IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 10, 2019, accepted December 22, 2019, date of publication December 31, 2019, date of current version February 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2963390

A Physics-Informed Neural Network Framework for PDEs on 3D Surfaces: Time Independent Problems

ZHIWEI FANG[1] AND JUSTIN ZHAN[2]
[1]Department of Mathematical Sciences, University of Nevada at Las Vegas, NV 89154-4020, USA
[2]Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA

Corresponding author: Justin Zhan (jzhan@uark.edu)

This work was supported in part by the United States Department of Defense under Grant W911NF1910482, and in part by the Arkansas Research Alliance Grant DN145250.

ABSTRACT Partial differential equations (PDEs) on surfaces are ubiquitous in all the nature science. Many traditional mathematical methods has been developed to solve surfaces PDEs. However, almost all of these methods have obvious drawbacks and complicate in general problems. As the fast growth of machine learning area, we show an algorithm by using the physics-informed neural networks (PINNs) to solve surface PDEs. To deal with the surfaces, our algorithm only need a set of points and their corresponding normal, while the traditional methods need a partition or a grid on the surface. This is a big advantage for real computation. A variety of numerical experiments have been shown to verify our algorithm.

INDEX TERMS PINN, PDEs on surfaces, Laplace-Beltrami operator.

I. INTRODUCTION

Partial differential equations (PDEs) on manifold in $\mathbb{R}^d$ arise in mathematical models for many nature phenomena. Image processing applications include the mapping an image on a given surface [1], the recovery of lost information on a surface [2] and the segmentation and deciphering of images on surfaces [3], [4]. It also widely used in biological and medical science. For example, simulation of animal coats [5], wound healing [6], brain wrapping [7], lipid interactions in biomembranes [8], and fluids in lungs [9]. In computer vision, this tool has been used in real time fluid visualization on surfaces [10] and vector field visualization [11].

Thus, many mathematical methods have been developed to deal with this problem. Way back in 1988, Dziuk [12] first came up with the analysis of finite element method based on surface triangulations. In [13], the authors avoid the surface triangulation and remeshing by extending the surface PDEs to a subset of $\mathbb{R}^d$ with a positive measure. However, this operation makes the extended PDEs degenerate. A review of finite element methods for surface PDEs can be found in [14]. Recently, Petras *et al.* [15], [16], and Petras and Ruuth *et al.* [19], [20] developed a systematic way to solve surface PDEs by using RBF-FD combined with a grid particle method. These methods reduce the computational workload tremendously, but still need a grid in the neighbourhood around the surface. And, extension of the solution in this neighbourhood has also been introduced as in [18].

Based on those drawbacks of traditional numerical methods, we cast our eyes on machine learning methods. With the growth of computing power, machine learning becomes the most popular topic in all the scientific area [19]. As Raissi, *et al.* [20] came up with physics-informed neural networks (PINNs) at 2019, there are explosive applications in mathematical and engineering area. In stochastic analysis and uncertainty quantification, Generative Adversarial Networks (GANs) has been combined with PINNs to solve PDEs with random inputs as shown in [21]. Stochastic inverse problem also has been studied in [22]. Numerical methods for fractional order PDEs has been studied in most recent work [23], [24]. We hence believe PINNs has the ability to conquer the difficulty of surface PDEs.

In this paper, we adopt the PINNs to solve the PDEs posed on surfaces. The mathematical principle for this method is the equivalence principle used in [15]. Then, we solve the modified PDEs with a normal constrain, which is much easier than the original surface PDEs. We apply our method to a high dimensional case. It worth mention that, our method do not need any partition or grid. The only information we need about the surface is a set of points on it, as well as their corresponding normal.

This paper is organized as follows. In section II, we briefly introduce the mathematical preliminary of surface PDEs. PINNs for traditional PDEs and our algorithm for surface

The associate editor coordinating the review of this manuscript and approving it for publication was Kathiravan Srinivasan.

PDEs have been shown in section III. Then, a variety of numerical experiments has been shown to verify our algorithm in section IV. We conclude this paper in section V.

II. PARTIAL DIFFERENTIAL EQUATIONS ON SURFACES

In this section, we briefly introduce the elementary definitions of PDEs on surfaces. For advance and detail of calculus on surfaces and manifolds, and PDEs theory on them, we recommend [14].

A. DIFFERENTIAL OPERATORS ON SURFACES

Let Γ be a smooth surface embedded in $\mathbb{R}^3$ with 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 $\overline{u}|_\Gamma = u$. The ∇, $\nabla\cdot$ and Δ denote ordinary gradient, divergence and Laplace operator in $\mathbb{R}^3$. Now, we can define the gradient, divergence and Laplace operator on Γ for u, which are essential for PDEs on surfaces.

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

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

Then, the divergence on Γ is defined as:

$$\nabla_\Gamma \cdot u = D_1 u + D_2 u + D_3 u$$

Definition 3: Let $u : \Gamma \mapsto \mathbb{R}$ has 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 surface is also known as Laplace-Beltrami operator.

