



**HAL**  
open science

# Operator approximation of the wave equation based on deep learning of Green's function

Ziad Aldirany, Régis Cottureau, Marc Laforest, Serge Prudhomme

## ► To cite this version:

Ziad Aldirany, Régis Cottureau, Marc Laforest, Serge Prudhomme. Operator approximation of the wave equation based on deep learning of Green's function. *Computers & Mathematics with Applications*, 2023, 159, pp.21-30. 10.1016/j.camwa.2024.01.018 . hal-04190380

**HAL Id: hal-04190380**

**<https://hal.science/hal-04190380>**

Submitted on 29 Aug 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Operator approximation of the wave equation based on deep learning of Green's function

Ziad Aldirany<sup>1</sup>, Régis Cottureau<sup>2</sup>, Marc Laforest<sup>1</sup>, and Serge Prudhomme<sup>1</sup>

<sup>1</sup>*Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, Québec, Canada*

<sup>2</sup>*Aix-Marseille Université, CNRS, Centrale Marseille, LMA UMR 7031, Marseille, France*

August 29, 2023

## Abstract

Deep operator networks (DeepONets) have demonstrated their capability of approximating nonlinear operators for initial- and boundary-value problems. One attractive feature of DeepONets is their versatility since they do not rely on prior knowledge about the solution structure of a problem and can thus be directly applied to a large class of problems. However, convergence in identifying the parameters of the networks may sometimes be slow. In order to improve on DeepONets for approximating the wave equation, we introduce the Green operator networks (GreenONets), which use the representation of the exact solution to the homogeneous wave equation in term of the Green's function. The performance of GreenONets and DeepONets is compared on a series of numerical experiments for homogeneous and heterogeneous media in one and two dimensions.

**Keywords:** Deep learning, Wave equation, Neural networks, Green's function, Fundamental solution, Deep operator networks, Physics-informed neural networks

## 1 Introduction

In the last few years, a large amount of work, see e.g. [10, 4, 20, 19], has been devoted to using deep learning methods for the solution of PDE-based problems, such as in fluid dynamics, elasticity, meteorology, etc. These works have been motivated by the ability of deep neural networks to approximate large classes of functions in high dimension over complex domains [9, 21].

Prominent deep learning methods for solving partial differential equations rely on either learning the solution to the problem, as presented in [22, 28, 29], or learning the operators that describe the physical problem, as introduced in [18, 16, 17, 27]. In the first approach, the solution is approximated with a neural network by minimizing the residual of the PDE, as in so-called physics-informed neural networks (PINNs) introduced by [22]. The second approach learns the differential operator for a given family of parameters, e.g. deep operator networks (DeepONets) as presented in [18], thus allowing one to subsequently approximate the solution to a physical problem for a specific parameter in the vicinity of the trained parameters. The training of such a neural network can turn out to be quite expensive, but needs to be done only once. Computing the solution for a new parameter requires only one forward pass in the online phase, which is usually cost-effective. This makes the operator approximation method very attractive when the physical problem needs to be solved for a wide range of parameter values. For example, in seismology, uncertainties

36 in the Earth’s properties often require thousands of simulations to obtain those solutions that best fit the  
37 measured data.

38 In physics-based deep learning approaches, the training of the network does not require using any data  
39 as it is essentially based on the physics of the problem, in the sense that it approximates the solution  
40 to the partial differential equations along with the boundary and initial conditions. This is achieved by  
41 minimizing the residual associated with the partial differential equations. The most common approach is  
42 to consider a Least Squares approach, in which the loss function is defined in terms of the  $L^2$ -norm of the  
43 strong form of the residual [22, 27]. In case the solution of the problem lacks regularity, one can define  
44 the loss function by minimizing the energy potential in the case of symmetric problems, such as the Ritz  
45 formulation [28], or by minimizing the norm of the residual functional, see e.g. [12, 29]. The choice of the  
46 loss function that one should consider for the solution of PDE problems with neural networks, is still viewed  
47 as an open question [26]. However, evaluation of the residual has been made more amenable thanks to  
48 automatic differentiation [3]. In this work, we will formulate the problem using the classical Least Squares  
49 method that involves the strong form of the residual due to its simplicity and convergence ability.

50 The main objective in this paper focuses on approximating the operator of the wave equation for a family  
51 of initial conditions. The architecture presented in DeepONets is general and can be applied to a large  
52 class of parametric PDEs. In order to improve on DeepONets, we propose here an approach based on the  
53 representation of the exact solution to the homogeneous wave equation on unbounded domains in terms of the  
54 Green’s function (for details on Green’s function, we refer the reader to [6]). The method will be heretofore  
55 referred to as the Green operator networks (GreenONets). GreenONets provide an approximation of the  
56 operator of the wave equation in terms of the corresponding Green’s function. Similar techniques based on  
57 the approximation of the Green’s function have been recently considered for the solution of linear and non-  
58 linear operators, see e.g. [5, 7, 17]. The architecture used in [5] is similar to the one presented in our work.  
59 However, in their work, the authors approximate the operator for a family of forcing terms and the training  
60 is performed by minimizing the error between the approximated solution and the exact solution, which  
61 makes their approach data-based, while the training in our work is based on the physics of the problem.  
62 In [7], a dual-autoencoder architecture is presented to approximate the operator for non-linear boundary  
63 value problems, by linearizing the problem and approximating the corresponding Green’s function. The  
64 authors in [17] introduced the graph neural operator, which is inspired by the Green’s function. However,  
65 the graph neural operator does not compute the Green’s function but aims at learning a corresponding kernel  
66 function using an iterative architecture. We will illustrate on a series of numerical examples involving the  
67 homogeneous and heterogeneous wave equations that the approximation of the operator using GreenONets  
68 exhibits in general better results in terms of accuracy and convergence when compared to DeepONets.

69 The paper is organized as follows. We introduce in Section 2 the model problem and some preliminary  
70 notations. In Section 3, we briefly present DeepONets and its architecture. We then describe in Section 4  
71 the main features of GreenONets. We compare the performance of GreenONets and DeepONets on several  
72 numerical results for the homogeneous and heterogeneous wave equation in Section 5. Finally, our main  
73 conclusions and potential extensions of the current research work are summarized in Section 6.

## 74 2 Preliminaries

75 We introduce here some preliminaries and notations in order to describe the notion of operator of the wave  
76 equation using neural networks. We first present the model problem and continue with a brief account of

77 neural networks and the use of PINNs to solve initial boundary-value problems.

