Deep Generalized Green's Functions

Rixi Peng
Department of ECE
Duke University
Durham, NC 27708
rixi.peng@duke.edu

Juncheng Dong
Department of ECE
Duke University
Durham, NC 27708
juncheng.dong@duke.edu

Jordan Malof
Department of Computer Science
University of Montana
Missoula, MT 59812
jordan.malof@umontana.edu

Willie J. Padilla
Department of ECE
Duke University
Durham, NC 27708
willie.padilla@duke.edu

Vahid Tarokh
Department of ECE
Duke University
Durham, NC 27708
vahid.tarokh@duke.edu

Abstract

In this study, we address the challenge of obtaining a Green's function operator for linear partial differential equations (PDEs). The Green's function is well-sought after due to its ability to directly map inputs to solutions, bypassing the need for common numerical methods such as finite difference and finite elements methods. However, obtaining an explicit form of the Green's function kernel for most PDEs has been a challenge due to the Dirac delta function singularity present. To address this issue, we propose the Deep Generalized Green's Function (DGGF) as an alternative, which can be solved for in an efficient and accurate manner using neural network models. The DGGF provides a more efficient and precise approach to solving linear PDEs while inheriting the reusability of the Green's function, and possessing additional desirable properties such as mesh-free operation and a small memory footprint. The DGGF is compared against a variety of state-of-the-art (SOTA) PDE solvers, including direct methods, namely physics-informed neural networks (PINNs), Green's function approaches such as networks for Gaussian approximation of the Dirac delta functions (GADD), and numerical Green's functions (NGFs). The performance of all methods is compared on four representative PDE categories, each with different combinations of dimensionality and domain shape. The results confirm the advantages of DGGFs, and benefits of Generalized Greens Functions as an novel alternative approach to solve PDEs without suffering from singularities.

1 Introduction

Efficiently solving partial differential equations (PDEs) poses a significant challenge in the realm of scientific computing. These equations are ubiquitous, appearing in a multitude of disciplines ranging from physics and engineering to social science. However, the majority of PDEs cannot be explicitly solved, necessitating the deployment of numerical methods such as the finite difference method (FDM) or the finite element method (FEM). These methods operate by discretizing space and / or time, thus generating solutions on pre-established grids. The accuracy of these numerical solutions is contingent upon the preciseness of the discretization. Enhancing the grid density to achieve superior accuracy often implies a substantial escalation in computational resources, including memory, CPU time, and effort.
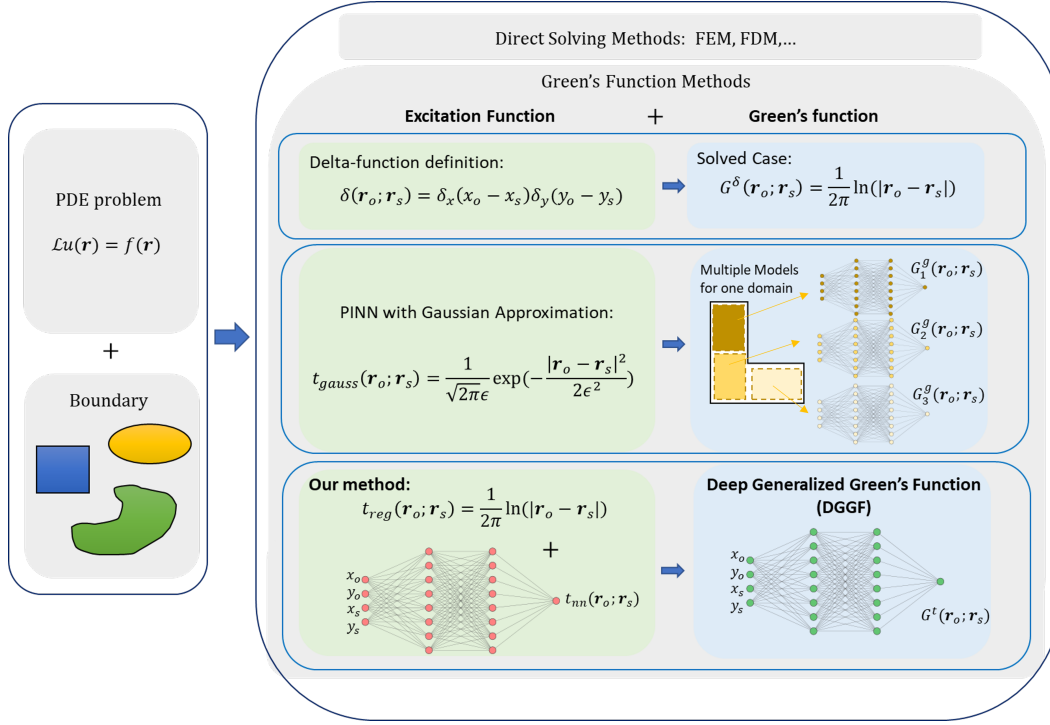
Preprint. Under review.

Figure 1: Schematic showing two types of PDE solvers: Direct Methods solve each PDE problem individually, and the Green's function method retrieves a kernel function first and constructs the solution in convolutional forms.

In order to mitigate these constraints, the development and implementation of alternative, mesh-free methods for PDEs is highly desirable. Recently, deep neural networks (DNNs) have emerged as a promising mesh-free solution capable of addressing a wide array of PDEs across various fields such as science, engineering, and mathematics. The most notable DNN-based PDE solving method is the physics-informed neural network (PINN) [1]. In this approach, a DNN is used to represent a solution function which satisfies the PDE at every test point within the domain. Despite their potential benefits, a significant drawback of these DNN-based strategies lies in their training cost, which remains invariable for each unique PDE problem since a DNN model must be trained for each new problem . Furthemore, training a DNN to accurately solve a specific PDE problem can be a laborious process, dependent on the unique attributes of each problem as dictated by the differential operator, domain, boundary condition, and input function.

