A Physics-Informed Neural Network Framework For Partial Differential Equations on 3D Surfaces: Time-Dependent Problems

ZHIWEI FANG[1], JUSTIN ZHAN[2], AND XIU YANG[3]

[1]University of Nevada, Las Vegas, 4505 S Maryland Pkwy, Las Vegas, NV 89154, USA

[2]Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA

[3]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA

LEHIGH
U N I V E R S I T Y.

A Physics-Informed Neural Network Framework For Partial Differential Equations on 3D Surfaces: Time-Dependent Problems

Zhiwei Fang [*]   Justin Zhan [†]   Xiu Yang [‡]

Abstract

In this paper, we show a physics-informed neural network solver for the time-dependent surface PDEs. Unlike the traditional numerical solver, no extension of PDE and mesh on the surface is needed. We show a simplified prior estimate of the surface differential operators so that PINN's loss value will be an indicator of the residue of the surface PDEs. Numerical experiments verify efficacy of our algorithm.

1 Introduction

Partial differential equations (PDEs) on manifolds has been widely used in various areas. Especially, its application in image processing include mapping an image on a given surface [1], recovering lost information [2], and segmenting and deciphering images [3, 4]. Such techniques are widely used in biological and medical sciences, e.g., simulating animal coats [5], wound healing [6], brain wrapping [7], lipid interactions in membranes [8], and fluids in lungs [9].

Different numerical methods have been developed to solve PDEs on surfaces [18]. In 1988, G. Dziuk [10] established the finite element scheme for the surface PDEs, including the weak formulation and finite element spaces. Then, an extension was introduced in [11]. However, this method needs to extend both the PDEs and their solution to a neighborhood of the surface. A. Petras et al. [19, 20, 22] report many works on using the radial basis function finite difference (RBF-FD) method to solve surface PDEs. This method is more efficient and easier to implement than finite element methods. But one needs to adjust the shape parameters of radial basis functions to balance the stability and accuracy.

Rather than the traditional numerical solvers, machine learning approaches attract more and more attentions. In particular, physics-informed neural network (PINN) [23] has become a popular method for solving PDEs and inverse problems. Further, the combination of PINN and adversarial networks can be used for uncertainty quantification (UQ) problems in PDE [13]. The finite element methods have been combined with the PINN to enhance its performance [16, 17]. One can even solve the PDEs without the concrete form of it as proposed in [24]. NVIDIA also develops a scalable PINN solver SIMNET [31]

---

[*]University of Nevada, Las Vegas, 4505 S Maryland Pkwy, Las Vegas, NV 89154, USA (`fangz1@unlv.nevada.edu`),

[†]Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA(`jzhan@uark.edu`),

[‡]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA (`xiy518@lehigh.edu`).

In this work, we will proceed with PINN's work for surface PDEs [14] and develop the PINN algorithm for time-dependent PDEs on the surface. We will show a simplified proof about a prior estimate of the surface differential operator. Continuous-time and discrete-time method have been studied for the surface PDEs. The numerical examples will verify our algorithm.

This paper is organized as follows. In section 2, we briefly introduce the mathematical preliminary of surface PDEs. We will show a prior estimate of the surface differential operators so that PINN's loss value will serve as an indicator of surface PDE's residue. PINNs for traditional PDEs and our algorithm for surface PDEs have been shown in section 3. Then, a variety of numerical experiments have been shown to verify our algorithm in section 4. We conclude this paper in section 5.

2 Partial Differential Equations on Surfaces

In this section, we briefly introduce the elementary definitions of PDEs on surfaces. For more details about calculus on the manifolds, we refer interested readers to [18].

2.1 Differential operators on surfaces

Let Γ be a smooth surface embedded in $\mathbb{R}^3$ with unit normal $\boldsymbol{n}$, and $u : \Gamma \mapsto \mathbb{R}$ be a function on Γ. Denoted by $\overline{u}$ the smooth extension of $u : \Gamma \mapsto \mathbb{R}$ to $\overline{u} : U \mapsto \mathbb{R}$, where U is a neighborhood of Γ, such that $\overline{u}|_\Gamma = u$ and $\overline{u}$ is a constant along $\boldsymbol{n}$ for each point on Γ. We use ∇, $\nabla\cdot$ and Δ to denote the ordinary gradient, the divergence, and the Laplace operator in $\mathbb{R}^3$, respectively. Now, we can define the gradient, the divergence, and the Laplace operator on Γ for u, which are essential for PDE on the surface.

Definition 2.1 *Let $u : \Gamma \mapsto \mathbb{R}$ has a continuous derivative (of class C^1), and then the gradient of u on Γ is defined as:*

$$\nabla_\Gamma u = \nabla \overline{u} - \langle \nabla \overline{u}, \boldsymbol{n} \rangle \, \boldsymbol{n},$$

where $\langle \cdot, \cdot \rangle$ means the inner product in $\mathbb{R}^3$.

Definition 2.2 *Let $u : \Gamma \mapsto \mathbb{R}$ has a continuous derivative (of class C^1), and*

$$\nabla_\Gamma u = (D_1 u, D_2 u, D_3 u).$$

Then, the divergence of vector $\mathbf{v} = (v_1, v_2, v_3) \in (C^1)^3$ on Γ is defined by:

$$\nabla_\Gamma \cdot \mathbf{v} = D_1 v_1 + D_2 v_2 + D_3 v_3.$$

Definition 2.3 *Let $u : \Gamma \mapsto \mathbb{R}$ have a second-order continuous derivative (of class C^2), then the Laplace operator on Γ is defined as:*

$$\Delta_\Gamma u = \nabla_\Gamma \cdot \nabla_\Gamma u.$$

The Laplace operator on a surface is also known as the Laplace-Beltrami operator.

