.nctu.edu.tw (T.-S. Lin), yhtseng@nuk.edu.tw (Y.-H. Tseng), mclai@math.nctu.edu.tw (M.-C. Lai)

1 Introduction

In this paper, we aim to solve a d-dimensional ($d=2$ or 3) elliptic interface problem defined in a regular domain $\Omega\subset\mathbb{R}^d$, which is separated by an embedded interface Γ such that the subdomains inside and outside the interface are denoted by Ω^- and Ω^+, respectively. Along the interface Γ, there exists jump discontinuities that the solution must be satisfied. With the associated boundary condition, the problem takes the form

$$\Delta u(\mathbf{x})=f(\mathbf{x}),\quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{1.1}$$

$$[\![u(\mathbf{x})]\!]=\gamma(\mathbf{x}),\quad [\![\partial_n u(\mathbf{x})]\!]=\rho(\mathbf{x}),\quad \mathbf{x}\in\Gamma, \tag{1.2}$$

$$u(\mathbf{x})=u_b(\mathbf{x}),\quad \mathbf{x}\in\partial\Omega. \tag{1.3}$$

Here, the jump $[\![\cdot]\!]$ indicates the quantity approaching from Ω^+ side minus the one from Ω^- side; the shorthand $\partial_n u$ represents the normal derivative $\nabla u\cdot\mathbf{n}$ in which $\mathbf{n}$ is the normal vector pointing from Ω^- to Ω^+. Notice that, here the underlying differential equation is subject to the Dirichlet-type boundary condition for illustration purpose, while other types of boundary condition (Neumann or Robin) will not change the main ingredients presented here. Since the Poisson equation is considered in Eq. (1.1), we simply call the above problem as the Poisson interface problem hereafter.

As seen from Eqs. (1.1)-(1.3), the solution and its partial derivatives have jumps across the interface. So, when applying the finite difference discretization to this problem, an additional treatment accounting for those jump discontinuities must be employed at the grid points near the interface. Over the past few decades, different discretization methodologies have been successfully developed to capture those jump conditions sharply or to improve the overall numerical accuracy, such as the immersed interface method (IIM) [12, 13, 16, 18], ghost fluid method (GFM) [6, 22], Voronoi interface method [7], to name a few. Different approaches for solving interface problems such as the immersed finite element method (IFEM) [8, 10] or other methods can be found in [20] and the references therein.

On the other hand, much attention has recently been paid to applying deep neural networks (DNNs) to solve elliptic interface problems, rather than using traditional numerical methods to solve such problems. Despite the success of the two mainstream deep learning approaches (Physics-Informed Neural Networks (PINNs) [25, 26] and the deep Ritz method [5]) in solving partial differential equations with smooth solutions, learning methods based on these two frameworks for solving elliptic interface problems with jump discontinuities remain to be improved. The main and intrinsic difficulty may be attributed to the fact that the usual activation functions used in DNNs are generally smooth; thus, DNN function approximators seem to be incapable of representing discontinuous functions. To approximate such discontinuous solutions (or functions) and tackle the elliptic interface problems, multiple independent networks need to be established and linked with each other by imposing the jump conditions, see, e.g., piecewise DNNs [11], interfaced neural networks [27], and deep unfitted Nitsche method [9]. The resulting prediction errors in their test examples reach the magnitude $\mathcal{O}(10^{-3})$ to $\mathcal{O}(10^{-4})$ in relative L^2 norm. Moreover, training these DNN models comes at the cost of having to

train a separate neural network in each subdomain independently. Until very recently, the authors of this paper proposed a Discontinuity Capturing Shallow Neural Network (DCSNN) [14] that allows a single network to represent piecewise smooth functions via a simple augmentation technique. The network is completely shallow (one hidden layer), so the resulting number of trainable parameters is moderate (only a few hundred) and attains prediction accuracy as low as $\mathcal{O}(10^{-7})$ in relative L^2 norm for all tests in both 2D and 3D elliptic interface problems. Note that the above neural network methods are all completely mesh-free, but their convergence still requires further investigation.

In this work, we propose a novel hybrid method that combines neural network learning machinery and traditional finite difference methods to solve the Poisson interface problem (1.1)-(1.3). The entire computation only comprises a supervised learning task of function approximation and a fast direct solver of the Poisson equation, which can be easily and directly implemented regardless of interface geometry. Here, we want to emphasize that it is not our intention to replace traditional numerical methods such as the immersed interface method (IIM) or immersed finite element method (IFEM) nor to compete with them in every aspect. Instead, we want to provide an alternative (especially from the implementation aspect) to solve Poisson interface problems with non-homogeneous jump conditions in which the advantages of using fast Poisson solver and machine learning can be fully exploited. As known, the IIM and non-bodyfitted IFEM need some complicated treatments to handle the non-homogeneous jump conditions near the interface, especially in 3D case. However, in the present hybrid method, these interface conditions can be easily incorporated into a function constructed by supervised learning and thus regular finite difference scheme can be exploited. The numerical experiments for 2D and 3D Poisson interface problems in Section 3 indicate that the proposed method can achieve a similar accuracy with the IIM.

The rest of the paper is organized as follows. In Section 2, we present the methodology and list some features, including error analysis of the hybrid scheme. Numerical results for the Poisson interface problems and Stokes equations with singular forces are given in Sections 3 and 4, respectively, followed by some concluding remarks and future works in Section 5.

2 Hybrid neural-network and finite-difference methodology

By taking advantage of the machine learning techniques, our goal is to design an easy-to-implement fast solver for the Poisson interface problem (1.1)-(1.3). To this end, we propose a novel hybrid method that exploits the advantages of neural network learning machinery and traditional finite difference method. As we can see from the jump conditions in (1.2), the solution u is non-smooth across the interface. Thus, we start by decomposing the solution into

$$u(\mathbf{x}) = v(\mathbf{x}) + w(\mathbf{x}), \tag{2.1}$$

where v and w represent the singular (non-smooth) and regular (smooth) parts of u, respectively. More precisely, we require w to be fairly smooth over the entire domain Ω, so that the zero jumps $[\![w]\!]=[\![\partial_n w]\!]=[\![\Delta w]\!]=0$ on the interface are all satisfied. Now the singular solution v is responsible for having all the discontinuities across the interface; hereby, we construct this discontinuous function by assuming

$$v(\mathbf{x})=\begin{cases}\mathcal{V}(\mathbf{x}), & \mathbf{x}\in\Omega^-,\\ 0, & \mathbf{x}\in\Omega^+,\end{cases} \tag{2.2}$$

where $\mathcal{V}$ is a smooth function to be found. Using the above definition and plugging the decomposition (2.1) into the jump conditions (1.2) and differential equation (1.1), the unknown function $\mathcal{V}$ must satisfy the following constraints along the interface:

$$\mathcal{V}(\mathbf{x})=-\gamma(\mathbf{x}),\quad \partial_n\mathcal{V}(\mathbf{x})=-\rho(\mathbf{x}),\quad \Delta\mathcal{V}(\mathbf{x})=-[\![f(\mathbf{x})]\!],\quad \mathbf{x}\in\Gamma. \tag{2.3}$$