To address the aforementioned limitations of DNN-based PDE solutions, several strategies have been proposed to reduce their computational costs. Towards this goal, notable examples include the DeepRitz [2], which tackles the variational form of PDEs, and the Fourier Neural Operator (FNO) [3], which leverages the Fourier transform methods widely used in many PDE problems. Another promising yet under-explored technique is the combination of Green's function methods with neural networks. Typically the Green's function is limited to linear PDEs and attempts to solve for a general integral kernel which is then used to construct a solution by convolving with the system input (or stimulus). The Green's function possesses several useful characteristics:

- Property (i): Regardless of the input function or boundary conditions, it transforms the process of solving the PDE into a convolution computation across the domain utilizing the Green's function as the kernel.
- Property (ii): The Green's function naturally arises in boundary value and initial value PDE problems, and simplifies the problem by reducing the dimensionality.

Therefore, once a Green's function is constructed for a particular differential operator it is *re-usable*, so that solutions for that operator can be constructed efficiently even if the input function and boundary conditions vary.

Despite their remarkable advantages, explicit Green's functions are only available for a limited number of scenarios, such as the Helmholtz equation in an unbounded homogeneous domain. In most cases, either the Green's function cannot be derived analytically or its resulting form becomes excessively intricate, thereby diminishing its utility; this has motivated the use of DNNs to approximate Green's functions. Training DNNs for this purpose is challenging however, because it requires the derivatives of the network output to approximate a singularity function [4], namely a Dirac delta function. [5].

Here we propose and demonstrate a generalized *Green's function* method and the neural network representation, termed as *Deep Generalized Green's functions* (DGGF), which addresses the singularity issue while inheriting all of the benefits of both using mesh-free neural network method and using the Green's function method to solve the linear PDE problems including Property (i) and (ii) above. Using a number of benchmark PDE problems, we demonstrate that the DGGF method provides stable solutions, exhibits a fast convergence rate, and achieves high accuracy.

Our contributions are summarized below:

- We construct Deep Generalized Green's Functions (DGGFs) that are applicable to all linear PDEs, and amenable to stable and efficient numerical computation. We prove that DGGFs maintain desirable Properties (i) and (ii) of the original Green's function,

- We demonstrate achieved accuracy in constructing PDE solutions using DGGFs for four different types of PDEs in various dimensions and for different boundary shapes and conditions, and

- We experimentally demonstrate that our proposal affords faster neural network training with superior performance compared to state of the art DNN based Green's function approaches.

2 Related Work

We formally define a general *PDE problem* with a partial differential operator $\mathcal{L} : \mathcal{U} \to \mathcal{U}^*$ of order $p > 0$ from the solution function space $\mathcal{U}$ to its dual input function space $\mathcal{U}^*$ with support in a compact set $D \subset \mathbb{R}^n$. Let $\mathcal{U}$ be the Sobolev space of the same order of p. The boundary condition $\mathcal{B}$ specifies the solution function constraints on the domain boundary ∂D. Combining these two factors leads to,

$$
\begin{aligned}
\mathcal{L}u(\boldsymbol{r}) &= f(\boldsymbol{r}), \quad \forall \boldsymbol{r} \in D, \\
\mathcal{B}_{\boldsymbol{r}}u(\boldsymbol{r})|_{\boldsymbol{r} \in \partial D} &= h(\boldsymbol{r}),
\end{aligned}
\tag{1}
$$

where the input function $f \in \mathcal{U}^* : D \to \mathbb{R}$ is usually given and the solution function $u \in \mathcal{U} : D \to \mathbb{R}$ is to be solved.

Many established PDE solvers can be chosen to solve the generic problem defined in Eq. 1 based on the characteristics of the specific problem. Traditional methods include Fourier and Laplace Transform Methods, FEM and FDM etc. The idea of using a neural network to efficiently solve PDEs, to the best of our knowledge, dates back to the 1990s [6, 7, 8]. However, the lack of computational power and auto-differentiation capabilities severely limited the efficacy of neural networks in this domain. More recently, there has been a resurgence of the application of neural networks in solving PDEs due to works on physics-informed neural networks (PINNs) [9] in various scientific fields such as fluid dynamics [10, 11], thermodynamics [12], and electromagnetism [13]. A detailed discussion of PINNs and their applications can be found in [14]. Several recent studies have proposed PINN improvements including reweighting loss terms in the objective function [15], new loss terms [16], and more complicated neural network structures [9]. Different DNN-based PDE solving methods are listed and compared in the Table 1. Another notable method for solving PDEs is DeepRitz [2], which solves the variational form of the original strict differential form, resulting in relaxed constraints and often more robust convergence to the solution function. However, the limitation of applying this approach to calculating the Green's function is in their time-consuming process of evaluating a multidimensional integral at every epoch of the training process.

To the best of our knowledge, there have been a limited number of studies for approximating the Green's function with DNNs. A notable approach, referred to as GaussNet and described in [5], utilizes Gaussians to approximate the Dirac delta function. By employing smooth Gaussians, this method significantly relaxes the singularity of the Dirac delta function. The use of Gaussian functions offers a systematic approach to achieve high accuracy by reducing the width of the Gaussian. However,

this method represents one Green's function on a single domain with multiple networks, making it challenging to apply to larger domains. Training may also be complex depending on the desired accuracy level, and approximation errors may be of concern.

The BI-GreenNet method is another important technique for expressing the non-singular part of Green's functions on different domains using DNNs [17]. This method takes advantage of the linearity of the Green's function and performs well when the singular part of the Green's function can be computed analytically. However, a limitation of this method is the need for an analytic solution for the singular part.

Various related data-driven approaches [18, 19, 20] learn the Green's function of an unknown system by incorporating it into a convolution integral and attempting to construct the solution function. These methods address a different type of problem, namely inferring the properties of an unknown system, instead of directly solving the underlying PDE. By utilizing the Green's function in a convolutional form, these approaches eliminate the need for evaluating the Dirac delta function. However, they may require fine sampling of the input function making them potentially challenging to implement in various scenarios of interest.

