FRIEDRICHS LEARNING: WEAK SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS VIA DEEP LEARNING

FAN CHEN

SCHOOL OF MATHEMATICAL SCIENCES, AND MOE-LSC,
SHANGHAI JIAO TONG UNIVERSITY, SHANGHAI 200240, CHINA (ALEXNWISH@SJTU.EDU.CN)

JIANGUO HUANG

SCHOOL OF MATHEMATICAL SCIENCES, AND MOE-LSC,
SHANGHAI JIAO TONG UNIVERSITY, SHANGHAI 200240, CHINA (JGHUANG@SJTU.EDU.CN)

CHUNMEI WANG

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF FLORIDA, GAINESVILLE, FL 32611, USA
(CHUNMEI.WANG@UFL.EDU)

HAIZHAO YANG

DEPARTMENT OF MATHEMATICS, PURDUE UNIVERSITY, WEST LAFAYETTE, IN 47907, USA
(HAIZHAO@PURDUE.EDU)

Abstract. This paper proposes Friedrichs learning as a novel deep learning methodology that can learn the weak solutions of PDEs via a minimax formulation, which transforms the PDE problem into a minimax optimization problem to identify weak solutions. The name "Friedrichs learning" is to highlight the close relation between our learning strategy and Friedrichs theory on symmetric systems of PDEs. The weak solution and the test function in the weak formulation are parameterized as deep neural networks in a mesh-free manner, which are alternately updated to approach the optimal solution networks approximating the weak solution and the optimal test function, respectively. Extensive numerical results indicate that our mesh-free Friedrichs learning method can provide reasonably good solutions for a wide range of PDEs defined on regular and irregular domains, where conventional numerical methods such as finite difference methods and finite element methods may be tedious or difficult to be applied, especially for those with discontinuous solutions in high-dimensional problems.

Key words. Partial Differential Equation; Friedrichs' System; Minimax Optimization; Weak Solution; Deep Neural Network; High Dimension; Complex Domain.

AMS subject classifications. 65M75; 65N75; 62M45.

1. Introduction.

High-dimensional PDEs and PDEs defined on complex domains are important tools in physical, financial, and biological models, etc. [46, 17, 62, 23, 61]. Generally speaking, they do not have closed-form solutions making numerical solutions of such equations indispensable in real applications. First, developing numerical methods for high-dimensional PDEs has been a challenging task due to the curse of dimensionality in conventional discretization. Second, conventional numerical methods rely on mesh generation that requires profound expertise and programming skills. In particular, for problems defined in complicated domains, it is challenging and time-consuming to implement conventional methods. As an efficient parametrization tool for high-dimensional functions [7, 15, 52, 51, 58, 34, 37, 56, 57] with user-friendly software (e.g., TensorFlow and PyTorch), neural networks have been applied to solve PDEs via various approaches recently. The idea of using neural networks to solve PDEs dates back to 1990s [45, 24, 13, 44] and was revisited and popularized recently [14, 27, 16, 42, 59, 9, 47, 8, 36, 35, 11, 55, 48, 63, 6, 50, 41, 38].

Many network-based PDE solvers are concerned with the classical solutions that are differentiable and satisfy PDEs in common sense. Unlike classical solutions, weak solutions are functions for which the derivatives may not all exist but which are nonetheless deemed to satisfy the PDE in some precisely defined sense. These solutions are crucial because many PDEs in modeling real-world phenomena do not have sufficiently smooth solutions. Motivated by the seminal work in

[6], we propose Friedrichs learning as an alternative method that can learn the weak solutions of elliptic, parabolic, and hyperbolic PDEs in $L^2(\Omega)$ via a novel minimax formulation devised and analyzed in Section 2.3. Since the formulation is closely related to the work of Friedrichs theory on symmetric systems of PDEs (cf. [22]), we call our learning strategy the Friedrichs learning. The main idea is to transform the PDE problem into a minimax optimization problem to identify weak solutions. Note that no regularity for the solution is required in Friedrichs learning, which is the main advantage of the proposed method, making it applicable to a wide range of PDE problems, especially those with discontinuous solutions. In addition, Friedrichs learning is capable of solving PDEs with discontinuous solutions without a priori knowledge of the location of the discontinuity. Although the current version of Friedrichs learning may not be able to provide highly accurate solutions, obtaining a coarse solution without a priori knowledge of the discontinuity could provide a rough estimation of the discontinuity and could be a good initial guess of conventional computation approaches for highly accurate solutions following the Int-Deep framework in [33]. Finally, theoretical results are provided to justify the Firedrichs learning framework for various PDEs.

The main philosophy of Friedrichs learning is to reformulate a PDE problem into a minimax optimization, the solution of which is a test deep neural network (DNN) that maximizes the loss and a solution DNN that minimizes the loss. For a high-order PDE, we first reformulate it into a first-order PDE system by introducing auxiliary variables, the weak form of which naturally leads to a minimax optimization using integration by parts according to the theory of Friedrichs' system [22]. The above-mentioned feature is the crucial difference from existing deep learning methods for weak solutions [16, 63]. Let us introduce the formulation of Friedrichs learning using first-order boundary value problems (BVPs) with homogeneous boundary conditions without loss of generality. The initial value problems (IVPs) can be treated as BVPs, where the time variable is considered to be one more spatial variable. The non-homogeneous boundary conditions can be easily transferred to homogenous ones by subtracting the boundary functions from the solutions.

In the seminal results by Friedrichs in [22] and other investigations in [5, 21], an abstract framework of the boundary value problem of the first-order system was established, which is referred to as Friedrichs' system in the literature. Let us introduce the concept of Firedrichs' system using a concrete and simple example and illustrate the main idea and intuition of the Friedrichs learning proposed in this paper. A more detailed abstract framework of Friedrichs learning will be discussed later in Section 2. Let $r \in \mathbb{N}$ and $\Omega \subset \mathbb{R}^d$ be an open and bounded domain with Lipschitz boundary $\partial\Omega$. The notation $(\cdot)^\intercal$ denotes the transpose of a vector or a matrix throughout the paper. We assume: 1) $\boldsymbol{A}_k \in [L^\infty(\Omega)]^{r,r}$, $\sum_{k=1}^d \partial_k \boldsymbol{A}_k \in [L^\infty(\Omega)]^{r,r}$, $\boldsymbol{A}_k = \boldsymbol{A}_k^\intercal$ a.e. in Ω for $k = 1, \ldots, d$, and $\boldsymbol{C} \in [L^\infty(\Omega)]^{r,r}$; 2) the full coercivity holds true, i.e., $\boldsymbol{C} + \boldsymbol{C}^\intercal - \sum_{k=1}^d \partial_k \boldsymbol{A}_k \geq 2\mu_0 \boldsymbol{I}_r$ a.e. in Ω for some $\mu_0 > 0$ and the identity matrix $\boldsymbol{I}_r \in \mathbb{R}^{r\times r}$. Then the first-order differential operator $T : \mathcal{D} \to L$ with $L = [L^2(\Omega)]^r$ and $\mathcal{D} = [C_0^\infty(\Omega)]^r$ defined by $T\boldsymbol{u} := \sum_{k=1}^d \boldsymbol{A}_k \partial_k \boldsymbol{u} + \boldsymbol{C}\boldsymbol{u}$ is called the Friedrichs operator, while the first-order system of PDE $T\boldsymbol{u} = \boldsymbol{f}$ is called the Friedrichs' system, where $\boldsymbol{f}$ is a given data function in L. Throughout this paper, the bold font will be used for vectors and matrices in concrete examples. In our abstract framework, PDE solutions are considered as elements of a Hilber space and, hence, they are not denoted as bold letters.

Friedrichs [22] also introduced an abstract framework for representing boundary conditions via matrix valued boundary fields. First, let $\boldsymbol{A}_{\boldsymbol{n}} := \sum_{k=1}^d n_k \boldsymbol{A}_k \in L^\infty(\partial\Omega; M_r)$, where $\boldsymbol{n} = (n_1, \cdots, n_d) \in \mathbb{R}^d$ is the unit outward normal direction on $\partial\Omega$, M_r is the set of matrix of size $r \times r$, and let $\boldsymbol{M} : \partial\Omega \to M_r$ be a matrix field on the boundary. Then a homogeneous Dirichlet boundary condition of Friedrichs' system is prescribed by $(\boldsymbol{A}_{\boldsymbol{n}} - \boldsymbol{M})\boldsymbol{u} = \boldsymbol{0}$ on $\partial\Omega$ by choosing an appropriate $\boldsymbol{M}$ to ensure the well-poseness of Friedrichs' system. In real applications, $\boldsymbol{M}$ is given by physical knowledge. Let $V := \mathcal{N}(\boldsymbol{A}_{\boldsymbol{n}} - \boldsymbol{M})$ and $V^* := \mathcal{N}(\boldsymbol{A}_{\boldsymbol{n}} + \boldsymbol{M}^\intercal)$, where $\mathcal{N}$ is the null space of the

argument. It has been proved that $\boldsymbol{u}$ solves the BVP

$$T\boldsymbol{u} = \boldsymbol{f} \text{ in } \Omega \quad \text{and} \quad (\boldsymbol{A_n} - \boldsymbol{M})\boldsymbol{u} = \boldsymbol{0} \text{ on } \partial\Omega, \tag{1.1}$$

if and only if $\boldsymbol{u}$ solves the minimax problem

$$\min_{\boldsymbol{u}\in V}\max_{\boldsymbol{v}\in V^*} \mathcal{L}(\boldsymbol{u},\boldsymbol{v}) := \frac{|(\boldsymbol{u},\tilde{T}\boldsymbol{v})_L - (\boldsymbol{f},\boldsymbol{v})_L|}{\|\tilde{T}\boldsymbol{v}\|_L},$$

where $\tilde{T} : \mathcal{D} \to L$ is the formal adjoint of T. Hence, in our Friedrichs learning, DNNs are applied to parametrize $\boldsymbol{u}$ and $\boldsymbol{v}$ to solve the above minimax problem to obtain the solution of the BVP (1.1). Friedrichs learning also works for other kinds of boundary conditions.