B. THE EQUIVALENCE PRINCIPLES

Instead of solving surface PDEs directly, we solve the ordinary PDEs on the surfaces with a constrain. The idea of our methods comes from the following lemma [15]:

Lemma 4: Let u be any function on $\mathbb{R}^n$ that is constant along normal directions of Γ. Then, at the surface, intrinsic gradients are equivalent to standard gradients, say,

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

Let 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,

$$\nabla_\Gamma \cdot v = \nabla \cdot v$$

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

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

In the next subsection, we will elucidate our idea by using a surface PDE as an example.

C. PDES ON SURFACES

Now, we can consider the PDEs on surfaces. For classical PDEs defined on $\mathbb{R}^2$, $\mathbb{R}^3$ or their subdomains, we usually consider the following well-known PDEs:

$$\begin{aligned} -\Delta u &= f && \text{(Poisson's equation)}\\ u_t - \Delta u &= f && \text{(Heat equation)}\\ u_{tt} - \Delta u &= f && \text{(Wave equation)} \end{aligned}$$

We now may generalized these equations on a general surface Γ:

$$\begin{aligned} -\Delta_\Gamma u &= f && \text{(Poisson's equation)}\\ u_t - \Delta_\Gamma u &= f && \text{(Heat equation)}\\ u_{tt} - \Delta_\Gamma u &= f && \text{(Wave equation)} \end{aligned}$$

In this paper, we only consider the time independent problems. Namely, the surfaces are static and equations without time derivatives.

We now consider the following PDE on a closed Γ:

$$\Delta_\Gamma u + u = f \tag{1}$$

which is known as elliptic problem. Since Γ is closed, there is no boundary condition imposed. The solution u is purely derived by f. In time dependent problems, the solution can also be driven by the surface Γ. That is, even though $f = 0$, we may get non-trivial solution [14]. Since compute $\Delta_\Gamma u$ directly will be complicate, we then, by using the idea comes up with in the last subsection, solve the following equivalent problem instead:

$$\begin{cases} \Delta u + u = f \\ \langle\nabla u, \boldsymbol{n}\rangle = 0 \end{cases} \tag{2}$$

By the lemma 4, (1) and (2) are equivalent. That is, they share the exactly same solution. Notice that in the equivalent problem (2), we erase all the surface differential operator by using equivalence principle. Hence, only the computation of ordinary differential operators Δ and ∇ are needed, which are much easier. In the sections below, we will solve this problem by using PINNs via a least square approach.

III. PINNS FOR PDES ON SURFACES

As shown in [20], [25], the PINNs can solve a wide class of PDEs. In this section, we will recap the PINNs technology for PDEs. And then, we will come up with our idea to solve time independent PDEs' problems on surfaces.

A. PINNS FOR 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. Then, let us consider the following general PDEs problem on Ω:

$$\begin{cases} \mathcal{N}[u(x)] = f(x) & \text{in } \Omega \\ u(x) = u_b(x) & \text{on } \Gamma_b \end{cases} \tag{3}$$

where $u(x)$ is the latent (hidden) solution of PDE, $x = (x_1, x_2, \ldots, x_d)$ is the variable of u, and f and u_b are the given right hand side term and boundary condition, respectively. Our target now is solving (3) by using PINNs.

As suggested in [20], [25], we use a fully connected neural network (NN) with N_l hidden layers. Each hidden layer consist of $N_e(l)$ neurons, where l is the index of hidden layer, while input layer consists of d neurons $x_1, x_2, \ldots, x_d$, which are the variables of solution u. And, the output layer consists of one neuron u.

Hence, we established a NN with input $x = (x_1, x_2, \ldots, x_d)$ and output u. Now, we are going to setup the loss function so that we can train this NN. For the sake of simplicity, let us denote the prediction of the NN by u_h. Since we want u_h satisfies equation (3), the loss function has been defined by the following mean square error (MSE):

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

Here $\{x_u^i, f_u^i\}$ are training data for PDE restriction in (3), and $\{x_b^i, u_b^i\}$ are training data for boundary condition in (3). N_u and N_b are the sizes of those training data, respectively. The $\mathcal{N}[u_h(x_u^i)]$ can be computed by automatic differentiation technique [26]. In Tensorflow, this can be done by the function `tf.gradient`. In the ideal case, say $MSE = 0$, the u_h satisfies the equation (3) exactly. Given these two sets of training data, we may train our NN for u and then get the prediction for each point in Ω. This solves our PDE problem (3). We call this NN as PINN.

B. NETWORK STRUCTURE AND STABILITY OF PINNS

One question for solving PDEs by PINNs is: what is the best network structure to solve a PDE? In [20], [25], the authors tried different sample sizes and network structures, even add a small noise on the data. It turns out, when the sample size, number of hidden layers and number of neurons in each layer are increasing in a reasonable range, the predictive accuracy is almost increasing. Even though the data has been polluted by a small noise, the PINNs can recover the solution of PDEs with a satisfied accuracy.