Table 1: Properties of different PDE methods

METHOD	FIXED GRID	RE-USABILITY	DATA REQUIREMENT
PINN	No	No	No
DGGF	No	Yes	No
GAUSSNET	No	Yes	No
BI-GREENNET	No	Yes	No

3 Deep generalized Green's function

3.1 Preliminaries

The Green's function corresponding to a *linear* PDE problem described by Equation 1, if it exists, is defined as the solution for a Dirac delta input function. To be precise, *the Green's function problem* is formulated as, denoted by $G : D \times D \to \mathbb{R}$:

$$\mathcal{L}_{\boldsymbol{r}} G(\boldsymbol{r}, \boldsymbol{\xi}) = \delta(\boldsymbol{r} - \boldsymbol{\xi}), \, \forall \, \boldsymbol{r}, \boldsymbol{\xi} \in D \tag{2}$$

$$G(\boldsymbol{r}, \boldsymbol{\xi}) = 0, \, \forall \boldsymbol{\xi} \in D^{\circ} \, \forall \, \boldsymbol{r} \in \partial D \tag{3}$$

where the subscript on $\mathcal{L}_{\boldsymbol{r}}$ denotes that the differential operator only operates on $\boldsymbol{r}$ and D° denotes the interior of domain D. It should be noted that the Green's function must satisfy the same *type* of boundary condition (Dirichlet, Neumann, or Robin) as that of the original PDE problem and should be in general zero on the boundary. The Green's function method then gives the solution to the problem as,

$$u(\boldsymbol{r}) = \int_{D} G(\boldsymbol{r}, \boldsymbol{\xi}) f(\boldsymbol{\xi}) d\boldsymbol{\xi} + \int_{\partial D} h(\boldsymbol{r}, \boldsymbol{\xi}') \frac{\partial_{\boldsymbol{r}} G}{\partial \mathbf{n}} d\boldsymbol{\xi}', \tag{4}$$

where $\mathbf{n}$ is the outward normal of the boundary [21].

3.2 Problem setting

The goal of the Green's function is to find the operator which maps the input function f to the solution function u. The formal description of solving the Green's function is outlined below. Let $\mathcal{U}$ be the Sobolev space of the order p defined on a bounded domain $D \subset \mathbb{R}^n$. Let $\mathcal{L} : \mathcal{U} \to \mathcal{U}^*$ be the linear partial differential operator such that $\mathcal{L}u = f, u \in \mathcal{U}, f \in \mathcal{U}^*$. The corresponding Green's function operator $\mathcal{G} : \mathcal{U}^* \to \mathcal{U}$ can be formally written as,

$$u = (\mathcal{G}f)(\boldsymbol{r}) = \int_{D} G(\boldsymbol{r}, \boldsymbol{\xi}) f(\boldsymbol{\xi}) d\boldsymbol{\xi}, \tag{5}$$

where the kernel $G : D \times D \to \mathbb{R}$, is called the Green's function.

The singular nature of the Dirac delta function poses challenges. Instead of solving the Green's function kernel with the formal definition, in this work, we assume f is twice differentiable and subsequently we propose a generalized Green's function operator,

$$\mathcal{G}^t : h \rightarrow u, \text{ where } h = \Delta f \tag{6}$$

where the Laplacian operator Δ is given by $\Delta := \sum_i \frac{\partial^2}{\partial x_i^2}$. With this generalized operator, an ordinary functions can replace the Dirac delta function in the optimization problem, formulated as

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{\boldsymbol{r}, \boldsymbol{\xi} \in D} |\mathcal{L} G_\theta^\circ(\boldsymbol{r}, \boldsymbol{\xi}) - t(\boldsymbol{r}, \boldsymbol{\xi})|^2, \tag{7}$$

where $t(\boldsymbol{r}, \boldsymbol{\xi})$ is used in place of the Dirac delta function. We then derive an explicit form of the function $t(\boldsymbol{r}, \boldsymbol{\xi})$ and show that the resulting operator maps to the solution function of the PDE. The input function $t(\boldsymbol{r}, \boldsymbol{\xi})$ can be divided into the singular term $t_s(\boldsymbol{r}, \boldsymbol{\xi})$ and the regular term $t_r(\boldsymbol{r}, \boldsymbol{\xi})$. The singular term depends on the dimension n of the problem while the regular term depends on the domain boundary condition.

Lemma 3.1 *[Decomposition of the alternative input function] Assume that both the differential operator $\mathcal{L}$ is linear and the boundary condition is $t = 0$. Then,*

$$t = t_s + t_r \tag{8}$$

where t_s and t_r respectively satisfy:

$$\Delta t_s(\boldsymbol{r}, \boldsymbol{\xi}) = \delta(\boldsymbol{r} - \boldsymbol{\xi}), \quad \lim_{\boldsymbol{r} \to \infty} t_s(\boldsymbol{r}, \boldsymbol{\xi}) = 0, \tag{9}$$

$$\Delta t_r(\boldsymbol{r}, \boldsymbol{\xi}) = 0, \quad t_r(\boldsymbol{r}, \boldsymbol{\xi})|_{\partial D} = -t_s(\boldsymbol{r}, \boldsymbol{\xi})|_{\partial D}. \tag{10}$$

Equation 3.1 itself forms a Green's function problem of the Poisson equation subject to the boundary condition. Next, we replace the Dirac delta function in the definition of Green's function in Eq. (2) with $t(\boldsymbol{r}, \boldsymbol{\xi})$:

$$\mathcal{L}_{\boldsymbol{r}} G^t(\boldsymbol{r}, \boldsymbol{\xi}) = t(\boldsymbol{r}, \boldsymbol{\xi}). \tag{11}$$

We term the corresponding solution G^t as the *generalized Green's function* subject to the input t.