## 78 2.1 Model problem

79 The linear wave equation describes small perturbations from the steady state of a system that locally behaves  
 80 like an elastic body. The material system is entirely characterized in its domain  $\Omega \subset \mathcal{R}^d$ ,  $d = 1, 2$ , or  $3$ ,  
 81 by the distribution of a bounded function  $c(\mathbf{x})$  in  $\Omega$ . The perturbations are introduced either as initial  
 82 displacements  $u_0$ , initial velocities  $u_1$ , or as perturbations entering the domain through its boundary  $\partial\Omega$   
 83 at different times. For the sake of simplicity, we will assume that the boundary conditions are given by  
 84 homogeneous Dirichlet boundary conditions on the displacement. To be more specific, given a wave speed  
 85  $c(\mathbf{x})$ , initial displacement  $u_0(\mathbf{x})$ , initial velocity  $u_1(\mathbf{x})$ , and a final time  $T > 0$ , the problem is to find the  
 86 perturbation  $u(\mathbf{x}, t)$ , for all  $\mathbf{x} \in \bar{\Omega}$  and  $t \in (0, T)$  such that

$$\partial_{tt}u(\mathbf{x}, t) - c^2(\mathbf{x})\nabla^2u(\mathbf{x}, t) = 0, \quad \forall(\mathbf{x}, t) \in \Omega \times (0, T), \quad (1)$$

subjected to the initial and boundary conditions

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \quad \forall\mathbf{x} \in \Omega, \quad (2)$$

$$\partial_tu(\mathbf{x}, 0) = u_1(\mathbf{x}), \quad \forall\mathbf{x} \in \Omega, \quad (3)$$

$$u(\mathbf{x}, t) = 0, \quad \forall(\mathbf{x}, t) \in \partial\Omega \times (0, T). \quad (4)$$

87 Our goal is to obtain a neural network approximation of the (inverse) operator of the wave equation that  
 88 would provide the solution  $u = u(\mathbf{x}, t; u_0)$  for a family of initial conditions  $u_0$ . In this case, the networks will  
 89 be trained on a family on initial conditions generated by Gaussian random fields (GRF) [23]. The resulting  
 90 neural network solution will allow one to compute an approximation to the wave equation for any initial  
 91 condition. The hope is that this approximation should be accurate for initial conditions that are close to  
 92 those used in the training phase. For the sake of simplicity, we shall focus mostly on the case where  $u_1 = 0$   
 93 but we will indicate how our approach can be extended to non-zero initial velocities  $u_1$ .

## 94 2.2 Green's functions

95 The Green's function of the operator defined previously is defined as the solution of the same problem with  
 96 an impulse (localized in space) as initial condition. More precisely,  $g(\mathbf{x}, t, \xi)$  is defined, for all  $\mathbf{x} \in \Omega$  and  
 97  $t \in (0, T)$ , as the solution of

$$\partial_{tt}g(\mathbf{x}, t, \xi) - c^2(\mathbf{x})\nabla^2g(\mathbf{x}, t, \xi) = 0, \quad \forall(\mathbf{x}, t) \in \Omega \times (0, T), \quad (5)$$

subjected to the initial and boundary conditions

$$g(\mathbf{x}, 0, \xi) = \delta(\mathbf{x} - \xi), \quad \forall\mathbf{x} \in \Omega, \quad (6)$$

$$\partial_tg(\mathbf{x}, 0, \xi) = 0, \quad \forall\mathbf{x} \in \Omega, \quad (7)$$

$$g(\mathbf{x}, t, \xi) = 0, \quad \forall(\mathbf{x}, t) \in \partial\Omega \times (0, T). \quad (8)$$

98 An interesting feature of the Green's function is that the solution  $u(x, t)$  of the problem presented in the  
 99 previous section can be obtained as a simple convolution with the initial condition  $u_0(\mathbf{x})$ :

$$u(x, t) = \int_{\Omega} g(x, t, \xi)u_0(\xi)d\xi. \quad (9)$$

100 Although we have considered here the case when  $u_1(x)$  vanishes, an additional Green's function can be  
 101 defined with the Dirac delta function on Eq. (7) rather than on Eq. (6), so that the solution would become  
 102 a sum of two convolutions by the principle of superposition.

103 A Green's function can be defined for any geometry of the domain and any distribution of properties  $c(x)$ .  
 104 However, in the more simple case of an unbounded domain  $\Omega = \mathbb{R}$  and homogeneous properties  $c(x) = c$ ,  
 105 the Green's function takes the simple form

$$g(x, t, \xi) = \frac{1}{2} \delta(ct - |x - \xi|). \quad (10)$$

106 More examples of Green's function (for higher dimensions and bounded domains) can be found in [6] or [11].

### 107 2.3 Neural networks

108 Neural networks have been the subject of intensive research in the past decades [14, 8] and more recently  
 109 have been used as a discretization approach for solving differential equations [25, 22]. By definition, a neural  
 110 network maps an input into an output by a composition of linear and nonlinear functions, with adjustable  
 111 weights and biases. The objective is usually to train the network by adjusting its weights and biases in  
 112 order to minimize some measure of error between the output and the corresponding target values over a  
 113 specific training set. In this sense, the optimal neural network is very much like the least-squares fit of some  
 114 fixed model to experimental data, but in contrast to a least-squares fit, the minimization problem might not  
 115 always possess a unique solution. The resulting network can then be used as a predictive model that should  
 116 hopefully provide accurate output when considering a wider set of input. There exist several neural network  
 117 architectures, e.g. convolutional neural networks [14], feedforward neural networks [15]. We describe below  
 118 the feedforward neural networks (FNN) that will be used later with DeepONets and GreenONets.

119 Let us consider a FNN with  $d$  hidden layers, each layer having a width  $N_i$ ,  $i = 1, \dots, d$ , and let  $N_0$  denote  
 120 the size of the input data and  $N_{d+1}$  the size of the output layer. Denoting the activation function by  $\sigma$ , the  
 121 neural network with input  $(\mathbf{x}, t)$  and output  $u$  is defined as

$$\begin{aligned} \text{Input layer:} \quad & \mathbf{z}_0 = (\mathbf{x}, t), \\ \text{Hidden layers:} \quad & \mathbf{z}_i = \sigma(W_i \mathbf{z}_{i-1} + \mathbf{b}_i), \quad i = 1, \dots, d, \\ \text{Output layer:} \quad & u = W_{d+1} \mathbf{z}_d + \mathbf{b}_{d+1}, \end{aligned} \quad (11)$$

122 where  $W_i$  is the weights matrix of size  $N_i \times N_{i-1}$  and  $\mathbf{b}_i$  is the biases vector of size  $N_i$ . For convenience, we  
 123 will combine the weights and biases into the single parameter  $\theta$  of the neural network. In this work, we shall  
 124 consider the tanh activation function, but other activation functions could be used as well.

### 125 2.4 Physics-informed neural networks and operator approximation

126 We recall here the physics-informed neural network approach for solving partial differential equations, first  
 127 introduced in [22], as applied to the wave equation. We denote the residual associated with the partial  
 128 differential equation of the wave equation as

$$R(\mathbf{x}, t, u) = \partial_{tt} u(\mathbf{x}, t) - c^2(\mathbf{x}) \nabla^2 u(\mathbf{x}, t), \quad \forall (\mathbf{x}, t) \in \Omega \times (0, T), \quad (12)$$

129 introduce the residual associated with the Dirichlet boundary condition as:

$$B(\mathbf{x}, t, u) = u(\mathbf{x}, t), \quad \forall (\mathbf{x}, t) \in \partial\Omega \times (0, T), \quad (13)$$

and the residuals associated with the initial conditions as:

$$I_1(\mathbf{x}, u) = u(\mathbf{x}, t = 0) - u_0(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega, \quad (14)$$

$$I_2(\mathbf{x}, u) = \partial_t u(\mathbf{x}, t = 0), \quad \forall \mathbf{x} \in \Omega. \quad (15)$$

We note that if  $u_1(\mathbf{x})$  is different from zero, then it should be subtracted from Eq. (15).