Note that this function is not unique, in the sense that there exist infinitely many functions defined in the domain Ω that satisfy the restrictions (2.3). To find $\mathcal{V}$, we leverage the power of function expressibility of neural networks. Here, we simply employ a shallow (one hidden layer) fully-connected feedforward neural network to approximate $\mathcal{V}$, and learn the function via the supervised learning model. Specifically, given a dataset with M training data points $\{\mathbf{x}_\Gamma^i\in\Gamma\}_{i=1}^M$ and the target outputs $\gamma(\mathbf{x}_\Gamma^i)$, $\rho(\mathbf{x}_\Gamma^i)$ and $[\![f(\mathbf{x}_\Gamma^i)]\!]$, we find $\mathcal{V}(\mathbf{x})$ by minimizing the following mean squared error loss consisting of the residuals of conditions in Eq. (2.3):

$$\text{Loss}(\mathbf{p})=\frac{1}{M}\sum_{i=1}^{M}\left[\Big(\mathcal{V}(\mathbf{x}_\Gamma^i;\mathbf{p})+\gamma(\mathbf{x}_\Gamma^i)\Big)^2+\Big(\partial_n\mathcal{V}(\mathbf{x}_\Gamma^i;\mathbf{p})+\rho(\mathbf{x}_\Gamma^i)\Big)^2+\Big(\Delta\mathcal{V}(\mathbf{x}_\Gamma^i;\mathbf{p})+[\![f(\mathbf{x}_\Gamma^i)]\!]\Big)^2\right], \tag{2.4}$$

where $\mathbf{p}$ collects all trainable parameters (weights and biases) in the network. To train the above loss model, we adopted the Levenberg-Marquardt (LM) method [23], a full-batch optimization algorithm which is particularly efficient for least squares losses. We should also mention that the partial derivatives of the target function $\mathcal{V}(\mathbf{x})$ in the loss function (2.4) can be computed easily by automatic differentiation [2].

Once $\mathcal{V}$ is available, we can obtain w by solving the following Poisson equation:

$$\Delta w(\mathbf{x})=\Delta u(\mathbf{x})-\Delta v(\mathbf{x})=\begin{cases}f(\mathbf{x})-\Delta\mathcal{V}(\mathbf{x}), & \mathbf{x}\in\Omega^-,\\ f(\mathbf{x}), & \mathbf{x}\in\Omega^+,\end{cases} \tag{2.5}$$

$$w(\mathbf{x})=u_b(\mathbf{x}),\quad \mathbf{x}\in\partial\Omega. \tag{2.6}$$

Notice that, using the last jump constraint for $\mathcal{V}$ in Eq. (2.3), one can immediately see that the right-hand side function of (2.5) is continuous on the entire domain. Moreover, w is accompanied by exactly the same boundary conditions as the solution u, since v vanishes in Ω^+. As a result, w has sufficient regularity (recall the requirement $[\![w]\!]=[\![\partial_n w]\!]=$

$[\![\Delta w]\!]=0$) and satisfies the Poisson equation in the regular domain Ω that can be simply and efficiently solved using the well-developed public software Fishpack [1] or any fast Poisson solvers. Of course, other traditional numerical methods, such as finite volume or finite element methods, can also be used to find the solution w. Additionally, we remark that both the right-hand sides of Eq. (2.2) and the Poisson equation (2.5) involve the categorization of $\mathbf{x}$ in Ω^- or Ω^+, which can be easily done with the assistance of a level set function for which the zero level set represents the interface Γ.

Let us summarize the proposed hybrid neural-network and finite-difference method for solving the Poisson interface problem (1.1)-(1.3) as follows:

Step 1. With a given training dataset and a sufficient number of neurons (or trainable parameters) used in the network, find the neural network function $\mathcal{V}$ by minimizing the loss function (2.4) using the LM optimizer. Compute $\mathcal{V}$ at the finite difference discretization grid points in Ω^- and then obtain the singular part of the solution, v, at those grid points using Eq. (2.2).

Step 2. Evaluate $\Delta\mathcal{V}$ at the grid points in Ω^-, and solve the Poisson equation (2.5) by discretizing the Laplace operator using the standard five-point Laplacian and applying a fast direct solver to obtain the regular part of the solution, w, at those gird points.

Step 3. Recover the numerical solution $u=v+w$ at the gird points.

We conclude this section by introducing several features of the proposed method as follows:

1. One can immediately deduce that the source of numerical error comes from the network approximation (optimization and network approximation error) and the finite difference approximation (local truncation error). The solution accuracy clearly depends on these two approximations.

2. The right-hand side function of the Poisson equation for w (see Eq. (2.5)) is continuous, but, in general, has discontinuities in its derivatives across the interface. Under the finite difference discretization in Step 2, we have $[\![w]\!]=[\![\partial_n w]\!]=[\![\Delta w]\!]=0$ on the interface. So the local truncation error for grid points right adjacent to the interface (irregular points) is $\mathcal{O}(h)$ with mesh size h, while the one for other grid points (regular points) is $\mathcal{O}(h^2)$. Since the number of irregular points is one dimension lower than the number of regular points used in the problem, this $\mathcal{O}(h)$ truncation error on an interior set of relative small size does not affect the overall second-order accuracy. (For 1D case, this can be immediately seen by writing the solution with the discrete Green's function (scaled by h) so the $\mathcal{O}(h)$ local truncation error would make an $\mathcal{O}(h^2)$ contribution to the global error.) To prove this, Beale and Layton [4] first write this localized $\mathcal{O}(h)$ truncation error at irregular points as the discrete

divergence of a function which is only $\mathcal{O}(h^2)$ in magnitude, and then perform a maximum norm estimate for a discrete elliptic problem with a nonhomogeneous term of divergence form. They are able to prove the solution global error is $\mathcal{O}(h^2)$ and its gradient error is $\mathcal{O}(h^2\log(1/h))$ in maximum norm. The present numerical evidence indeed shows that the overall accuracy for solving w is second-order.