Once a new algorithm for numerical PDEs is developed, the natural questions from mathematician are: what is the advantage of the new algorithm? Is this method stable? As we all know, there are a plenty of numerical methods for solving PDEs and PDEs on surfaces, such as, finite difference methods, finite element methods, spectral methods, etc. However, all of these methods required a mesh or grid on the PDEs' domain and surface, which is complicate or even impossible for high dimensional problems. The PINNs, on the other hand, can be generalized to high dimension by just change the number of neurons for input layer. The meshless numerical methods, such as radial basis function methods, has been proved unstable for direct method [15].

Although there is no theoretical proof for the stability of PINNs, we may explain it by the following way. As shown in [20], the sin function is always be the best activation function among the commonly used activation function. In this case, our output of PINNs can be understood as a generalized Fourier series approximation for solution. If we only have one hidden layer (shallow NN), the PINNs is exactly the sine series. And the training procedure is actually finding the least square solution of the generalized Fourier approximation. So, the stability and accuracy theory of least square Fourier approximation may partly explain the the stability and accuracy of PINNs, and this has been proved by lots of experiments in [20], [25]. The stability and accuracy theory of PINNs will be a significant future work. At this time, we believe the PINNs give the reliable prediction of PDEs' solution, and we verify it by the cross-validation as explained below. Furthermore, by using PINNs, we can recover the unknown coefficients with additional data. This inverse problem is always computational expensive. But PINNs is a stable and easy solver for this inverse problem [20]. Therefore, the PINNs is competitive among these numerical methods.

C. PINNS FOR PDES ON 3D SURFACES

In this subsection, we come up with our algorithm for PDEs on 3D surfaces based on PINNs. In this work, we only consider the time independent PDEs on static surfaces. Mostly, these are elliptic PDEs. To illustrate our algorithm, let us consider the following general PDEs on 3D surfaces:

$$\begin{cases} \mathcal{N}_\Gamma[u(x)] = f(x) & \text{on } \Gamma \\ u(x) = u_b(x) & \text{on } \Gamma_b \end{cases} \tag{4}$$

where $\mathcal{N}_\Gamma$ is a differential operator on Γ, and all the other notations keep the same meaning. In this case, the domain of surface PDEs is 2 dimensional. However, instead of imposing the restriction of $\mathcal{N}_\Gamma$, we use

$$\begin{cases} \mathcal{N}[u(x)] = f(x) \\ \langle \nabla u, \boldsymbol{n} \rangle = 0 \end{cases} \tag{5}$$

on Γ, where $\mathcal{N}$ means the 3D extension of $\mathcal{N}_\Gamma$. We will explain this by using concrete example in the next section. We claim (4) and (5) are equivalent. This is because, by applying the equivalence principle in lemma 4, surface differential operators, such as ∇_Γ, Δ_Γ and $\nabla_\Gamma\cdot$, can be replaced by ∇, Δ and $\nabla\cdot$, respectively. This will reduce the workload tremendously as it is not easy to compute the surface differential operators, where first and second fundamental forms of surfaces get involved in. But in our method, only the ordinary differential operators in $\mathbb{R}^3$ need to be computed.

We hence use a PINN illustrated in the previous subsections to predict the solution of (4). The input layer consist of 3 neurons which are the variables in $\mathbb{R}^3$, say, $x = (x_1, x_2, x_3)$. Then the output is the solution of (4). Similarly, we denote the prediction of latent variable by u_h. Then the $\mathcal{N}[(u_h(x))]$ and ∇u_h can be computed by automatic differentiation. Therefore, (5) is computable, and the loss function is given by

$$\begin{aligned} MSE &= MSE_u + MSE_b + MSE_g \\ &= \frac{1}{N_u}\sum_{i=1}^{N_u} |\mathcal{N}[u_h(x_u^i)] - f_u^i|^2 \\ &\quad + \frac{1}{N_b}\sum_{i=1}^{N_b} |u_b^i - u_h(x_b^i)|^2 \\ &\quad + \frac{1}{N_u}\sum_{i=1}^{N_u} |\left\langle \nabla u_h(x_u^i), \boldsymbol{n}(x_u^i)\right\rangle|^2 \end{aligned} \tag{6}$$

All the notations have same meaning for the traditional PINNs introduced in the previous subsections. Notice that we use the same training set for first and third term as they come from one equation $\mathcal{N}_\Gamma[u(x)] = f(x)$ in (4). By training this PINN, we can get the prediction of u.

Remark 5: We point that as a closed surface has no boundary, the boundary condition $u(x) = u_b$ for $x \in \Gamma_b$ in (4) will be vanished. In this case, the loss function only contains two terms. Namely, $MSE = MSE_u + MSE_g$.