In addition to Eq. 11, we also impose the following two requirements on G^t on the boundary ∂D:

$$\Delta_{\boldsymbol{r}} G^t(\boldsymbol{r}, \boldsymbol{\xi})|_{\partial D} = 0, \tag{12}$$

$$\mathcal{B} G^t(\boldsymbol{r}, \boldsymbol{\xi}) = 0. \tag{13}$$

Definition 3.2 (Generalized Green's Function) *The generalized Green's Function $G^t : D \times D \rightarrow \mathbb{R}$ on a compact domain is defined to be the solution to the following two connected PDE problems:*

$$\Delta t(\boldsymbol{r}, \boldsymbol{\xi}) = \delta(\boldsymbol{r} - \boldsymbol{\xi}), \tag{14}$$

$$\mathcal{L}_{\boldsymbol{r}} G^t(\boldsymbol{r}, \boldsymbol{\xi}) = t(\boldsymbol{r}, \boldsymbol{\xi}), \tag{15}$$

$$\Delta_{\boldsymbol{r}} G^t(\boldsymbol{r}, \boldsymbol{\xi})|_{\partial D} = 0, \tag{16}$$

$$\mathcal{B} G^t(\boldsymbol{r}, \boldsymbol{\xi}) = 0. \tag{17}$$

We note that the third condition can be used to derive the boundary condition for the alternative input function t. For instance, if the original problem is the Poisson equation, i.e., $\mathcal{L} = \Delta$, the first condition on G^t directly leads to $t(\boldsymbol{r}, \boldsymbol{\xi}) = 0$ on the boundary. For other types of operators, the boundary condition can be derived similarly.

A generic analytic solution to the singular part $t_s(\boldsymbol{r}, \boldsymbol{\xi})$ for various domain dimensions is known with vanishing boundary condition, i.e., $\lim_{\boldsymbol{r} \to \infty} t_s(\boldsymbol{r}, \boldsymbol{\xi}) = 0$. For 2-dimensional domain, $t_s(\boldsymbol{r}, \boldsymbol{\xi}) = ln(|\boldsymbol{r} - \boldsymbol{\xi}|)$ and for 3-dimensional domain, $t_s(\boldsymbol{r}, \boldsymbol{\xi}) = 1/|\boldsymbol{r} - \boldsymbol{\xi}|$. Since $t_s(\boldsymbol{r}, \boldsymbol{\xi})$ is expressed in the analytic form, the boundary value for the regular part $t_r(\boldsymbol{r}, \boldsymbol{\xi})$ can be easily calculated on arbitrarily shaped finite domain by $t_r(\boldsymbol{r}, \boldsymbol{\xi}) = -t_s(\boldsymbol{r}, \boldsymbol{\xi})$. Then the problem reduces to solving for $t_r(\boldsymbol{r}, \boldsymbol{\xi})$ which is a standard boundary value problem in PDE that can be easily solved. Once the format of $t(\boldsymbol{r}, \boldsymbol{\xi})$ is fully determined via the procedures above, the Green's function problem is transformed to the generalized one with $t(\boldsymbol{r}, \boldsymbol{\xi})$ as the alternative input function, which is expressed exactly as normal functions.

3.3 Model

In this section, we propose a unified paradigm for solving PDEs with neural networks and the generalized Green's function. Our method involves three steps which generate one auxiliary neural network and one primary neural network for the generalized Green's function. Following the concept of physics-informed neural network, we represent the solution function with parameterized neural networks while using the partial differential equations and boundary conditions to define the loss function which is minimized during training. The three steps are

- Step 1: Solve for the alternative input function t.
- Step 2: Solve the generalized Green's function G^t.
- Step 3: Construct the solution using the generalized Green's function neural network.

It should be noted that step 1 and 2 are only executed once for a fixed PDE operator and domain D but the trained neural network can be reused in step 3. We elaborate on each step in the following sections.

3.3.1 Solving the alternative input function t.

As stated in the Lemma 3.1, the regular part $t_r(\boldsymbol{r}, \boldsymbol{\xi})$ is the solution to the boundary value problem defined with $\mathcal{L}_{\boldsymbol{r}} t_r(\boldsymbol{r}, \boldsymbol{\xi}) = 0, t_r(\boldsymbol{r}, \boldsymbol{\xi}) = g(\boldsymbol{r}, \boldsymbol{\xi})$ where the boundary values are determined by the analytic function of $t_s(\boldsymbol{r}, \boldsymbol{\xi})$. Suppose a parameterized neural network $\hat{t}_{r,\theta}(\boldsymbol{r}, \boldsymbol{\xi})$ is used to approximate $t_r(\boldsymbol{r}, \boldsymbol{\xi})$. For each training iteration, a batch of random interior point pairs of $(\boldsymbol{r}_i \in D^\circ, \boldsymbol{\xi}_i \in D^\circ), i = 1, 2, ..., N'_{dm}$ and a batch of random boundary point pairs $(\boldsymbol{r}_j \in \partial D, \boldsymbol{\xi}_j \in D^\circ), j = 1, 2, ..., N'_{bd}$ are independently sampled using the Latin hypercube sampling method. These sampled points across the domain or on the boundary are used to evaluate the residual loss or boundary loss, respectively. The total loss function at each iteration is defined as,

$$L_\theta = \frac{\lambda_{res}}{N'_{dm}} \sum_i^{N'_{dm}} L_{res} + \sum_j^{N'_{bd}} \lambda_{bd} L_{bd} \tag{18}$$

$$= \frac{\lambda_{res}}{N'_{dm}} \sum_i^{N_{dm}} |\mathcal{L}\hat{t}_\theta(\boldsymbol{r}_i, \boldsymbol{\xi}_i)|^2 + \frac{\lambda_{bd}}{N'_{bd}} \sum_j^{N_{bd}} |\hat{t}_{r,\theta}(\boldsymbol{r}_j, \boldsymbol{\xi}_j) - g(\boldsymbol{r}_j, \boldsymbol{\xi}_j)|^2 \tag{19}$$

$\lambda_{res}, \lambda_{db}$ are the weights to balance the magnitude between the PDE residual loss and the boundary loss. The partial derivatives in $\mathcal{L}$ are all computed using the AutoGrad package in PyTorch. The stochastic gradient descent (SGD) method is used to minimize the loss and results in an accurate neural network representation of $\hat{t}_{r,\theta}(\boldsymbol{r}, \boldsymbol{\xi})$.