3. With only one-time training, the obtained network function $\mathcal{V}$ is defined in a continuous sense so that it can be used in Step 2 for any grid resolutions.

4. The advantage of traditional grid-based methods is that the boundary conditions are exactly satisfied. The present hybrid method shares the same advantage thanks to the design of Eq. (2.2). In contrast, most modern deep learning approaches either adopt a penalty term in the loss function (e.g., PINNs [25] and DCSNN [14]) or introduce an energy functional to enforce the boundary condition (e.g., shallow Ritz method [15] and deep Nitsche method [21]), leading to an inevitable prediction error along the domain boundary.

5. The proposed hybrid algorithm is easy to implement and efficient. It comprises a supervised learning task (for $\mathcal{V}$) and a fast direct Poisson solver (for w), and there are already many well-developed and efficient packages for both tasks. We should also point out that the regular part solver is not limited to finite difference method nor the five-point Laplacian discretization. Finite volume, finite element, or spectral methods can also be used to find the solution w.

6. The present method can be applied to any domain that has available fast solvers, such as a two-dimensional disk, a three-dimensional sphere, higher dimensional cube with periodic boundary conditions.

7. It is straightforward to implement the present method when multiple embedded interfaces are considered.

3 Numerical results

In this section, we check the accuracy of the proposed method by performing two numerical tests, including solving two- and three-dimensional Poisson interface problems. In each test, the neural net function $\mathcal{V}$ is simply represented via a shallow network with a sigmoid activation function, in which only a single hidden layer is employed. Thanks to the shallow network structure, it only needs to train a moderate amount of parameters (a few hundred parameters used throughout all numerical examples), so learning this network function is efficient, for example, it can be done in seconds on iMac (2021). Since all the computational domains considered in the following problems are regular (square in 2D and cube in 3D), to solve the regular part w, we set up a uniform Cartesian grid layout with the same mesh size h in each spatial direction.

Example 1. We start by solving a two-dimensional Poisson interface problem and compare the results with the ones obtained by the 2D IIM [16]. The problem is defined in the square domain $\Omega=[-1,1]^2$ in which the embedded interface is an ellipse given by $\Gamma:(x/0.8)^2+(y/0.2)^2=1$. The exact solution is chosen as

$$u_e(x,y)=\begin{cases}\exp(x)\cos(y), & \text{if} \quad (x,y)\in\Omega^-,\\ \exp(x^2)\cos(y), & \text{if} \quad (x,y)\in\Omega^+,\end{cases}$$

so the corresponding right-hand side $f(x,y)$, and the jump information $\gamma(x,y)$, $\rho(x,y)$ and $[\![f(x,y)]\!]$ used in the loss function can be calculated accordingly. In this example, the network for $\mathcal{V}(x,y)$ is equipped with 40 neurons in the hidden layer and is trained using 200 randomly sampled training points on the interface Γ. We finish the training process when the stopping condition $\text{Loss}(\mathbf{p})<10^{-12}$ or the maximum iteration number (epoch$=$1000) is met.

In the left panel of Fig. 1, we report the mesh refinement study for maximum norm error $\|u-u_e\|_\infty$ as a function of mesh size h, where u denotes the numerical solution. One can see that the results obtained by the present hybrid method (solid blue line with circular markers) and IIM (solid red line with triangular markers) are almost equally well, and both achieve a second-order convergence rate. We then use the computed solution to find $\nabla u=(\partial_x u,\partial_y u)$ simply by applying standard central difference for the regular part w and automatic differentiation for the singular part v. As can be seen in the right panel of Fig. 1, the gradient of the numerical solution attains a second-order convergence too.

As discussed in the previous section, the induced numerical error comes from both neural network approximation and finite difference approximation. Although not shown here, the final loss value here is about $10^{-13}\sim10^{-14}$, which leads to the predictive accuracy of the target function $\mathcal{V}$ and its Laplacian are of magnitude $10^{-6}\sim10^{-7}$. Thus, we can conclude that the error is mainly dominated by the second-order finite difference approx-

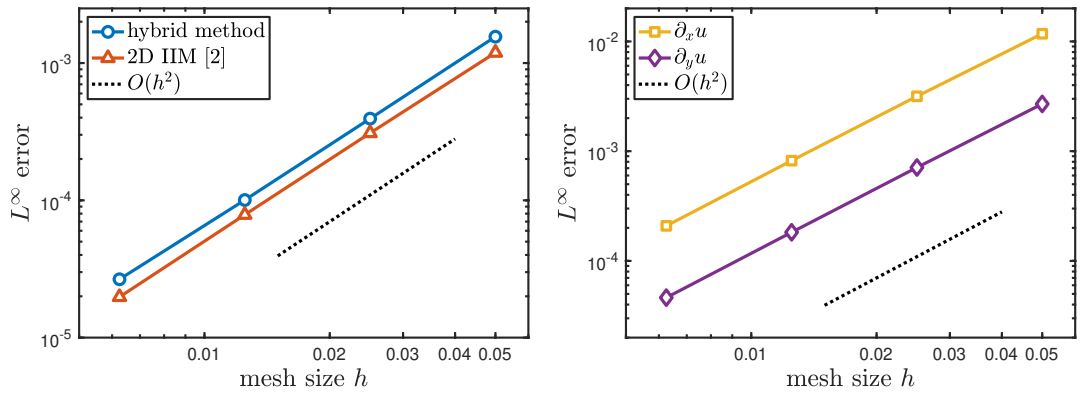

Figure 1: Mesh refinement results for the 2D Poisson interface problem in Example 1. Left: Comparison of maximum norm errors of u between the present hybrid method and 2D IIM [16]. Right: Maximum norm error of the gradient $\nabla u=(\partial_x u,\partial_y u)$.

imation error when $h^2 \gtrsim 10^{-6}$. To verify our error estimation, we have run more refining numerical tests with $h^2 < 10^{-6}$. As expected, the resulting error cannot be reduced when refining the mesh width h with higher resolutions (not shown here).