This paper is organized as follows. In Section 2, we devise and analyze Friedrichs minimax formulation for weak solutions of PDEs. In Section 3, several concrete examples of PDEs and their minimax formulations are provided. In Section 4, network-based optimization is introduced to solve the minimax problem in Friedrichs formulation. In Section 5, a series of numerical examples are provided to demonstrate the effectiveness of the proposed Friedrichs learning. Finally, we conclude this paper in Section 6.

2. Friedrichs Minimax Formulation for Weak Solutions. We shall first briefly review Friedrichs' system in a Hilbert space setting [1, 10]. We shall introduce and analyze Friedrichs minimax formulation for weak solutions which is the foundation of Friedrichs learning.

2.1. An Abstract Framework of Friedrichs' System. Firstly, we recall some basic results on Friedrichs' system for later uses [10, 1]. We assume L is a real Hilbert space associated with the inner product $(\cdot,\cdot)_L$ and the induced norm $\|\cdot\|_L$. The dual space of L, denoted by L', can be identified naturally with L by the Riesz representation theorem. For a dense subspace $\mathcal{D}$ of L, we consider two linear operators $T : \mathcal{D} \to L$ and $\tilde{T} : \mathcal{D} \to L$ satisfying the following properties: for any $u, v \in \mathcal{D}$, there exists a positive constant C such that

$$(Tu, v)_L = (u, \tilde{T}v)_L, \tag{2.1}$$

$$\|(T + \tilde{T})u\|_L \le C\|u\|_L. \tag{2.2}$$

Here, $\tilde{T}$ is called the formal adjoint of T. As shown in [1, Lemma 2.1], we define a graph space W

$$W = \{u \in L;\ Tu \in L\}, \tag{2.3}$$

which is a Hilbert space with respect to the graph norm $\|\cdot\|_T = (\cdot,\cdot)_T^{1/2}$ induced by the inner product $(\cdot,\cdot)_T = (\cdot,\cdot)_L + (T\cdot, T\cdot)_L$. In addition, owing to (2.2), we have

$$W = \{u \in L;\ \tilde{T}u \in L\}.$$

In other words, W is also a graph space associated with $\tilde{T}$.

The abstract framework of Friedrichs' system concerns the solvability of the problem

$$Tu = f \in L, \tag{2.4}$$

and its solution falls in the graph space W. Obviously, the problem (2.4) may not be well-posed since its solution in W may not be unique. We are interested in constructing a subspace $V \subseteq W$ such that $T : V \to L$ is an isomorphism. A standard way is carried out as follows. We first define a self-adjoint boundary operator $B \in \mathcal{L}(W, W')$ as follows (cf. [1]). For any $u, v \in W$,

$$\langle Bu, v\rangle_{W'\times W} = (Tu, v)_L - (u, \tilde{T}v)_L. \tag{2.5}$$

This operator plays a key role in the forthcoming analysis. Moreover, the identity (2.5) can be reformulated in the form

$$(Tu, v)_L = (u, \tilde{T}v)_L + \langle Bu, v\rangle_{W'\times W},$$

which is usually regarded as an abstract integration by parts formula (cf. [1]).

Next, we assume that there exists an operator $M \in \mathcal{L}(W, W')$ such that

$$\langle Mw, w\rangle \geq 0, \quad \forall\, w \in W, \tag{2.6}$$

$$W = \mathcal{N}(B - M) + \mathcal{N}(B + M), \tag{2.7}$$

where $\mathcal{N}$ is the null space of its argument. Furthermore, let $M^* \in \mathcal{L}(W, W')$ denote the adjoint operator of M given by $\langle M^*u, v\rangle_{W'\times W} = \langle Mv, u\rangle_{W'\times W}, \forall\, u, v \in W$.

To find V such that the problem (2.4) is well-posed, we should make an additional assumption for L as follows; i.e.,

$$((T + \tilde{T})v, v)_L \geq 2\mu_0\|v\|_L^2, \quad \forall\, v \in L, \tag{2.8}$$

where μ_0 is a positive constant. Then we choose

$$V = \mathcal{N}(B - M), \quad V^* = \mathcal{N}(B + M^*). \tag{2.9}$$

We have the following important theory for Friedrichs' system [1, Lemma 3.2 and Theorem 3.1].

THEOREM 2.1. *Assume* (2.2) (2.8), (2.6) *and* (2.7) *hold true. Let* V *and* V^* *be given by* (2.9). *The following statements hold true:*

1. *For any* $v \in W$, *there holds*

$$\mu_0\|v\|_L \leq \|Tv\|_L, \quad \mu_0\|v\|_L \leq \|\tilde{T}v\|_L. \tag{2.10}$$

2. *For any* $f \in L$, *problem* (2.4) *has a unique solution in* V. *In other words,* T *is an isomorphism from* V *onto* L. *Moreover,* $\tilde{T}$ *is an isomorphism from* V^* *onto* L.

2.2. First Order PDEs of Friedrichs Type. As a typical application of the above framework, we restrict L to be the space of square integral (vector-valued) functions over an open and bounded domain $\Omega \subset \mathbb{R}^d$ with Lipschitz boundary, $\mathcal{D}$ to be the space of test functions, and T to be a first-order differential operator with its formal adjoint $\tilde{T}$. In particular, we take $L = [L^2(\Omega)]^r$, $r \in \mathbb{N}$ and $\mathcal{D} = [D(\Omega)]^r$, where $D(\Omega) = C_0^\infty(\Omega)$. $\mathcal{D}$ is thus dense in L. Consider $T : \mathcal{D} \to L$ as follows

$$T\boldsymbol{u} = \sum_{k=1}^{d} \boldsymbol{A}_k\partial_k\boldsymbol{u} + \boldsymbol{C}\boldsymbol{u} = \boldsymbol{f}, \quad \forall\, \boldsymbol{u} \in \mathcal{D}. \tag{2.11}$$

The standard assumptions are imposed on $\boldsymbol{A}_k$ and $\boldsymbol{C}$ for Friedrichs' system [18, 19, 22]:

$$\boldsymbol{C} \in [L^\infty(\Omega)]^{r,r}, \tag{2.12}$$

$$\boldsymbol{A}_k \in [L^\infty(\Omega)]^{r,r}, k = 1, \cdots, d \quad \text{and} \quad \sum_{k=1}^{d} \partial_k\boldsymbol{A}_k \in [L^\infty(\Omega)]^{r,r} \tag{2.13}$$

$$\boldsymbol{A}_k = \boldsymbol{A}_k^\intercal, \quad \text{a. e. in } \Omega, k = 1, \cdots, d. \tag{2.14}$$

The formal adjoint $\tilde{T}:\mathcal{D}\to L$ of T can be defined by

$$\tilde{T}\boldsymbol{u}=-\sum_{k=1}^{d}\boldsymbol{A}_k\partial_k\boldsymbol{u}+(\boldsymbol{C}^{\intercal}-\sum_{k=1}^{d}\partial_k\boldsymbol{A}_k)\boldsymbol{u},\quad\forall\,\boldsymbol{u}\in\mathcal{D}. \tag{2.15}$$

It is easy to see that T and $\tilde{T}$ satisfy (2.1)-(2.2). All the results in this section hold true for Friedrichs' system satisfying (2.12)-(2.14).

Regarding to the abstract Friedrichs' system, an explicit representation of B could be found; however, it is impossible for M on the abstract level. Assume $\mathcal{B}=\sum_{k=1}^{d}n_k\boldsymbol{A}_k$ is well-defined a.e. on $\partial\Omega$ where $\boldsymbol{n}=(n_1,\cdots,n_d)^{\intercal}$ is the unit outward normal vector of $\partial\Omega$. For simplicity of notations, we set $\mathcal{H}^s=[H^s]^r$ with H^s being the usual Sobolev space of order s, and $\mathcal{C}^1=[C^1]^r$ with C^1 being the space of continuously differentiable functions.

LEMMA 2.2. *[3, 39] For $\boldsymbol{u},\boldsymbol{v}\in\mathcal{H}^1(\Omega)\subset W(\Omega)$, there holds*

$$\langle B\boldsymbol{u},\boldsymbol{v}\rangle_{W'(\Omega)\times W(\Omega)}=\langle\mathcal{B}\boldsymbol{u},\boldsymbol{v}\rangle_{\mathcal{H}^{-\frac{1}{2}}(\partial\Omega)\times\mathcal{H}^{\frac{1}{2}}(\partial\Omega)},$$

where $W(\Omega)=\{\boldsymbol{u}\in L(\Omega);\ T\boldsymbol{u}\in L(\Omega)\}$ and $W'(\Omega)$ is the dual space of $W(\Omega)$. Specifically, $\langle B\boldsymbol{u},\boldsymbol{v}\rangle_{W'(\Omega)\times W(\Omega)}=\int_{\partial\Omega}\boldsymbol{v}^{\intercal}\mathcal{B}\boldsymbol{u}ds$, for any $\boldsymbol{u},\boldsymbol{v}\in C_0^\infty(\mathbb{R}^d)$.

If Ω has segment property [4], $\mathcal{C}^1(\overline{\Omega})$ is thus dense in $\mathcal{H}^1(\Omega)$ and further is dense in $W(\Omega)$. Therefore, the representation could be uniquely extended to the whole space $W(\Omega)$ in the sense that for any $\boldsymbol{u}\in W(\Omega)$ and $\boldsymbol{v}\in\mathcal{H}^1(\Omega)$,

$$\langle B\boldsymbol{u},\boldsymbol{v}\rangle_{W'(\Omega)\times W(\Omega)}=\langle\mathcal{B}\boldsymbol{u},\boldsymbol{v}\rangle_{\mathcal{H}^{-\frac{1}{2}}(\partial\Omega)\times\mathcal{H}^{\frac{1}{2}}(\partial\Omega)}. \tag{2.16}$$

The coercivity condition on T dictated by the positiveness condition on the coefficients $\boldsymbol{A}_k$ and $\boldsymbol{C}$ [18, 19, 20] is needed to show the well-posedness of PDEs of Friedrichs type. After some direct manipulation, the abstract coercivity condition (2.8) is equivalent to the following full coercivity for Friedrichs PDEs:

$$\boldsymbol{C}+\boldsymbol{C}^{\intercal}-\sum_{k=1}^{d}\partial_k\boldsymbol{A}_k\geq 2\mu_0\boldsymbol{I}_r,\quad\text{a.e., in }\Omega, \tag{2.17}$$

where μ_0 is a positive constant and $\boldsymbol{I}_r$ is the $r\times r$ identity matrix. If a system does not satisfies the coercivity condition (2.17) we can introduce a feasible transformation so that the modified system satisfies this condition. In [10], the authors introduced the so-called partial coercivity condition to study the mathematical theory of the corresponding system. Readers are referred to [10] for more details.

