Δ-PINNs: PHYSICS-INFORMED NEURAL NETWORKS ON COMPLEX GEOMETRIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Physics-informed neural networks (PINNs) have demonstrated promise in solving forward and inverse problems involving partial differential equations. Despite recent progress on expanding the class of problems that can be tackled by PINNs, most of existing use-cases involve simple geometric domains. To date, there is no clear way to inform PINNs about the topology of the domain where the problem is being solved. In this work, we propose a novel positional encoding mechanism for PINNs based on the eigenfunctions of the Laplace-Beltrami operator. This technique allows to create an input space for the neural network that represents the geometry of a given object. We approximate the eigenfunctions as well as the operators involved in the partial differential equations with finite elements. We extensively test and compare the proposed methodology against traditional PINNs in complex shapes, such as a coil, a heat sink and a bunny, with different physics, such as the Eikonal equation and heat transfer. We also study the sensitivity of our method to the number of eigenfunctions used, as well as the discretization used for the eigenfunctions and the underlying operators. Our results show excellent agreement with the ground truth data in cases where traditional PINNs fail to produce a meaningful solution. We envision this new technique will expand the effectiveness of PINNs to more realistic applications.

1 MOTIVATION

Physics-informed neural networks (PINNs) Raissi et al. (2019) are a exciting new tool for blending data and known physical laws in the form of differential equations. They have been successfully applied to multiple physical domains such as fluid mechanics Raissi et al. (2020), solid mechanics Haghighat et al. (2021), heat transfer Cai et al. (2021) and biomedical engineering Kissas et al. (2020); Ruiz Herrera et al. (2022), to name a few. Even though this technique can be used to solve forward problems, they tend to excel when performing inverse problems. Since their inception, there have been multiple attempts to improve PINNs, in areas such as training strategies Nabian et al. (2021); Wang et al. (2022) and activation functions Jagtap et al. (2020).

Despite recent progress, many works still consider simple geometric domains to solve either forward or inverse problems, hindering the applicability of PINNs to real world problems, where the objects of study may have complicated shapes and topologies. In this area, there have been multiple attempts to introduce complexity to the input domain of the neural networks. One approach is to describe the boundary of the domain with a signed distance function Sukumar & Srivastava (2022); Berg & Nyström (2018); McFall & Mahan (2009). In this way, the boundary conditions can be satisfied exactly instead of relying on a penalty term in the loss function. Another approach is to use domain decomposition to model smaller but simpler domains Jagtap & Karniadakis (2020). Coordinate mappings between a simple domain have also been proposed for convolutional Gao et al. (2021) and fully connected networks Li et al. (2022). Nonetheless, all these works demonstrate examples of 2-dimensional shapes. When using PINNs in 3-dimensional surfaces there have been attempts to ensure that the vector fields that may appear in the partial differential equations remain tangent to the surface by introducing additional terms in the loss function Fang et al. (2021); Sahli Costabal et al. (2020). This approach works well when the surface is relatively simple and smooth, specifically when the Euclidean distance of the embedding 3-D space between two points in the domain is similar to the intrinsic geodesic distance on the manifold. However, in several applications the two distances may sensibly differ, as exemplified in Figure 1.

In this work, we propose to represent the coordinates of the input geometry with a positional encoding based on the eigenfunctions of the Laplace-Beltrami operator of the manifold. In this way, points that are close in the surface will remain close in the positional encoding space. Next, the input of the neural network will be the value of a finite number of the eigenfunctions evaluated at a point in the surface and the output will remain the physical quantity that we are modeling with PINNs. Positional encoding have shown great success in improving the capabilities of neural networks and are used in transformers Vaswani et al. (2017), neural radiance fields Mildenhall et al. (2020) and PINNs Wang et al. (2021) to name a few. For our positional encoding, the Laplace-Beltrami eigenfunctions can be approximated numerically for any shape with the finite element method. By approximating the eigenfunctions we lose the ability to use automatic differentiation to compute the operators of the partial differential equations. However, we show that common operators such as the gradient and the Laplacian can be efficiently computed with finite elements applied to the output of the neural network. We demonstrate the capabilities of the proposed method by testing different geometries, such as a coil, a heat sink and a bunny and different physics, such as the Eikonal equation and heat transfer.

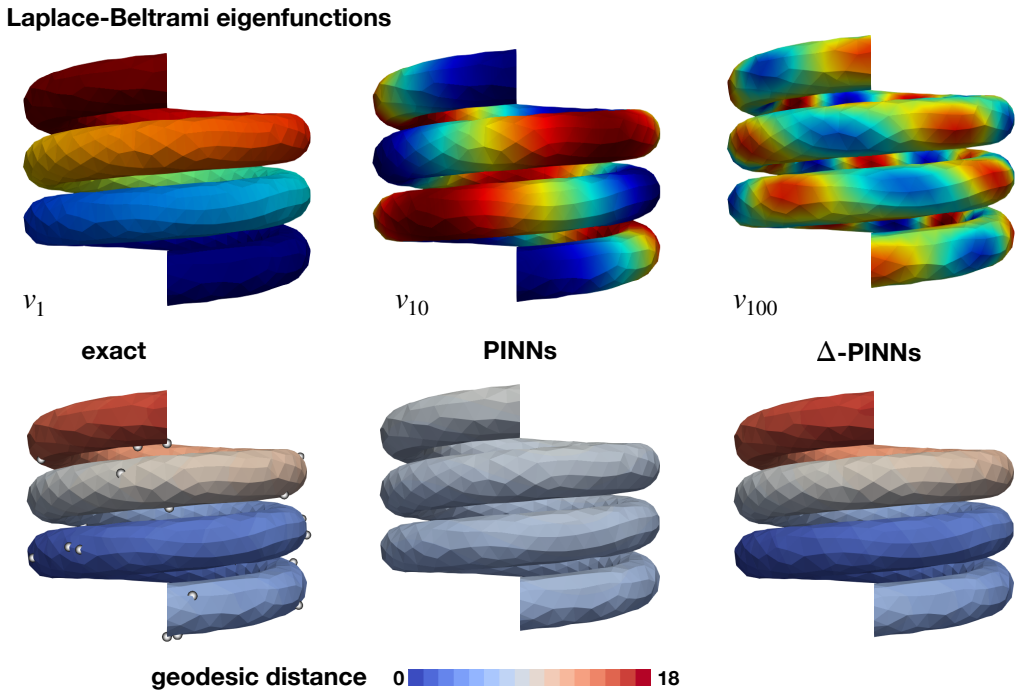

Figure 1: **Learning the Eikonal equation on a coil**. Top row: the 1st, 10th and 100th Laplace-Beltrami eigenfunction of the geometry. Bottom row, left: the ground truth solution of the Eikonal equation and the data points used for training shown as gray spheres. Center, the traditional PINN approximation. Right, the solution of Δ-PINNs, our proposed method.