2.2 The equivalence principles

Instead of solving surface the PDEs directly, we solve the ordinary PDEs on the surface with constraints. The idea of our method comes from the following lemma [20]:

Lemma 2.1 *Let u be any function on $\mathbb{R}^n$ that is constant along with normal directions of Γ. Then, at the surface, intrinsic gradients are equivalent to standard gradients:*

$$\nabla_\Gamma u = \nabla u.$$

Let $\mathbf{v}$ be any vector field on $\mathbb{R}^n$ that is tangent to Γ and tangent to all surfaces displaced by a fixed distance from Γ. Then, at the surface,

$$\nabla_\Gamma \cdot \mathbf{v} = \nabla \cdot \mathbf{v}.$$

Therefore, by using this lemma, we may replace the surface gradient $\nabla_\Gamma u$ with ∇u under the constraint $\langle \nabla u, \boldsymbol{n} \rangle = 0$. Then, the Laplace-Beltrami operator $\Delta_\Gamma u$ can be replaced by Δu with constraint $\langle \nabla u, \boldsymbol{n} \rangle = 0$. This is because under the constraint $\langle \nabla u, \boldsymbol{n} \rangle = 0$, ∇u is a vector field tangent to Γ. Hence,

$$\Delta_\Gamma u = \nabla_\Gamma \cdot \nabla_\Gamma u = \nabla_\Gamma \cdot \nabla u = \nabla \cdot \nabla u = \Delta u,$$

if we extend u as a constant along the $\boldsymbol{n}$.

The original idea of PINNs is to set up a loss function by using the residue of the PDE [23]. When the loss function approaches zero, the residue of the PDE goes to zero, which implies taht the PDE is satisfied on training points approximately. However, we will show that this is not true for surface PDEs. Therefore, we develop a new algorithm in [14] to solve this problem. The following prior estimate will be used to improve our algorithm in the next sections.

Theorem 2.1 *Let $\Gamma \subset \mathbb{R}^3$ be a smooth surface whose mean curvature $|H| < \infty$, $\mathbf{v}$ and $\mathbf{f}$ be two vector fields on Γ, and u and g are two scalar functions on Γ, then we have*

$$\|\nabla_\Gamma u - \mathbf{f}\| \leq \|\nabla \overline{u} - \overline{\mathbf{f}}\| + \|\langle \boldsymbol{n}, \nabla \overline{u} \rangle\|, \tag{1}$$

$$\|\nabla_\Gamma \cdot \mathbf{v} - g\| \leq \|\nabla \cdot \overline{\mathbf{v}} - \overline{g}\| + \|\boldsymbol{n}^\top \nabla \overline{\mathbf{v}} \boldsymbol{n}\|, \tag{2}$$

$$\|\Delta_\Gamma u - g\| \leq C(\|\Delta \overline{u} - \overline{g}\| + \|\langle \boldsymbol{n}, \nabla \overline{u} \rangle\| + \|\boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n}\|) \tag{3}$$

on Γ, where $C = \max(1, 2|H|)$ is a constant, column vector $\boldsymbol{n}$ is the unit normal on Γ, $\|\cdot\|$ is the L^2 norm for the scalar or the vector functions, and ∇^2 is the Hessian operator. The overline means the $\mathbb{R}^3$ smooth extension as mentioned above.

Proof.

1. By using the definition of ∇_Γ, we have

$$\begin{aligned}
\|\nabla_\Gamma u - \mathbf{f}\| &= \|\nabla \overline{u} - \boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle - \overline{\mathbf{f}}\| \\
&\leq \|\nabla \overline{u} - \overline{\mathbf{f}}\| + \|\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle\| \\
&\leq \|\nabla \overline{u} - \overline{\mathbf{f}}\| + \|\langle \boldsymbol{n}, \nabla \overline{u} \rangle\|.
\end{aligned}$$

This proves (1).

3

2. By using the definition of $\nabla_\Gamma \cdot$, we have

$$\|\nabla_\Gamma \cdot \mathbf{v} - \overline{g}\| = \|\nabla \cdot \mathbf{v} - \boldsymbol{n}^\top \nabla \overline{\mathbf{v}} \boldsymbol{n} - \overline{g}\|$$
$$\leq \|\nabla \cdot \mathbf{v} - \overline{g}\| + \|\boldsymbol{n}^\top \nabla \overline{\mathbf{v}} \boldsymbol{n}\|.$$

This proves (2).

3. By using the definition of Δ_Γ, we have

$$\Delta_\Gamma u - g = \nabla_\Gamma \cdot \nabla_\Gamma u - g$$
$$= \nabla \cdot (\nabla \overline{u} - \boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) - \boldsymbol{n}^\top \nabla (\nabla \overline{u} - \boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) \boldsymbol{n} - \overline{g}$$
$$= \Delta \overline{u} - \nabla \cdot (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) - \boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n} + \boldsymbol{n}^\top \nabla (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) \boldsymbol{n} - \overline{g}.$$

Note that, by the vector product rule, and $\nabla \cdot \boldsymbol{n} = -2H$,

$$\nabla \cdot (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) = \langle \boldsymbol{n}, \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \rangle + \langle \boldsymbol{n}, \nabla \overline{u} \rangle (\nabla \cdot \boldsymbol{n})$$
$$= \langle \boldsymbol{n}, \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \rangle - 2H \langle \boldsymbol{n}, \nabla \overline{u} \rangle.$$