2.3. Friedrichs Minimax Formulation. Throughout this subsection, we assume all the conditions given in Theorem 2.1 hold true. Recall that $V=\mathcal{N}(B-M)$ and $V^*=\mathcal{N}(B+M^*)$ with $M\in\mathcal{L}(W,W')$ satisfying conditions (2.6)-(2.7). For a given $f\in L$, find the solution $u\in V$ such that

$$Tu=f, \tag{2.18}$$

or equivalently,

$$(Tu,v)_L=(f,v)_L,\quad\forall\,v\in L. \tag{2.19}$$

In most cases, L is a differential operator whose action on a function should be understood in the sense of distributions. u is thus called the weak solution of the primal variational equation (2.19). We restrict $v \in V^* \subset L$. From (2.5),

$$\begin{aligned}(Tu, v)_L =& (u, \tilde{T}v)_L + \langle Bu, v\rangle_{W'\times W} \\ =& (u, \tilde{T}v)_L + \langle \frac{B-M}{2}u, v\rangle_{W'\times W} + \langle \frac{B+M}{2}u, v\rangle_{W'\times W} \\ =& (u, \tilde{T}v)_L + \langle u, \frac{B+M^*}{2}v\rangle_{W'\times W} = (u, \tilde{T}v)_L,\end{aligned}$$

where we used $u \in V = \mathcal{N}(B-M)$ and $v \in V^* = \mathcal{N}(B+M^*)$. This, combined with (2.19), gives

$$(u, \tilde{T}v)_L = (f, v)_L, \quad \forall\, v \in V^*. \tag{2.20}$$

For $u \in V$, (2.20) is equivalent to (2.19). For $u \in L$ satisfying (2.20), u is called the weak solution of the dual variational equation (2.20).

For $u \in V$, $v \in V^*$, we define

$$\mathcal{L}(u, v) := \frac{|(u, \tilde{T}v)_L - (f, v)_L|}{\|\tilde{T}v\|_L}. \tag{2.21}$$

According to the estimate (2.10), we have

$$|(u, \tilde{T}v)_L - (f, v)_L| \le \|u\|_L\|\tilde{T}v\|_L + \|f\|_L\|v\|_L \le (\|u\|_L + \frac{1}{\mu_0}\|f\|_L)\|\tilde{T}v\|_L,$$

where μ_0 is given in (2.8). Therefore, the functional $\mathcal{L}(u, v)$ is bounded with respect to $v \in V^*$ for a fixed $u \in L$.

Thus we can reformulate the problem (2.18) or equivalently the problem (2.19) as the following minimax problem formally:

$$\min_{u\in V}\max_{v\in V^*}\mathcal{L}(u, v) := \min_{u\in V}\max_{v\in V^*}\frac{|(u, \tilde{T}v)_L - (f, v)_L|}{\|\tilde{T}v\|_L}, \tag{2.22}$$

to identify the weak solution of the primal variational equation (2.19).

THEOREM 2.3. *Assume all the conditions given in Theorem 2.1 hold true. Then u is the unique weak solution of the primal variational equation* (2.19) *if and only if u is the unique solution that solves the minimax problem* (2.22).

Proof. On one hand, if $u \in V$ is a weak solution of (2.19), we have from (2.20) that $\mathcal{L}(u, v) = 0$ for all $v \in V^*$. Thus, u is a solution of the minimax problem (2.22).

One the other hand, if u is a solution of the minimax problem (2.22), then

$$\max_{v\in V^*}\mathcal{L}(u, v) = \max_{v\in V^*}\frac{|(u, \tilde{T}v)_L - (f, v)_L|}{\|\tilde{T}v\|_L} = 0.$$

Thus, we have $\mathcal{L}(u, v) = 0$ for all $v \in V^*$. This implies

$$(u, \tilde{T}v)_L - (f, v)_L = 0, \quad \forall\, v \in V^*.$$

Since u is in V, the above equation gives

$$(Tu - f, v)_L = 0, \quad \forall\, v \in V^*.$$

Observing that $\mathcal{D}$ belongs to V^* and is dense in L, the above equation implies that u is a weak solution of the primal variational equation (2.19).

Finally, under the the conditions given in Theorem 2.1, it is well known that the weak solution u of the primal variational equation (2.19) exists and is unique. This completes the proof of this theorem.

□

Note that the above discussion and Theorem 2.3 are concerned with the weak solution of the primal variational equation (2.19) with a solution u being in V. It is also of interest to discuss the weak solution u of the dual variational equation (2.20) with u being in L due to Friedrichs (cf. [22]). According to similar arguments for proving Theorem 2.3, we have the following theorem.

THEOREM 2.4. *Assume all the conditions given in Theorem 2.1 hold true. Then u is a weak solution of the dual variational equation* (2.20) *if and only if u is a solution of the following minimax problem:*

$$\min_{u\in L}\max_{v\in V^*}\mathcal{L}(u,v)=\min_{u\in L}\max_{v\in V^*}\frac{|(u,\tilde{T}v)_L-(f,v)_L|}{\|\tilde{T}v\|_L}. \tag{2.23}$$

Note that the weak solution of the dual variational equation (2.20) in L may not be unique which is also true for the minimax problem (2.23). However, their solution is unique for Friedrichs' system mentioned in the Subsection 2.2, due to the equivalence between the weak solution and the strong solution (cf. [22]). In this case, the two problems (2.22) and (2.23) are equivalent.

Theorems 2.3-2.4 have covered various interesting equations in real applications. However, we would like to mention that Friedrichs learning can be extended to a more general setting, e.g., $u \in L$ but the data function f in $Tu = f$ is not necessarily in L. Since the solution space L is more generic than V including solutions with discontinuity, this setting has a wide range of applications in fluid mechanics. We will show this application by a numerical example for the advection-reaction problem in Section 5. Theoretical analysis for more general cases is left as future work.

3. Examples of PDEs and the Corresponding Minimax Formulation. Using the abstract framework and the minimax formulation developed in Section 2, we will derive the minimax formulations for several typical PDEs. From now on, we will denote by $(\cdot,\cdot)_\Omega$ the standard L^2 inner product, which induces the L^2 norm $\|\cdot\|_\Omega$. These notations also apply to L^2 smooth vector-valued functions. For simplicity, we will focus on PDEs with homogeneous boundary conditions throughout this section.

3.1. Advection-Reaction Equation. The advection-reaction equation seeks u such that

$$\mu u+\boldsymbol{\beta}\cdot\nabla u=f, \tag{3.1}$$

where $\boldsymbol{\beta}=(\beta_1,\cdots,\beta_d)^\intercal\in[L^\infty(\Omega)]^d$, $\nabla\cdot\boldsymbol{\beta}\in L^\infty(\Omega)$, $\mu\in L^\infty(\Omega)$ and $f\in L^2(\Omega)$. Compared with (2.11), (3.1) is a Friedrichs' system by setting $\boldsymbol{A}_k=\beta_k$ for $k=1,2,...,d$ and $\boldsymbol{C}=\mu$.

We assume there exists $\mu_0>0$ such that

$$\mu(\boldsymbol{x})-\frac{1}{2}\nabla\cdot\boldsymbol{\beta}(\boldsymbol{x})\geq\mu_0>0,\quad\text{a.e. in }\Omega. \tag{3.2}$$

Thus, the full coercivity condition in (2.17) holds true. The graph space W given by (2.3) is

$$W=\{w\in L^2(\Omega);\ \boldsymbol{\beta}\cdot\nabla w\in L^2(\Omega)\}.$$

We define the inflow and outflow boundary for the advection-reaction equation (3.1):

$$\partial\Omega^-=\{\boldsymbol{x}\in\partial\Omega;\boldsymbol{\beta}(\boldsymbol{x})\cdot\boldsymbol{n}(\boldsymbol{x})<0\},\quad\partial\Omega^+=\{\boldsymbol{x}\in\partial\Omega;\boldsymbol{\beta}(\boldsymbol{x})\cdot\boldsymbol{n}(\boldsymbol{x})>0\}. \tag{3.3}$$

To enforce boundary conditions, we choose from the physical interpretation that

$$V = \{v \in W; v|_{\partial\Omega^-} = 0\}, V^* = \{v \in W; v|_{\partial\Omega^+} = 0\}. \tag{3.4}$$

In this case, it is easy to check that the conditions (2.6) and (2.7) hold true. By (2.11),

$$\tilde{T}v = -\sum_{i=1}^{d}\Big(\beta_i\frac{\partial v}{\partial x_i} + \frac{\partial}{\partial x_i}\beta_i v\Big) + \boldsymbol{C}^\intercal v = -\boldsymbol{\beta}\cdot\nabla v - (\nabla\cdot\boldsymbol{\beta})v + \mu v.$$

The minimax problem is thus given as follows

$$\min_{u\in V}\max_{v\in V^*}\mathcal{L}(u,v) = \min_{u\in V}\max_{v\in V^*}\frac{|\big(u, -(\boldsymbol{\beta}\cdot\nabla v + (\nabla\cdot\boldsymbol{\beta})v) + \mu v\big)_\Omega - (f,v)_\Omega|}{\|\boldsymbol{\beta}\cdot\nabla v + (\nabla\cdot\boldsymbol{\beta})v - \mu v\|_\Omega}.$$

Note that if the coercivity condition (3.2) does not hold true, we can introduce a transformation $u = e^{\lambda_0 t}\tilde{u}$, so that the advection-reaction equation (3.1) in $\tilde{u}$ satisfies (3.2) for sufficiently large constant $\lambda_0 > 0$.