2 Methods

2.1 Physics-informed neural networks

We consider the problem where we have partial observations of an unknown function $u(\boldsymbol{x})$, with an input domain $\boldsymbol{x} \in \mathcal{B}$, where $\mathcal{B}$ is an open and connected domain in $\mathbb{R}^d$, $d = 2, 3$. We also assume that $u(\boldsymbol{x})$ satisfies a partial differential equation of the form $\mathcal{N}[u, \boldsymbol{\lambda}] = 0$, where $\mathcal{N}[\cdot, \boldsymbol{\lambda}]$ is a potentially non-linear operator parametrized by $\boldsymbol{\lambda}$. The partial observations u_i are located in $\bar{\mathcal{B}}$ at positions $\boldsymbol{x}_i$, $i = 1, \ldots, N$, which may therefore fall on the boundary $\partial\mathcal{B}$. Boundary conditions, such as Neumann boundary conditions $\nabla u \cdot \boldsymbol{n} = g_i$, may also be enforced at boundary points $\boldsymbol{x}_i^b$, $i = 1, \ldots, B$. We proceed by approximating the unknown function with a neural network $u \approx \hat{u} = NN(\boldsymbol{x}, \boldsymbol{\theta})$, parametrized with trainable parameters $\boldsymbol{\theta}$. In order to learn a function that

satisfies the observed data, the boundary conditions and the model, we postulate a loss function composed of three terms:

$$\mathcal{L}(\{u_i, \boldsymbol{x}_i\}, \{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}, \boldsymbol{\lambda}) = MSE_{\text{data}}(\{u_i, \boldsymbol{x}_i\}, \boldsymbol{\theta}) + MSE_{\text{PDE}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + MSE_{\text{boundary}}(\{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}). \tag{1}$$

The first term favours the network to learn the available data:

$$MSE_{\text{data}}(\{u_i, \boldsymbol{x}_i\}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}(u_i - \hat{u}(\boldsymbol{x}_i; \boldsymbol{\theta}))^2. \tag{2}$$

The second term ensures that the model, in the form of a partial differential equation, is approximately satisfied:

$$MSE_{\text{PDE}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{R}\sum_{i}^{R}(\mathcal{N}[\hat{u}(\boldsymbol{x}_i^r, \boldsymbol{\theta}; \boldsymbol{\lambda}]))^2. \tag{3}$$

Here, we will evaluate the model at locations $\boldsymbol{x}_i^r$, called collocation points, that can be generated randomly within the domain at every optimization iteration or can have predefined values. Minimizing this term will ensure that the model is approximately satisfied in the entire domain. We can also define the parameters of the operator $\mathcal{N}[\cdot, \boldsymbol{\lambda}]$ as trainable variables that will be learned during the optimization procedure. The final term in the loss function ensures that the boundary conditions are satisfied

$$MSE_{\text{boundary}}(\{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}) = \frac{1}{B}\sum_{i}^{B}(\nabla\hat{u}(\boldsymbol{x}_i^b; \boldsymbol{\theta}) \cdot \boldsymbol{n} - g_i)^2. \tag{4}$$

We note that Dirichlet boundary conditions can be included as data in $\{u_i, \boldsymbol{x}_i\}$. This is the basic structure of physics-informed neural networks that can be accommodated for a large class of problems Raissi et al. (2019). One issue of this formulation is that the neural network that represents the solution is parametrized with a vector $\boldsymbol{x} \in \mathbb{R}^3$, which is a Euclidean space. In this way, it is difficult to inform the network about the geometry of the domain or, in the case of a complex domain in $\mathbb{R}^d$ where the boundaries lie. This is important in the computation of the differential operators. Furthermore, points in the manifold $\mathcal{B}$ that are close in the Euclidean space are not necessarily close in the intrinsic distance, as illustrated in Figure 1.

2.2 Eigenfunctions of Laplace-Beltrami operator

In this work, we propose to represent the manifold of interest $\mathcal{B}$ with the eigenfunctions of the Laplace-Beltrami operator $-\Delta[\cdot] = \nabla \cdot \nabla[\cdot]$ instead of using the Euclidean coordinates $\boldsymbol{x}$. These functions $v_i(\boldsymbol{x})$ can be computed by solving the eigenvalue problem: $-\Delta v_i = \lambda_i v_i$, where λ_i is the correspondent eigenvalue of $v_i(\boldsymbol{x})$. In the case of a domain with open boundaries, this problem is usually accompanied with either homogeneous Dirichlet or Neumann boundary conditions. In this work, we will use the latter. The set of eigenvalues provide a spectral signature of the domain, albeit not unique Gordon et al. (1992). It is possible to show that the eigenvalues of the Laplace-Beltrami operator, endowed with homogeneous boundary conditions, are real, non-negative, and tending to infinity. When ordered by their magnitude, the first eigenvalue λ_0 is always zero, whereas the second one λ_1 is strictly positive. We note that the physical interpretation that small eigenvalues are associated with low frequency functions, represented by the corresponding eigenfunction. Although these functions can be computed analytically for shapes, such as a line, a rectangle or a sphere, they can be computed numerically for an arbitrary manifold. Here, we represent the domain $\mathcal{B}$ using a mesh with nodes and elements and solve the eigenvalue problem using finite element shape functions. As such, we can obtain the discrete Laplacian matrix $\boldsymbol{A}$ and mass matrix $\boldsymbol{M}$ as:

$$\boldsymbol{A}_{ij} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \int_{\mathcal{B}} \nabla N_i \cdot \nabla N_j d\mathcal{B}, \quad \boldsymbol{M}_{ij} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \int_{\mathcal{B}} N_i N_j d\mathcal{B}, \tag{5}$$

where $\mathbf{A}$ represents the assembly of the local element matrices, and N are the finite element shape functions. Then, we solve the eigenvalue problem: $\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{M}\boldsymbol{v}$.

2.3 Physics-informed neural networks on manifolds

We propose to represent the position of a point in the manifold by N eigenfunctions of the Laplace-Beltrami operator associated with the N lowest eigenfunctions, such that $\boldsymbol{v}_i =$

$[v_1(\boldsymbol{x}_i), v_2(\boldsymbol{x}_i), ..., v_N(\boldsymbol{x}_i)]$. Now, instead of parametrizing the neural network with the Euclidian position, we use: $u \approx \hat{u} = NN(\boldsymbol{v}(\boldsymbol{x}), \boldsymbol{\theta})$. Since $\boldsymbol{v}$ varies smoothly in the manifold, intrinsically close points will be also close in the space $\boldsymbol{v}$.

Another interpretation of the proposed method is that we are constructing features or positional encodings for points in the manifold. Then, the approximation of the solution $\hat{u}$ can be seen as a non-linear combination of these features. In our case, the eigenfunctions represent features of increasing frequency, as seen in Figure 1. A similar concept has been proposed for cartesian domains with Fourier feature mappings Tancik et al. (2020); Wang et al. (2021). Here, the Cartesian coordinates are transformed with sine and cosine functions, which have random frequencies that can be tuned for the problem of interest. The main difference is that in our method the frequencies are given by the eigenfunctions, and are defined in the manifold where want to solve the problem. We can select different eigenfunctions depending on the nature of problem, where high frequency problems might need a larger amount of eigenfunctions.