And,

$$\boldsymbol{n}^\top \nabla (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) \boldsymbol{n} = \boldsymbol{n}^\top (\nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \boldsymbol{n}^\top + \langle \boldsymbol{n}, \nabla \overline{u} \rangle \nabla \boldsymbol{n}) \boldsymbol{n}$$
$$= \boldsymbol{n}^\top \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \boldsymbol{n}^\top \boldsymbol{n} + \boldsymbol{n}^\top \langle \boldsymbol{n}, \nabla \overline{u} \rangle \nabla \boldsymbol{n} \boldsymbol{n}$$
$$= \boldsymbol{n}^\top \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle + \langle \boldsymbol{n}, \nabla \overline{u} \rangle \boldsymbol{n}^\top \nabla \boldsymbol{n} \boldsymbol{n}$$
$$= \langle \boldsymbol{n}, \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \rangle.$$

Here, we use the fact that $\boldsymbol{n}^\top \boldsymbol{n} = 1$ and $\boldsymbol{n}^\top \nabla \boldsymbol{n} \boldsymbol{n} = 0$. The second equality is because the symmetric curvature tensor $\nabla \boldsymbol{n}$ is generated by the basis of Γ hence is orthogonal to $\boldsymbol{n}$. Combining these together, we have

$$\Delta_\Gamma u - g = \Delta \overline{u} - \nabla \cdot (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) - \boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n} + \boldsymbol{n}^\top \nabla (\boldsymbol{n} \langle \boldsymbol{n}, \nabla \overline{u} \rangle) \boldsymbol{n} - \overline{g}$$
$$= \Delta \overline{u} - \overline{g} - \boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n} - \langle \boldsymbol{n}, \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \rangle + 2H \langle \boldsymbol{n}, \nabla \overline{u} \rangle + \langle \boldsymbol{n}, \nabla \langle \boldsymbol{n}, \nabla \overline{u} \rangle \rangle$$
$$= \Delta \overline{u} - \overline{g} - \boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n} + 2H \langle \boldsymbol{n}, \nabla \overline{u} \rangle.$$

Therefore,

$$\|\Delta_\Gamma u - g\| \leq \|\Delta \overline{u} - \overline{g}\| + \|\boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n}\| + 2|H| \|\langle \boldsymbol{n}, \nabla \overline{u} \rangle\|$$
$$\leq \max(1, 2|H|)(\|\Delta \overline{u} - \overline{g}\| + \|\boldsymbol{n}^\top \nabla^2 \overline{u} \boldsymbol{n}\| + \|\langle \boldsymbol{n}, \nabla \overline{u} \rangle\|).$$

This proves (3)

□

Remark 2.1 *We point out that there is a more general result about (3) has been proved at [12] (Theorem 2.1 and Corollary 2.2). Our result presents above provide a much easier proof in $\mathbb{R}^3$, which can be extended to high order derivative cases.*

Additionally, if the $\mathbf{v}$ is a constant vector along normal, we have $\nabla_\Gamma \cdot \mathbf{v} = \nabla \cdot \mathbf{v}$, which is the equivalent principle. However, if we only have $\mathbf{v} \cdot \boldsymbol{n} = 0$, then

$$\nabla_\Gamma \cdot \mathbf{v} = \frac{1}{|\nabla \phi|} \nabla \cdot (|\nabla \phi| \mathbf{v}) \ on \ \Gamma,$$

where $\Gamma = \{\mathbf{x} \in \mathbb{R}^3 | \phi(\mathbf{x}) = 0\}$. This result has been proved in [21].

3 PINNs for time-evolving PDEs on surfaces

As shown in [23, 24], the PINN can solve a large class of PDEs. In this section, we first recap the PINN method for PDEs. Then, we propose a new algorithm to solve time-dependent PDEs on the surface.

3.1 PINNs for classic PDEs

Let $\mathcal{N}[\cdot]$ be a general differential operator defined on $\Omega \subset \mathbb{R}^d$, and Γ_b is a union of (inner or outer) boundaries of Ω. We consider the following PDE on Ω:

$$\begin{cases} \mathcal{N}[u(\mathbf{x})] & = f(\mathbf{x}) \quad \text{in } \Omega, \\ u(\mathbf{x}) & = u_b(\mathbf{x}) \quad \text{on } \Gamma_b, \end{cases} \tag{4}$$

where $u(\mathbf{x})$ is the solution of PDE, $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ is the variable of u, and f and u_b are the given right-hand side term and the boundary condition, respectively. As suggested in [23, 24], we use a fully connected neural network (NN) with N_l hidden layers to solve Eq. (4). Each hidden layer consists of $N_e(l)$ neurons, where l is the index of the hidden layer. The input layer consists of d neurons $x_1, x_2, \cdots, x_d$ which are the variables of the solution u, and the output layer consists of one neuron u.

After establishing an NN with input $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ and output u, we need to set up a loss function to train it. Here, we denote the prediction of the NN by u_h. Since we want u_h to satisfy Eq. (4), the loss function can be defined by the following mean square error (MSE):

$$\begin{aligned} MSE &= MSE_u + MSE_b \\ &= \frac{1}{N_u} \sum_{i=1}^{N_u} |\mathcal{N}[u_h(\mathbf{x}_u^i)] - f_u^i|^2 + \frac{1}{N_b} \sum_{i=1}^{N_b} |u_b^i - u_h(\mathbf{x}_b^i)|^2. \end{aligned}$$

Here, $\{\mathbf{x}_u^i, f_u^i\}$ are collocation points in Ω, $\{\mathbf{x}_b^i, u_b^i\}$ are training data on Γ_b, and N_u and N_b are the sizes of these points, respectively. The term $\mathcal{N}[u_h(\mathbf{x}_u^i)]$ can be computed through the automatic differentiation technique [26]. In Tensorflow, this can be performed using the function `tf.gradient`. Ideally, $MSE = 0$, which indicates taht u_h satisfies Eq. (4) exactly. Given these two sets of collocation points, we may train our NN for u and then get the prediction for each point in Ω, and this NN is called PINN.