3.2. Scalar Elliptic PDEs. Consider the second-order PDE to find u satisfying

$$-\Delta u + \mu u = f, \quad \text{in } \Omega, \tag{3.5}$$

where $\Omega \subset \mathbb{R}^d$, $\mu \in L^\infty(\Omega)$ is positive and uniformly bounded away from zero, $f \in L^2(\Omega)$. This PDE can be rewritten into a first-order PDE system by introducing an auxiliary function $\boldsymbol{v}$; i.e.,

$$\boldsymbol{v} + \nabla u = 0, \qquad \mu u + \nabla\cdot\boldsymbol{v} = f.$$

This first order system could be formulated into a Friedrichs' system with $r = d+1$. The Hilbert space L is chosen as $L = [L^2(\Omega)]^r$. Let $\tilde{\boldsymbol{u}} = (\boldsymbol{v}^\intercal, u)^\intercal \in L$. For $k = 1, 2, ..., d$, $\boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{e}^k \\ (\boldsymbol{e}^k)^\intercal & \boldsymbol{0} \end{bmatrix}$, $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \boldsymbol{0} & \mu \end{bmatrix}$, where $\boldsymbol{e}^k$ is the k-th canonical basis of $\mathbb{R}^d$. Since $\mu > 0$ and has a lower bound away from zero, the full coercivity condition (2.17) is satisfied. The graph space is

$$W = H(\mathrm{div};\Omega)\times H^1(\Omega).$$

One possible choice of the Dirichlet boundary condition is as follows

$$V = V^* = H(\mathrm{div};\Omega)\times H_0^1(\Omega) = \{(\boldsymbol{v}^\intercal, u)^\intercal \in W;\ u|_{\partial\Omega} = 0\}. \tag{3.6}$$

The choices of boundary conditions are not unique obviously. By introducing auxiliary variables, the second-order linear PDE can be reformulated into a first-order PDE system. Finally, the weak solution of (3.5) can be found by solving the equivalent minimax problem in (2.22).

Denote the test function by $\boldsymbol{\psi} = (\psi_{\boldsymbol{v}}^\intercal, \psi_u)^\intercal$ in the space V^*. The minimax problem can be presented as

$$\min_{\tilde{\boldsymbol{u}}\in V}\max_{\boldsymbol{\psi}\in V^*}\mathcal{L}(\tilde{\boldsymbol{u}},\boldsymbol{\psi}) = \min_{\tilde{\boldsymbol{u}}\in V}\max_{\boldsymbol{\psi}\in V^*}\frac{|\big(-\boldsymbol{v}, \psi_{\boldsymbol{v}} - \nabla\psi_u\big)_\Omega + (u, \mu\psi_u - \nabla\cdot\psi_{\boldsymbol{v}})_\Omega - (f,\psi_u)_\Omega|}{\|((\psi_{\boldsymbol{v}} - \nabla\psi_u)^\intercal, \mu\psi_u - \nabla\cdot\psi_{\boldsymbol{v}})^\intercal\|_\Omega}.$$

To reduce the computational cost, we will reformulate the above formulation into a minimax problem in a primal form. To this end, letting $\psi_{\boldsymbol{v}} = \nabla\psi_u$, and noting that $\tilde{\boldsymbol{u}} = (\nabla u^\intercal, u)^\intercal$, we have by a direct manipulation that

$$\mathcal{L}(\tilde{\boldsymbol{u}},\boldsymbol{\psi}) = \frac{|(u, \mu\psi_u - \Delta\psi_u)_\Omega - (f,\psi_u)_\Omega|}{\|\mu\psi_u - \Delta\psi_u\|_\Omega},$$

which induces the following minimax problem

$$\min_{u\in H_0^1(\Omega)}\max_{\psi_u\in H_0^1(\Omega)}\mathcal{L}(u,\psi_u)=\min_{u\in H_0^1(\Omega)}\max_{\psi_u\in H_0^1(\Omega)}\frac{|(u,\mu\psi_u-\Delta\psi_u)_\Omega-(f,\psi_u)_\Omega|}{\|\mu\psi_u-\Delta\psi_u\|_\Omega}. \tag{3.7}$$

In fact, we can derive the above minimax problem in a rigorous way. From (3.5), we have

$$(-\Delta u+\mu u,\psi_u)=(f,\psi_u),\qquad \forall\,\psi_u\in H_0^1(\Omega),$$

which, from the usual integration by parts twice, gives

$$(u,\mu\psi_u-\Delta\psi_u)_\Omega=(f,\psi_u)_\Omega,\qquad \forall\,\psi_u\in H_0^1(\Omega).$$

This will naturally give the minimax problem (3.7).

3.3. Maxwell's Equation in the Diffusion Regime. The Maxwell's equations in $\mathbb{R}^3$ in the diffusive regime could be considered as

$$\mu\boldsymbol{H}+\nabla\times\boldsymbol{E}=\boldsymbol{f},\qquad \sigma\boldsymbol{E}-\nabla\times\boldsymbol{H}=\boldsymbol{g}, \tag{3.8}$$

with μ and σ being two positive functions in $L^\infty(\Omega)$ and uniformly bounded away from zero. Three-dimensional functions $\boldsymbol{f},\boldsymbol{g}$ lie in the space $[L^2(\Omega)]^3$ and the solution functions $(\boldsymbol{H}^\intercal,\boldsymbol{E}^\intercal)^\intercal$ are in the space $[L^2(\Omega)]^3\times[L^2(\Omega)]^3$. In Equation (2.11), set $r=6$ and let $\boldsymbol{A}_k\in\mathbb{R}^{6\times6}$ and $\boldsymbol{C}$ be $\boldsymbol{A}_k=\begin{bmatrix}\boldsymbol{0}&\boldsymbol{\mathcal{R}^k}\\(\boldsymbol{\mathcal{R}^k})^\intercal&\boldsymbol{0}\end{bmatrix}$, $\boldsymbol{C}=\begin{bmatrix}\mu\cdot\boldsymbol{I}_3&\boldsymbol{0}\\\boldsymbol{0}&\sigma\cdot\boldsymbol{I_3}\end{bmatrix}$, for $k=1,2,3$. Here, the entries of $\mathcal{R}^k_{ij}=\text{sign(i}-\text{j)}$ if $i=k+1(\text{mod }3)$ and $\mathcal{R}^k_{ij}=0$ otherwise. The graph space is defined as $W=[H(\text{curl};\Omega)]^3\times[H(\text{curl};\Omega)]^3$. One example of the boundary condition is $V=V^*=[H(\text{curl};\Omega)]^3\times[H_0(\text{curl};\Omega)]^3$. The function pair $\boldsymbol{u}:=(\boldsymbol{H}^\intercal,\boldsymbol{E}^\intercal)^\intercal\in W$ is in V whenever $\boldsymbol{E}\times\boldsymbol{n}|_{\partial\Omega}=0$. Let $\boldsymbol{\psi}=(\boldsymbol{\psi}_{\boldsymbol{H}}^\intercal,\boldsymbol{\psi}_{\boldsymbol{E}}^\intercal)^\intercal$ be the test function in V^*. Then the minimax problem in (2.22) becomes

$$\min_{\boldsymbol{u}\in V}\max_{\boldsymbol{\psi}\in V^*}\frac{|\big(\boldsymbol{H},-\nabla\times\boldsymbol{\psi}_{\boldsymbol{E}}+\mu\boldsymbol{\psi}_{\boldsymbol{H}}\big)_\Omega+\big(\boldsymbol{E},\nabla\times\boldsymbol{\psi}_{\boldsymbol{H}}+\sigma\boldsymbol{\psi}_{\boldsymbol{E}}\big)_\Omega-(\boldsymbol{f},\boldsymbol{\psi}_{\boldsymbol{H}})_\Omega-(\boldsymbol{g},\boldsymbol{\psi}_{\boldsymbol{E}})_\Omega|}{\|((-\nabla\times\boldsymbol{\psi}_{\boldsymbol{E}}+\mu\boldsymbol{\psi}_{\boldsymbol{H}})^\intercal,(\nabla\times\boldsymbol{\psi}_{\boldsymbol{H}}+\sigma\boldsymbol{\psi}_{\boldsymbol{E}})^\intercal)^\intercal\|_\Omega}. \tag{3.9}$$

4. Deep Learning-Based Solver. To complete the introduction of Friedrichs learning, we introduce a deep learning-based method to solve the minimax optimization in (2.22) or (2.23) for the weak solution of (2.18) or (2.20) in this section. For simplicity, we will focus on the minimax optimization (2.22) to identify the weak solution of (2.18).

4.1. Overview. In the deep learning-based method, one solution DNN, $\phi_s(\boldsymbol{x};\theta_s)$, is applied to parametrize the weak solution u in (2.22) and another test DNN, $\phi_t(\boldsymbol{x};\theta_t)$, is used to parametrize the test function ψ in (2.22). Here, θ_s and θ_t are the parameters to be identified such that

$$\begin{aligned}(\bar\theta_s,\bar\theta_t)&=\arg\min_{\theta_s}\max_{\theta_t}L(\phi_s(\boldsymbol{x};\theta_s),\phi_t(\boldsymbol{x};\theta_t))\\&=\arg\min_{\theta_s}\max_{\theta_t}\frac{|(\phi_s(\boldsymbol{x};\theta_s),\tilde T\phi_t(\boldsymbol{x};\theta_t))_\Omega-(f,\phi_t(\boldsymbol{x};\theta_t))_\Omega|}{\|\tilde T\phi_t(\boldsymbol{x};\theta_t)\|_\Omega},\end{aligned} \tag{4.1}$$

under the constraints

$$\phi_s(\boldsymbol{x};\theta_s)\in V\text{ and }\phi_t(\boldsymbol{x};\theta_t)\in V^*.$$

For simplicity, we use $L(\theta_s,\theta_t)$ for short to represent $L(\phi_s(\boldsymbol{x};\theta_s),\phi_t(\boldsymbol{x};\theta_t))$ from now on.