We conclude this section by pointing out the advantage of our method. Note that our methods only require a batch of sample points and their corresponding normal on the surface, and no partition and connectivity information about the surface, such as first and second fundamental forms, is needed. Therefore, our method is based on a points cloud information. This is a big advantage compared with traditional methods where the Lagrangian and Euclidean information may needed [17]. As the finite element methods for surface PDEs, the first step is generate partitions on the surface or the neighborhood of the surface (bulk finite element methods), which is complicate and will cause error between the mesh and original surface. And then, one should figure out the weak formulation of surface PDEs, which contains involved mathematical inference. While our method only relies on points cloud on the surface so there is no error arising from the surface. Additionally, our method based on the equivalence principle as shown in lemma 4, thus is much easier. In the next section, we will show several numerical experiments to verify our method.

IV. NUMERICAL EXAMPLES

In this section, we mainly consider the following elliptic equation:

$$\Delta_\Gamma u + au = f \quad \text{on } \Gamma \tag{7}$$

where a is a constant and Γ is usually a closed surface. When Γ is open, boundary condition on $\partial\Gamma$ should be added. In this case, $\mathcal{N}_\Gamma[u(x)] = \Delta_\Gamma u(x) + au(x)$.

All the experiments below have been done by Tensorflow 1.8 on a Dell workstation with Intel(R) Xeon(R) CPU E5-1630 v4 at 3.70GHz and NVIDIA GeForce GTX 1080 GPU.

A. EXEMPLARY EXPERIMENT FOR LAPLACE-BELTRAMI EQUATION ON 3D SPHERE

We illustrate our algorithm in this concrete example. In this example, we solve (7) with $a = 0$ on a 3D unit sphere $\mathbb{S}^2$. We set $f = 18x_1x_2x_3$ so that the exact solution of (7) is $u(x) = x_1x_2x_3$. To generate the training data, we first generate sample points on $\mathbb{S}^2$ by Fibonacci lattices rule [27]–[29]. This is a structured sample points on $\mathbb{S}^2$. As shown in [27]–[29], this method has an amazing performance in numerical computation, especially for quasi Monte Carlo method. We hence use this method instead of uniform random sampling methods. However, we also point out that, in a general surface, where Fibonacci lattices is not applied, we should use other sampling methods such as uniform random sampling methods. No matter what method we use, once the sample points has been generated, we may also compute the corresponding normal for each sample point. If the equation of the surface is available, we may use its closed form equation or formula to find out the normal. In the case that the equation of the surface is not available, we may use numerical methods, such as local quadratic interpolation [30].

After generating N_u sample points $\{x_u^i\}_{i=1}^{N_u}$ on $\Gamma = \mathbb{S}^2$, we may compute $f_u^i = f(x_u^i)$ for each sample x_u^i. Then, according to the last section, the loss function of this example reads

$$\begin{aligned} MSE &= MSE_u + MSE_g \\ &= \frac{1}{N_u}\sum_{i=1}^{N_u} |\Delta u_h(x_u^i) - f_u^i|^2 \\ &\quad + \frac{1}{N_u}\sum_{i=1}^{N_u} |\left\langle \nabla u_h(x_u^i), \boldsymbol{n}(x_u^i)\right\rangle|^2 \end{aligned}$$

The MSE_b term vanished because there is no boundary on $\mathbb{S}^2$ (closed surface). In (7), the differential operator $\mathcal{N}_\Gamma = \Delta_\Gamma$. Hence, the corresponding 3D extension is $\mathcal{N} = \Delta$. As our most recent work [31], we mentioned that the activation function $\sigma(s) = \sin(\pi s)$ is more powerful than $\sigma(s) = \sin(s)$ that suggested in [20]. That is, $\sigma(s) = \sin(\pi s)$ can deal with much extremely case, such as high frequency PDEs' problems. This is probably because, after batch normalization, the data locate in $[-1, 1]$, and $\sigma(s) = \sin(\pi s)$ is a surjective in this interval. Therefore, we choose $\sigma(s) = \sin(\pi s)$ in our experiments. All the parameters in the PINNs have been initialized by Xavier [32].

By setting $N_u = 2,000$, we train our PINNs for surface PDEs by $10,000$ steps via Adam algorithm. After that, we use the L-BFGS packed in Scipy package continues the

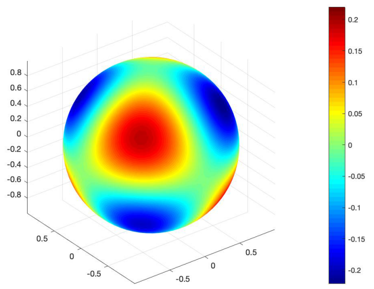

FIGURE 1. Solution of Laplace-Beltrami equation on $\mathbb{S}^2$. Exact solution: $u = x_1x_2x_3$.

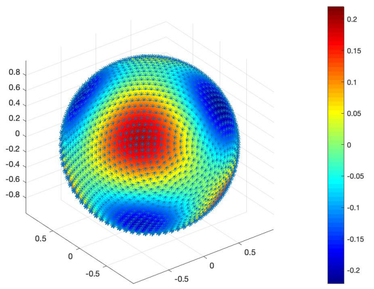

FIGURE 2. Solution of Laplace-Beltrami equation on $\mathbb{S}^2$ with sample points (marked as stars) Exact solution: $u = x_1x_2x_3$.