Example 2. Because of the mesh-free nature of the neural network, the proposed method is robust to handle complicated interface geometries in which one only has to input the interface description (such as interface parametric form and normal vector) in implementing supervised learning for the singular part solution. We test the problem where the interface is of super-ellipse shape $(x/\sqrt{0.7})^4+(y/\sqrt{0.1})^4=1$, and choose the same analytic solution u_e and follow the same setups as in Example 1. In the left panel of Fig. 2, we depict the numerical solution u with the mesh size $h=1/160$; as one can see that the discontinuity is indeed captured sharply across the interface. In the right panel we report the mesh refinement result for the numerical solution and its gradient. As anticipated, the maximum norm errors converge with second-order accuracy.

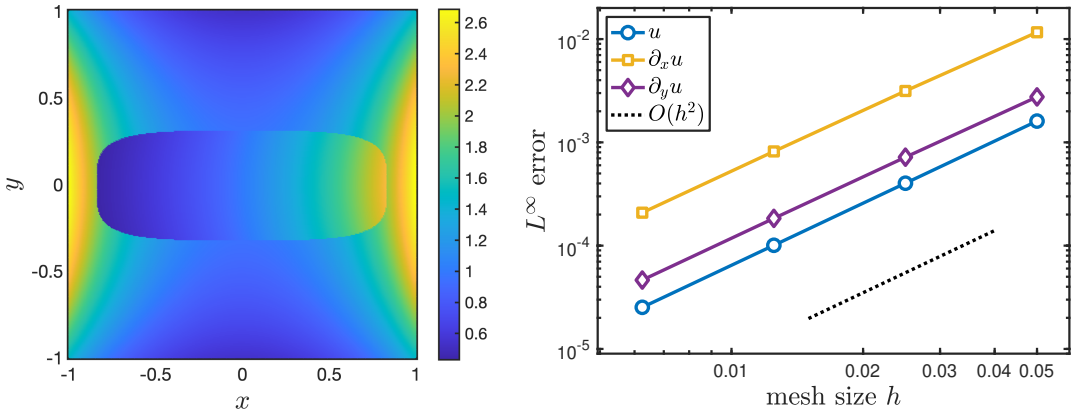

Figure 2: Left: The numerical solution u obtained by the present hybrid method with the mesh size $h=1/160$. Right: Maximum norm errors of the numerical solution u and its gradient $\nabla u=(\partial_x u,\partial_y u)$.

Example 3. To showcase the reliability of the hybrid method, we present an example in the scenario that the exact solution is unavailable. We again embed an ellipse Γ with parametric form $(x(s),y(s))=(\sqrt{0.7}\cos s,\sqrt{0.1}\sin s), s\in[0,2\pi)$ in the square domain $\Omega=[-1,1]^2$. Specifically, we choose the right-hand side function of the Poisson interface problem as

$$f(x,y)=\begin{cases} \exp(x\sin y), & \text{if } (x,y)\in\Omega^-, \\ \exp(y\cos x), & \text{if } (x,y)\in\Omega^+. \end{cases}$$

Along the domain boundary $\partial\Omega$, we set the Dirichlet boundary condition $u(x,y)=0$; along the interface Γ, we choose the jump condition $[\![u]\!](s)=\gamma(s)=\sin s$, $[\![\partial_n u]\!](s)=\rho(s)=$

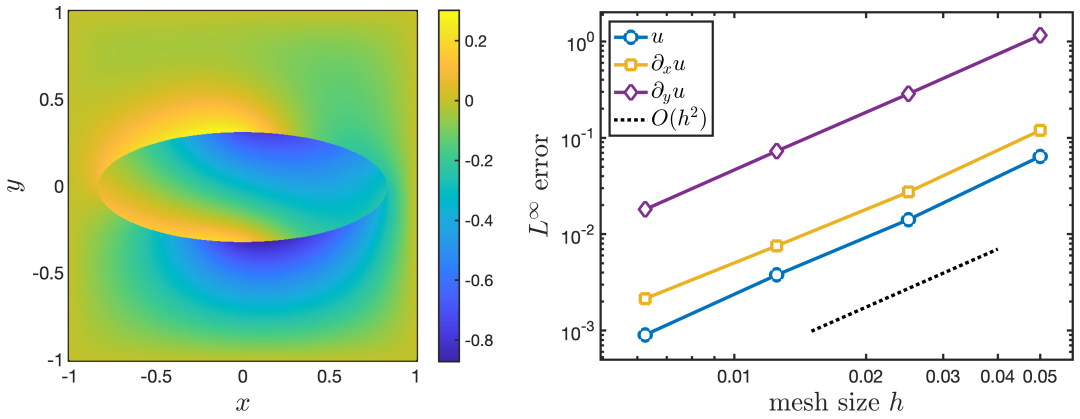

Figure 3: Left: The numerical solution u obtained by the present hybrid method with the mesh size $h=1/320$. Right: Maximum norm errors of the numerical solution u and its gradient $\nabla u=(\partial_x u,\partial_y u)$.

cos$s$, and the jump $[\![f]\!](s)$ can be accordingly obtained from the above given function. The singular part solution is trained using 150 neurons in one hidden layer with 300 randomly sampled training points on the interface Γ.

We need to point out that, since the exact solution is not available in this test, we measure the L^∞ error by the successive error $\|u_h-u_{h/2}\|_\infty$, where u_h denotes the solution with the grid size h. We depict the numerical solution u in the left panel of Fig. 3 with the finest resolution $h=1/320$ in this test. In the right panel we show the maximum norm errors for u and its gradient. Again, all the quantities converge with second-order accuracy.