4.2. Network Implementation. Now, we will introduce the network structures of the solution DNN and test DNN used in the previous section. In this paper, all DNNs are chosen as ResNet [29] defined as follows. Let $\phi(\boldsymbol{x};\theta)$ denote such a network with an input $\boldsymbol{x}$ and parameter θ, which is defined recursively using a nonlinear activation function σ as follows:

$$\boldsymbol{h}_0 = \boldsymbol{V}\boldsymbol{x}, \boldsymbol{g}_\ell = \sigma(\boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1} + \boldsymbol{b}_\ell), \boldsymbol{h}_\ell = \bar{\boldsymbol{U}}_\ell \boldsymbol{h}_{\ell-2} + \boldsymbol{U}_\ell \boldsymbol{g}_\ell, \ell = 1, 2, \ldots, L, \phi(\boldsymbol{x};\theta) = \boldsymbol{a}^\intercal \boldsymbol{h}_L, \tag{4.2}$$

where $\boldsymbol{V} \in \mathbb{R}^{m\times d}$, $\boldsymbol{W}_\ell \in \mathbb{R}^{m\times m}$, $\bar{\boldsymbol{U}}_\ell \in \mathbb{R}^{m\times m}$, $\boldsymbol{U}_\ell \in \mathbb{R}^{m\times m}$, $\boldsymbol{b}_\ell \in \mathbb{R}^m$ for $\ell = 1, \ldots, L$, $\boldsymbol{a} \in \mathbb{R}^m$, $\boldsymbol{h}_{-1} = \boldsymbol{0}$. Throughout this paper, $\boldsymbol{U}_\ell$ is set as an identity matrix in the numerical implementation of ResNets for the purpose of simplicity. Furthermore, as used in [16], we set $\bar{\boldsymbol{U}}_\ell$ as the identify matrix when ℓ is even and set $\bar{\boldsymbol{U}}_\ell = \boldsymbol{0}$ when ℓ is odd. θ consists of all the weights and biases $\{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l=0}^L$. The number m and L are called the width and the depth of the network, respectively. The activation function σ is problem-dependent. For example, if the DNN as a test function is required to be continuously differentiable, the Tanh activation function can be chosen to guarantee that our DNN is in $\mathcal{C}^\infty$; if it is desired that $\phi(\boldsymbol{x};\theta)$ is in the H^k space for $k \in \mathbb{N}$, the activation function $\mathrm{ReLU}^{k+1}(x)$ could be used, where $\mathrm{ReLU}(x) := \max\{0, x\}$.

4.3. Unconstrained Minimax Problem. When the domain becomes relatively complex, the penalty method may be employed to solve the constrained minimax optimization in (4.1). For this purpose, we shall introduce a distance to quantify how good the solution DNN and test DNN satisfy their constraints. Such a distance is specified according to the boundary conditions. Denote by $\mathrm{dist}(\phi(\boldsymbol{x};\theta), V)$ the distance between a DNN $\phi(\boldsymbol{x};\theta)$ and a space V. Therefore, the penalty terms of boundary conditions can be written as

$$L_b(\theta_s, \theta_t) := \lambda_1 \mathrm{dist}(\phi_s(\boldsymbol{x};\theta_s), V) + \lambda_2 \mathrm{dist}(\phi_t(\boldsymbol{x};\theta_t), V^*), \tag{4.3}$$

where λ_1 and λ_2 are two positive hyper-parameters. Finally, the constraint minimax problem (4.1) can be formulated into the following unconstrained minimax problem

$$(\bar{\theta}_s, \bar{\theta}_t) = \arg\min_{\theta_s}\max_{\theta_t} \big(L(\theta_s, \theta_t) + L_b(\theta_s, \theta_t)\big), \tag{4.4}$$

which can be solved to obtain the solution DNN $\phi_s(\boldsymbol{x};\bar{\theta}_s)$ as the weak solution of the given PDE in (2.18) by Friedrichs Learning.

4.4. Special Networks for Different Boundary Conditions. As discussed in [26, 25], it is possible to build special networks to satisfy various boundary conditions automatically, which can simplify the unconstrained optimization (4.4) into

$$(\bar{\theta}_s, \bar{\theta}_t) = \arg\min_{\theta_s}\max_{\theta_t} L(\theta_s, \theta_t). \tag{4.5}$$

This optimization problem (4.5) is easier to solve compared to (4.4) since two hyperparamters λ_1 and λ_2 in (4.3) are dropped. Note that for a regular PDE domain, e.g., a hypercube or a ball, it is simple to construct such special networks satisfying various boundary conditions automatically.

Let us take the case of a homogeneous Dirichlet boundary condition as an example. For other cases, the readers are referred to [26, 25]. A DNN satisfying Dirichlet boundary condition $\psi(\boldsymbol{x}) = g(\boldsymbol{x})$ on $\partial\Omega$ can be constructed by $\phi(\boldsymbol{x};\theta) = h(\boldsymbol{x})\hat{\phi}(\boldsymbol{x};\theta) + b(\boldsymbol{x})$, where $\hat{\phi}$ is a generic network as in (4.2), and $h(\boldsymbol{x})$ is a specifically chosen function such that $h(\boldsymbol{x}) = 0$ on $\partial\Omega$, and $b(\boldsymbol{x})$ is chosen such that $b(\boldsymbol{x}) = g$ on $\partial\Omega$. For example, if Ω is a d-dimensional unit ball, then $\phi(\boldsymbol{x};\theta)$ can take the form $\phi(\boldsymbol{x};\theta) = (|\boldsymbol{x}|^2 - 1)\hat{\phi}(\boldsymbol{x};\theta) + b(\boldsymbol{x})$. For another example, if Ω is the d-dimensional hyper-cube $[-1, 1]^d$, then $\phi(\boldsymbol{x};\theta)$ can take the form $\phi(\boldsymbol{x};\theta) = \prod_{i=1}^d (x_i^2 - 1)\hat{\phi}(\boldsymbol{x};\theta) + b(\boldsymbol{x})$.

4.5. Network Training. Once the solution DNN and test DNN have been set up, the rest is to train them to solve the minimax problem in (4.4). The stochastic gradient descent (SGD) method or its variants (e.g., RMSprop [30] and Adam [43]) is an efficient tool to solve this problem numerically. Although the convergence of SGD for the minimax problem is still an active research topic [54, 12, 60], empirical success shows that SGD can provide a good approximate solution. The training algorithm and main numerical setup are summarized in Algorithm 1.

In Algorithm 1, the outer iteration loop takes n iterations. Each inner iteration loop contains n_s steps of θ_s updates and n_t steps of θ_t updates. In each inner iteration for updating θ_s, we generate two new sets of random samples $\{\boldsymbol{x}_i^1\}_{i=1}^N \subset \Omega$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_b} \subset \partial\Omega$ following uniform distributions. In most of the examples, the Latin Hyper-cube Sampling method is employed to generate random points in order to simulate the distributional characteristics even for the relatively small number of samples. We define the empirical loss of these training points for the Friedrichs' system (2.11) as

$$L_t(\theta_s, \theta_t) := \hat{L}(\theta_s, \theta_t) + \hat{L}_b(\theta_s, \theta_t), \tag{4.6}$$

where $\hat{L}(\theta_s, \theta_t) := \frac{|\hat{L}_n(\theta_s,\theta_t)|}{\hat{L}_d(\theta_s,\theta_t)}$ with

$$\begin{aligned}\hat{L}_n(\theta_s, \theta_t) \;:=\; & \frac{A(\Omega)}{N_1}\sum_{i=1}^{N_1}\big(\sum_{j=1}^{d}\frac{\partial}{\partial x_j}(-\boldsymbol{A}_j\phi_t(\boldsymbol{x}_i^1;\theta_t)), \phi_s(\boldsymbol{x}_i^1;\theta_s)\big) + \frac{A(\Omega)}{N_1}\sum_{i=1}^{N_1}\big(\boldsymbol{C}^\intercal\phi_t(\boldsymbol{x}_i^1;\theta_t), \phi_s(\boldsymbol{x}_i^1;\theta_s)\big) \\ & -\frac{A(\Omega)}{N_1}\sum_{i=1}^{N_1}\big(f(\boldsymbol{x}_i^1), \phi_t(\boldsymbol{x}_i^1;\theta_t)\big) + \frac{A(\partial\Omega)}{N_2}\sum_{i=1}^{N_2}\big((\sum_{j=1}^{d}\boldsymbol{A}_j n_j)\phi_s(\boldsymbol{x}_i^2;\theta_s), \phi_t(\boldsymbol{x}_i^2;\theta_t)\big),\end{aligned}$$

$$\hat{L}_d(\theta_s, \theta_t) := \frac{A(\Omega)}{N_1}\sum_{i=1}^{N_1}\|\sum_{j=1}^{d}\frac{\partial}{\partial x_j}(-\boldsymbol{A}_j\phi_t(\boldsymbol{x}_i^1;\theta_t)) + \boldsymbol{C}^\intercal\phi_t(\boldsymbol{x}_i^1;\theta_t)\|_2^2,$$

where $(\cdot,\cdot)$ denotes the inner product of two vectors, $\|\cdot\|_2$ denotes the 2-norm of vectors and $A(\cdot)$ is denoted as the area or volume of the integral region. As for the boundary loss, let us take the Dirichlet boundary condition $u(\boldsymbol{x}) = g_d(\boldsymbol{x})$ as an example. In this case, the boundary loss can be formulated as

$$\hat{L}_b(\theta_s, \theta_t) := \frac{A(\partial\Omega)}{N_2}\sum_{i=1}^{N_2}\|\phi_s(\boldsymbol{x}_i^2, \theta_s) - g_d(\boldsymbol{x}_i^2)\|_2^2.$$

As mentioned in Section 4.4, if the solution DNN and test DNN are built to satisfy their boundary conditions automatically, $\hat{L}_b(\theta_s, \theta_t)$ is zero.