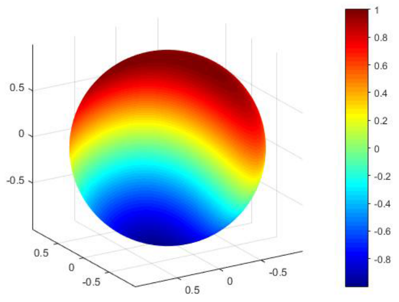

FIGURE 3. Solution of Laplace-Beltrami equation on $\mathbb{S}^2$. Exact solution: $u = x_1 \sin(x_2) + x_3$.

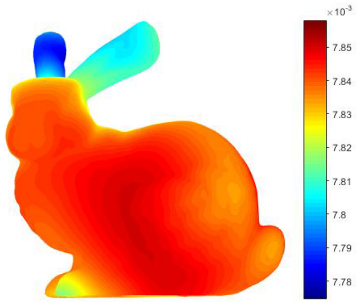

FIGURE 4. Solution of Laplace-Beltrami equation on the surface of a bunny (front).

optimization until the absolute value of difference between loss functions in the consecutive two steps less than 10^{-16}. The solution is shown in Figure 1. To show the Fibonacci lattices, we also display the solution with sample points which is marked as stars in Figure 2.

To evaluate this result, we define the relative error of prediction u_h given by PINNs. Let u be the exact solution, and $\{x_c^i\}_{i=1}^{N_c}$ be a set of sample points, which is differ from training set $\{x_u^i\}_{i=1}^{N_u}$, for cross-validation. Then, the relative error Err is given by

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

Essentially, this is discrete relative L^2 norm which widely used in numerical PDEs. Hence, we generate $N_c = 2,500$ points randomly on $\mathbb{S}^2$ and then get the prediction of $u_h(x_c^i)$ on those points via the trained PINN. In our experiment, $Err = 1.109493e - 02$. This result justifies our algorithm. We will study the relationship between the accuracy and structure in the experiments below.

Since $u(x) = x_1x_2x_3$ is an eigenfunction of Laplace-Beltrami operator on $\mathbb{S}^2$ (with eigenvalue 18), we also do the same test with different exact solution $u(x) = x_1 \sin(x_2) + x_3$ to verify the correctness of our algorithm. All the settings keep the same. The result is shown in Figure 3 with $Err = 2.354291e - 02$.

Compared with traditional numerical algorithms, such as radial basis finite difference method [15] and finite element methods [14], our method is much simpler to implement and more flexible to design new algorithms. This is because we do not need any partition or grid on the surface, which are important ingredients of traditional methods. All we need is a set of points on the surface and their corresponding normal.

B. LAPLACE-BELTRAMI EQUATION ON MORE COMPLEX GEOMETRY

In this experiment, we display the result of a benchmark problem. We consider (7) with $a = 0$ on a surface of bunny. The original surface data of the bunny consist of 34835 points, which is available at www.numerical-tours.com. We randomly choose 20,000 points as our training data. The rest will be used for cross-validation. The normal of these points will computed by local quadratic interpolation. In this experiment, the right hand side function $f(x) = x_1 \sin(x_2) + x_3$. Notice there is no closed form exact solution for this problem, we hence evaluate our result by the loss function value of cross-validation. Actually, the loss function is an ideal indicator for the residue of the PDE. This is because, when the loss function $MSE = 0$, the PDE is satisfied exactly on the test points. The result is shown in Figure 4 (front) and 5 (back). The loss function value is $5.389461e - 5$.

C. LAPLACE-BELTRAMI EQUATION ON OPEN SURFACE

In this experiment, we justify our algorithm with an open surface. Notice the only difference between the closed and open surface is if there is a boundary of the surface. We consider (7) with $a = 0$ on upper unit hemisphere Γ. Thus, the boundary of the surface is the equator. In this case, (6) will

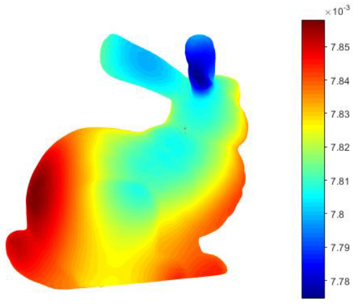

FIGURE 5. Solution of Laplace-Beltrami equation on the surface of a bunny (back).

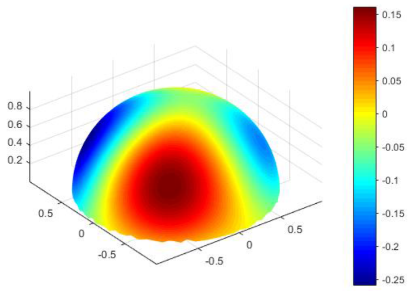

FIGURE 6. Solution of Laplace-Beltrami equation on upper unit hemisphere.

be considered as the loss function. We still use the Fibonacci lattices rule on Γ with $N_u = 2,000$. For the boundary, we randomly take $N_b = 100$ sample points on the $\partial\Gamma$, namely, equator. The exact solution has been set as $u(x) = x_1x_2x_3$ and then the right hand side term is $f(x) = 18x_1x_2x_3$. Then, the corresponding data $\{x_u^i, f(x_u^i)\}$ and $\{x_b^i, u_b^i\}$ as well as the normal $\{\boldsymbol{n}(x_u^i)\}$ has been computed. The numerical result is shown in Figure 6 with $Err = 8.560961e-3$.