Example 4. We proceed to consider the three-dimensional Poisson interface problem, in which the interface is an ellipsoid $\Gamma:(x/0.7)^2+(y/0.5)^2+(z/0.3)^2=1$, embedded in a cube $\Omega=[-1,1]^3$. The exact solution is given by

$$u_e(x,y,z)=\begin{cases}\exp(x+y+z), & \text{if} \quad (x,y,z)\in\Omega^-,\\ \sin(x)\sin(y)\sin(z), & \text{if} \quad (x,y,z)\in\Omega^+.\end{cases}$$

Again, one can obtain f, γ, ρ, and $[\![f]\!]$ accordingly. We use the same shallow network structure as in the previous example, i.e., we set 40 neurons in one hidden layer and 200 training points on the interface Γ to learn $\mathcal{V}(x,y,z)$. Fig. 4 shows the mesh refinement results for the present method (solid blue line with circular markers) and 3D IIM solver developed in [12] (solid red line with triangular markers), as well as the maximum norm errors for the numerical gradient. Similar to the 2D case, one can clearly see that the results of the present hybrid method and IIM are almost identical, and a second-order accuracy is achieved for both the numerical solution and its gradient. It is important to point out that, the implementation for learning $\mathcal{V}$ and solving w in 3D problems is indeed

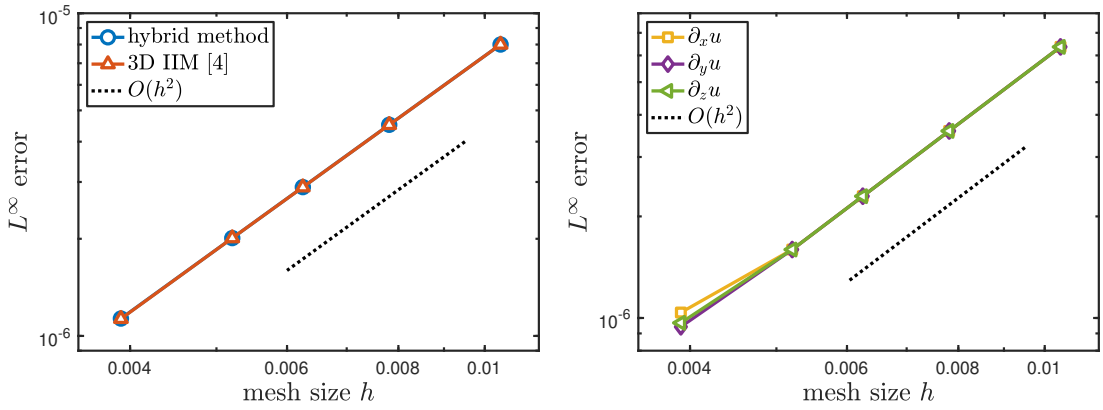

Figure 4: Mesh refinement results for the 3D Poisson interface problem in Example 2. Left: Comparison of maximum norm errors of u between the present hybrid method and 3D IIM [12]. Right: Maximum norm error of the gradient $\nabla u=(\partial_x u,\partial_y u,\partial_z u)$.

straightforward as in 2D. By contrast, calculating the extra correction terms (incorporating all jump information) in the IIM implementation can be quite tedious in 3D problems, especially when the interface geometry is complex.

4 Application: 2D Stokes equations with singular forces

In this section, we apply the proposed method to solve the two-dimensional Stokes equations with singular forces on an interface Γ, which result from the immersed boundary formulation [24] of fluid-structure interaction problems. The governing equations are written the same as in [16]:

$$-\nabla p(\mathbf{x})+\mu\Delta\mathbf{u}(\mathbf{x})+\int_\Gamma \mathbf{F}(s)\delta(\mathbf{x}-\mathbf{X}(s))\,\mathrm{d}s+\mathbf{g}(\mathbf{x})=\mathbf{0},\quad \mathbf{x}\in\Omega, \tag{4.1}$$

$$\nabla\cdot\mathbf{u}(\mathbf{x})=0,\quad \mathbf{x}\in\Omega, \tag{4.2}$$

$$\mathbf{u}(\mathbf{x})=\mathbf{u}_b(\mathbf{x}),\quad \mathbf{x}\in\partial\Omega, \tag{4.3}$$

where $\mathbf{u}=(u_1,u_2)$ denotes the fluid velocity, p is the pressure, and μ is the constant viscosity. There are two different forces acting to the fluid; namely, the external force field $\mathbf{g}$ (might be discontinuous) to the fluid domain Ω, and the singular force represented by the interfacial force $\mathbf{F}$ in terms of delta function formulation on Γ. Here, the notation $\mathbf{X}(s)$ represents the configuration of the interface Γ with the arc-length parameter s. (Notice that, the delta function in Eq. (4.1) is two-dimensional but the integration is over the one-dimensional interface which leaves the integral term has one-dimensional singularity.) Furthermore, the interfacial force $\mathbf{F}(s)$ can be written as a sum of the tangential ($\boldsymbol{\tau}$) and normal ($\mathbf{n}$) components as $\mathbf{F}=F_\tau\boldsymbol{\tau}+F_n\mathbf{n}$. Instead of using the delta function formulation, one can rewrite Eqs. (4.1)-(4.3) into the following Stokes equations with jump conditions

across the interface (see the reference in [19]):

$$-\nabla p(\mathbf{x})+\mu\Delta\mathbf{u}(\mathbf{x})+\mathbf{g}(\mathbf{x})=\mathbf{0}, \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{4.4}$$

$$\nabla\cdot\mathbf{u}(\mathbf{x})=0, \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{4.5}$$

$$\mathbf{u}(\mathbf{x})=\mathbf{u}_b(\mathbf{x}), \quad \mathbf{x}\in\partial\Omega, \tag{4.6}$$

$$[\![p(\mathbf{X}(s))]\!]=F_n(s), \quad [\![\mathbf{u}(\mathbf{X}(s))]\!]=\mathbf{0}, \quad [\![\partial_n\mathbf{u}(\mathbf{X}(s))]\!]=-F_\tau(s)\boldsymbol{\tau}(s)/\mu, \quad \mathbf{X}(s)\in\Gamma. \tag{4.7}$$

To solve the above Stokes equations by the present methodology given in Section 2, following the implementation in [19], we firstly apply the divergence operator to Eq. (4.4) and use the divergence-free condition (4.5) to obtain the pressure Poisson equation as

$$\Delta p(\mathbf{x})=\nabla\cdot\mathbf{g}(\mathbf{x}), \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{4.8}$$

$$[\![p(\mathbf{X}(s))]\!]=F_n(s), \quad [\![\partial_n p(\mathbf{X}(s))]\!]=\frac{\partial F_\tau}{\partial s}+[\![\mathbf{g}(\mathbf{X}(s))]\!]\cdot\mathbf{n}(s), \quad \mathbf{X}(s)\in\Gamma. \tag{4.9}$$