The main challenge of our methodology is that the differential operators cannot be computed directly from the neural network with automatic differentiation. Since the eigenfunctions cannot be computed in closed form for an arbitrary manifold, we have a map between $\boldsymbol{x}$ and $\boldsymbol{v}$ that is defined with finite elements. The only way to differentiate the eigenfunctions $\boldsymbol{v}$ with respect to the position $\boldsymbol{x}$ is using finite elements. Therefore, we use this numerical approximation of the differential operators that are required to compute the PDE operator $\mathcal{N}[\cdot]$. Commonly used operators, such as the gradient and the Laplacian can be easily computed with linear finite elements. We also note that one could employ the chain rule to compute, for example, the gradient, such that $\nabla_{\boldsymbol{x}}\hat{u} = \nabla_{\boldsymbol{v}}\hat{u} \cdot \nabla_{\boldsymbol{x}}\boldsymbol{v}$. The term $\nabla_{\boldsymbol{v}}\hat{u}$ can be computed with automatic differentiation since it only depends on the neural network, and the second term $\nabla_{\boldsymbol{x}}\boldsymbol{v}$ needs to be computed with finite elements. We found small advantages with this approach, and it becomes cumbersome for second order operators. Instead, we take the simplest route: we use the neural network predictions at the nodal locations and use finite elements to evaluate the operator. For instance, to compute $\nabla_{\boldsymbol{x}}\hat{u}$ at the centroid of a linear triangular element, we need to evaluate the neural network $\hat{u}_i$ at every $\boldsymbol{v}_i$, which corresponds to the selected eigenfunctions evaluated at the nodal locations $\boldsymbol{x}_i$. Then, the gradient can be computed as $\nabla_{\boldsymbol{x}}\hat{u} \approx \boldsymbol{B}^e \cdot [\hat{u}_1, \hat{u}_2, \hat{u}_3]^T$, where the matrix $\boldsymbol{B}^e$ defines the gradient operator for a linear triangular element in particular. The details on the computation of this matrix can be found in the accompanying code. One of the advantages of this methodology is that the gradient is guaranteed to be tangent to the manifold, which is not the case for the traditional physics-informed neural network formulation Sahli Costabal et al. (2020). In a similar fashion, we can compute the Laplacian $\Delta\hat{u}_i$ at any given nodal location by predicting $\hat{u}_j$ at all the neighboring nodes, including the node of interest. Then, we can use the definition of the discrete Laplacian presented in equation 5 and approximate $\Delta\hat{u}_i \approx \boldsymbol{A}_{ij}\hat{u}_j$.

The advantage of being able to compute the operators locally at the element or node level is that we still use mini-batch techniques to estimate the loss function and its gradient. In this way, we avoid predicting the operator for the entire manifold, which could slow the training process for large meshes. In general, we will randomly select nodes or elements for each training iteration to evaluate the loss term MSE_{PDE} defined in equation 3.

3 Numerical experiments

3.1 Eikonal equation on a coil

Our first numerical experiment concerns the computation of geodesic distances through the Eikonal equation:

$$\sqrt{\nabla u \cdot \nabla u} = 1, \; u(\boldsymbol{x}_b) = 0 \tag{6}$$

This equation has multiple applications, for instance, in cardiac electrophysiology Sahli Costabal et al. (2020) and seismology Smith et al. (2020), because it can be used to model the arrival times of a traveling wave. In the particular form shown in equation 6, the solution $u(\boldsymbol{x})$ can be interpreted as the geodesic distance from the point $\boldsymbol{x}_b$ to any point in the manifold $\boldsymbol{x}$.

In this example, we will solve this equation on the surface of a coil, which is generated by an helix with 30 mm of diameter and 12 mm per revolution of pitch. This curve is extruded with

a circular cross-section with 5 mm of radius. The resulting geometry can be seen in Figure 1, which is discretized with 1546 points and 3044 triangles. We can generate a ground truth solution of the Eikonal equation by randomly selecting a point on the mesh for $\boldsymbol{x}_b$ and then applying the exact geodesic algorithm Mitchell et al. (1987) as implemented in libigl Jacobson et al. (2018). We randomly select 40 points of this solution to use as data u_i. In this case, the partial differential equation loss defined in 3 takes the form:

$$MSE_{\text{PDE}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{i}^{R}\left(\sqrt{(\boldsymbol{B}_r^e\hat{\boldsymbol{u}}_r^e)\cdot(\boldsymbol{B}_r^e\hat{\boldsymbol{u}}_r^e)} - 1\right)^2 \tag{7}$$

where $\boldsymbol{B}_r^e\hat{\boldsymbol{u}}_r^e$ is the approximation of the gradient for a particular triangle r. Here, the vector $\hat{\boldsymbol{u}}_r^e$ is the prediction of the neural network at the nodes of the triangle r, which depend on the trainable parameters $\boldsymbol{\theta}$. We use 50 eigenfuctions of the Laplace-Beltrami operator as input of the neural network, which is then followed by a single hidden layer with 100 neurons and hyperbolic tangent activations. We train the neural network with the ADAM optimizer Kingma & Ba (2014) for 10,000 iterations with default parameters. We use a batch size of 10 samples to evaluate both the MSE_{data} and MSE_{PDE} term. For comparison, we also train the same neural network by removing the physics, that is not considering the partial differential equation loss shown equation 7. Additionally, we train a regular physics-informed neural network, where we use the Cartesian coordinates as the input of the neural network. We implemented the Eikonal equation in the MSE_{PDE} computing ∇u through automatic differentiation. This network also had only single hidden layer of 100 neurons. We also tested deeper architectures without any meaningful change in the results, using the same optimization procedure as described above.

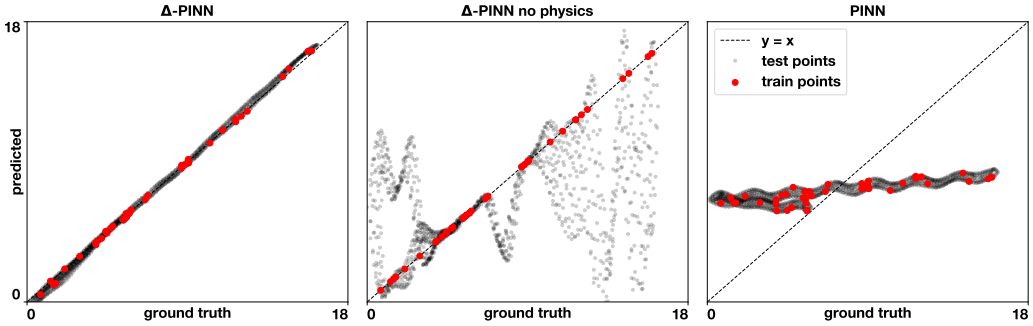

Figure 2: **Accuracy the coil example with the Eikonal equation.** Correlation between predicted and ground truth values of geodesic distance for $\Delta-$PINNs (left), $\Delta-$PINNs trained without the loss functions that includes the Eikonal equation (center) and traditional PINNs (right).