The equator in Figure 6 is not smooth because the figure is generated based on triangle partition. So, extrapolation is occasionally occurred.

D. ELLIPTIC EQUATION ON TORUS

In this experiment, we solve (7) with $a = 1$ on a torus $\Gamma = \{x \in \mathbb{R}^3 | (\sqrt{x_1^2+x_2^2}-3)^2+x_3^2 = 1\}$. The most important topological feature of Γ is that the Euler characteristic is 0, while for $\mathbb{S}^2$ is 1. This may sometimes cause trouble during computation and graphing [14]. However, there is totally no difference to the previous examples in implementation of our algorithm except a little bit tricky for sampling. Notice Γ can be also defined by the following parametric equation equivalently:

$$\begin{cases} x_1(\theta,\phi) = (3+\cos(\theta))\cos(\phi) \\ x_2(\theta,\phi) = (3+\cos(\theta))\sin(\phi) \\ x_3(\theta,\phi) = \sin(\theta) \end{cases} \tag{8}$$

where $\theta, \phi \in [0, 2\pi)$. Therefore, we use Latin hypercube sampling method to take $N_u = 2,000$ sample of (θ, ϕ) on $[0, 2\pi)^2$. Then, we generate the sample points $\{x_u^i\}$ by means of (8). The exact solution is given by $u(x) = x_1x_2x_3$. The numerical result is shown in Figure 7 with $Err = 1.450341e-2$.

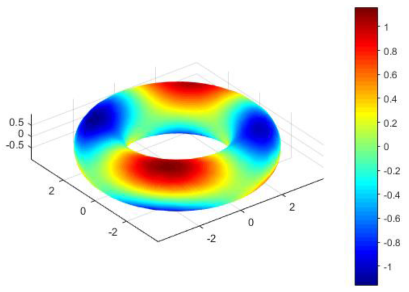

FIGURE 7. Solution of elliptic equation on torus.

E. LAPLACE-BELTRAMI EQUATION ON HIGH DIMENSIONAL SURFACE

In this subsection, we solve a high dimensional problem which is very intractable, or almost impossible by using traditional numerical methods. Let us consider (7) on $\Gamma = \mathbb{S}^4 \subset \mathbb{R}^5$ with $a = 0$.

For finite element method, we must first generate partition on $\mathbb{S}^4$, which, as the authors' best knowledge, is impossible. Even though we have the partition on $\mathbb{S}^4$, we must establish finite element space on $\mathbb{S}^4$, and then solve the discrete weak formulation. This procedure will include solving a super large linear system, which is an expensive cost. For meshless methods, such as radial basis function method, can form a relative easy linear system by the direct method. Unfortunately, this method is unstable as suggested in [15]. Thus, lots of researchers study the radial basis function finite difference methods. But once finite difference scheme is getting involved in, we must generate grid on $\mathbb{S}^4$ and its neighbourhood, which is also a huge work. So, this is a very challenging problem.

Our PINNs methods, once again, only need points cloud on $\mathbb{S}^4$ and their corresponding normal. And then, all the other part of the algorithm just keep the same. Since there is no way to visualized the result in $\mathbb{R}^5$, we only consider the relative error in this experiment. Instead of just show the result with a specific setting, we also study the relationship among the accuracy and PINNs' structure as well as the sample size.

We set the exact solution $u(x) = x_1 \sin(x_2x_3) + x_4e^{x_5}$. And generate uniform distributed sample points on $\mathbb{S}^4$. To do this, we first generate samples $\mathbf{y}_u^i$ followed by 5 dimensional standard normal distribution. Then the uniform distributed sample on $\mathbb{S}^4$ is given by $x_u^i = \frac{\mathbf{y}_u^i}{\|\mathbf{y}_u^i\|}$, where the $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^5$. This method can be generalized to any dimensional case.

First, we study how the sample size N_u of the PINN affect the accuracy. We use 6 layer PINN with 100 neurons each layer. By varying the sample size N_u from 8,000 to 18,000 and keep the sample size for cross-validation $N_c = 10,000$, we summarize the Err in Figure 8. The label in the

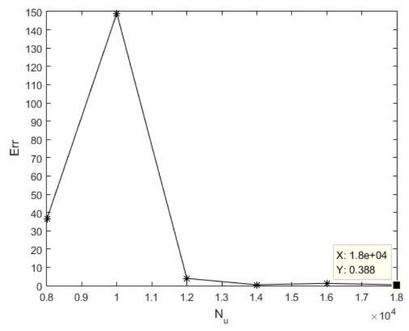

FIGURE 8. ***Err* with different N_u.**

TABLE 1. ***Err* with different PINNs' structure.**

Layers \ Neurons	80	100	120
2	$2.702997e+01$	$1.876405e+01$	$3.563460e+01$
4	$1.090280e+00$	$3.563426e-01$	$2.370108e-01$
6	$3.693945e+00$	$4.655468e-01$	$2.203986e-01$
8	$9.351051e-01$	$3.509825e-01$	$2.145151e-01$

figure shows the best case $Err = 3.884185e-01$ at $N_u = 18,000$. Although this is not a perfect result compared with our previous 3D problems, it is acceptable considering this is a high dimensional problem.