One can immediately see that an extra normal derivative jump condition of the pressure is needed since now the equation (4.8) involves second-order derivatives rather than the first-order derivatives in Eq. (4.4). For completeness, we list the derivation of the jump conditions in Eq. (4.9) in Appendix, in which the proof is slightly different from the one given in [19]. Once the pressure is found, the velocity can be obtained by solving

$$\Delta\mathbf{u}(\mathbf{x})=\frac{1}{\mu}(\nabla p(\mathbf{x})-\mathbf{g}(\mathbf{x})), \quad \mathbf{x}\in\Omega^-\cup\Omega^+, \tag{4.10}$$

$$[\![\mathbf{u}(\mathbf{X}(s))]\!]=\mathbf{0}, \quad [\![\partial_n\mathbf{u}(\mathbf{X}(s))]\!]=-F_\tau(s)\boldsymbol{\tau}(s)/\mu, \quad \mathbf{X}(s)\in\Gamma, \tag{4.11}$$

$$\mathbf{u}(\mathbf{x})=\mathbf{u}_b(\mathbf{x}), \quad \mathbf{x}\in\partial\Omega. \tag{4.12}$$

One can immediately see that these are Poisson interface problems so the present method can be applied directly.

Example. For the numerical test, we use the example as in [16]. We consider a square domain $\Omega=[-2,2]^2$ and the interface is simply a unit circle with the center located at the origin: $\Gamma=\{\mathbf{X}(s)=(\cos s,\sin s)|s\in[0,2\pi)\}$. The interfacial force comprises both tangential ($\boldsymbol{\tau}(s)=\mathbf{X}'(s)$) and normal ($\mathbf{n}(s)=\mathbf{X}(s)$) directions as $\mathbf{F}(s)=2\sin(3s)\boldsymbol{\tau}(s)-\cos^3(s)\mathbf{n}(s)$. Here, we set $\mu=1$, and the exact velocity written in polar coordinates is chosen as

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

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

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

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

Substituting the above velocity $\mathbf{u}=(u_1,u_2)$ and pressure p into Eq. (4.4), one can obtain the corresponding discontinuous external force $\mathbf{g}$.

We establish a three-output neural network, $\boldsymbol{\mathcal{V}}=(\mathcal{V}_{u_1},\mathcal{V}_{u_2},\mathcal{V}_p)$, to learn the target functions simultaneously. Consequently, the loss function now consists all the jump residuals for $\mathbf{u}$ and p. Notice that, one requires the jump information of the right-hand side of Eq. (4.10), $\frac{1}{\mu}([\![\nabla p]\!]-[\![\mathbf{g}]\!])$; the latter is given directly and the former can be obtained by using the identity $[\![\nabla p]\!]=\partial_s[\![p]\!]\boldsymbol{\tau}+[\![\partial_n p]\!]\mathbf{n}$ that links to the given quantities using Eq. (4.9). In the present test, we use a one-hidden-layer network that consists 50 neurons in the hidden layer. We use 200 randomly distributed sampling points and train the model using the LM optimizer.

Once the singular parts are obtained, we use fast Poisson solver to find the regular parts. Inside Ω, the fluid variables are defined at usual staggered grid layout with uniform mesh width h. Namely, the velocity components u_1 and u_2 are correspondingly defined at the cell edges

$$(x_{i-1/2},y_j)=(-2+(i-1)h,-2+(j-1/2)h),\quad (x_i,y_{j-1/2})=(-2+(i-1/2)h,-2+(j-1)h),$$

while the pressure p is defined at the cell center

$$(x_{i-1/2},y_{j-1/2})=(-2+(i-1/2)h,-2+(j-1/2)h).$$

We should point out that, for testing purpose, the pressure boundary condition is chosen to be the Neumann type (which is commonly used in projection method for the pressure increment in Navier-Stokes flows [3]). After the numerical solution for the regular part of pressure is obtained, we compute ∇p at the cell edges (which coincides with the location of $\mathbf{u}$) by applying standard central difference for the regular part and auto differentiation for the singular part.

The resulting errors for all fluid variables are shown in the top panel of Fig. 5. As expected, the accuracy obtained by the present hybrid method is comparable with the ones by IIM [16], and roughly achieves second-order convergence in maximum norm error. In the bottom panel, we can also see that the numerical errors of $\nabla\cdot\mathbf{u}$ and ∇p are around second-order as well.

5 Conclusion and future work

In this paper, we propose a new class of numerical methods to solve an elliptic interface problem whose solution and derivatives are known to have jump discontinuities across