Next, we compute the gradient of $L_t(\theta_s, \theta_t)$ with respect to θ_s, denoted by g_s, which is known as the gradient descent direction. The gradient is evaluated via the autograd in PyTorch, which is essentially equivalent to a sequence of chain rules to compute the gradient since the loss function is the composition of several simple functions with explicit formulas. Thus, θ_s can be updated by applying one step of gradient descent with a step size η_s as follows: $\theta_s \leftarrow \theta_s - \eta_s g_s$. In each outer iteration of Algorithm 1, we repeatedly sample new training points and update θ_s for n_s steps.

In each inner iteration, θ_t can be updated similarly to maximize the empirical loss $L_t(\theta_s, \theta_t)$. In each inner iteration for updating θ_t, we generate random samples and evaluate the gradient of the empirical loss with respect to θ_t, denoted by g_t. Then θ_t can be updated via one step of gradient ascent with a step size η_t as follows: $\theta_t \leftarrow \theta_t + \eta_t g_t$. In each outer iteration of Algorithm 1, we repeatedly sample new training points and update θ_t for n_t steps.

We would like to emphasize that minimax optimization problems are in general more challenging to solve than minimization problems arising in network-based PDE solvers in the strong form. Note

that, when the test DNN is fixed, the loss function in (4.1) is a convex functional of the solution DNN. Hence, the difficulty of the minimization problem when the test DNN is fixed is the same as the network-based least squares method. No matter what the test function is, the gradient descent update of θ_s can improve the solution DNN as long as the gradient is not zero and the step size is appropriate. For a fixed solution DNN, the maximization problem over the test DNN is not convex both in the parameter space and in the DNN space.

To further facilitate the convergence of Friedrichs learning, a restarting strategy is employed to obtain the restarted Friedrichs learning in Algorithm 1, which is in the same spirit of typical restarted iterative solvers in numerical linear algebra, e.g., the restarted GMRES [40], or the restart strategies in optimization [2, 28, 53, 32]. For simplicity and without loss of generality, the restarted Friedrichs learning is introduced for PDEs with Dirichlet boundary conditions. For other boundary conditions, the restarted Friedrichs learning can be designed similarly.

Algorithm 1 Friedrichs Learning for Weak Solutions of PDEs.

Require: The desired PDE.

Ensure: Parameters θ_t and θ_s solving the minimax problem in (4.4).

Set iteration parameters n, n_s, and n_t. Set sample size parameters N and N_b. Set step sizes $\eta_s^{(k)}$ and $\eta_t^{(k)}$ in the k-th outer iteration.
Initialize $\phi_s(\boldsymbol{x};\theta_s^{0,0})$ and $\phi_t(\boldsymbol{x};\theta_t^{0,0})$.
for $k = 1, \cdots, n$ **do**
  for $j = 1, \cdots, n_s$ **do**
    Generate uniformly distributed sample points $\{\boldsymbol{x}_i^1\}_{i=1}^N \subset \Omega$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_b} \subset \partial\Omega$.
    Compute the gradient of the loss function in (4.6) at the point $(\theta_s^{k-1,j-1}, \theta_t^{k-1,0})$ with respect to θ_s and denote it as $g(\theta_s^{k-1,j-1}, \theta_t^{k-1,0})$.
    Update $\theta_s^{k-1,j} \leftarrow \theta_s^{k-1,j-1} - \eta_s^{(k)} g(\theta_s^{k-1,j-1}, \theta_t^{k-1,0})$ with a step size $\eta_s^{(k)}$.
  end for
  $\theta_s^{k,0} \leftarrow \theta_s^{k-1,n_s}$.
  for $j = 1, \cdots, n_t$ **do**
    Generate uniformly distributed sample points $\{\boldsymbol{x}_i^1\}_{i=1}^N \subset \Omega$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_b} \subset \partial\Omega$.
    Compute the gradient of the loss function in (4.6) at $(\theta_s^{k,0}, \theta_t^{k-1,j-1})$ with respect to θ_t and denote it as $g(\theta_s^{k,0}, \theta_t^{k-1,j-1})$.
    Update $\theta_t^{k-1,j} \leftarrow \theta_t^{k-1,j-1} + \eta_t^{(k)} g(\theta_s^{k,0}, \theta_t^{k-1,j-1})$ with a step size $\eta_t^{(k)}$.
  end for
  $\theta_t^{k,0} \leftarrow \theta_t^{k-1,n_t}$.
  if Stopping criteria is satisfied **then**
    Return $\theta_s = \theta_s^{k,0}$ and $\theta_t = \theta_t^{k,0}$.
  end if
end for

5. Numerical Experiments. In this section, all hyperparameters are listed in Table 5.1. We set the solution DNN $\phi_s(\boldsymbol{x}, \theta_s)$ as a fully connected ResNet with ReLU activation functions, depth 7, and width m_s, where m_s is problem dependent. The activation of $\phi_s(\boldsymbol{x}, \theta_s)$ is chosen as ReLU due to its ability to approximate functions with low regularity. The test DNN $\phi_t(\boldsymbol{x}, \theta_t)$ has the same structure with depth 7 and width m_t. To ensure the smoothness of $\phi_t(\boldsymbol{x}, \theta_t)$, we employ the Tanh activation function. The optimizers for updating $\phi_s(\boldsymbol{x}, \theta_s)$ and $\phi_t(\boldsymbol{x}, \theta_t)$ are chosen as Adam and RMSprop, respectively. All of our experiments share the same setting for network structures and optimizers. At the pre-training phase we always set the learning rate to be larger

Notation	Meaning
d	the dimension of the problem
n_p	the number of pre-training iterations
n	the number of outer iterations
η_s^p	the pre-training learning rate for optimizing the solution network
η_t^p	the pre-training learning rate for optimizing the test network
$\eta_s^{(0)}$	the initial learning rate for optimizing the solution network
$\eta_t^{(0)}$	the initial learning rate for optimizing the test network
ν_s	the decaying rate for η_s
ν_t	the decaying rate for η_t
m_s	the width of each layer in the solution network
m_t	the width of each layer in the test network
n_s	the number of inner iterations for the solution network
n_t	the number of inner iterations for the test network
N	the number of training points inside the domain
N_b	the number of training points on the domain boundary

TABLE 5.1
Parameters in the model and algorithm.

than the following training phase. Thereafter, to ensure an effective and stable training process, the learning rate in the optimization is updated in an exponentially decaying scheme. More precisely, at the k-th iteration, we set the learning rate $\eta_s^{(k)} = \eta_s^{(0)}(\frac{1}{10})^{(k/\nu_s)}$ for the solution DNN, where $\eta_s^{(0)}$ is the initial learning rate and ν_s is the decaying rate. Similarly, we set $\eta_t^{(k)} = \eta_t^{(0)}(\frac{1}{10})^{(k/\nu_t)}$ for test DNN.

Throughout this section, special networks satisfying boundary conditions automatically are used to avoid tuning the parameters λ_1 and λ_2 in (4.3); the inner iteration numbers are set as $n_s = 1$ and $n_t = 1$. The values of other parameters listed in Table 5.1 will be specified later.

To measure the solution accuracy, the following discrete relative L^2 error at uniformly distributed test points in the domain is applied; i.e.,

$$e_{L^2}(\theta_s) := \left(\frac{\sum_i \|\phi_s(\boldsymbol{x}_i;\theta_s) - u^*(\boldsymbol{x}_i)\|_2^2}{\sum_i \|u^*(\boldsymbol{x}_i)\|_2^2} \right)^{\frac{1}{2}},$$

where u^* is the exact solution, $\|\cdot\|_2$ denotes the 2-norm of a vector. In the case when the true solution is continuous, the following discrete relative L^∞ error at uniformly distributed test points in the domain is also applied; i.e.,

$$e_{L^\infty}(\theta_s) := \frac{\max_i(\|\phi_s(\boldsymbol{x}_i;\theta_s) - u^*(\boldsymbol{x}_i)\|_\infty)}{\max_i(\|u^*(\boldsymbol{x}_i)\|_\infty)},$$

where $\|\cdot\|_\infty$ denotes the L^∞-norm of a vector. In most examples, we choose at least $10,000$ testing points for error evaluation. When the dimension is high or the value of function surges, we may choose $50,000$ or even $100,000$ testing points.

5.1. Advection-Reaction Equation with Plain Discontinuity. In the first example, we identify the weak solution in $L^2(\Omega)$ of the advection-reaction equation in (3.1) with discontinuous

solutions. Following Example 2 in [31], we choose the velocity $\boldsymbol{\beta} = (1, 9/10)^\intercal$ and $\mu = 1$ in the domain $\Omega = [-1, 1]^2$. We choose the right-hand-side function f and the boundary function g such that the exact solution is

$$u^*(x,y) = \begin{cases} \sin(\pi(x+1)^2/4)\sin(\pi(y-\frac{9}{10}x)/2) & \text{for } -1 \le x \le 1,\ \frac{9}{10}x < y \le 1, \\ e^{-5(x^2+(y-\frac{9}{10}x)^2)} & \text{for } -1 \le x \le 1,\ -1 \le y < \frac{9}{10}x. \end{cases} \tag{5.1}$$

The exact solution is visualized in Figure 5.1(b). The discontinuity of the initial value function will propagate along the characteristic line $y = 9x/10$. Hence, the derivative of the exact solution does not exist along that line. Classical network-based least square algorithms in the strong form will encounter a large residual error near the characteristic line and hence its accuracy may not be very attractive, which motivates our Friedrichs Learning in the weak form.

As discussed in [31], a priori knowledge of the characteristic line is crucial for conventional finite element methods with adaptive mesh to obtain high accuracy. In [31], the streamline diffusion method (SDFEM) can obtain a solution with $O(10^{-2})$ accuracy using $O(10^4)$ degrees of freedom when the mesh is aligned with the discontinuity, i.e., when the priori knowledge of the characteristic line is used in the mesh generation. The discontinuous Galerkin method (DGFEM) in [31] can obtain $O(10^{-8})$ accuracy under the same setting. When the mesh is not aligned with the discontinuity, e.g., when the characteristic line is not used in mesh generation, DGFEM converges as slow as SDFEM and the accuracy is not better than $O(10^{-2})$ with $O(10^4)$ degrees of freedom according to the discussion in [31].