Next, we study how the PINNs' structure affect the accuracy. To this end, we keep sample size $N_u = 12,000$ and $N_c = 10,000$ and varying the number of layers and number of neurons for each layer. The result is shown in table 1.

These two results show that, when we use more sample points and larger PINN (more layers and neurons), the accuracy is almost increasing except some exceptions. This also fit the conclusions in [20], [25]. However, the best case will take most computational time. Hence, we suggest that in a practical problem, one could choose a sample size and network structure for the specific problem to balance the workload and accuracy. This also answer a reads' potential question: why we choose 6 layers with 100 neurons and such sample sizes in all our previous experiments? This is because, as our experience, when the relative error of cross-validation is around or less than 0.1, the result is acceptable. Hence, we just choose this structure and such sample sizes (2, 000 for all problems except the bunny surface and high dimensional problems). It turns out the results are not bad.

We also point out that, the PINN's structure is also related to its capacity. For example, if we solve a PDE on 1, 000 dimensional surface, then we must enhance the number of layers and number of neurons. This is because, the solution of a surface PDE can be manifold. So, the PINN must has enough ability, or degrees of freedom, to fit the solution.

V. CONCLUSION

In this paper, we develop a brand new algorithm for time independent surface PDEs by PINNs. The basis of our method is equivalence principle as shown in lemma 4. A plenty of numerical experiments have been shown to verify our algorithm. The relationship among sample size, PINN's structure and accuracy has been discussed. Compared with traditional numerical methods, such as finite element methods, finite difference methods and radial basis function methods etc., our algorithm do not need any partition, grid and extension of surface(, where they called h-narrow band around the surface in [18]). The only needed information for t he surface is a set of points and their corresponding normal. That is, our method is based on points cloud information, so it is more flexible and easier to implement. Nevertheless, the drawback of this method is that there is no theoretical proof for the error, which is similar to traditional numerical methods.

Some future work can be expected based on this method. First, we may use this method to solve time dependent problem, especially for moving surface problems. Second, different neuron network can be used to solve the surface PDEs, such as convolutional neural networks (CNNs) and Generative Adversarial Networks (GANs). Finally, points cloud methods can be used when we only have sparse sample points on the surface instead of knowing all the information of the surface.

REFERENCES

[1] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, pp. 289–298, Jul. 1991.

[2] M. Bertalmio, A. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Aug. 2005, p. 1.

[3] L. Tian, C. B. Macdonald, and S. J. Ruuth, "Segmentation on surfaces with the closest point method," in *Proc. 16th IEEE Int. Conf. Image Process. (ICIP)*, Nov. 2009, pp. 3009–3012.

[4] H. Biddle, I. Von Glehn, C. B. Macdonald, and T. Marz, "A volume-based method for denoising on curved surfaces," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2013, pp. 529–533.

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

[6] L. Olsen, P. K. Maini, and J. A. Sherratt, "Spatially varying equilibria of mechanical models: Application to dermal wound contraction," *Math. Biosci.*, vol. 147, no. 1, pp. 113–129, Jan. 1998.

[7] A. W. Toga, *Brain Warping*. Amsterdam, The Netherlands: Elsevier, 1998.

[8] C. M. Elliott and B. Stinner, "Modeling and computation of two phase geometric biomembranes using surface finite elements," *J. Comput. Phys.*, vol. 229, no. 18, pp. 6585–6612, Sep. 2010.

[9] D. Halpern, O. E. Jensen, and J. B. Grotberg, "A theoretical study of surfactant and liquid delivery into the lung," *J. Appl. Physiol.*, vol. 85, no. 1, pp. 333–352, Jul. 1998.

[10] S. Auer, C. Macdonald, M. Treib, J. Schneider, and R. Westermann, "Real–time fluid effects on surfaces using the closest point method," *Comput. Graph. Forum*, vol. 31, no. 6, pp. 1909–1923, Sep. 2012.

[11] U. Diewald, T. Preusser, and M. Rumpf, "Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces," *IEEE Trans. Vis. Comput. Graphics*, vol. 6, no. 2, pp. 139–149, Apr./Jun. 2000.

[12] G. Dziuk, "Finite elements for the Beltrami operator on arbitrary surfaces," in *Partial Differential Equations and Calculus of Variations*. Berlin, Germany: Springer, 1988, pp. 142–155.