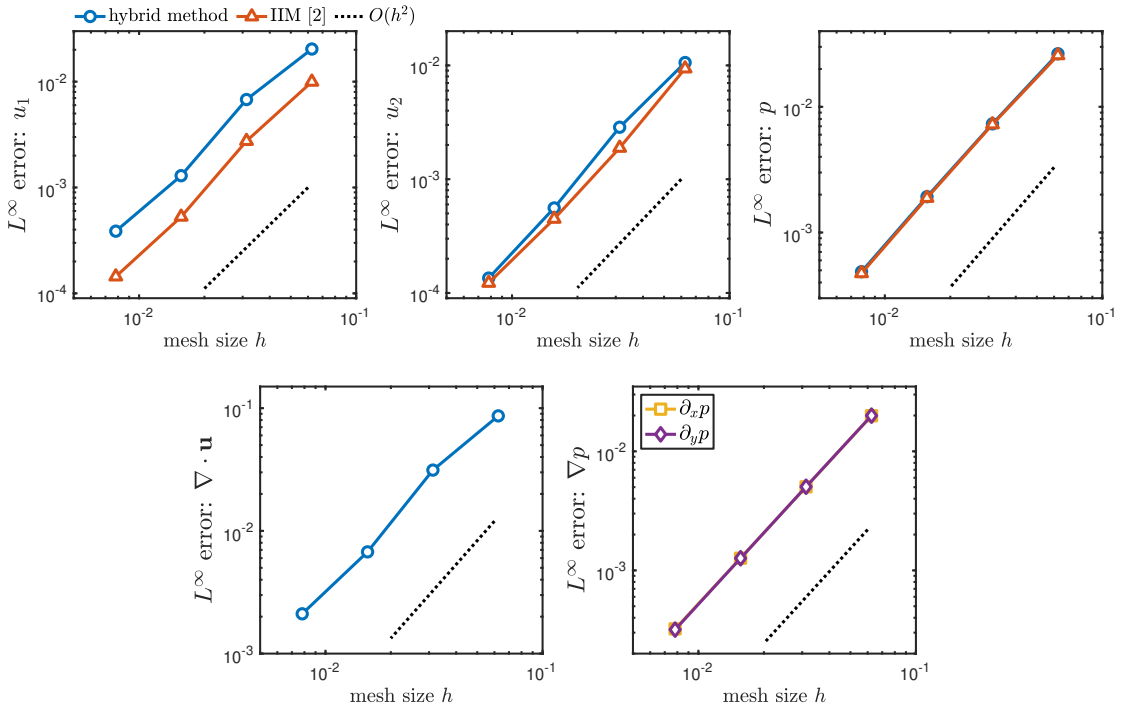

Figure 5: Mesh refinement results for 2D Stokes equations with singular forces. Top: Comparison results between the present hybrid method and 2D IIM [16]. Bottom: Maximum norm errors of $\nabla \cdot \mathbf{u}$ and ∇p.

an interface. The crucial idea is to decompose the solution into singular (non-smooth) and regular (smooth) parts. The singular part is formed by a neural network representation found by using supervised learning machinery that incorporates all given jump information into the loss function. The regular part, however, is a solution to the Poisson equation, which can be obtained efficiently by several well-developed numerical methods, such as the fast direct solver based on finite difference discretization. Therefore, it is simple to implement our proposed method, and it is straightforward to handle multiple interfaces or high-dimensional problems. The numerical experiments for 2D and 3D Poisson interface problems show that the proposed neural-network and finite-difference hybrid method can achieve second-order accuracy for the solution and its derivatives. Although all illustrated examples consider a single embedded interface only, it is straightforward to implement the hybrid method with multiple interfaces. As an application, we use the present methodology to solve 2D Stokes equations with singular forces. Again, the numerical result shows that all the fluid variables and their derivatives have second-order convergence in maximum norm error as well.

The present hybrid method readily serves as a fast solver for Poisson interface problems involved in the projection step of Navier-Stokes flow problems. Our future work aims to extend the present methodology for solving variable-coefficient elliptic interface problems in regular or even irregular domains.

Acknowledgments

W.-F. Hu, T.-S. Lin, Y.-H. Tseng, 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, 111-2115-M-390-002, 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.

Appendix

In this appendix, we present the derivation of the pressure jump condition in Eq. (4.9). Let us recall that the domain $\Omega\subset\mathbb{R}^d$, is separated by an embedded interface Γ such that the subdomains inside and outside the interface are denoted by Ω^- and Ω^+, respectively. And the shorthand ∂_n represents the normal derivative of a quantity where $\mathbf{n}$ is the unit outward normal vector pointing from Ω^- to Ω^+. By taking the divergence operator to Eq. (4.1) and using the divergence-free condition (4.2), we obtain the pressure Poisson equation as

$$\Delta p(\mathbf{x}) = \nabla\cdot\int_\Gamma \mathbf{F}(s)\delta(\mathbf{x}-\mathbf{X}(s))\,\mathrm{d}s+\nabla\cdot\mathbf{g}(\mathbf{x}).$$

Since the right-hand side of the above equation involves taking the divergence operator on the Dirac delta function and discontinuous function $\mathbf{g}$, we should regard them in the sense of distributions. In other words, the above equation should be represented as $\langle\Delta p,\phi\rangle=\langle p,\Delta\phi\rangle$, for all test functions $\phi\in C_0^\infty(\Omega)$. So in the following derivations, the test function ϕ and its normal derivative $\partial_n\phi$ will be vanished on the outside boundary $\partial\Omega^+$. Applying the derivative properties of the Dirac delta function, we have

$$\begin{aligned}
\langle\Delta p,\phi\rangle &= \int_\Omega \nabla\cdot\int_\Gamma \mathbf{F}(s)\delta(\mathbf{x}-\mathbf{X}(s))\,\mathrm{d}s\phi(\mathbf{x})\,\mathrm{d}\mathbf{x}+\langle\nabla\cdot\mathbf{g},\phi\rangle\\
&= -\int_\Gamma \mathbf{F}(s)\cdot\nabla\phi(\mathbf{X}(s))\,\mathrm{d}s-\langle\mathbf{g},\nabla\phi\rangle \quad \text{(by definition of derivative distributions)}\\
&= -\int_\Gamma (F_\tau\boldsymbol{\tau}+F_n\mathbf{n})\cdot\nabla\phi\,\mathrm{d}s-\int_{\Omega^+}\mathbf{g}\cdot\nabla\phi\,\mathrm{d}\mathbf{x}-\int_{\Omega^-}\mathbf{g}\cdot\nabla\phi\,\mathrm{d}\mathbf{x}\\
&= -\int_\Gamma (F_\tau\boldsymbol{\tau}+F_n\mathbf{n})\cdot\nabla\phi\,\mathrm{d}s+\int_{\Omega^+}(\nabla\cdot\mathbf{g})\phi\,\mathrm{d}\mathbf{x}+\int_\Gamma(\mathbf{g}^+\cdot\mathbf{n})\phi\,\mathrm{d}s\\
&\quad +\int_{\Omega^-}(\nabla\cdot\mathbf{g})\phi\,\mathrm{d}\mathbf{x}-\int_\Gamma(\mathbf{g}^-\cdot\mathbf{n})\phi\,\mathrm{d}s\\
&\qquad\qquad \text{(by applying Green's first identity to } \Omega^\pm \text{ separately)}\\
&= \int_\Gamma\frac{\partial F_\tau}{\partial s}\phi\,\mathrm{d}s-\int_\Gamma F_n\partial_n\phi\,\mathrm{d}s+\int_{\Omega^-\cup\Omega^+}(\nabla\cdot\mathbf{g})\phi\,\mathrm{d}\mathbf{x}+\int_\Gamma([\![\mathbf{g}]\!]\cdot\mathbf{n})\phi\,\mathrm{d}s, \qquad (A.1)
\end{aligned}$$

where the first term in the last equation is obtained using integration by parts. Meanwhile,