In PINNs, the goal is to obtain the solution  $u$  to the problem by approximating  $u$  with a neural network  $u_\theta(\mathbf{x})$ . The training is usually performed by minimizing the following loss function:

$$L(\theta) = w_r \int_0^T \int_\Omega R(\mathbf{x}, t, u_\theta)^2 dx dt + w_{bc} \int_0^T \int_{\partial\Omega} B(\mathbf{x}, t, u_\theta)^2 dx dt + w_{ic} \int_\Omega I_1(\mathbf{x}, u_\theta)^2 + I_2(\mathbf{x}, u_\theta)^2 dx, \quad (16)$$

where  $w_r$ ,  $w_{bc}$ , and  $w_{ic}$  are weighting coefficients.

The minimization is usually done using a gradient-based method, e.g. ADAM [13], since the minimization problem is non-convex with respect to the trained parameters. Some of the main advantages of the PINNs is that it is a meshless method, and therefore we eliminate the process of mesh construction that can be very time consuming. Moreover, the implementation of the different types of boundary and initial conditions is similar in all cases and can be simply done by adding an extra weighted term in the loss function as presented in (16).

### 3 Deep operator networks

If one seeks to calculate the solution using PINNs for several values of some parameters  $\mathbf{s}$  of the model problem, one should have to recompute the approximate solution  $u_\theta$  for each instance of  $\mathbf{s}$  by training the network from the beginning. This becomes quickly inefficient if the solution has to be evaluated for multiple values of the parameters, such as in a multi-query approach for uncertainty quantification or optimization. An alternative approach is to construct a surrogate model, in which the solution is searched as an operator acting on the parameters  $\mathbf{s}$ , i.e.  $u = Q(\mathbf{s})$ . Similarly to PINNs, the operator  $Q$  can be approximated using deep learning by minimizing the loss function associated with the residuals of the partial differential equation and the boundary and initial conditions for a family of parameters. This approach becomes more attractive when the initial boundary value problem should be solved multiple times for different parameters, since it requires only one forward pass to compute the solution of a new parameter in the online phase.

We briefly review the deep operator networks first introduced by [18]. In this work, we will be learning the operator from the partial differential equation and the initial and boundary conditions. In other words, the physics-informed DeepONets, described in [27], will be presented.

Given Banach spaces  $\mathcal{U}$  and  $\mathcal{S}$ , we want to learn the operator  $Q : \mathcal{S} \rightarrow \mathcal{U}$  such that for any input parameter  $s \in \mathcal{S}$  (which in our case represents an initial condition),  $Q(s) \equiv u \in \mathcal{U}$  is the solution of problem (12) with the boundary conditions (13). In order to approximate the operator  $Q$ , we present the unstacked DeepONets architecture originally introduced in [18] and schematically shown in Figure 1. We start by defining the input vector  $\mathbf{s}$  of initial conditions  $[u_0(\mathbf{x}_i)]_{i=1, \dots, m}$  evaluated at a collection of  $m$  points  $\{\mathbf{x}_i\}_{i=1}^m$ , known as sensors. Then, as illustrated in Figure 1, the operator is approximated as

$$\hat{Q}(\mathbf{s})(\mathbf{x}, t) = \sum_{k=1}^q b_k(\mathbf{s}) t_k(\mathbf{x}, t), \quad (17)$$

where  $\{b_k\}_{k=1}^q$  is the output of the branch network that takes  $\mathbf{s}$  as an input and  $\{t_k\}_{k=1}^q$  is the output of the trunk network that takes  $\mathbf{x}$  as an input. The value of  $q$  will be chosen in the numerical experiments as the

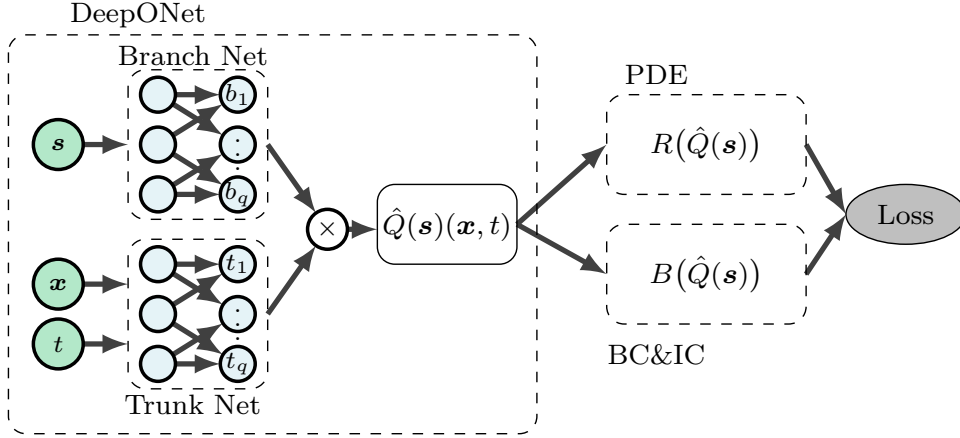


Figure 1: Illustration of the architecture of the unstacked DeepONets. It consists of two networks, the branch net and the trunk net. The branch net takes as input the input vector while the trunk net takes the input coordinates. Their outputs are merged with dot product to give the approximated operator. The network is trained to minimize the loss function that consists of the residuals of the partial differential equation and the initial and boundary conditions.

162 width of the layers in the neural network. We will consider a simple FNN for both the branch and trunk  
 163 networks. We note that a convolutional neural network [14] could also be used for the branch network when  
 164 working with uniformly distributed sensors.

We consider here the physics-informed DeepONets, where the network is trained by penalizing the residuals associated with the governing partial differential equation and with the initial and boundary conditions for a family of  $N$  input functions  $\{\mathbf{s}^{(i)}\}_{i=1}^N$ . The loss function in this case reads:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N \left[ w_r \int_0^T \int_{\Omega} R(\mathbf{x}, t, \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}, t))^2 dx dt + w_{bc} \int_0^T \int_{\partial\Omega} B(\mathbf{x}, t, \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}, t))^2 dx dt + w_{ic} \int_{\Omega} I_1(\mathbf{x}, \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}, t))^2 + I_2(\mathbf{x}, \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}, t))^2 dx \right]. \quad (18)$$

## 165 4 Green operator networks

166 The architecture presented in the DeepONets is a general architecture that works can accommodate different  
 167 problems with different input parameters. Our objective here is to develop an approach that improves upon  
 168 the efficiency of the DeepONets for certain types of parameters. We will focus on the solution of the wave  
 169 equation for homogeneous and heterogeneous materials, as presented in Section 2.1. We thus propose the  
 170 Green operator networks (GreenONets), that approximate the Green's function of the operator, to solve the  
 171 aforementioned problem.