Figures 1 and 2 show the results of these experiments. In the bottom row of Figure 1, we see the exact solution to the geodesic problem, the physics-informed neural network learned solution and the proposed method. On one hand, we see that PINNs fail to capture the fast variations in the Cartesian space and then to predict a nearly constant intermediate value. On the other hand, we see that Δ-PINNs can represent the exact solution with high accuracy. This is reflected in Figure 2, where we show the correlation between the predicted and ground truth data for train and test points. We see that the proposed method achieves a high correlation in both training and testing data, while PINNs fail to fit both. Interestingly, when the physics are removed from Δ-PINNs, the method can only overfit the training data with a poor performance in the test data. Qualitatively, the normalized mean squared errors on the combined train and test dataset correspond to $3.97 \cdot 10^{-3}$ for Δ-PINNs, 1.23 for for Δ-PINNs without physics, and 1.02 for PINNs. These results show more than 2 orders of magnitude improvement for the proposed method over the traditional formulation, at the expense of small pre-processing step of computing the eigenfunctions of the Laplace-Beltrami operator for that geometry. It is worth noting that generating the solution of the Eikonal equation with the exact geodesic algorithm does not lead to a zero MSE_{PDE} loss as defined in equation 7 due to the underlying numerical discretization. If we evaluate this term using all the points in the mesh we obtain a residual value of $2.11 \cdot 10^{-2}$. Hence, the proposed method works even in the presence of imperfect physics, when the model is not exactly satisfied.

3.2 HEAT TRANSFER IN A HEAT SINK

In this experiment, we simulate a heat sink with convection losses in 2 dimensions and steady state. This can be expressed as the following boundary value problem:

$$\begin{aligned} \Delta u(\boldsymbol{x}) &= 0,\, \boldsymbol{x} \in \mathcal{B}; \quad u(\boldsymbol{x}) = 1,\, \boldsymbol{x} \in d\mathcal{B}_D \\ \nabla u(\boldsymbol{x}) \cdot \boldsymbol{n} &= 0.1u(\boldsymbol{x}),\, \boldsymbol{x} \in d\mathcal{B}_C; \quad \nabla u(\boldsymbol{x}) \cdot \boldsymbol{n} = 0,\, \boldsymbol{x} \in d\mathcal{B}_N. \end{aligned} \tag{8}$$

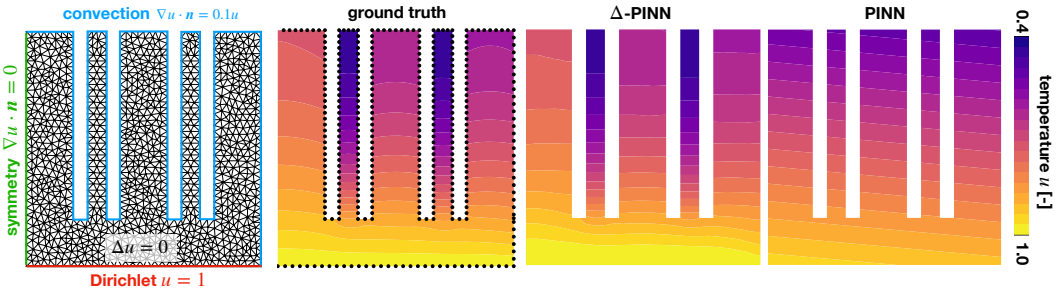

Figure 3: **Learning the temperature distribution of a heat sink from boundary measurements.** Left, the boundary conditions and the finite element mesh used to create the solution. Right, the ground truth, $\Delta-$PINNs, $\Delta-$PINNs not informed by the heat equations and traditional PINNs predicted temperature distributions. The data points are shown on the ground truth panel in black.

We include three different types of boundary conditions: Dirichlet in $d\mathcal{B}_D$, Robin in $d\mathcal{B}_C$ and Neumann $d\mathcal{B}_N$. The Dirichlet boundary condition is to represent another body with constant, higher than ambient, temperature, which the heat sink $\mathcal{B}$ is attempting to dissipate. The Robin boundary condition represents a convective flux, which is proportional to the temperature u. The Neumann boundary condition is used to represent the symmetry of the domain. The geometry and location of the boundaries are shown in Figure 3. This example could be interpreted as a simplified model for the heat sink that is typically placed above a central processing unit (CPU) of a computer. To generate a ground truth solution for this problem we performed finite element simulations using the FEniCS library Logg et al. (2012) on a mesh with 1289 nodes and 2183 linear triangular elements. In this experiment we will attempt to solve the inverse problem of learning the temperature inside the domain $\mathcal{B}$ from the information of the temperature in the boundary $d\mathcal{B}$. A practical realization of this experiment could be achieved by using a thermal camera and attempt to predict the temperature inside the heat sink. We will again use the proposed Δ-PINNs with and without physics, as well conventional PINNs to solve this problem. In all cases, we will not inform the type of boundary conditions applied, and instead we only provide temperature measurements u_i at the boundary $d\mathcal{B}$. Thus, we will not consider the term MSE_{boundary} in the loss function defined in equation 1. In practice, this is important because the true form of the convective fluxes is often unknown or empirically approximated. For Δ-PINNs we will approximate the Laplacian operator Δu with triangular finite elements using the definition of discrete Laplacian $\boldsymbol{A}$ shown in equation 5. Then, we can define the loss term:

$$MSE_{\text{PDE}}(\boldsymbol{\theta}) = \sum_{i}^{R} \left(\sum_{j \in N(i)} A_{ij} \hat{u}_j \right)^2 \tag{9}$$

where the index i represents the nodes of the mesh inside the domain $\mathcal{B}$, $j \in N(i)$ represents the neighbor nodes of the node i, including itself. The term $\hat{u}_j$ represents the output of the neural network at the nodal locations j that depends on trainable parameters $\boldsymbol{\theta}$. For traditional PINNs, we use automatic differentiation to compute $\Delta\hat{u}$ and construct the loss term MSE_{PDE}. We use 50 eigenfunctions to represent the domain $\mathcal{B}$ for Δ-PINNs, and 3 hidden layers of 100 neurons for all cases. We train the network with ADAM Kingma & Ba (2014) for 50,000 iterations using a batch size of 30 samples to evaluate each of the terms of the loss function. We use 361 values of u located at the boundaries as training data, as shown in Figure 3. We note that we use the same operator, the Laplacian, for the PDE that governs this problem and to construct the input of the neural network. However, the eigenfunctions are computed with homogeneous Neumann boundary

conditions, which guarantees that the eigenfunctions v_i are not solutions to the boundary value problem shown in 9 since they have different boundary conditions.