$$\begin{aligned}
\langle p,\Delta\phi\rangle &= \int_{\Omega^+} p\Delta\phi \,\mathrm{d}\mathbf{x} + \int_{\Omega^-} p\Delta\phi \,\mathrm{d}\mathbf{x} \\
&= -\int_{\Omega^+} \nabla p\cdot\nabla\phi \,\mathrm{d}\mathbf{x} - \int_\Gamma p^+\partial_n\phi \,\mathrm{d}s - \int_{\Omega^-} \nabla p\cdot\nabla\phi \,\mathrm{d}\mathbf{x} + \int_\Gamma p^-\partial_n\phi \,\mathrm{d}s \\
&= -\int_{\Omega^+} \nabla p\cdot\nabla\phi \,\mathrm{d}\mathbf{x} - \int_{\Omega^-} \nabla p\cdot\nabla\phi \,\mathrm{d}\mathbf{x} - \int_\Gamma [\![p]\!]\partial_n\phi \,\mathrm{d}s \\
&= \int_{\Omega^+} \Delta p\phi \,\mathrm{d}\mathbf{x} + \int_\Gamma \partial_n p^+\phi \,\mathrm{d}s + \int_{\Omega^-} \Delta p\phi \,\mathrm{d}\mathbf{x} - \int_\Gamma \partial_n p^-\phi \,\mathrm{d}s - \int_\Gamma [\![p]\!]\partial_n\phi \,\mathrm{d}s \\
&= \int_{\Omega^+} (\nabla\cdot\mathbf{g})\phi \,\mathrm{d}\mathbf{x} + \int_{\Omega^-} (\nabla\cdot\mathbf{g})\phi \,\mathrm{d}\mathbf{x} + \int_\Gamma [\![\partial_n p]\!]\phi \,\mathrm{d}s - \int_\Gamma [\![p]\!]\partial_n\phi \,\mathrm{d}s \\
&= \int_{\Omega^-\cup\Omega^+} (\nabla\cdot\mathbf{g})\phi \,\mathrm{d}\mathbf{x} + \int_\Gamma [\![\partial_n p]\!]\phi \,\mathrm{d}s - \int_\Gamma [\![p]\!]\partial_n\phi \,\mathrm{d}s. \qquad \text{(A.2)}
\end{aligned}$$

By equating Eqs. (A.1) and (A.2), one can immediately obtain the jump conditions

$$[\![p]\!] = F_n, \quad [\![\partial_n p]\!] = \frac{\partial F_\tau}{\partial s} + [\![\mathbf{g}]\!]\cdot\mathbf{n}.$$

References

[1] J. Adams, P. Swarztrauber, R. Sweet. Fishpack – a package of fortran subprograms for the solution of separable elliptic partial differential equations [online] (1980).

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: A survey, *J. Mach. Learn. Res.*, 18 (2018), 1–43.

[3] D. L. Brown, R. Cortez, M. L. Minion, Accurate projection methods for the incompressible Navier-Stokes equations, *J. Comput. Phys.*, 168 (2001), 464–499.

[4] J. .T. Beale, A. T. Layton, On the accuracy of finite difference methods for elliptic interface problems with interfaces, *Comm. App. Math. and Comp. Sci.*, 1 (2006), 91–119.

[5] W. E., B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, *Commun. Math. Stat.*, 6 (2018), 1–12.

[6] R. Egan, F. Gibou, xGFM: Recovering convergence of fluxes in the ghost fluid method, *J. Comput. Phys.*, 409 (2020), 109351.

[7] A. Guittet, M. Lepilliez, S. Tanguy, F. Gibou, Solving elliptic problems with discontinuities on irregular domains – the Voronoi interface method, *J. Comput. Phys.*, 298 (2015), 747–765.

[8] Y. Gong, B. Li, and Z. Li, Immersed-interface finite-element Methods for elliptic interface problems with non-homogeneous jump conditions, *SIAM J. Numer. Anal.*, 46 (2008), 472–495.

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

[10] X. He, T. Lin, Y. Lin, Immersed finite element methods for elliptic interface problems with non-homogeneous jump conditions, *Int. J. Numer. Anal. Model.* 8 (2011), 284–301.

[11] C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems, *J. Comput. Appl. Math.*, 412 (2022), 114358.

[12] S.-H. Hsu, W.-F. Hu, M.-C. Lai, A coupled immersed interface and grid based particle method for three-dimensional electrohydrodynamic simulations, *J. Comput. Phys.*, 398 (2019), 108903.
[13] W.-F. Hu, M.-C. Lai, Y.-N. Young, A hybrid immersed boundary and immersed interface method for electrohydrodynamic simulations, *J. Comput. Phys.*, 282 (2015), 47–61.
[14] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, *J. Comput. Phys.*, 469 (2022), 111576.
[15] M.-C. Lai, C.-C. Chang, W.-S. Lin, W.-F. Hu, T.-S. Lin, A shallow Ritz method for elliptic problems with singular sources, *J. Comput. Phys.*, 469 (2022), 111547.
[16] M.-C. Lai and H.-C. Tseng, A simple implementation of the immersed interface methods for Stokes flows with singular forces, *Computers & Fluids*, 37 (2008), 99–106.
[17] R. J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations, SIAM, 2007.
[18] R. J. LeVeque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, *SIAM J. Numer. Anal.*, 31 (1994) 1019–1044.
[19] R. J. LeVeque, Z. Li, Immersed interface methods for Stokes flow with elastic boundaries or surface tension, *SIAM J. Sci. Comput.*, 18 (1997), 709–735.
[20] Z. Li, K. Ito, The Immersed Interface Method, SIAM, 2006.
[21] Y. Liao, P. Ming, Deep Nitsche method: Deep Ritz method with essential boundary conditions, *Commun. Comput. Phys.*, 29 (2021), pp. 1365–1384.
[22] X.-D. Liu, R. P. Fedkiw, M. Kang, A boundary condition capturing method for Poisson's equation on irregular domains, *J. Comput. Phys.*, 160 (2000), 151–178.
[23] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, *SIAM J. Appl. Math.*, 11 (1963), 431–441.
[24] C. S. Peskin, The immersed boundary method, *Acta Numer.*, 8 (1999), 143–195.
[25] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, 378 (2019), 686–707.
[26] Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, *Commun. Comput. Phys.*, 28 (2020), 2042–2074.
[27] S. Wu, B. Lu, INN: Interfaced neural networks as an accessible meshless approach for solving interface PDE problems, *J. Comput. Phys.*, 470 (2022), 111588.