172 In this work, we are interested in learning the operator of the wave equation for different initial conditions,  
 173 i.e. the input function is defined as  $s = u_0$ . Therefore, instead of using a general architecture as the  
 174 DeepONets, we introduce the Green operator networks, shown in Figure 2, as a discrete approximation of  
 175 the integral in Eq. (9). The GreenONet is defined as

$$\hat{Q}(\mathbf{s})(\mathbf{x}, t) = \frac{1}{m} \sum_{i=1}^m G(\mathbf{x}, t, \mathbf{x}_i) u_0(\mathbf{x}_i), \quad (19)$$

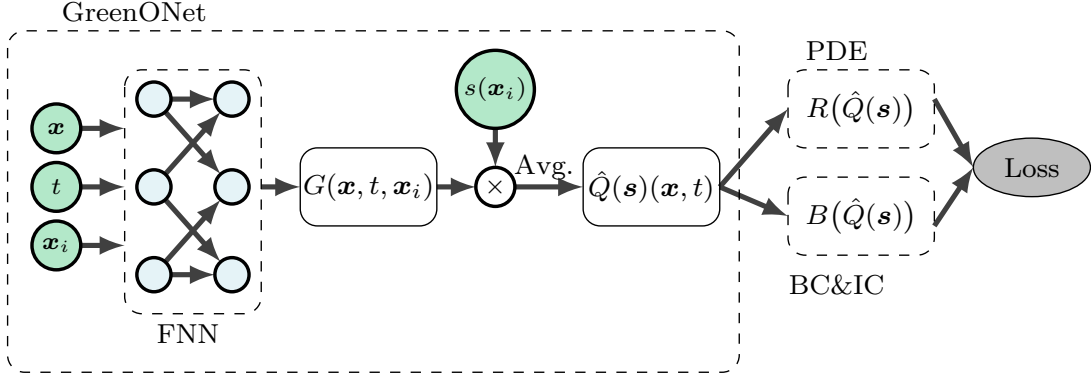


Figure 2: Illustration of the architecture of the GreenONets. A FNN takes as input the coordinates and sensor points and outputs an approximated Green’s function  $G$  with respect to each sensor point. Then, the operator is computed by averaging the product of the Green’s function and the input function over the sensor points. The network is then trained to minimize the loss function that consists of the residuals associated with the partial differential equation and the initial and boundary conditions.

176 where  $G$  is a simple feedforward neural network. We note that the formulation of GreenONets depends  
 177 explicitly on the sensor points  $\{\mathbf{x}_i\}_{i=1}^m$ . Similar to the physics-informed DeepONets, the GreenONets are  
 178 trained by minimizing the loss function (18). We notice that in our approximation, the solution is zero if  
 179 the initial condition is zero. However, if the Dirichlet boundary condition or the initial speed are different  
 180 from zero, we should add a term in (19), that should be approximated by a new network, to compensate for  
 181 these conditions.

182 Although the exact solution in (9) is only presented for a homogeneous material and an unbounded  
 183 domain, the following numerical results show that GreenONets yield better results when compared to Deep-  
 184 ONets for bounded domains with homogeneous or heterogeneous properties.

## 185 5 Numerical results

186 In this section, we approximate the operator of the wave equation for homogeneous and heterogeneous  
 187 materials in the case of a family of initial conditions, in order to show the effectiveness of GreenONets when  
 188 compared to DeepONets.

189 We consider  $\Omega = (-1, 1)^d$  and we set  $u_1(x) = 0$  in all cases. In order to define the training set, on which  
 190 we want to minimize our residuals, we start by defining a family of  $N$  input functions  $\{\mathbf{s}^{(i)}\}_{i=1}^N$ . For each  
 191  $\mathbf{s}^{(i)}$ , we randomly define  $\{(\mathbf{x}_{ic,j}^{(i)}, 0)\}_{j=1}^{P_{ic}}$  on which we will penalize the initial conditions,  $\{(\mathbf{x}_{bc,j}^{(i)}, t_{bc,j}^{(i)})\}_{j=1}^{P_{bc}}$   
 192 on the boundary of the domain at different times, for which we will penalize the boundary conditions, and  
 193  $\{(\mathbf{x}_{r,j}^{(i)}, t_{r,j}^{(i)})\}_{j=1}^{P_r}$  on  $(-1, 1)^d \times (0, T)$ , for which we will penalize the bulk residual. Using the sampling points  
 194 to estimate numerically the integrals, the loss function (18) is approximated as

$$\mathcal{L}(\theta) = w_r \mathcal{L}_r(\theta) + w_{bc} \mathcal{L}_{bc}(\theta) + w_{ic} \mathcal{L}_{ic}(\theta), \quad (20)$$



195 where

$$\begin{aligned}\mathcal{L}_r(\theta) &= \frac{1}{NP_r} \sum_{i=1}^N \sum_{j=1}^{P_r} \left| \partial_{tt} \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}_{r,j}^{(i)}, t_{r,j}^{(i)}) - c(\mathbf{x}_{r,j}^{(i)})^2 \nabla^2 \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}_{r,j}^{(i)}, t_{r,j}^{(i)}) \right|^2, \\ \mathcal{L}_{bc}(\theta) &= \frac{1}{NP_{bc}} \sum_{i=1}^N \sum_{j=1}^{P_{bc}} \left| \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}_{bc,j}^{(i)}, t_{bc,j}^{(i)}) \right|^2, \\ \mathcal{L}_{ic}(\theta) &= \frac{1}{NP_{ic}} \sum_{i=1}^N \sum_{j=1}^{P_{ic}} \left| \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}_{ic,j}^{(i)}, 0) - u_0(\mathbf{x}_{ic,j}^{(i)}) \right|^2 + \left| \partial_t \hat{Q}(\mathbf{s}^{(i)})(\mathbf{x}_{ic,j}^{(i)}, 0) \right|^2.\end{aligned}$$

196 Again,  $w_r$ ,  $w_{ic}$ , and  $w_{bc}$  are the weighting coefficients. The initial conditions  $\mathbf{s}^{(i)}$  are randomly sampled  
 197 from a Gaussian random field (GRF), as presented by [18], with a defined length scale  $l$ . In the following  
 198 experiments, the FNNs in the DeepONets and GreenONets are defined with  $d = 6$  hidden layers and  $N_i = 50$   
 199 for all hidden layers. The loss function is minimized using the ADAM optimizer [13] with the default hyper-  
 200 parameters, while considering different learning rates for each experiment.

## 201 5.1 One-dimensional problems

202 We start by comparing the GreenONets with the DeepONets for the one-dimensional case, i.e.  $d = 1$ . The  
 203 input functions  $s$  are defined by a GRF and then modified to verify the homogeneous Dirichlet boundary  
 204 conditions by subtracting the proper linear function. Figure 3 shows examples of the modified GRF for length  
 205 scales  $l = 0.5$  and  $l = 0.1$ . The numerical comparison is performed for homogeneous and heterogeneous  
 206 material properties with different length scales in the GRF.