3.3.2 Solving for the generalized Green's function G^t.

With the help of the auxiliary neural network model $\hat{t}_{r,\theta}(\boldsymbol{r}, \boldsymbol{\xi})$, the alternative excitation function $t(\boldsymbol{r}, \boldsymbol{\xi})$ can be readily evaluated at every point across the domain, i.e. $t(\boldsymbol{r}, \boldsymbol{\xi}) = t_s(\boldsymbol{r}, \boldsymbol{\xi}) + \hat{t}_{r,\theta}(\boldsymbol{r}, \boldsymbol{\xi})$. Now we use the same framework as above to train the generalized Green's function $\hat{G}^t_\phi$. For each training iteration, a batch of random interior point pairs of $\boldsymbol{r}_i \in D^\circ, \boldsymbol{\xi}_i \in D, i = 1, 2, ..., N_{dm}$ and a batch of random boundary point pairs $(\boldsymbol{r}_j \in \partial D, \boldsymbol{\xi}_j \in D^\circ), j = 1, 2, ..., N_{bd}$ are independently sampled using again the Latin hypercube sampling method. For this problem, the residual loss takes the alternative excitation function t and sets the boundary values are zero, which leads to the definition of the total loss as,

$$L_\phi = \frac{\lambda_{res}}{N_{dm}} \sum_i^{N_{dm}} L_{res} + \frac{\lambda_{res}}{N_{bd}} \sum_j^{N_{bd}} \lambda_{bd} L_{bd} \tag{20}$$

$$= \frac{\lambda_{res}}{N_{dm}} \sum_i^{N_{dm}} |\mathcal{L}\hat{G}^t_\phi(\boldsymbol{r}_i, \boldsymbol{\xi}_i) - (t_s + \hat{t}_{r,\theta})(\boldsymbol{r}_i, \boldsymbol{\xi}_i)|^2 + \frac{\lambda_{res}}{N_{bd}} \sum_j^{N_{bd}} |\hat{G}^t_\phi(\boldsymbol{r}_j, \boldsymbol{\xi}_j)|^2. \tag{21}$$
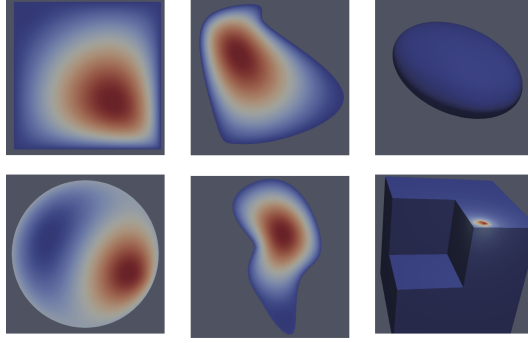
Figure 2: Different types of domain boundaries explored in the experiments. The color map indicates the generalized Green's function of a given fixed source location for the Poisson equation.

Once the generalized Green's function network finishes training, it can be used in Eq. (15) to determine the solution function for any excitation function f with,

$$u(\boldsymbol{r}_0) = \sum_j w_j \hat{G}^t_\phi(\boldsymbol{r}_0, \boldsymbol{\xi}_j) \Delta f(\boldsymbol{\xi}_j). \tag{22}$$

where $\boldsymbol{\xi}_{i,j} = 1, 2, ..., n$ are the quadrature points for fast evaluation of the multi-dimensional integration, and w_j are corresponding quadrature weights. It should be noted that the solution function values at multiple $\boldsymbol{r}_0$ can be computed in parallel.

4 Numerical experiments

We demonstrate the performance of DGGF in solving PDE problems compared to several recent state-of-the-art baseline methods for different types of problems. We include different domains, including a square (SQ), a circle (CR), and two B-spline curve enclosed loops (B1 & B2), and two 3D boundaries including a cube with 1/8 corner cut (CC), and an ellipsoid (EP), along with four different PDE types, listed in Table 2. The exact domain shapes are illustrated in Fig. 4.

Specifically, we demonstrate that DGGF has higher accuracy compared to GaussNet and constructs faster results compared to PINN on several classes of widely-studied PDEs. We present results of different dimensions and boundary shapes in our experiments. For each type of the PDE problems, we are interested in the performance of DGGF compared to the following baseline models:

- Method (I): Gaussian Approximation of the Dirac delta function (GaussNet),

- Method (II): Physics-informed neural network (PINN),

- Method (III): Numeric Green's Function (NGF),

Table 2: PDE problems explored in the experiments. Boundary shapes include square (SQ), circular (CR), two Jordan curves B-spline 1 (B1) and B-spline 2 (B2), corner cut cube (CC), and an ellipsoid (EP).

CLASS	EXPRESSION	BOUNDARY SHAPE
POISSON	$\sum_{x,y,z} \partial^2_{x,y,z}$	SQ, CR, B1, B2, CC, EP
HELMHOLTZ	$\sum_{x,y,z} \partial^2_{x,y,z} + k^2$	SQ, CR, B1, B2, CC, EP
HEAT	$\sum_{x,y,z} \partial^2_{x,y,z} - \partial_t$	SQ, CR, B1, B2, CC, EP
KLEIN-GORDON	$\sum_{x,y,z} \partial^2_{x,y,z} - \partial^2_t - k^2$	SQ, CR, B1, B2, CC, EP

We use the FEM solver FEniCSx [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32] to obtain the solution close to the ground truth for all the experiments using very fine meshes.