The results of this experiment are summarized in Figures 3 and 4. First, we see that the geometry of the heat sink greatly influences the solution, as shown in Figure 3. The narrow fins have more temperature reduction than the wide ones. Qualitatively, we observe that $\Delta-$PINN produces the closest temperature distribution to the ground truth solution. The method is able to produce different temperature distributions for the different fins, which may be close in Euclidean distance, but are far apart in the intrinsic distance. When we remove the physics, we observe a good fit at the boundaries, but the method completely fails to approximate the solution within the domain, producing unrealistic temperature distributions. Finally, traditional PINNs tend to favor satisfying the MSE_{PDE}, producing effectively a linear solution for which trivially $\Delta\hat{u} = 0$. However, in this process the data provided at the boundaries is not fitted. Also, this method is not able to produce different temperature profiles for the different fins, correlating points that are close only in the cartesian space. Quantitatively, we see that the normalized mean squared error on the combined train and test sets drops 2 orders of magnitude for Δ-PINNs ($1.4 \cdot 10^{-3}$) when compared to traditional PINNs ($2.52 \cdot 10^{-1}$). When we remove the physics from Δ-PINNs, the normalized mean squared error increases to $7.44 \cdot 10^{-2}$. It can be seen in Figure 4, central panel, that this method is able to fit well all the training points at the boundary, but fails at the test points that are inside the domain. Also in Figure 4, we can see that for PINNs, the correlations reflect exactly structure of the different fins. With this example, we have shown a problem with simple physics, a linear inverse problem, with some degree of complexity in the geometry of the domain may cause PINNs to fail while Δ-PINN may successfully work.

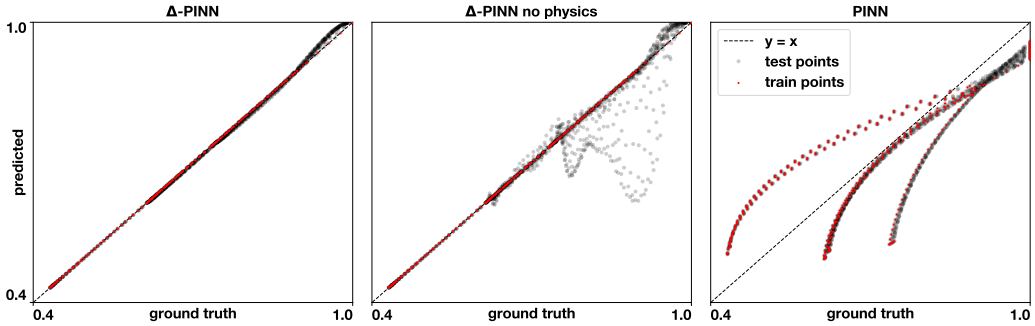

Figure 4: **Accuracy the heat sink example.** Correlation between predicted and ground truth values of geodesic distance for $\Delta-$PINNs (left), $\Delta-$PINNs trained without the loss functions that includes the heat equation (center) and traditional PINNs (right).

3.3 Physics-informed deep operator network

We finalize this section by presenting a deep operator network Lu et al. (2019); Wang et al. that learns the geodesic distance between two points on a manifold. This type of networks learn to transform functions instead of single points. They are composed by a branch network that encodes the input functions and a trunk net that encodes the input position. The input function in our case corresponds to boundary condition of the Eikonal equation, shown in equation 6 and the input coordinate corresponds to points in the manifold. We note two properties of the geodesic distance function $d(\cdot,\cdot)$: first, it is symmetric, $d(\boldsymbol{x}_1, \boldsymbol{x}_2) = d(\boldsymbol{x}_2, \boldsymbol{x}_1)$, since the distance from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ is the same as the distance from $\boldsymbol{x}_2$ to $\boldsymbol{x}_1$. Second, the distance from a point to itself is zero, that is $d(\boldsymbol{x}, \boldsymbol{x}) = 0$ for every point in the manifold. As in the rest of the experiments, we will represent points in the manifold by the eigenfunctions of the Laplace-Beltrami operator $\boldsymbol{v}_i = [v_1(\boldsymbol{x}_i), v_2(\boldsymbol{x}_i), ..., v_N(\boldsymbol{x}_i)]$. With this is mind, we propose the following deep operator network architecture $\hat{d}$ to approximate d:

$$\bar{d}(\boldsymbol{v}_1, \boldsymbol{v}_2) = NN(\boldsymbol{v}_1; \boldsymbol{\theta}_T) \cdot NN(\boldsymbol{v}_2; \boldsymbol{\theta}_B) \tag{10}$$

$$\hat{d}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \frac{1}{2}\left(1 - \frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{\|\boldsymbol{v}_1\|\|\boldsymbol{v}_2\|}\right)\left(\bar{d}(\boldsymbol{v}_1, \boldsymbol{v}_2) + \bar{d}(\boldsymbol{v}_2, \boldsymbol{v}_1)\right) \tag{11}$$

where $\boldsymbol{\theta}_T$ and $\boldsymbol{\theta}_B$ represent the trainable parameters of the trunk and branch networks respectively. This architecture automatically satisfies the symmetry and zero distance properties of the geodesic distance function. The network will be informed by informed by the Eikonal equation, which should be satisfied at any point of the domain. In this case, we use the chain rule to compute the gradient $\nabla_{\boldsymbol{x}_1}\hat{d} = \nabla_{\boldsymbol{v}_1}\hat{d} \cdot \nabla_{\boldsymbol{x}_1}\boldsymbol{v}$. The gradient of the eigenfuctions with respect to the input positions $\nabla_{\boldsymbol{x}_1}\boldsymbol{v}$ is computed with finite elements and gradient of the predicted distance function with respect to the input eigenfunctions $\nabla_{\boldsymbol{v}_1}\hat{d}$ is computed with automatic differentiation. The loss function to train the neural network still consists of the one detailed in equation 1, but in this case, we do not have boundary data and the partial differential equation loss takes the form:

$$MSE_{\text{PDE}}(\boldsymbol{\theta}_T, \boldsymbol{\theta}_B) = \frac{1}{R}\sum_{i}^{R}\left(\sqrt{(\nabla_{\boldsymbol{x}_1}\hat{d} \cdot \nabla_{\boldsymbol{x}_1}\hat{d})} - 1\right)^2 \tag{12}$$

which is evaluated at the R triangle centroids of the mesh representing the manifold with a mini-batch strategy.

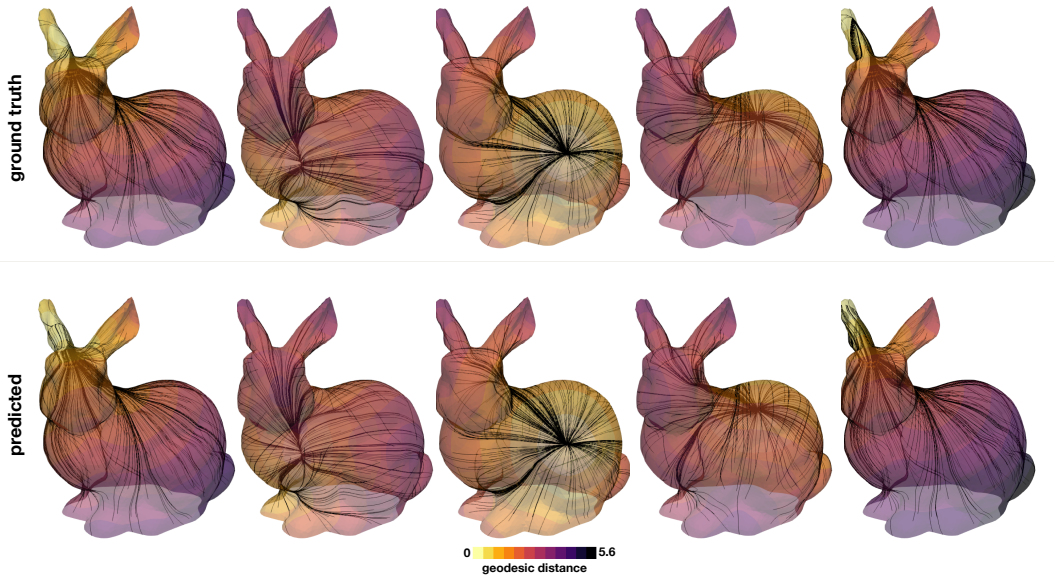

Figure 5: **Geodesics on a bunny.** We learn the geodesic distance between any two points in the bunny surface. We randomly select 5 points and show the ground truth and predicted geodesic distance between these points and the rest of the surface, and geodesic paths between 300 randomly selected points and these origin points.