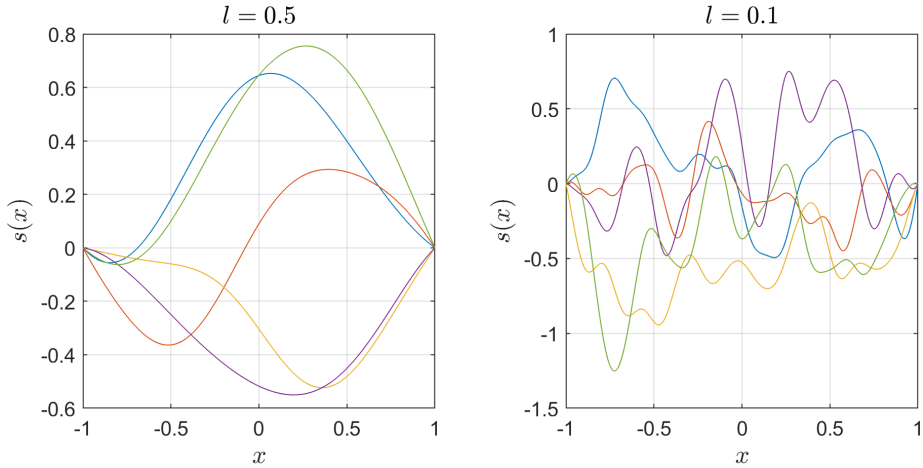


Figure 3: Examples of the input functions  $s^{(i)}$  for a length scale  $l = 0.5$  (left) and  $l = 0.1$  (right).

### 207 5.1.1 Homogeneous case with a length scale of 0.5

208 We first approximate the operator for a homogeneous material using DeepONets and GreenONets. The  
 209 sensor points are chosen uniformly with  $m = 21$  while the input parameters  $\mathbf{s}$  are generated using a GRF  
 210 with length scale  $l = 0.5$ . We take the initial learning rate as  $10^{-3}$  and let it decrease with a rate of 0.9995  
 211 at each epoch. In this example, we choose  $N = 1000$  and  $P_r = P_{bc} = P_{ic} = 10$ . The weights in (20) are set  
 212 to  $w_r = 0.1$  and  $w_{ic} = w_{bc} = 10$ . The training is done for 5000 epochs with 16 mini-batches.

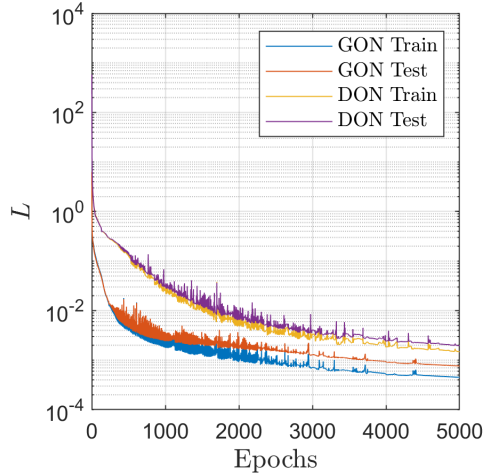


Figure 4: The evolution of the loss function on the training and testing sets during the training with GreenONets and DeepONets for the example of Section 5.1.1.

213 Figure 4 compares the evolution of the loss function on the training and testing sets for the GreenONets  
 214 and DeepONets. We observe that the loss functions decrease faster with GreeONets and after 5000 epochs  
 215 we have smaller losses when compared to the DeepONets loss functions. In order to verify our operators, we  
 216 compute the solution at  $t = 2$  for the initial conditions  $u_0(x) = (1 - x^2)^k$ , with  $k = 2$  and  $k = 10$ , as shown  
 217 in Figure 5 (left). We observe in Figure 5 (middle), that the pointwise error at  $t = 2$  for  $k = 2$  is slightly  
 218 larger when using DeepONets. However, as shown in Figure 5 (right), for  $k = 10$  the DeepONets solution  
 219 exhibits a maximum pointwise error of 0.14 while the maximum pointwise error for the GreenONets solution  
 220 is 0.04. Therefore, the GreenONets solutions seem to generalize better at higher frequencies.

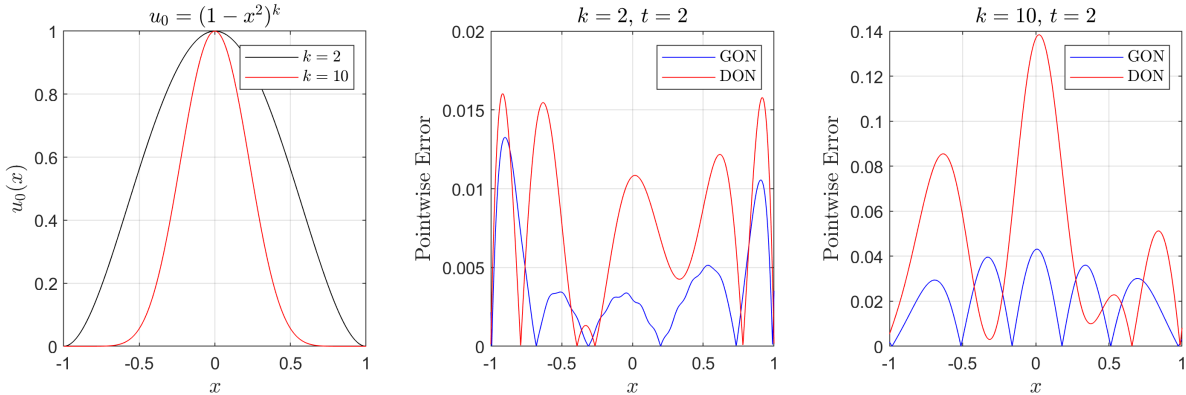


Figure 5: Example of Section 5.1.1: (left) Initial conditions with which we test the networks. (middle) Pointwise error at  $t = 2$  for  $k = 2$  using GreenONets and DeepONets. (right) Pointwise error at  $t = 2$  for  $k = 10$  using GreenONets and DeepONets.

### 221 5.1.2 Homogeneous case with a length scale of 0.1

222 Here, we solve the same problem as in the previous section but the input parameters  $s$  are defined using  
 223 a GRF with length scale  $l = 0.1$ . In other words, we now compare the two methods for higher frequency

224 solutions. The sensor points are defined uniformly with  $m = 60$ . We initialize the learning rate to  $5 \times 10^{-4}$   
 225 and let it decrease with a rate of 0.999 at each epoch. In this example, we take  $N = 3000$ ,  $P_r = 30$ , and  
 226  $P_{bc} = P_{ic} = 3$ . The weights associated with each component of the loss function (20) are  $w_r = 0.2$  and  
 227  $w_{ic} = w_{bc} = 100$ . We train both networks for 2000 epochs with 128 mini-batches.