3.2 The loss function for surface PDEs

In this subsection, we will use Theorem 2.1 to refine the loss function we have used in [14]. We begin with the time-independent problem $\Delta_\Gamma u = f$. In [14], the loss function is given by

$$MSE = \frac{1}{N_u} \sum_{i=1}^{N_u} |\Delta u_h(\mathbf{x}_u^i) - f_u^i|^2 + \frac{1}{N_u} \sum_{i=1}^{N_u} |\langle \nabla u_h(\mathbf{x}_u^i), \boldsymbol{n}(\mathbf{x}_u^i) \rangle|^2, \tag{5}$$

where $\{\mathbf{x}_u^i\}_{i=1}^{N_u}$ are the training points on Γ, and u_h is the solution by the PINN. However, Theorem 2.1 shows that this is not a good choice as we discussed in Section 2. Instead, Theorem 2.1 motivates

us to set the loss function as

$$MSE = \frac{1}{N_u} \sum_{i=1}^{N_u} |\Delta u_h(\mathbf{x}_u^i) - f_u^i|^2 + \frac{1}{N_u} \sum_{i=1}^{N_u} |\langle \nabla u_h(\mathbf{x}_u^i), \boldsymbol{n}(\mathbf{x}_u^i) \rangle|^2 + \frac{1}{N_u} \sum_{i=1}^{N_u} |\boldsymbol{n}(\mathbf{x}_u^i)^\top \nabla^2 u_h \boldsymbol{n}(\mathbf{x}_u^i)|^2. \quad (6)$$

With this setting, the residue of the surface PDE approaches zero as $MSE \to 0$. We note that, in practice, Eq.(5) works for most of the smooth surfaces, and Eq.(6) provides an indicator of PDEs' residue.

Now, we can handle the spatial derivative such as ∇_Γ, $\nabla_\Gamma \cdot$, and Δ_Γ. The next step is to consider the operator ∂_t. We follow the continuous-time and discrete-time method for time-dependent PDE in [23] but mofidy the spatial derivatives based on Theorem 2.1.

3.3 Continuous-time PINN for surface PDE

The most natural idea to solve the time-dependent problem is to deal with the time derivatives in the same manner as the spatial variable. Let us consider the following time-dependent surface PDE problem

$$\partial_t u(\mathbf{x}, t) = \mathcal{N}_\Gamma[u(\mathbf{x}, t)] \text{ for } (\mathbf{x}, t) \in \Gamma \times [0, T], \quad (7)$$
$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \text{ for } \mathbf{x} \in \Gamma, \quad (8)$$

where $\Gamma \subset \mathbb{R}^3$ is a closed surface, and $\mathcal{N}_\Gamma$ is a spatial differential operator on Γ, consisting of Δ_Γ, $\nabla_\Gamma \cdot$ and ∇_Γ. Although this assumption on $\mathcal{N}_\Gamma$ does not include all cases of the surface differential operators, it covers the most scenarios we focus on. For more complicated operators, such as Δ_Γ^2, we may develop a similar result, as shown in Theorem 2.1, and then establish the corresponding loss function accordingly. The key point is that the loss function should be an indicator for the residue of the surface PDE.

Base on the Theorem 2.1, we define the following loss function for (7):

$$
\begin{aligned}
MSE &= \frac{1}{N_u} \sum_{i=1}^{N_u} \left| \partial_t u_h(\mathbf{x}_u^i, t_u^i) - \mathcal{N}_\Gamma[u(\mathbf{x}_u^i, t_u^i)] \right|^2 + \frac{1}{N_u} \sum_{i=1}^{N_u} \left| \langle \nabla u_h(\mathbf{x}_u^i, t_u^i), \boldsymbol{n}(\mathbf{x}_u^i) \rangle \right|^2 \\
&\quad + \frac{1}{N_u} \sum_{i=1}^{N_u} \left| \boldsymbol{n}(\mathbf{x}_u^i)^\top \nabla^2 u(\mathbf{x}_u^i, t_u^i) \boldsymbol{n}(\mathbf{x}_u^i) \right|^2 + \frac{1}{N_0} \sum_{i=1}^{N_0} \left| u_h(\mathbf{x}_0^i, 0) - u_0(\mathbf{x}_0^i) \right|^2, \quad (9)
\end{aligned}
$$

where $\{\mathbf{x}_u^i, t_u^i\}_{i=1}^{N_u}$ are collocation points for the PDE, $\{\mathbf{x}_0^i\}_{i=1}^{N_0}$ are collocation points for the initial data, and $\boldsymbol{n}$ is the unit outer normal on Γ. According to Theorem 2.1, Eqs. (7) and (8) are enforced by minimizing Eq.(9).

There are several methods to generate collocation points on $\Gamma \times [0, T]$. In the numerical examples shown below, we used the following two methods. If the surface is defined by implicit equations and homeomorphic to sphere, for example, $\frac{x^2}{2} + y^2 + z^2 = 1$, we may generate Fibonacci lattice on the unit sphere, and then map them to the target surface. We then obtain collocation points on Γ. Next, we choose a set of partition points on $[0, T]$, and the collocation points $\{\mathbf{x}_u^i, t_u^i\}_{i=1}^{N_u}$ are given by the tensor product of the collocation points on Γ and the partition points on $[0, T]$. If the surface is given by the parametric form, we may use the Latin hypercube sampling (LHS) to generate a set of points $(\alpha_i, \beta_i, t_i)_{i=1}^{N_u}$. Next, we map (α_i, β_i) to the surface through the parametric equation of the surface. Then, we get the collocation points $\{\mathbf{x}_u^i, t_u^i\}_{i=1}^{N_u}$.