We test the proposed methodology in the Stanford bunny Turk & Levoy (1994), obtained from the software MeshMixer. This mesh has 5,550 vertices and 10,996 triangles, which results in 15,404,025 pairs of distance to be learned. We randomly select 50,000 pairs (0.32% of the total pairs) as training data, for which we compute the exact geodesic distance Mitchell et al. (1987). We use the 50 eigenfunctions associated with the lowest eigenvalues of the Laplace-Beltrami operator to define $\boldsymbol{v}(\boldsymbol{x}_i)$. Both for the trunk and branch network, we use 10 hidden layers of 200 neurons each. We train for 50,000 iterations with ADAM Kingma & Ba (2014) with exponential decay of the learning rate. To demonstrate the potential of this method, we also compute geodesic paths between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ by solving the following initial value problem: $\dot{\boldsymbol{x}} = -\nabla_{\boldsymbol{x}} d(\boldsymbol{x}, \boldsymbol{x}_2)$, $\boldsymbol{x}(0) = \boldsymbol{x}_1$. Some representative results of this method are shown in Figure 5, where we display the geodesic distance from 5 randomly selected points to the rest of the manifold for our method and the ground truth. We also show geodesic paths between 300 randomly selected points to each of the 5 origin points. We see qualitatively very good agreement in both for the predicted geodesic distances and geodesic paths. Furthermore, the geodesic path are actually directed to the origin point, which shows that the learned geodesic distances continuously grow out of the origin point. Quantitatively, we evaluate the test error using 10,000 randomly selected pairs and obtained a mean absolute error normalized by the maximum geodesic distance of 0.57%.

4 DISCUSSION

In this work, we present a novel method to use physics-informed neural networks in complex geometries by representing manifolds by the eigenfunctions of the Laplace-Beltrami operator. Given a mesh representing the geometry, we can compute these functions with standard finite elements. We show that this method outperforms the original formulation of PINNs, especially in cases where there points in the input domain are close in Eucledian distance but far in the intrinsic distance. We test our methodology by solving an inverse problem with the Eikonal equation and the heat equation, a forward Poisson problem, and an operator learning problem for inferring the geodesic distance between two points on a complex manifold. The Laplace-Beltrami eigenfunctions have been used in other applications of machine learning, such as point signatures in shapes Rustamov et al. (2007); Sun et al. (2009), shape descriptors Fang et al. (2015), for neural fields Koestler et al. (2022) and to approximate kernels on manifolds for Gaussian process regression Borovitskiy et al. (2020) and classification Gander et al. (2022). To our knowledge this is the first time they have been used in the context of physics-informed neural networks.

The proposed method can be seen as an extension of positional encoding methods Vaswani et al. (2017), such as Fourier features Tancik et al. (2020), to arbitrary geometries. Furthermore, combining the Laplacian eigenfunctions in $\mathbb{R}^N$, we can recover a series of cosine functions when using Neumann boundary conditions and sine functions when using Dirichlet boundary conditions, which correspond to the encodings proposed in Vaswani et al. (2017). These techniques have significantly improved the performance of neural networks, including PINNs Wang et al. (2021). Nonetheless, we observed in our experiments that only using the eigenfunctions from the Laplacian with Neumann boundary conditions was enough to obtain accurate results. We see that including the physics of the problem greatly improves the results. Furthermore, we tested our method in a pure regression tasks in B and compared it to Gaussian process regression which can be naturally extended to work on manifolds. We see that neural networks with the eigenfunctions as input tend to overfit the data and the results depend heavily on the number of eigenfunctions used. This confirms the regularization effect of the physics, explaining the excellent performance in problems shown here. We also note that Gaussian process are an effective tool for regression on manifolds, but extending then to include physical knowledge of non-linear problems is cumbersome, requires approximate inference techniques and will suffer from the classical poor scaling of Gaussian process Raissi et al. (2017).

In our experiment of the Poisson equation, in Appendix A, where we can directly compare to traditional PINNs, we observe that is possible to obtain similar levels of errors, showing that our method can also be used for simple domains. Furthermore, we did not see a decrease in performance when we compared the numerically obtained eigenfunctions versus the exact ones. The number of eigenfunctions used did alter the accuracy of our method. This quantity is a common hyper-parameter that needs to be tuned in some sort of shape for all positional encodings Tancik et al. (2020). We also observed that the size of the discretization to compute the operators also has an impact on the accuracy in a non-monotonic fashion. The way that changing the discretization affects the loss landscape and the training process requires further studies, which we believe are beyond the scope of this work.

Our current approach has some limitations. First, we numerically approximate the operators involved in the partial differential equations using finite elements. Although the gradient and the Laplacian, which are the most commonly used operators, can be approximated with linear elements, more sophisticated elements might be needed for other operators. Second, computing eigenfunctions of the Laplace-Beltrami operator can become expensive for large meshes. Nonetheless, since we only need a small number of eigenfunctions and this is a common problem in geometry processing, there are specialized algorithms for task that could be exploited Nasikun et al. (2018).

As future work, we plan to expand this methodology to work on 3D solid geometries, for which we can compute the eigenfunctions of the Laplace operator. There are multiple opportunities in this area, as most partial differential equations are actually solved in this type of domains. We would also like to extend this approach to time dependent problems, where we can include time as an additional feature of the neural network. We envision that $\Delta-$PINNs will enlarge the possibilities for applications of physics-informed neural networks in more complex and realistic geometries.

REFERENCES

Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, et al. Matérn gaussian processes on riemannian manifolds. *Advances in Neural Information Processing Systems*, 33:12426–12437, 2020.

Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6), 2021.

Yi Fang, Jin Xie, Guoxian Dai, Meng Wang, Fan Zhu, Tiantian Xu, and Edward Wong. 3d deep shape descriptor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2319–2328, 2015.

Zhiwei Fang, Justin Zhang, and Xiu Yang. A physics-informed neural network framework for partial differential equations on 3d surfaces: Time-dependent problems. *arXiv preprint arXiv:2103.13878*, 2021.

Lia Gander, Simone Pezzuto, Ali Gharaviri, Rolf Krause, Paris Perdikaris, and Francisco Sahli Costabal. Fast characterization of inducible regions of atrial fibrillation models with multi-fidelity gaussian process classification. *Frontiers in Physiology*, pp. 260, 2022.

Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.

Carolyn Gordon, David L Webb, and Scott Wolpert. One cannot hear the shape of a drum. *Bulletin of the American Mathematical Society*, 27(1):134–138, 1992.

Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 2021.

Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

Lukas Koestler, Daniel Grittner, Michael Moeller, Daniel Cremers, and Zorah Lähner. Intrinsic neural fields: Learning functions on manifolds. *arXiv preprint arXiv:2203.07967*, 2022.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36(8):962–977, 2021.

Ahmad Nasikun, Christopher Brandt, and Klaus Hildebrandt. Fast approximation of laplace-beltrami eigenproblems. In *Computer Graphics Forum*, volume 37, pp. 121–134. Wiley Online Library, 2018.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

Carlos Ruiz Herrera, Thomas Grandits, Gernot Plank, Paris Perdikaris, Francisco Sahli Costabal, and Simone Pezzuto. Physics-informed neural networks to learn cardiac fiber orientation from multiple electroanatomical maps. *Engineering with Computers*, pp. 1–17, 2022.

Raif M Rustamov et al. Laplace-beltrami eigenfunctions for deformation invariant shape representation. In *Symposium on geometry processing*, volume 257, pp. 225–233, 2007.

Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

Jonathan D Smith, Kamyar Azizzadenesheli, and Zachary E Ross. Eikonet: Solving the eikonal equation with deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 59 (12):10685–10696, 2020.

N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pp. 1383–1392. Wiley Online Library, 2009.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 311–318, 1994.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

A POISSON EQUATION WITH EXACT EIGENFUNCTIONS

In this example, we select a case that should favor traditional PINNs: solving a Poisson equation in a simple square domain. Here, we fabricate the following solution:

$$u(x,y) = \left(x^2-1\right)\left(y^2-1\right)\exp\left(-(x-y)^2/l^2\right) \tag{13}$$

which can be seen in the left panel of Figure 6. Then, we define the following boundary value problem to be tackled by PINNs.

$$\begin{aligned} \Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}),\ \boldsymbol{x}=\{x,y\}\in\mathcal{B} \\ u(\boldsymbol{x}) &= 0,\ \boldsymbol{x}\in d\mathcal{B}. \end{aligned} \tag{14}$$

The domain $\mathcal{B}$ is defined by a square in $[-1,1]\times[-1,1]$. The expression for $f(\boldsymbol{x})$ can be obtained by applying the Laplacian to the manufactured solution $u(\boldsymbol{x})$. In this domain, we can compute the eigenfunctions of the Laplace operator with homogeneous Neumann boundary conditions analytically, which correspond to:

$$v_{l,m}(x,y) = \cos\left(\pi\frac{lx}{2}\right)\cos\left(\pi\frac{my}{2}\right) \tag{15}$$

with $l,m=1,...,\sqrt{N}$, where N is the total number of eigenfunctions. We note that the boundary value problem described in equation 15 has Dirichlet boundary conditions, such that these eigenfunctions do not provide a suitable basis to solve this problem. In this case, we only provide data at the boundary $\boldsymbol{x}\in d\mathcal{B}$ and the values of $f(\boldsymbol{x})$ in the domain $\boldsymbol{x}\in\mathcal{B}$, effectivelly solving the forward boundary value problem shown in (15). The partial differential equation loss in this example corresponds to:

$$MSE_{\text{PDE}}(\boldsymbol{\theta}) = \sum_{i}^{R}\left(\sum_{j\in N(i)} A_{ij}\hat{u}_j - f_i\right)^2 \tag{16}$$

where the index i represents the nodes of the mesh inside the domain $\mathcal{B}$, $j\in N(i)$ represents the neighbor nodes of the node i, including itself. The term $\hat{u}_j$ represents the output of the neural network at the nodal locations j that depends on trainable parameters $\boldsymbol{\theta}$, and f_i corresponds to $f(\boldsymbol{x}_i)$, with $\boldsymbol{x}_i$ the position of node i. For traditional PINNs, we use automatic differentiation to compute $\Delta\hat{u}$ and construct the loss term MSE_{PDE}. We use neural networks with 3 hidden layers of 100 neurons and train with ADAM for 50,000 iterations.

We perform a sensitivity study to understand the influence of the number of eigenfunctions and the size of the discretization used to evaluate the operator $\Delta[\cdot]$ and the eigenfunctions computed numerically. We vary the number of eigenfuctions computed analytically and numerically from 9 to 400, and the size of finite elements between 0.02 to 0.2. Figures 6 and 7 summarize the results of this experiment. First, we note that traditional PINNs perform very well as expected, obtaining a mean squared error of $1.72\cdot10^{-5}$. Then, we observe that the results of Δ-PINNs are sensitive to both the number of eigenfuctions and the discretization. When using the exact, analytical eigenfunctions, we are able to obtain low errors if we use more than 9 eigenfunctions for certain discretizations. When using the numerically obtained eigenfunctions, the same is true, expect for the case of 400

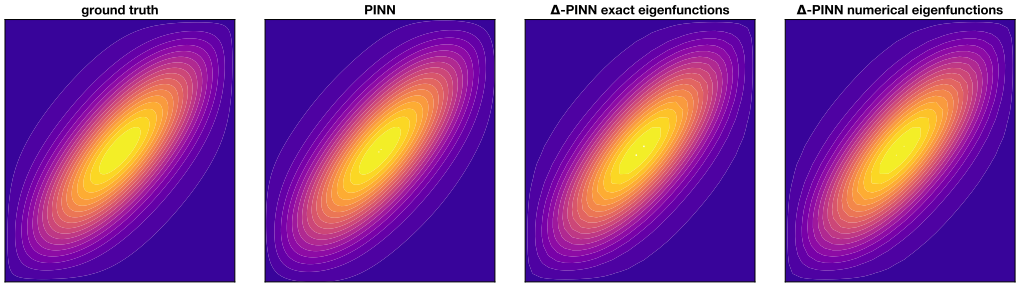

Figure 6: **Poisson example.** The ground truth manufactured solution, the approximation of traditional PINNs, $\Delta-$PINNs using the exact eigenfunctions as input and $\Delta-$PINNs using numerically approximated eigenfunctions by the finite element methods. For both cases of $\Delta-$PINNs, we use 100 eigenfunctions and an element size = 0.066 to discretize the operators.

eigenfunctions. In this case, we cannot use a bigger discretization than 0.1 because the number of nodes becomes less than 400. The discretization size has a bigger impact on the errors for both cases. There is a sweet spot between 0.066 and 0.1 where most cases produce low error. The error when the discretization is big is expected, since the Laplace operator will not be accurately approximated. The error when the discretization is small is not straightforward to explain, but we hypothesize that in these cases the MSE_{PDE} term becomes less important during training as the differences for $\hat{u}$ become smaller as the scale of the element decreases. In Figure 6, we can see qualitatively that both PINNs and Δ-PINNs can successfully solve this boundary value problem.