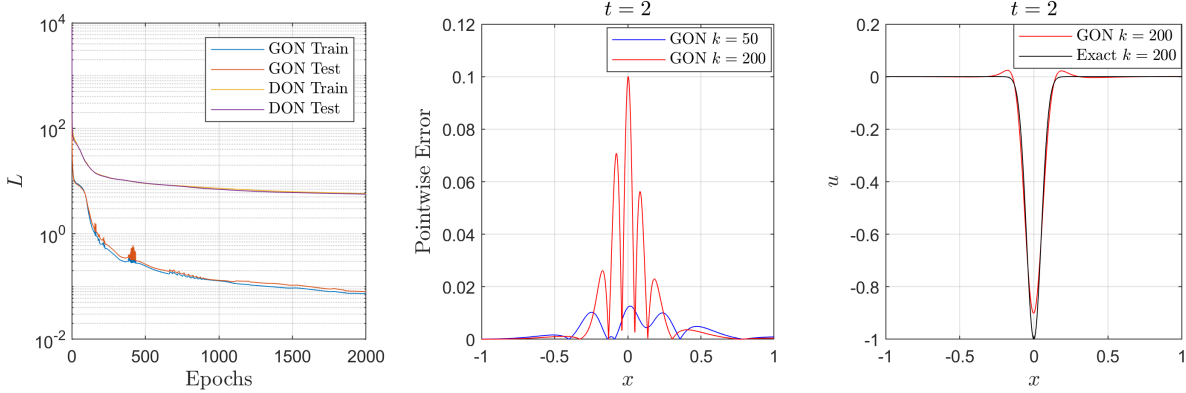


Figure 6: Example of Section 5.1.2: (left) Evolution of the loss function on the training and testing sets with GreenONets and DeepONets. (middle) Pointwise error at  $t = 2$  for  $k = 50$  and  $k = 200$  using GreenONets. (right) Comparison of the solution obtained by GreenONets at  $t = 2$  for  $k = 200$  with the exact solution.

228 We observe in Figure 6 (left) that, using the same hyper-parameters, the loss functions for the GreenONets  
 229 attain  $8 \times 10^{-2}$  in 2000 epochs while those for the DeepONets plateau earlier. In Figure 6 (middle), we show  
 230 the pointwise errors of the GreenONets solutions for  $u_0 = (1 - x^2)^k$ , with  $k = 50$  and  $k = 200$ . The maximum  
 231 pointwise error is around 0.01 for  $k = 50$  and around 0.1 for  $k = 200$ . The pointwise error of the DeepONets  
 232 solutions is not available since the loss functions did not converge. To better characterize the error for  
 233  $k = 200$ , we compare the solution using GreenONets to the exact solution at  $t = 2$  in Figure 6 (right). We  
 234 remark that the large errors are close to the propagating wave and did not spread in the rest of the solution.

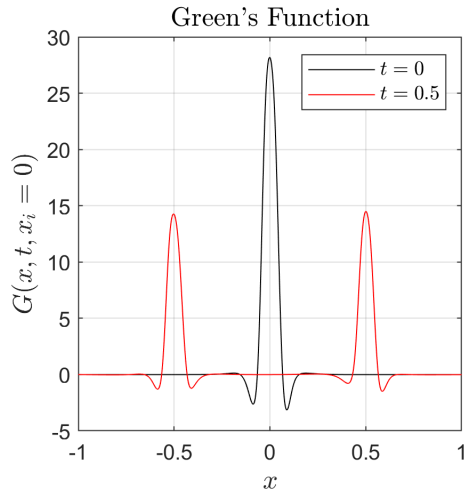


Figure 7: Example of Section 5.1.2: The approximated Green's function  $G(x, t, x_i)$ , for  $x_i = 0$  at  $t = 0$  and  $t = 0.5$ .

235 We show in Figure 7 the approximated Green's function  $G(x, t, x_i)$ , as computed by Equation (19), for

236  $x_i = 0$  at  $t = 0$  and  $t = 0.5$ . As expected, we observe that  $G$  approximates a Dirac delta function around  
 237  $x = 0$  at  $t = 0$ , which splits into two functions at  $t = 0.5$  with half the amplitude of the original one.

### 238 5.1.3 Heterogeneous case with a length scale of 0.3

239 In this section, we use the GreenONets to approximate the operator of the heterogeneous wave equation.  
 240 We consider the wave speed  $c(x)^2 = 1 + \mathcal{H}(x - 0.5)$ , where  $\mathcal{H}$  is the Heaviside function. We consider a family  
 241 of initial conditions with a length scale  $l = 0.3$  and use uniformly distributed sensor points with  $m = 30$ .  
 242 The learning rate is  $10^{-3}$  and decreases with a rate of 0.9995 per epoch. We set  $N = 2000$ ,  $P_r = 15$ , and  
 243  $P_{bc} = P_{ic} = 3$ . The weights of the loss functions are  $w_r = 1$  and  $w_{ic} = w_{bc} = 100$ . We divide our training  
 244 set into 32 mini-batches and train the networks for 2500 epochs.

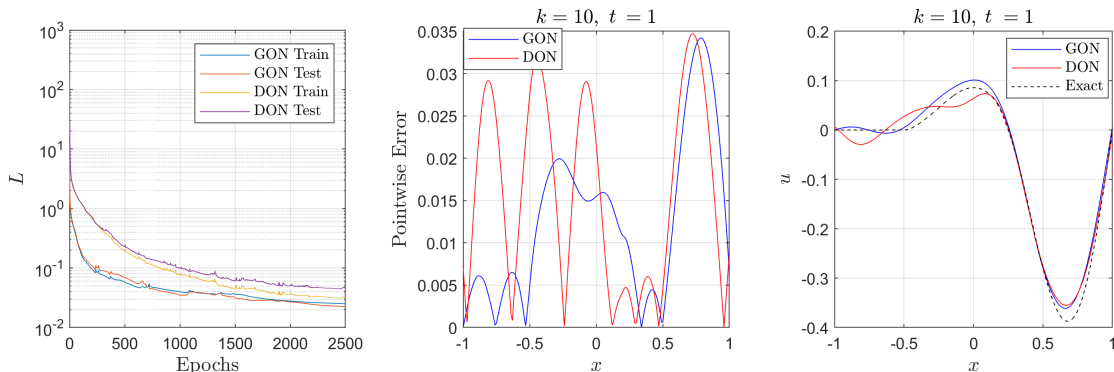


Figure 8: Example of Section 5.1.3: (left) Evolution of the loss function on the training and testing sets with GreenONets and DeepONets. (middle) Pointwise error at  $t = 1$  for  $k = 10$ , using GreenONets and DeepONets. (right) Comparison of the solutions obtained by GreenONets and DeepONets at  $t = 1$  for  $k = 10$  with an overkill solution using the spectral element method.

245 As shown in Figure 8 (left), the loss functions on the training set and testing set decrease faster using  
 246 GreenONets than with DeepONets. Figure 8 (middle) shows the pointwise error in the solution at  $t = 1$   
 247 using GreenONets and DeepONets with an initial condition  $u_0 = (1 - x^2)^{10}$ . The maximum pointwise errors  
 248 are similar for both networks with a value close to 0.035. We also plot the solutions at  $t = 1$  along with  
 249 the exact solution – actually, an overkill solution using the spectral element method, see e.g. [2, 24, 1],  
 250 and references therein – in Figure 8 (right). We remark that the errors in the solution of the GreenONets  
 251 remain localized around the main pulses and are proportional to the wave amplitude. However, in the case  
 252 of DeepONets, the pointwise errors have the tendency to spread over the whole domain. Therefore, one  
 253 could conclude that GreenONets tend to provide better approximations of the propagating waves without  
 254 introducing large errors away from the main pulses.