We can predict the solution for any $(\mathbf{x}, t) \in \Gamma \times [0, T]$ by training this PINN. But the training set is large because it includes all the data between $[0, T]$. This issue can be fixed by the discrete-time method.

3.4 Discrete-time PINN for surface PDE

Alternatively, the discrete-time method first discretize the time derivative by using high order Runge-Kutta methods. Then the PINN provides a sequence of function evaluations at different time partition points. We use Eq. (7) to describe the entire procedure. We follow the notations in [23], and the general form of q stages Runge-Kutta scheme of (7) is given by

$$
\begin{aligned}
u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \qquad i = 1, \cdots, q, \\
u^{n+1} &= u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}].
\end{aligned}
\tag{10}
$$

Here $u^{n+c_j}(\mathbf{x}) = u(\mathbf{x}, t^n + c_j \Delta t)$, is the hidden state of the system at time $t^n + c_j \Delta t$ for $j = 1, \cdots, q$. In this problem, we set $t^n = 0$ and $t^{n+1} = T$. We proceed by placing a multi-output neural network prior with input $\mathbf{x}$ and output

$$
(u^{n+c_1}, \cdots, u^{n+c_q}, u^{n+1}).
$$

The theoretical error estimate implies that the temporal error of this method is $O(\Delta t^{2q})$. Therefore, the prerequisite of this method is that $\Delta t < 1$, otherwise the solution may not converge. For $T \geq 1$, we can set a reference time $0 < \widetilde{T} < 1$, and apply the transform $\widetilde{t} = \frac{t}{T} \widetilde{T}$. The resulting PDE is

$$
\partial_{\widetilde{t}} u(\mathbf{x}, \widetilde{t}) = \frac{T}{\widetilde{T}} \mathcal{N}_{\Gamma}[u(\mathbf{x}, \widetilde{t})].
\tag{11}
$$

Then we can use the same method to handle it.

To design the loss function, we use the following notation:

$$
u_i^n = u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \qquad i = 1, \cdots, q,
\tag{12}
$$

$$
u_{q+1}^n = u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}].
\tag{13}
$$

Therefore, given the collocation points $\{\mathbf{x}_u^i, t_u^i\}_{i=1}^{N_u}$ as mentioned above, the loss function is given by

$$
MSE = \frac{1}{N_u} \sum_{j=1}^{q+1} \sum_{i=1}^{N_u} \left(|u_j^n(\mathbf{x}^i) - u_0(\mathbf{x}^i)|^2 + |\langle \nabla u_j^n(\mathbf{x}^i), \boldsymbol{n}(\mathbf{x}^i) \rangle|^2 + |\boldsymbol{n}(\mathbf{x}^i)^\top \nabla^2 u_j^n(\mathbf{x}^i) \boldsymbol{n}(\mathbf{x}^i)|^2 \right).
$$

By training this PINN, we will get the solution of (7)-(8) not only at the final time but also some intermediate time points. In this case, we cannot get the predicted solution directly for any given time $t \in [0, T]$ other than the Runge Kutta nodes. But we can use the interpolation method to recover it. Compared with the continuous-time method, this method saves time because we have fewer training points than the continuous-time method.

4   Numerical Experiments

In this section, we present numerical results using continuous- and discrete-time approaches to demonstrate the efficacy of our PINN method for surface PDE. All numerical experiments were conducted by using Tensorflow 1.13 on DGX-2 with NVIDIA Tesla V100. For the continuous-time method, we set up a PINN of 4 hidden layers with 100 neurons each layer, and 1 neuron for the output layer. For the discrete-time method, we set up a PINN of 4 hidden layers, with 200 neurons each layer and $q + 1$ neurons for the output layer, where q stages Runge-Kutta method has been used. We choose $\sigma(s) = \sin(\pi s)$ as the activation function [14, 15].

4.1   Continuous-Time Method

Consider the following heat equation

$$\partial_t u(\mathbf{x}, t) = \Delta_\Gamma u(\mathbf{x}, t) + f(\mathbf{x}, t) \text{ for } (\mathbf{x}, t) \in \Gamma \times [0, T], \tag{14}$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \text{ for } \mathbf{x} \in \Gamma. \tag{15}$$

We first test the accuracy of the continuous-time method. To this end, we set $u(\mathbf{x}, t) = x_1 \sin(tx_2) + x_3$ as the exact solution, where $\mathbf{x} = (x_1, x_2, x_3)$ and $\Gamma = \mathbb{S}^2$ is the unit sphere. We choose a suitable f to make the PDE hold, and set $u_0(\mathbf{x}) = u(\mathbf{x}, 0)$. To evaluate the performance of our algorithm, we define the relative error of prediction u_h by PINNs. Let u be the exact solution, and $\{\mathbf{x}_c^i\}_{i=1}^{N_c}$ be a set of sample points for the accurcy test, which differs from training set $\{\mathbf{x}_u^i\}_{i=1}^{N_u}$. Then, the relative error Err is given by

$$Err = \frac{\sqrt{\sum_{i=1}^{N_c} |u_h(\mathbf{x}_c^i) - u(\mathbf{x}_c^i)|^2}}{\sqrt{\sum_{i=1}^{N_c} |u(\mathbf{x}_c^i)|^2}}.$$

We generate $50,000$ collocation points on $\Gamma \times [0, 1]$ by using the tensor product of Fibonacci lattice, as mentioned above.

Figure 1 shows the results at different time. We can see that the relative error is around 10^{-2}. Of note, in this example, if we drop out the $\boldsymbol{n}\nabla^2 u\boldsymbol{n}$ term in the loss function, the PINN will converge slightly faster. This is probably because Γ is a very smooth surface with a constant curvature. This also enlightens us to adjust the weight of this second order term in the loss function, which is currently 1. However, the pattern of how this weight affects the convergence is unclear at this time and will remain as future work.