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

[14] G. Dziuk and C. M. Elliott, "Finite element methods for surface PDEs," *Acta Numerica*, vol. 22, pp. 289–396, May 2013.

[15] A. Petras, L. Ling, and S. J. Ruuth, "An RBF-FD closest point method for solving PDEs on surfaces," *J. Comput. Phys.*, vol. 370, pp. 43–57, Oct. 2018.

[16] A. Petras, L. Ling, C. Piret, and S. Ruuth, "A least-squares implicit RBF–FD closest point method and applications to PDEs on moving surfaces," *J. Comput. Phys.*, vol. 381, pp. 146–161, Mar. 2019.

[17] A. Petras, and S. J. Ruuth, "PDEs on moving surfaces via the closest point method and a modified grid based particle method," *J. Comput. Phys.*, vol. 312, pp. 139–156, May 2016.

[18] K. Deckelnick, G. Dziuk, C. M. Elliott, and C. J. Heine, "An h-narrow band finite-element method for elliptic equations on implicit surfaces," *IMA J. Numer. Anal.*, vol. 30, no. 2, pp. 351–376, Apr. 2010.

[19] C. Chiu and J. Zhan, "Deep learning for link prediction in dynamic networks using weak estimators," *IEEE Access*, vol. 6, pp. 35937–35945, 2018.

[20] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.

[21] Y. Yang and P. Perdikaris, "Adversarial uncertainty quantification in physics-informed neural networks," Nov. 2018, *arXiv:1811.04026*. [Online]. Available: https://arxiv.org/abs/1811.04026

[22] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, "Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems," *J. Comput. Phys.*, vol. 397, Nov. 2019, Art. no. 108850.

[23] G. Pang, L. Lu, and G. E. Karniadakis, "FPINNs: Fractional Physics–Informed Neural Networks," *SIAM J. Sci. Comput.*, vol. 41, no. 4, pp. A2603–A2626, Jan. 2019.

[24] M. Gulian, M. Raissi, P. Perdikaris, and G. Karniadakis, "Machine learning of space–fractional differential equations," *SIAM J. Sci. Comput.*, vol. 41, no. 4, pp. A2485–A2509, Jan. 2019.

[25] M. Raissi and G. E. Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," *J. Comput. Phys.*, vol. 357, pp. 125–141, Mar. 2018.

[26] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. March. Learn. Res.*, vol. 18, pp. 1–43, Apr. 2018.

[27] Á. González, "Measurement of areas on a sphere using fibonacci and latitude–longitude lattices," *Math. Geosci.*, vol. 42, no. 1, pp. 49–64, Jan. 2010.

[28] J. Q. You, J. R. Yan, T. Xie, X. Zeng, and J. X. Zhong, "Generalized Fibonacci lattices: Dynamical maps, energy spectra and wavefunctions," *J. Phys., Condens. Matter*, vol. 3, no. 38, pp. 7255–7268, Sep. 1991.

[29] A. Mojsilovic and E. Soljanin, "Color quantization and processing by Fibonacci lattices," *IEEE Trans. Image Process.*, vol. 10, no. 11, pp. 1712–1725, Nov. 2001.

[30] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 3206–3211.

[31] Z. Fang and J. Zhan, *Deep Physical Informed Neural Network for Metamaterial Design*.

[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, Mar. 2010, pp. 249–256.

ZHIWEI FANG is currently pursuing the Ph.D. degree in mathematics with the Department of Mathematical Sciences, University of Nevada at Las Vegas, Las Vegas. His research interests include machine learning, numerical methods for partial differential equations, computational electromagnetic fields, and uncertainty quantification.

JUSTIN ZHAN is an ARA Scholar and a Professor of data science with the Department of Computer Science and Computer Engineering, University of Arkansas. He is also an Adjunct Professor with the Department of Biomedical Informatics, University of Arkansas for Medical Sciences. He has been the Director of the Big Data Hub and a Professor of the Department of Computer Science, College of Engineering, University of Nevada at Las Vegas. He has published 230 articles in peer-reviewed journals and conferences and delivered more than 30 keynote speeches and invited talks. His research interests include data science, biomedical informatics, artificial intelligence, information assurance, and social computing. He was the Steering Chair of the IEEE International Conference on Social Computing (SocialCom), the IEEE International Conference on Privacy, Security, Risk, and Trust (PASSAT), and the IEEE International Conference on BioMedical Computing (BioMedCom). He has served as a Conference General Chair, a Program Chair, a Publicity Chair, and a Workshop Chair. He has also served as a Program Committee Member of 150 international conferences. He has also served as an Editor-in-Chief, Editor, Associate Editor, Guest Editor, Editorial Advisory Board Member, and Editorial Board Member for 30 journals. He has been involved in more than 50 projects as a Principal Investigator (PI) or the Co-PI, which were funded by the National Science Foundation, the Department of Defense, and the National Institute of Health. He has been an Editor-in-Chief of the *International Journal of Privacy, Security and Integrity* and the *International Journal of Social Computing and Cyber-Physical Systems*.