## 255 5.2 A two-dimensional example

256 In this section, we present some numerical results obtained with DeepOnets and GreenOnets for the two-  
 257 dimensional homogeneous wave equation with  $c(x, y) = 1$ , where  $(x, y)$  denote the spatial coordinates. The  
 258 input functions  $s$  are defined as

$$s(x, y) = (1 - x^2)(1 - y^2)h(x, y),$$

259 where  $h$  is randomly sampled from a zero-mean Gaussian random field with a length scale  $l = 1$ . We choose  
 260  $m = 49$  sensor points uniformly distributed in  $\Omega = [-1, 1] \times [-1, 1]$ . We consider an initial learning rate

261  $10^{-3}$  that decreases with a rate 0.9995 at each epoch. The training set is defined with  $N = 50000$ , and  
 262  $P_r = P_{bc} = P_{ic} = 1$ . The weights in Equation (20) are set to  $w_r = 1$  and  $w_{ic} = w_{bc} = 100$ . The networks  
 263 are trained for 400 epochs with 100 mini-batches.

264 Similarly to the one-dimensional case, the loss functions have decreased faster for the GreenONets after  
 265 400 epochs, see Figure 9. We compare in Figure 10 the error in the solutions obtained with DeepONets  
 266 and GreenONets. In this figure, we show the initial condition  $u_0 = \cos(x\pi/2)\cos(y\pi/2)$ , with which we  
 267 test our networks, and the pointwise errors at  $t = 1.5$  using DeepONets and GreenONets. We observe that  
 268 the maximum pointwise error remains consistently smaller when using GreenONets. Moreover, we plot in  
 269 Figure 11 the approximated Green’s function  $G(x, y, t, x_i, y_i)$  for  $(x_i, y_i) = (0, 0)$  at  $t = 0$  and  $t = 0.4$ . The  
 270 Green’s function initially peaks at the origin and then radially propagates through the domain, as expected.

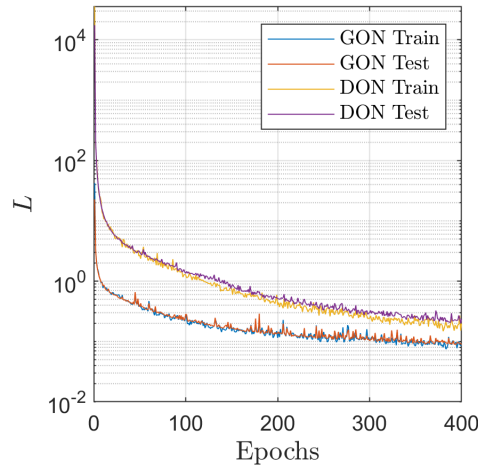


Figure 9: Example of Section 5.2: Evolution of the loss function on the training and testing sets.

## 271 6 Conclusions

272 In this work, we have introduced the Green operator networks, that approximate the operator of the wave  
 273 equation in homogeneous and heterogeneous domains for a family of initial conditions. The GreenONets  
 274 architecture is inspired by the exact representation of solution of the wave equation in terms of the Green’s  
 275 function in unbounded domains. This architecture yields better results, when approximating the wave oper-  
 276 ator for homogeneous and heterogeneous domains in one and two dimensions, when compared to DeepONets.  
 277 The increased performance is attributed to the fact that the GreenONets architecture is better suited to  
 278 this type of problems, but we also recognize that the DeepONets architecture is more general. We have  
 279 in particular showed that the loss functions associated with the GreenONets always converge with fewer  
 280 epochs. The numerical results also highlighted the fact that the pointwise errors are generally smaller with  
 281 GreenONets and that the solutions generalize better when tested on initial conditions with frequencies higher  
 282 than that of the training set. Finally, we have observed that the errors in the GreenONets solutions remained  
 283 localized around the peak amplitudes while the errors with DeepONets had the tendency to spread within  
 284 the domain. Plans for future works will focus on testing the GreenONets for the two- and three-dimensional  
 285 wave equation involving materials with various heterogeneous properties and to extend the methodology to  
 286 other model problems.

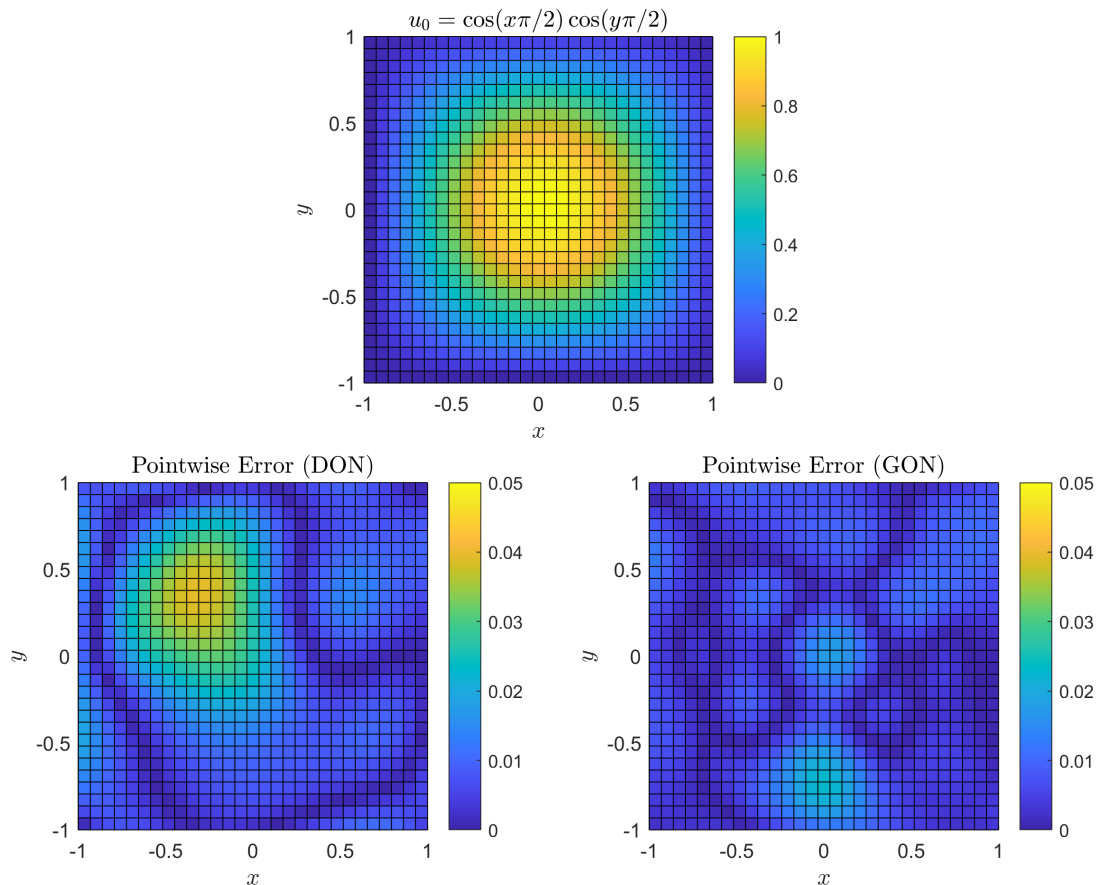


Figure 10: Example of Section 5.2: (left) Initial condition; (middle) Pointwise error at  $t = 1.5$  using DeepONets; (right) Pointwise error at  $t = 1.5$  using GreenONets.