4.1 Model training

For our method, the PINN, and the GaussNet, we use the exact same DNN structure and training procedures throughout for all experiments for fair comparison. The DNN comprises 8 hidden layers with Rectified Linear Units (ReLU) activation functions, and 100 neurons in each layer. Note that the GaussNet method prescribes using multiple networks, instead of a single network, to represent the fast-varying Green's function on the whole domain. We tested this multi-network approach on simple 2D domain problems with 16 networks and results are included in the Appendix. The complexity of arbitrary segmentation of irregular domains hinders the implementation of this method on all the cases explored in this study. The hyperparameters involved in the training of the models are N_{dm}, N_{bd}, N'_{dm}, N'_{bd}, λ_{res}, λ_{bd}. Please see Sec. 3.3 for interpretation of these hyperparameters. For simplicity, we always set $\lambda_{res} = 1$ and search $\lambda_{bd} \in [1, 5, 50]$, $N_{dm} \in [50000, 100000, 150000, 200000]$, $N_{bd} \in [25000, 50000, 100000, 75000, 150000, 200000]$. The lists of values for N'_{dm} and N'_{bd} are the same as that of N_{dm} and N_{bd}. We use the learning rate of 1×10^{-3} with ADAM [33] and zero regularization. The GaussNet method requires one additional hyperparameter, which is the width of the Gaussian used to approximate the Dirac delta function. For this, we tested two values $\epsilon \in \{.05, 0.1\}$, for every problem considered below.

4.2 Results

4.2.1 Accuracy

A comparison of the accuracy between DGGF and baseline methods are present in Figure 4. It can be observed that the DGGF outperforms both the GaussNet for both choices of the Gaussian width explored with the single network representation. In particular, the DGGF realizes at least three orders of magnitude smaller errors than that of the GuassNet, and is comparable to the PINN method. These results indicate that by transforming the Green's function to the generalized Green's function, the neural network model can learn a better kernel to construct the solution functions. Despite comparable accuracy, our method is 20,000 times faster than the PINN in the solution function construction step, with the parallel convolution in our method taking 0.08 s for 2D domains, compared to a 30 min training time for the PINN method on any single 2D problem.
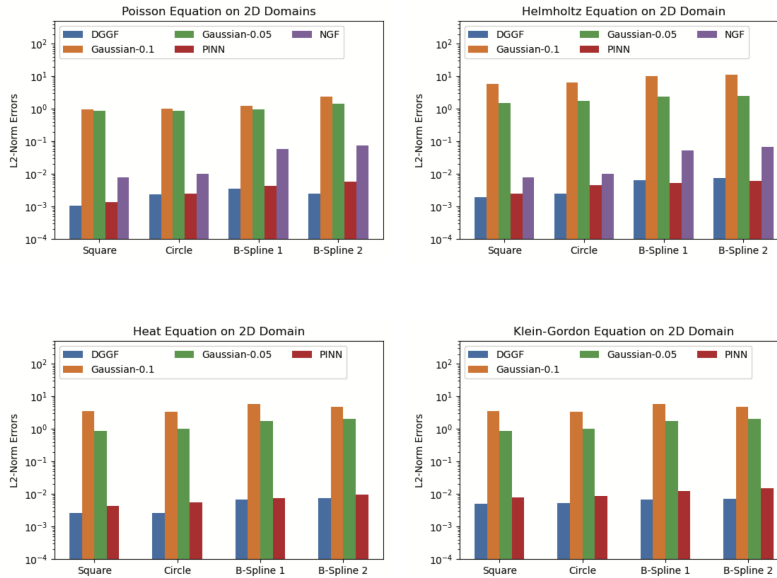

Figure 3: Results of 2D experiments with different PDEs and boundaries showing the mean L2 norm error, including the DFFG (our approach shown as the blue colors), Gaussian (0.05 green colors) and (0.1 shown as the orange colors), PINN (purple), and NGF (red).
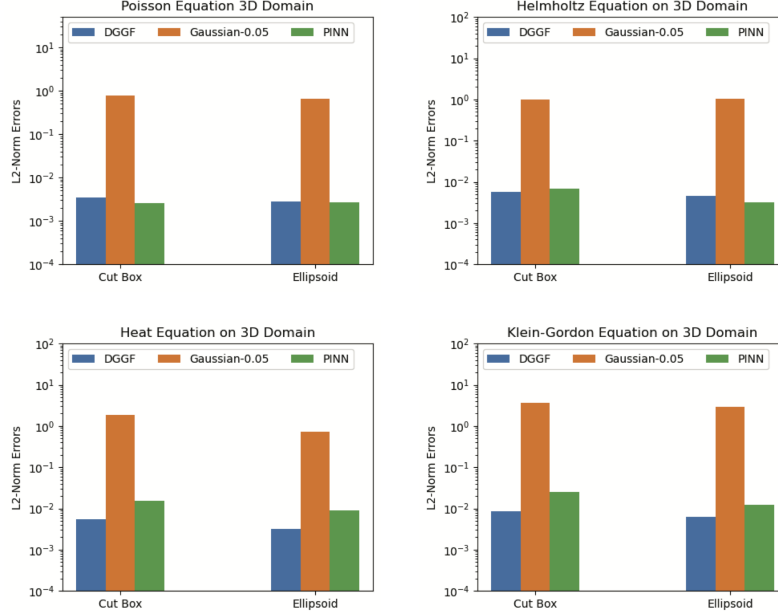
Figure 4: Results of 3D experiments with different PDEs and boundaries showing the mean L2 norm error, including the DFFG (our approach shown as the blue colors), Gaussian (0.05 green colors) and (0.1 shown as the orange colors), and PINN (red).
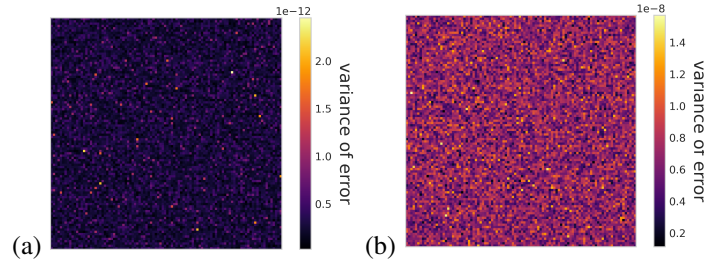

Figure 5: Stability of DGGF. Here we perform only Step 2 (G^t) in (a) and Step 1 + Step 2 (b). The variance of accuracies across points in the domain (due to random initialization of neural networks) is used to measure the stability of DGGF. Shown in (a) and (b) are variances of accuracies for each point in the 2-d Box domain.