As a deep learning algorithm, Friedrichs Learning is a mesh-free method and the weak solution can be identified without the priori knowledge of the characteristic line. By the discussion in Section 4.4, a special network $\phi_s(\boldsymbol{x}, \theta_s)$ is constructed as follows to fulfill the boundary condition of the solution:

$$\phi_s(\boldsymbol{x}, \theta_s) = \cos(-\frac{\pi}{4} + \frac{\pi}{4}x)\cos(-\frac{\pi}{4} + \frac{\pi}{4}y)\hat{\phi}_s(\boldsymbol{x}, \theta_s) + b(x,y), \tag{5.2}$$

where $b(x, y)$ is constructed directly from the boundary condition as

$$b(x,y) = \begin{cases} 0, & \text{for } -1 \le x \le 1,\ -0.4 + x/2 < y \le 1, \\ e^{-5[x^2+(-1-9/10x)^2]} + e^{-5[(-1)^2+(y+9/10)^2]} & \\ \quad -e^{-5[(-1)^2+(-1+9/10)^2]}, & \text{for } -1 \le x \le 1,\ -1 \le y \le -0.4 + x/2, \end{cases} \tag{5.3}$$

satisfying $b(x, y) = u(x, y)$ on the outflow boundary $\partial\Omega^-$. For test function, we fix its structure so that $\phi_t(\boldsymbol{x}, \theta_t) = 0$ on $\partial\Omega^+$ defined in (3.3).

First of all, pre-training the base function is employed. The special network structure satisfying the Dirichlet boundary conditions for solution DNN ϕ_s is constructed as

$$\phi_s(\boldsymbol{x}; \theta_s) = h(\boldsymbol{x})\hat{\phi}_s(\boldsymbol{x}; \theta_s) + b(\boldsymbol{x}), \tag{5.4}$$

where $b(\boldsymbol{x})$ satisfies the boundary condition which also can be regarded as an initial guess; $h(\boldsymbol{x}) = 0$ on the Dirichlet boundary. We observe that if $b(\boldsymbol{x})$ is closer to the true solution, it is easier to train a generic DNN $\hat{\phi}_s$ to obtain the solution DNN ϕ_s that approximates the true solution more accurately. Therefore, after a few rounds of outer iterations in the original Friedrichs learning, we obtain a rough solution DNN, which can be served as a better b function in (5.4) to construct a new solution DNN. After that, we will continue training to obtain a more accurate solution.

Secondly, we choose $b(x, y)$ to be discontinuous along a random line rather than the true discontinuous line of the exact solution. This could be a reasonable reproduction of the real application

Parameters	n	m_s	m_t	N	N_b
Value	$50,000$	pre-train 50, after 250	150	$90,000$	$45,000$
Parameters	$\eta_s^{(0)}$	$\eta_t^{(0)}$	ν_s	ν_t	parameter number
Value	3×10^{-4}	3×10^{-3}	$9,000$	$9,000$	$327,700$

TABLE 5.2
The parameters for the Friedrichs learning solver of the experiment in Section 5.1.

Parameters	n	m_s	N	$\eta_s^{(0)}$	ν_s
Value	$50,000$	250	$90,000$	1×10^{-3}	$10,000$

TABLE 5.3
The parameters of the comparative experiment in Section 5.1.

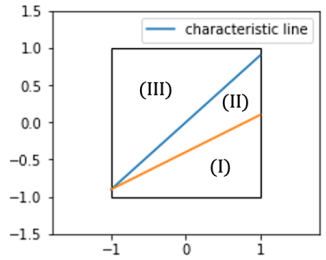
scenarios. Indeed, our choice of $b(x, y)$ above actually make the problem more challenging. The true solution is discontinuous along the characteristic line, the blue line in Figure 5.1(a), and $b(x, y)$ is discontinuous along the orange line in Figure 5.1(a). Hence, to make the solution DNN ϕ_s in (5.2) approximate the true solution well, one algorithm needs to find and correct these two lines automatically and the DNN $\hat{\phi}_s$ in (5.2) should be approximately discontinuous along these two lines. As shown by Figure 5.1(d), with Friedrichs learning the solution DNN ϕ_s has a configuration similar to the true solution in Figure 5.1(b), which means that it has successfully learned these two lines. This feature can be significant because no prior knowledge of the discontinuity of the exact solution is needed during the training, as long as the boundary condition is satisfied.

Thirdly, we can observe the mechanism of Friedrichs learning from Figure 5.1(e), where the test DNN ϕ_t surges and has a larger magnitude near these two lines to emphasize the error of the solution DNN ϕ_s. It can make the update of the configuration of ϕ_s more focused on these two lines than other places, which in turns facilitates the expected convergence of the solution DNN.

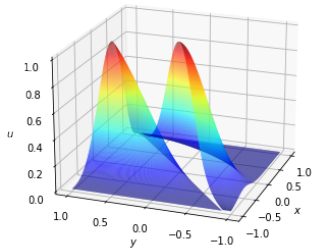
The whole training process can be divided into two phases. In Phase I of pre-training, we train a ResNet of width 50 for $1,000$ outer iterations to get a rough solution with an L^2 relative error $2.76e-1$. All other parameters are shown in Table 5.2. As shown in Figure 5.1(c), the rough solution has already captured basically the shape of the solution. In Phase II of training, we set this rough solution as base function $b(\boldsymbol{x})$ and again set up a ResNet with width 150. It is shown that $50,000$ outer iterations are enough to make the L^2 error of the solution DNN decrease to $2.27e-2$, as shown in Figure 5.1(d) and Figure 5.1(f). Our method is comparable with the SDFEM in [31] considering the same order of degrees of freedom summarized in Table 5.2. However, SDFEM in [31] requires the prior knowledge of the characteristic line while our method does not. Therefore, from the perspective of practical computation, our method would be more convenient in real applications.

To compare Friedrichs Learning and the DNN-based least square (LS) algorithm [13, 44, 55], we conduct comparative experiments with very similar hyper-parameters shown in Table 5.3. After $50,000$ iterations we obtain a solution with the relative error in L^2 norm which is $3.29e-2$. Though Friedrichs learning is more accurate, the DNN-based least square algorithm and the Friedrichs learning have errors of the same order in this numerical test.

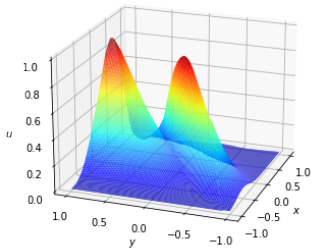
5.2. Advection-Reaction Equation with Curved Discontinuity. Consider a domain $\Omega = \{(x, y)|x^2 + y^2 \leq 1, y \geq 0\}$. The velocity $\beta = (\sin\theta, -\cos\theta)^\intercal = (y/\sqrt{x^2+y^2}, -x/\sqrt{x^2+y^2})$ with θ being the polar angle and $\mu = 0$. The Dirichlet boundary condition on the inflow boundary



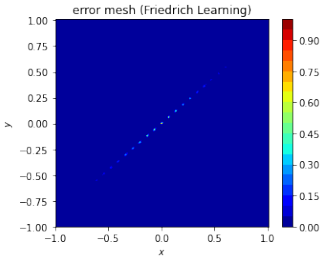
(a) The characteristic line (blue) of the exact solution and the line (orange) along which $b(x, y)$ in (5.2) is discontinuous.



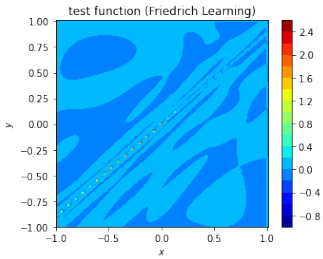
(b) Exact solution.



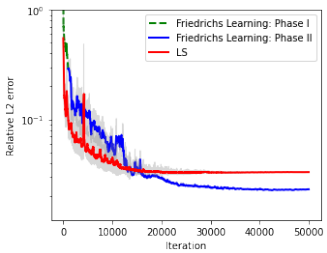
(c) The solution DNN right before restarting.



(d) The point-wise error of approximate solution at epoch $50,000$ by $50,000$. Friedrichs learning.



(e) The test DNN value at epoch by DNN-based least square and Friedrichs learning



(f) The relative L^2 error curve

FIG. 5.1. *Numerical results of Equation* (3.1) *when the exact solution is chosen as* (5.1).

is given as $u(x, 0) = 1$ for $-1 \le x \le -\frac{1}{2}$, $u(x, 0) = 0$ for $-\frac{1}{2} < x \le 0$. The true solution is

$$u^*(x, y) = \begin{cases} 0, & x^2 + y^2 < 1/4 \\ 1, & x^2 + y^2 \ge 1/4 \end{cases}. \tag{5.5}$$

Again, without the prior knowledge of the characteristic line, to create a network satisfying the boundary condition, we choose a solution DNN ϕ_s as

$$\phi_s(\boldsymbol{x}, \theta_s) = \big(\frac{\pi}{2} - \arctan(\frac{-x}{y})\big) \sin(\frac{\pi}{2} r) \hat{\phi}_s(\boldsymbol{x}, \theta_s) + b(x, y), \tag{5.6}$$

where

$$b(x, y) = \begin{cases} 0, & x \ge -1/2 \\ 1, & x < -1/2 \end{cases}, \quad \text{and} \quad r = \sqrt{x^2 + y^2}.$$

ϕ_s will be applied as the solution network of Friedrichs learning. Similarly,

$$\phi_t(\boldsymbol{x}, \theta_t) = \big(-\frac{\pi}{2} - \arctan(\frac{-x}{y})\big) \hat{\phi}_t(\boldsymbol{x}, \theta_t). \tag{5.7}$$

By applying Friedrichs learning with ϕ_s and ϕ_t as the solution and test DNN, respectively, we get an approximate solution with an L^2 relative error 2.48e−2 with the iteration error visualized in Figure 5.3(b). Figure 5.2(a) shows the point-wise error after 100,000 iterations by Friedrichs learning. Friedrichs learning can capture the discontinuous locations well with a sharp characterization. The test function value is relatively large around the discontinuous location, resulting in a large weight for samples around there, which can help to obtain a more accurate PDE solution.