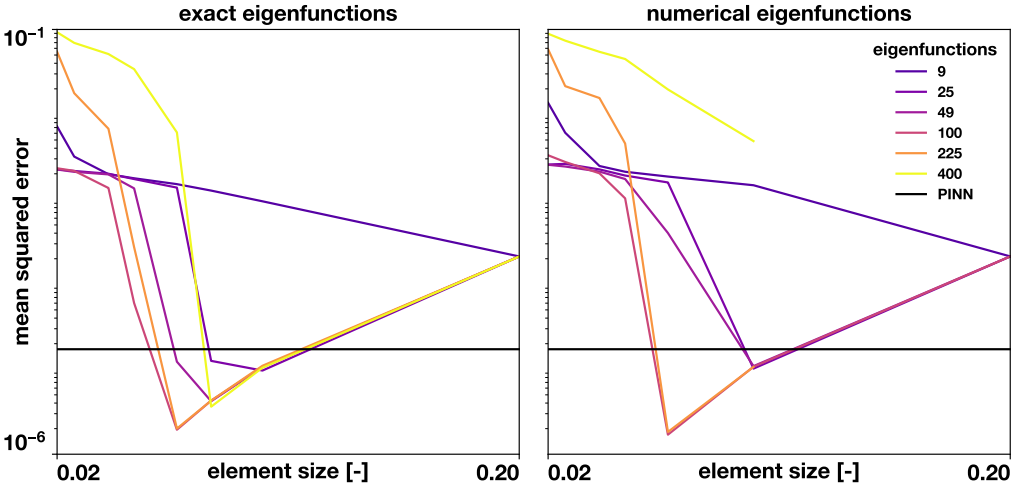

Figure 7: **Poisson example sensitivity analysis**. We run the Poisson example varying the two hyper parameters introduced by $\Delta-$PINNs: the number of eigenfunctions used as input and the element size used to discretize the operators. We test $\Delta-$PINNs using the exact eigenfunctions and the ones approximated by finite elements to understand the impact of this approximation.

B Regression with $\Delta-$NN

In this section we test the capabilities of a neural network that takes as input the Laplace-Beltrami eigenfunctions to perform a regression task on a manifold. In this case, we do not include any physics that may act as a regularization of the predicted function. For this benchmark, we compare against Gaussian process regression with a Matérn kernel, which can be approximated on the manifold using the Laplace-Beltrami eigenfunctions as well Borovitskiy et al. (2020). We choose a dragon geometry

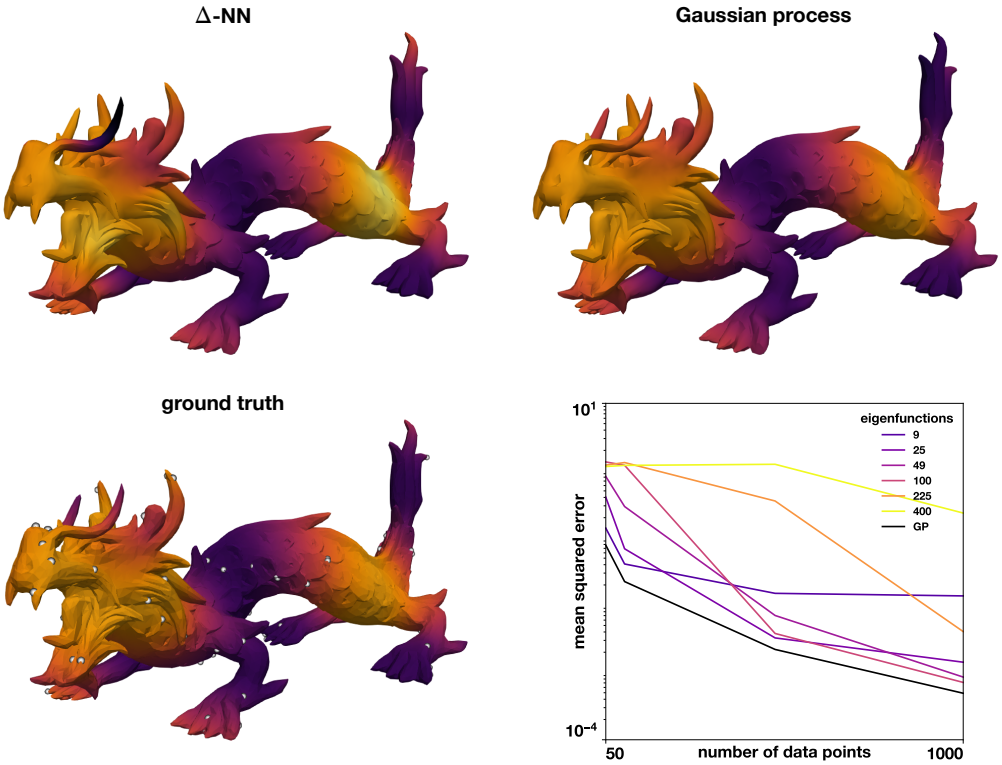

Figure 8: **Regression on a dragon.** We show the results when 100 data points for $\Delta-$NN using 25 eigenfunctions and Gaussian process. The ground truth is shown in the lower left, along with the training points, shown in white. The error comparison for different levels of data availability and different number of eigenfunctions for $\Delta-$NN are shown in the lower right panel.

9,057 vertices and 18,110 triangles. We create a synthetic ground truth function by computing the geodesic distance from a vertex $d(\boldsymbol{x})$:

$$f(\boldsymbol{x}) = \sin\left(\pi \frac{d(\boldsymbol{x})}{d_{max}}\right) \tag{17}$$

where d_{max} corresponds to the maximum geodesic distance from the selected vertex. We train with 50, 100, 500 and 1000 data points, randomly selecting them. We repeat this process five times and report the median error. For the neural network ($\Delta-$NN), we train using 9, 25, 49, 100, 225 or 400 eigenfunctions. We use one hidden layer of 100 neurons and hyperbolic tangent activations. For the Gaussian process, we select $\nu = 3/2$ for the Matérn kernel and approximate it with 1000 eigenfunctions. The results are shown in Figure 8. In the lower right panel, we can see that the accuracy of $\Delta-$NN depends on the number of eigenfunctions used as input. When there are few data points available, it is preferible to have less eigenfunctions of lower frequencies to avoid overfitting. As the number of data points increases, more capacity is needed to approximate the data and the error decreases when more eigenfunctions are used. Nonetheless, using a high number of eigenfunctions, such as 225 and 400, never produces good results. On the other hand, Gaussian process regression excels in approximating this smooth function and always provides lower errors than $\Delta-$NNs. We also show in Figure 8, top row, a comparison of the output produced by $\Delta-$NN and Gaussian process when training with 100 points. In general, where there is data, both methods do a good job approximating the function. However, in the absence of data points, such as the horns of the dragon, $\Delta-$NN tends to predict more extreme values that increase the overall error. In this case, the overall mean squared error drops from $6.8 \cdot 10^{-2}$ $\Delta-$NN to $1.8 \cdot 10^{-2}$ for Gaussian process regression. We see that, in general, $\Delta-$NNs tend to overfit the data, and the addition of known physics across the domain has a regularization effect that improves the prediction, as seen in Figure 2 and 4.