4.3 Ablation Study

4.3.1 Stability

We use 2-d Poisson problem with Box boundary to study the stability of DGGF, i.e., the variance of error caused by different random initializations of the neural networks. We study two scenarios: (1) only training the networks in step 2 of DGGF and (2) training networks in both step 1 and step 2. Specifically, in both scenarios, we use network of 8 layers, all with 100 neurons. We set $N_{dm} = 100000$, $N_{bd} = 300000$ and $\lambda_{bd} = 5$. Then we train 30 neural networks with different initializations. The variances of their accuracies for estimating Green's Function at various points in domain are then computed. As can be seen in Figure 5, the variance of accuracy across the domain is uniformly small. In particular, the variance of only training the network for G^t is to the order of 10^{-11}, demonstrating the stability of DGGF.

5 Conclusions

To harness the full potential of Green's function reusability, we introduce the concept of deep generalized Green's function. By circumventing the singularity associated with the Dirac delta function while preserving theoretical accuracy, our method offers a practical solution. Empirical tests conducted on a comprehensive selection of partial differential equations (PDEs) have confirmed the effectiveness of our approach. Notably, DGGF exhibits lower computational resource requirements, faster convergence, and, crucially, enables efficient reuse for solving PDEs of the same type.

References

[1] 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.

[2] Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[3] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[4] Armen H Zemanian. *Distribution theory and transform analysis: an introduction to generalized functions, with applications*. Courier Corporation, 1987.

[5] Yuankai Teng, Xiaoping Zhang, Zhu Wang, and Lili Ju. Learning green's functions of linear reaction-diffusion equations with application to fast numerical solver. In *Mathematical and Scientific Machine Learning*, pages 1–16. PMLR, 2022.

[6] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[7] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[8] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91:110–131, 11 1990.

[9] Pu Ren, Chengping Rao, Yang Liu, Jian-Xun Wang, and Hao Sun. PhyCRNet: Physics-informed convolutional-recurrent network for solving spatiotemporal PDEs. *Computer Methods in Applied Mechanics and Engineering*, 389:114399, feb 2022.

[10] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, page 1457–1466, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, feb 2021.

[12] Navid Zobeiry and Keith D. Humfeld. A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications. *Engineering Applications of Artificial Intelligence*, 101:104232, 2021.

[13] Marco Baldan, Giacomo Baldan, and Bernard Nacke. Solving 1d non-linear magneto quasi-static maxwell's equations using neural networks. *IET Science, Measurement and Technology*, 15, 03 2021.

[14] Salvatore Cuomo, Vincenzo Schiano di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next, 2022.

[15] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[16] Pao-Hsiung Chiu, Jian Cheng Wong, Chinchun Ooi, My Ha Dao, and Yew-Soon Ong. CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method. *Computer Methods in Applied Mechanics and Engineering*, 395:114909, may 2022.

[17] Guochang Lin, Fukai Chen, Pipi Hu, Xiang Chen, Junqing Chen, Jun Wang, and Zuoqiang Shi. Bi-greennet: learning green's functions by boundary integral network. *Communications in Mathematics and Statistics*, 11(1):103–129, 2023.

[18] Craig R Gin, Daniel E Shea, Steven L Brunton, and J Nathan Kutz. Deepgreen: Deep learning of green's functions for nonlinear boundary value problems. *Scientific reports*, 11(1):1–14, 2021.

[19] Nicolas Boullé, Christopher J Earls, and Alex Townsend. Data-driven discovery of green's functions with human-understandable deep learning. *Scientific reports*, 12(1):1–9, 2022.

[20] Nicolas Boullé, Christopher J Earls, and Alex Townsend. Data-driven discovery of green's functions with human-understandable deep learning. *Scientific reports*, 12(1):4824, 2022.

[21] Erwin Kreyszig, K Stroud, and G Stephenson. Advanced engineering mathematics. *Integration*, 9(4), 2008.

[22] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3, 2015.

[23] M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells. Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes. *ACM Transactions on Mathematical Software*, 2022. To appear.

[24] M. W. Scroggs, I. A. Baratta, C. N. Richardson, and G. N. Wells. Basix: a runtime finite element basis evaluation library. *Journal of Open Source Software*, 7(73):3982, 2022.

[25] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[26] A. Logg and G. N. Wells. DOLFIN: automated finite element computing. *ACM Transactions on Mathematical Software*, 37, 2010.

[27] A. Logg, G. N. Wells, and J. Hake. DOLFIN: a C++/Python finite element library. In A. Logg, K.-A. Mardal, and G. N. Wells, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, chapter 10. Springer, 2012.

[28] R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32, 2006.

[29] A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. FFC: the FEniCS form compiler. In A. Logg, K.-A. Mardal, and G. N. Wells, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, chapter 11. Springer, 2012.

[30] K. B. Ølgaard and G. N. Wells. Optimisations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software*, 37, 2010.

[31] M. S. Alnaes, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, 40, 2014.

[32] R. C. Kirby. FIAT: numerical construction of finite element basis functions. In A. Logg, K.-A. Mardal, and G. N. Wells, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, chapter 13. Springer, 2012.