Next, we apply our method to a benchmark problem named heating a torus [18]. Let

$$\Gamma = \left\{ \mathbf{x} \in \mathbb{R}^3 \left| \left(\sqrt{x_1^2 + x_2^2} - 1 \right)^2 + x_3^2 = \frac{1}{16} \right. \right\},$$
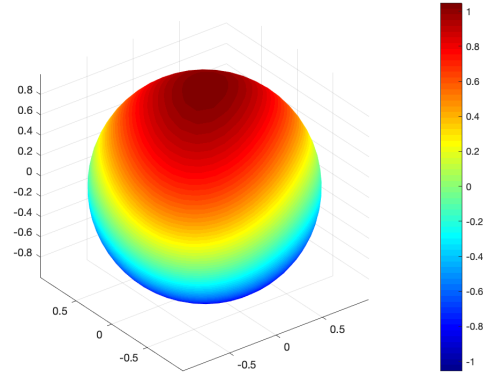
with the right-hand side being a regularized version of the characteristic function

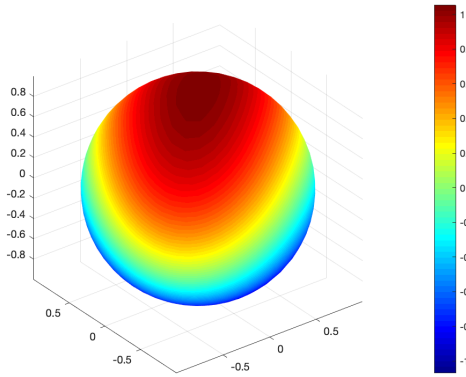$$f(\mathbf{x}, t) = 100\chi_G(\mathbf{x}), \qquad \mathbf{x} \in \Gamma,$$

where $x \in \Gamma | |\mathbf{x} - (0, 1, 0)| \le 0.25\}$ and χ_G is the characteristic function on G. The initial condition is set as $u = 0$. We generate $50,000$ collocation points by using LHS as mentioned above. The results are shown in Figure 2.
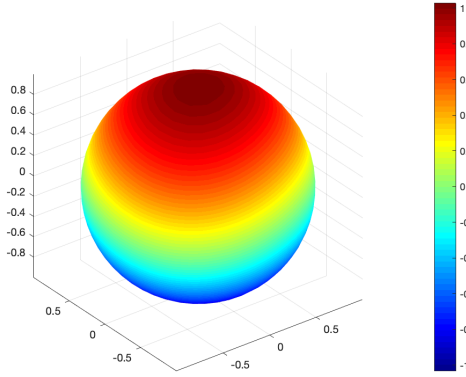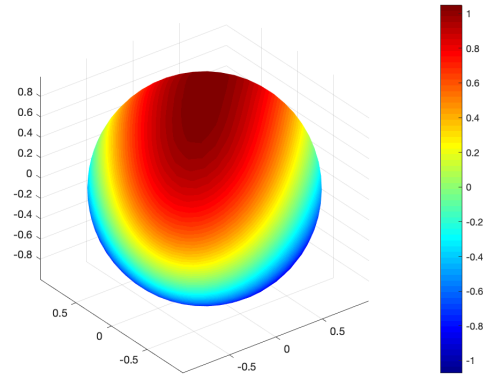
(a) Time: 0.25, $Err = 3.098151e - 02$

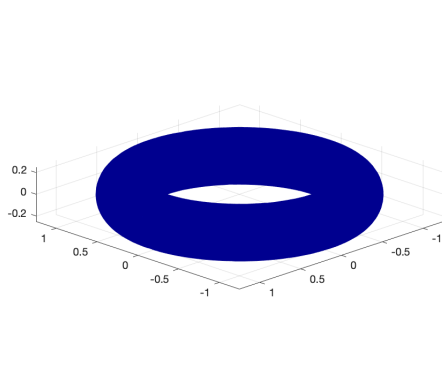(b) Time: 0.5, $Err = 5.168469e - 02$
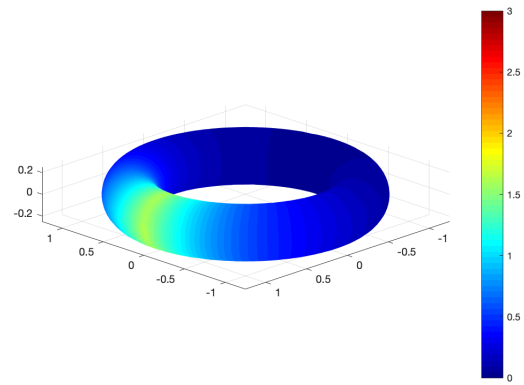
(c) Time: 0.75, $Err = 6.373482e - 02$