287 **Acknowledgements.** SP and ML are grateful for the support from the Natural Sciences and Engineering  
 288 Research Council of Canada (NSERC) Discovery Grants [grant numbers RGPIN-2019-7154, PGPIN-2018-  
 289 06592]. This research was also partially supported by an NSERC Collaborative Research and Development  
 290 Grant [grant number RDCPJ 522310-17] with the Institut de Recherche en Électricité du Québec and  
 291 Prompt. ZA and SP are thankful to the Laboratoire de Mécanique et d’Acoustique UMR 7031, in Marseille,  
 292 France, for hosting them. This work received support from the French government under the France 2030  
 293 investment plan, as part of the Initiative d’Excellence d’Aix-Marseille Université - A\*MIDEX - AMX-19-IET-  
 294 010. Finally, the authors also thank J.-P. Ampuero, for having shared SEMLAB [2] with the community.

## 295 References

- 296 [1] Z. Aldirany, R. Cottereau, M. Laforest, and S. Prudhomme. Optimal error analysis of the spectral  
 297 element method for the 2D homogeneous wave equation. *Computers and Mathematics with Applications*,  
 298 119:241–256, 2022.
- 299 [2] J.-P. Ampuero. SEMLAB. MATLAB Central File Exchange, ([https://www.mathworks.com/  
 300 matlabcentral/fileexchange/6154-semmlab](https://www.mathworks.com/matlabcentral/fileexchange/6154-semmlab)), 2023. (Accessed 1 June 2023).

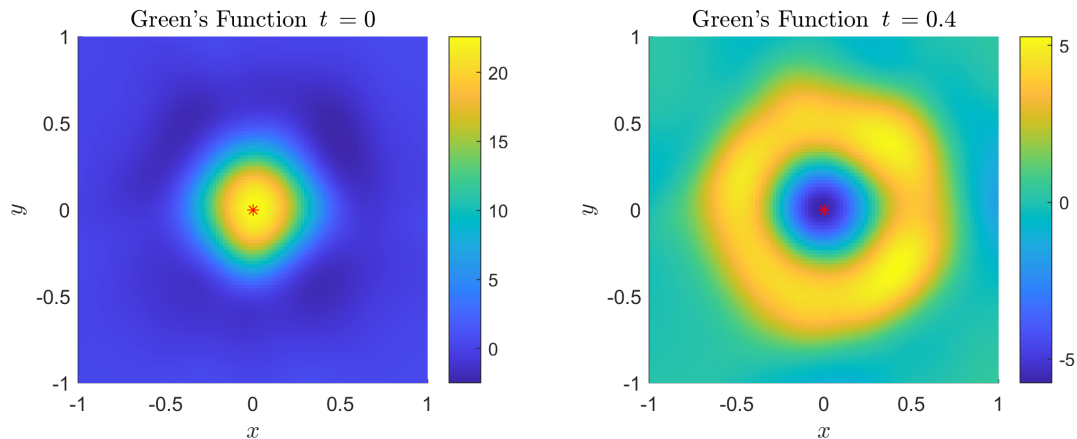


Figure 11: Example of Section 5.2: The approximated Green's function  $G(x, y, t, x_i, y_i)$  for  $(x_i, y_i) = (0, 0)$  at  $t = 0$  and  $t = 0.4$ . The red dot denotes the point  $(x_i, y_i) = (0, 0)$ .

- 301 [3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine  
302 learning: a survey. *Journal of Machine Learning Research*, 18:1–43, 2018.
- 303 [4] A. Bihlo and R. O. Popovych. Physics-informed neural networks for the shallow-water equations on the  
304 sphere. *Journal of Computational Physics*, 456:111024, 2022.
- 305 [5] N. Boullé, C. J. Earls, and A. Townsend. Data-driven discovery of Green's functions with human-  
306 understandable deep learning. *Scientific reports*, 12(1):4824, 2022.
- 307 [6] D. G. Duffy. *Green's functions with applications*. CRC press, Boca Raton, FL, 2015.
- 308 [7] C. R. Gin, D. E. Shea, S. L. Brunton, and J. N. Kutz. DeepGreen: Deep learning of Green's functions  
309 for nonlinear boundary value problems. *Scientific reports*, 11(1):21614, 2021.
- 310 [8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Ben-  
311 gio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- 312 [9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approxima-  
313 tors. *Neural networks*, 2(5):359–366, 1989.
- 314 [10] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed  
315 neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*,  
316 426:109951, 2021.
- 317 [11] E. Kausel. *Fundamental Solutions in Elastodynamics - A Compendium*. Cambridge University Press,  
318 2006.
- 319 [12] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving  
320 partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.
- 321 [13] D. P. Kingma and J. Ba. ADAM: A method for stochastic optimization. In *International Conference*  
322 *on Learning Representations*, 2015.

- 323 [14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural  
324 networks. *Communications of the ACM*, 60(6):84–90, 2017.
- 325 [15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- 326 [16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar.  
327 Fourier neural operator for parametric partial differential equations. arXiv:2010.08895, 2020.
- 328 [17] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar.  
329 Neural operator: Graph kernel network for partial differential equations. arXiv:2003.03485, 2020.
- 330 [18] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet  
331 based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229,  
332 2021.
- 333 [19] B. Moseley, A. Markham, and T. Nissen-Meyer. Solving the wave equation with physics-informed deep  
334 learning. arXiv preprint:2006.11894, 2020.
- 335 [20] C. L. Pettit and D. K. Wilson. A physics-informed neural network for sound propagation in the at-  
336 mospheric boundary layer. In *Proceedings of Meetings on Acoustics 179ASA*, volume 42, page 022002.  
337 Acoustical Society of America, 2020.
- 338 [21] A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195,  
339 1999.
- 340 [22] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning  
341 framework for solving forward and inverse problems involving nonlinear partial differential equations.  
342 *Journal of Computational Physics*, 378:686–707, 2019.
- 343 [23] C. E. Rasmussen, C. K. Williams, et al. *Gaussian processes for machine learning*, volume 1. Springer,  
344 2006.
- 345 [24] G. Seriani and E. Priolo. Spectral element method for acoustic wave simulation in heterogeneous media.  
346 *Finite Elem. Anal. Des.*, 16:337—348, 1994.
- 347 [25] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential  
348 equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
- 349 [26] C. Wang, S. Li, D. He, and L. Wang. Is  $L^2$  physics-informed loss always suitable for training physics-  
350 informed neural network? *arXiv preprint arXiv:2206.02016*, 2022.
- 351 [27] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential  
352 equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.
- 353 [28] E. Weinan and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving  
354 variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.
- 355 [29] Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional partial differential  
356 equations. *Journal of Computational Physics*, 411:109409, 2020.