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

A Gaussian Approximation with Segmented Domains

In the main text, one neural network $G_\theta(\boldsymbol{r}, \boldsymbol{\xi})$ is used to represent the Green's function of some PDE problem for all $\boldsymbol{\xi}$ on the interior domain. In this section, we aim to directly compare the GaussNet scheme proposed in [5] which uses multiple neural network models, to our method which uses a single model. In more detail, for the GaussNet method the entire interior domain is segmented into several subdomains where each is represented by a neural network $G_{\theta,i}(\boldsymbol{r}, \boldsymbol{\xi}), \forall \boldsymbol{r} \in D, \forall \boldsymbol{\xi} \in D_i, \bigcup_i D_i = D^\circ$. It should be noted that using multiple networks should decrease the representation error especially on edge areas of the domain and potentially decrease the training difficulty. However, it cannot solve the approximation error inherent in the Gaussian approximation itself, which is solely determined by the Gaussian width. Furthermore, using multiple networks for one single domain will drastically increase the computational burden for training, as well as the storage requirement for each domain. Here, we used a Gaussian width of $s = 0.02$ and $6x6$ squares to segment three different domains, (Square, Annulus, and L-Shape), following the scheme in [5]. We study the Poisson equation subject to the second order polynomial input functions $f(x, y) = a_1 x^2 + a_2 xy + a_3 y^2 + a_4 x + a_5 y + a_6$ for 10 different random realizations $a_i \sim \mathcal{N}(0, 2)$, and for $i = 1, ..., 6$. The ground truth is computed using FEM with a dense mesh for these domains.

Table 3: Averaged Numerical Error of Multiple Network Models and DGGF on 2D Domains

METHOD	SQUARE	ANNULUS	L-SHAPE
GAUSSNET	3.37E-2(6X6)	9.97E-2 (6X6)	1.43E-2(6X6)
GAUSSNET W/ SYMMETRY LOSS	2.76E-2(6X6)	7.45E-2 (6X6)	1.06E-2(6X6)
DGGF	1.13E-3	3.47E-3	1.42E-3

The symmetry loss, introduced in [5], is defined as $\min |G_\theta(\boldsymbol{r}, \boldsymbol{\xi}) - G_\theta(\boldsymbol{\xi}, \boldsymbol{r})|$. However, this symmetry property is valid for reciprocal systems and may not hold generally. For this reason, we did not include this symmetry constraint in the training of DGGFs, which allows our method to generalize to a wider range of PDE problems. The average run-time for these experiments is given in Table 4.

Table 4: Training Computational Complexity for The Experiments Presented in Table **??** in GPU hours

METHOD	SQUARE	ANNULUS	L-SHAPE
GAUSSNET	1.89H X 36	2.48H X 36	3.04H X 27
GAUSSNET W/ SYMMETRY LOSS	1.53H X 36	2.12H X 36	2.34H X 27
DGGF	0.12H + 0.08H	0.42H + 0.46H	0.12H + 0.13H

For GaussNet, the averaged computation time is reported since all the neural networks for subdomains are trained in parallel. Note that for the L-shaped domain, although it is segmented into 6×6, this domain only fills out 27 subdomains. For DGGF, there are two seriel steps of training two neural networks and the computation time is reported as the sum in the table. Since the GaussNet uses LBFGS for optimization, it is in general more time-consuming for each epoch compared to ADAM. In DGGF, the relaxed singularity means the generalized Green's function vary slower on the domain than the original Green's function, making it converge fast with ADAM.

B Deriving the Generalized Green's Function

In this section, we demonstrate that the generalized Green's function solved above can also construct the solution via integral operation with the input function f.

Theorem B.1 (Equivalence between Generalized Green's Function and Green's Function) *We have*

$$u(\boldsymbol{r}) = \int_D G^t(\boldsymbol{r}, \boldsymbol{\xi}) \Delta f(\boldsymbol{\xi}) d\boldsymbol{\xi} + \int_{\partial D} f \frac{\partial G^t}{\partial n} - G^t \frac{\partial f}{\partial n} ds \tag{23}$$

This equation can be further simplified if G^t, f or their partial derivatives vanish on the boundary. In that case, the relationship of the generalized Green's function and the solution function is simply,

$$u(\boldsymbol{r}) = \int_D G^t(\boldsymbol{r}, \boldsymbol{\xi}) \Delta f(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{24}$$

In other words, the solution can be expressed as the integral of the generalized function and the Laplacian of the input functions.

The problem to be solved is to find the solution function u given the input function f in Eq.(1). We start with formulating a Green's identity integral of G^t and $\mathcal{L}u$ with the operator of Laplacian,

$$\int_D d\boldsymbol{r} G^t \Delta(\mathcal{L}u) - (\mathcal{L}u)\Delta G^t = \int_{\partial D} ds G^t \frac{\partial \mathcal{L}u}{\partial n} - \mathcal{L}u \frac{\partial G^t}{\partial n} \tag{25}$$

Assume $\mathcal{L}$ to be self-adjoint and apply again the Green's identity on it, we could get

$$\int_D \Delta G^t(\mathcal{L}u) - u\Delta(\mathcal{L}G^t) d\boldsymbol{r} = \int_{\partial D} ds \gamma_{bd} \tag{26}$$

where interchangeability of $\mathcal{L}$ and the Dirac delta function is assumed and the boundary terms are involved with $\Delta G^t, u$ and their partial derivatives on the boundary depending on $\mathcal{L}$. Since we have specified $\Delta G^t = 0$ on the boundary, the boundary terms in the above equation vanish and $\int \mathcal{L}u\Delta G^t = \int u\mathcal{L}\Delta G^t = \int u\Delta t = \int u\Delta = u$. Therefore, by inserting this equation into Eq.(12), we have

$$u(\boldsymbol{r}) = \int_D G^t(\boldsymbol{r}, \boldsymbol{\xi}) \Delta f(\boldsymbol{\xi}) d\boldsymbol{\xi} + \int_{\partial D} f \frac{\partial G^t}{\partial n} - G^t \frac{\partial f}{\partial n} ds \tag{27}$$