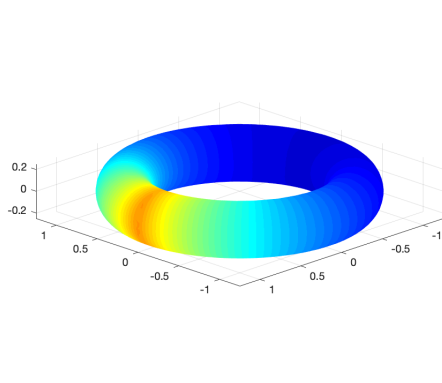(d) Time: 1.0, $Err = 6.761617e - 02$

Figure 1: Error of continuous-time method for different time steps.
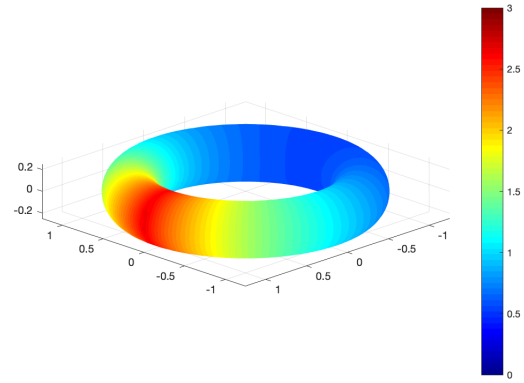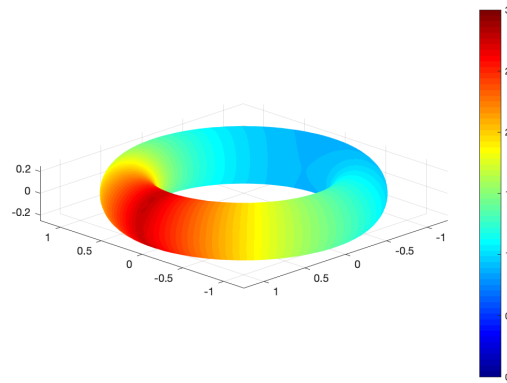
(a) Time: 0.0

(b) Time: 0.75

(c) Time: 1.5

(d) Time: 2.25

(e) Time: 3.0

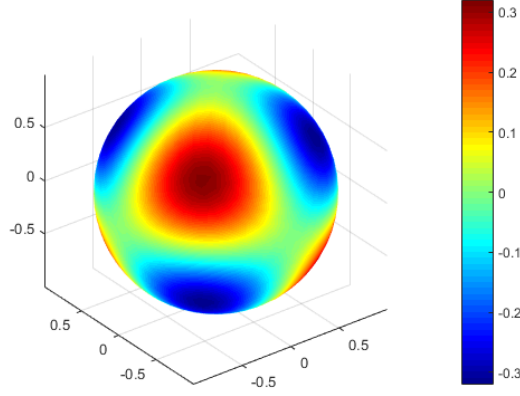Figure 2: Heating a torus. The color indicates the magnitude of the solution. Viewpoint: $(-1, 1, 0.5)$.

Figure 3: Discrete-time method with $T = 0.5$. $Err = 1.467495e - 02$.

4.2 Discrete-Time Method

In this subsection, we test the discrete-time method. We first set $0.5 = T < 1$ and reconsider (14)-(15) on $\Gamma = \mathbb{S}^2$. For the discrete-time method, we only need collocation points on Γ as the time derivative has already been discretized by the Runge-Kutta method. Therefore, 500 Fibonacci lattices on $\mathbb{S}^2$ has been generated as the collocation points. This number is 100 times smaller than that of the continuous-time method. Thus, it saves memory and computational time significantly. A 100 stages Runge-Kutta method has been used. The theoretical result indicates that the time error of this $0.5^{200} \approx 6.223015278 \times 10^{-61}$, which is much smaller than machine precision. We set the exact solution as $u(\mathbf{x}, t) = x_1 x_2 x_3 \exp(t)$.

The result of $u(\mathbf{x}, T)$ is shown in Figure 3. As we can see, the relative error is of the same order as the continuous-time method, because in this case, the spatial PINN error dominates the total error.

Next, we set $3 = T > 1$ to verify this method works after the time rescaling. We choose $\widetilde{T} = 0.5$ and transform $[0, T]$ to $[0, \widetilde{T}]$, then we apply our discrete-time method to (11). We set our exact solution $u(\mathbf{x}, t)$ as $x_1 x_2 x_3 \exp(t)$ and $x_1 \sin(tx_2) + x_3$, respectively. All the other settings keep the same. The result of $u(\mathbf{x}, T)$ is shown in Figure 4, whose relative error is still around 10^{-2}.

5 Conclusion

In this paper, we put forward the PINN method for time-evolving PDEs on surfaces. A simplified proof of the prior estimate of the differential operator on the surface has been shown in theorem 2.1. To deal with the temporal differential operator, we come up with the continuous-time method and discrete-time method. The continuous-time method is more flexible and straightforward. Although, in general, there is no restriction on the length of the time interval. But this method will take more sample points and then more time to train the PINN.

On the other hand, the discrete-time method needs a relatively small collocation points set and, hence, less training time. But it requires the time interval less than 1 due to the error analysis of the Range Kutta scheme. The numerical experiments verified our algorithm.
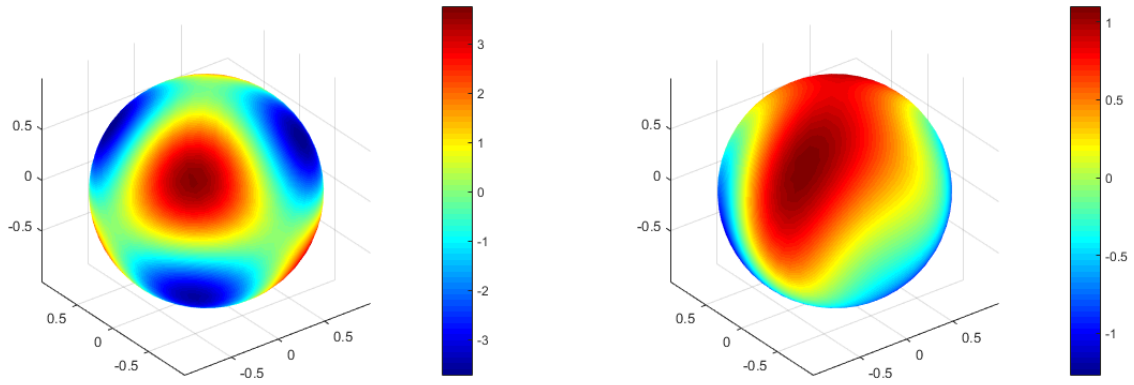
(a) $u(\mathbf{x}, t) = x_1 x_2 x_3 \exp(t)$. $Err = 5.967653e - 02$

(b) $u(\mathbf{x}, t) = x_1 \sin(t x_2) + x_3$. $Err = 8.814725e - 02$

Figure 4: Discrete-time method with $T = 3$.

References

[1] Turk, G., 1991, July. *Generating textures on arbitrary surfaces using reaction-diffusion.* In ACM SIGGRAPH Computer Graphics (Vol. 25, No. 4, pp. 289-298). ACM.

[2] Bertalmio, M., Bertozzi, A.L. and Sapiro, G., 2001, December. *Navier-Stokes, fluid dynamics, and image and video inpainting.* In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001 (Vol. 1, pp. I-I). IEEE.

[3] Tian, L., Macdonald, C.B. and Ruuth, S.J., 2009, November. *Segmentation on surfaces with the closest point method.* In 2009 16th IEEE International Conference on Image Processing (ICIP) (pp. 3009-3012). IEEE.

[4] Biddle, H., von Glehn, I., Macdonald, C.B. and März, T., 2013, September. *A volume-based method for denoising on curved surfaces.* In 2013 IEEE International Conference on Image Processing (pp. 529-533). IEEE.

[5] Murray, J.D., 2003. II. *Spatial Models and Biomedical Applications.* Springer.

[6] Olsen, L., Maini, P.K. and Sherratt, J.A., 1998. *Spatially varying equilibria of mechanical models: Application to dermal wound contraction.* Mathematical biosciences, 147(1), pp.113-129.

[7] Toga, A.W. ed., 1998. *Brain warping.* Elsevier.

[8] Elliott, C.M. and Stinner, B., 2010. *Modeling and computation of two phase geometric biomembranes using surface finite elements.* Journal of Computational Physics, 229(18), pp.6585-6612.

[9] Halpern, D., Jensen, O.E. and Grotberg, J.B., 1998. *A theoretical study of surfactant and liquid delivery into the lung.* Journal of Applied Physiology, 85(1), pp.333-352.

[10] Dziuk, G., 1988. *Finite elements for the Beltrami operator on arbitrary surfaces.* In Partial differential equations and calculus of variations (pp. 142-155). Springer, Berlin, Heidelberg.

[11] Bertalmio, M., Cheng, L.T., Osher, S. and Guillermo, S., 2000. *Variational problems and partial differential equations on implicit surfaces: The framework and examples in image processing and pattern formation.*

[12] Cheung, K.C. and Ling, L., 2018. A kernel-based embedding method and convergence analysis for surfaces PDEs. SIAM Journal on Scientific Computing, 40(1), pp.A266-A287.

[13] Yang, Y. and Perdikaris, P., 2019. Adversarial uncertainty quantification in physics-informed neural networks. Journal of Computational Physics, 394, pp.136-152.

[14] Fang, Z. and Zhan, J., 2019. A Physics-Informed Neural Network Framework For Partial Differential Equations on 3D Surfaces: Time Independent Problems. IEEE Access.

[15] Fang, Z. and Zhan, J., 2019. Deep Physical Informed Neural Networks for Metamaterial Design. IEEE Access, 8, pp.24506-24513.

[16] Kharazmi, E.H.S.A.N., Zhang, Z. and Karniadakis, G.E., 2019. Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. arXiv preprint arXiv:1912.00873.

[17] Kharazmi, E., Zhang, Z. and Karniadakis, G.E., 2020. hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition. arXiv preprint arXiv:2003.05385.

[18] Dziuk, G. and Elliott, C.M., 2013. Finite element methods for surface PDEs. Acta Numerica, 22, pp.289-396.

[19] Petras, A. and Ruuth, S.J., 2016. *PDEs on moving surfaces via the closest point method and a modified grid based particle method.* Journal of Computational Physics, 312, pp.139-156.

[20] Petras, A., Ling, L. and Ruuth, S.J., 2018. An RBF-FD closest point method for solving PDEs on surfaces. Journal of Computational Physics, 370, pp.43-57.

[21] Deckelnick, K., Dziuk, G., Elliott, C.M. and Heine, C.J., 2010. An h-narrow band finite-element method for elliptic equations on implicit surfaces. IMA Journal of Numerical Analysis, 30(2), pp.351-376.

[22] Petras, A., Ling, L., Piret, C. and Ruuth, S.J., 2019. A least-squares implicit RBF-FD closest point method and applications to PDEs on moving surfaces. Journal of Computational Physics, 381, pp.146-161.

[23] Raissi, M., Perdikaris, P. and Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, pp.686-707.

[24] Raissi, M., 2018. Deep hidden physics models: Deep learning of nonlinear partial differential equations. The Journal of Machine Learning Research, 19(1), pp.932-955.

[25] Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).

[26] Baydin, A.G., Pearlmutter, B.A., Radul, A.A. and Siskind, J.M., 2018. Automatic differentiation in machine learning: a survey. Journal of Marchine Learning Research, 18, pp.1-43.

[27] Goodfellow, I.J., Shlens, J. and Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[28] Rastegari, M., Ordonez, V., Redmon, J. and Farhadi, A., 2016, October. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision (pp. 525-542). Springer, Cham.

[29] Gawehn, E., Hiss, J.A. and Schneider, G., 2016. Deep learning in drug discovery. Molecular Informatics, 35 (1), 3–14.

[30] Alipanahi, B., Delong, A., Weirauch, M.T. and Frey, B.J., 2015. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. Nature biotechnology, 33(8), p.831.

[31] NVDIA SimNET `https://developer.nvidia.com/simnet`