Parameters	n	m_s	m_t	N	N_b
Value	$100,000$	150	150	$45,000$	$5,000$
Parameters	$\eta_s^{(0)}$	$\eta_t^{(0)}$	ν_s	ν_t	parameter number
Value	3×10^{-4}	3×10^{-3}	$15,000$	$15,000$	113,850

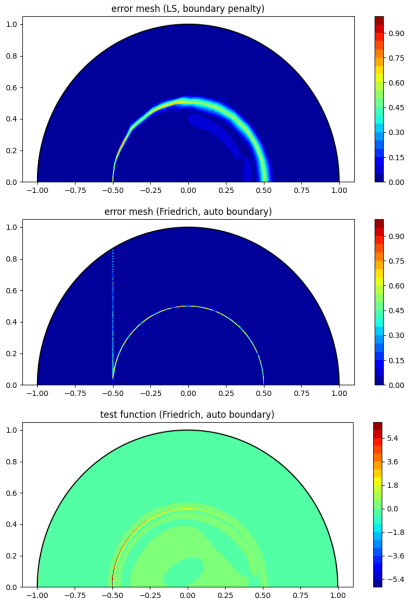
TABLE 5.4
The parameters for the Friedrichs learning solver of the experiment in Section 5.2.

As a comparison with traditional PDE solvers, note that the same PDE was solved by the adaptive least-squares finite element method (LSFEM) in [49] with the same order of degrees of freedom ($\approx 1.1 \times 10^5$) as in Friedrichs learning. The L^2 relative error of LSFEM is 4.59e−2, which is larger than the one by Friedrichs learning. We would like to emphasize that LSFEM in [49] has applied extra computational resources to adaptively generate discretization mesh, without which the error would be poorer. Besides, the DGFEM[1] with adaptive mesh is also applied to solve the same PDE with the same order of degrees of freedom (107,332) as in Friedrichs learning. The L^2 relative error of DGFEM is 2.05e−2, which is very similar to the error by Friedrichs learning. Following the idea in [49] to visualize the solution, we project the approximate solutions by DGFEM and Friedrichs learning to the radius axis in Figure 5.3(b) and plot the scatters corresponding to the angle θ ranging from 0 to π. This visualization makes it easier to compare the solutions near the discontinuous location. It is easy to see that the solution by DGFEM has a larger error than the one by Friedrichs learning near the discontinuous location.

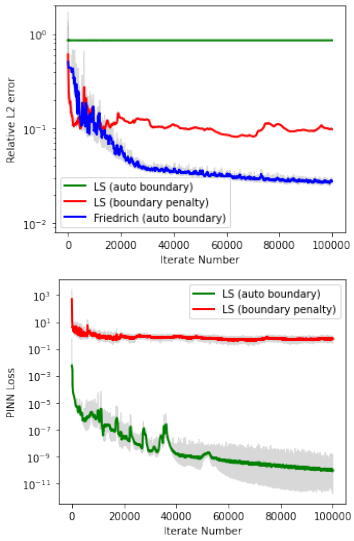
DNN-based least square is also applied to solve the same problem as comparison. Two options of DNN-based least square are tested: one with ϕ_s as the solution network so that there is no penalty terms to enforce the boundary condition in the loss function; another one with a standard neural network as the solution network and, hence, a penalty term in the loss function is required to enforce the boundary condition. The first option, i.e., DNN-based least square with the special network structure described in (5.6) to parametrize the PDE solution, fails to find a reasonable solution even though the optimization loss is almost zero as shown by Figure 5.2(b). One possible reason is due to the fact that the square loss in the strong form is 0 for $b(x, y)$, since DNN-based least square samples points randomly in the "interior" but not on the discontinuous line with probability almost 1. Therefore, even if the generic network $\hat{\phi}_s(\boldsymbol{x}, \theta_s)$ is not 0 at the beginning, no information of the discontinuity is captured by the strong form in DNN-based least square and, hence, the solution network will converge to 0, resulting in a fake solution satisfying the equation almost everywhere in the strong sense. However, this solution is mathematically wrong in the weak sense. For instance, the derivatives across the discontinuity contain Diracs delta functions.

The second option of DNN-based least square can provide a meaningful solution and serves as a good baseline for Friedrichs learning. Figure 5.2(a) shows the point-wise error after 100,000 iterations by DNN-based least square with a boundary penalty term and Friedrichs learning. Friedrichs learning can capture the location of discontinuous line with better accuracy than DNN-based least square. The error curve of DNN-based least square in the L^2 norm is shown in 5.2(b) (the red line) and the iteration error cannot be improved anymore at the early beginning. DNN-based least square with a boundary penalty term provides a solution with an L^2 error 9.35e−2 after $100,000$ iterations and this error is almost 4 times of the error by Friedrichs learning.

[1] Available at `https://github.com/dealii/dealii`.

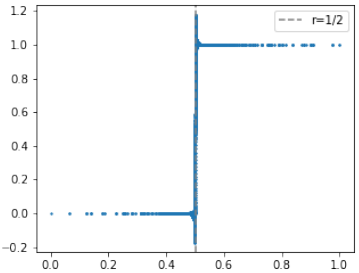


(a) Top, the point-wise error for solution by DNN-based least square; Middle, the point-wise error for solution by Friedrichs learning; Bottom, the point-wise test function value by Friedrichs learning.

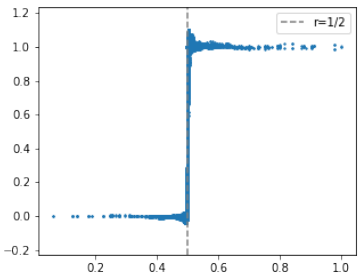


(b) Upper, the relative L^2 error curve with respect to the iteration number for three different algorithm setting; Lower, the running DNN-based least square loss with respect to the iteration number.

FIG. 5.2. *Numerical results of Equation* (3.1) *when the exact solution is chosen as* (5.5).



(a) Projected solution of DGFEM with adaptive mesh grid.



(b) Projected solution of Friedrichs learning. The value of points 0.005 Euclid distance away from $x = -\frac{1}{2}$ is adjusted to true value.

FIG. 5.3. *Numerical results of Equation* (3.1) *when the exact solution is chosen as* (5.5).

5.3. Green's Function. The next example is to identify the Green's function of the Laplacian operator by solving

$$\Delta u(\boldsymbol{x}) = \delta_0(\boldsymbol{x}), \tag{5.8}$$

where $\delta_0(\boldsymbol{x})$ is the Dirac's delta function at the origin. In this example, we solve the above equation on a 3D unit ball $\Omega = \{\boldsymbol{x} \in \mathbb{R}^3 || \boldsymbol{x}| \leq 1\}$, where $|\cdot|$ is the standard L^2 norm of a vector. The true solution is

$$u^*(\boldsymbol{x}) = \frac{1}{8\pi|\boldsymbol{x}|}, \tag{5.9}$$

Parameters	n	N	$\eta_s^{(0)}$	ν_s	m_s
Value	$100,000$	$45,000$	1×10^{-3}	$15,000$	150

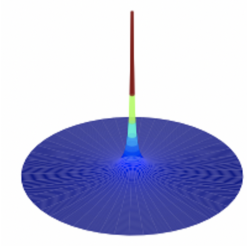
TABLE 5.5
The parameters of the comparative experiment in Section 5.2.

Parameters	n	m_s	m_t	N	N_b	η_s^p
Value	$20,000$	100	100	$45,000$	$5,000$	1×10^{-4}
Parameters	η_t^p	$\eta_s^{(0)}$	$\eta_t^{(0)}$	ν_s	ν_t	parameter number
Value	2×10^{-4}	1×10^{-5}	2×10^{-5}	$10,000$	$10,000$	$51,000$

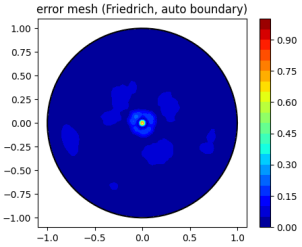
TABLE 5.6
The parameters for the Friedrichs learning solver of the experiment in Section 5.3.

and the given Dirichlet boundary condition is $u(\boldsymbol{x}) = \frac{1}{8\pi}$ on $\partial\Omega$. Although the exact solution is in H^1 and has strong singularity near the origin, Friedrichs learning can provide an approximate solution with a small error as shown in Figure 5.4(a) and 5.4(b). Figure 5.4(b) visualizes the point-wise relative error of the solution by Friedrichs learning. We can see that, except for those locations that are very close to the origin, the relative errors are not greater than 1e−1. In Table 5.7, we summarize the relative L^2 errors of the solution by Friedrichs learning in the region of $\Omega\backslash\mathcal{B}(\mathbf{0},\varepsilon)$ with ε equal to $0.001, 0.01, 0.1, 0.2$, respectively. Therefore, the solution is accurate when the location is not very close to the origin.

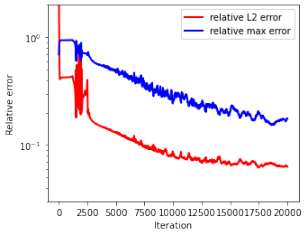
As a comparison, the DNN-based least square method cannot find a meaningful solution for the Green's function. The right hand side function of (5.8) is a Dirac Delta function and, hence, cannot be captured by the discrete analog of the least square loss function via random sampling. Therefore, even if the DNN-based least square method can be applied to form an optimization problem, the minimizer of this problem will return a constant function as a solution, which has a large error.



(a) The cross section of the Green's function at $x_3 = 0$. The Green's function has strong singularity near the origin.



(b) The projected point-wise relative error by Friedrichs learning on the slice $x_3 = 0$.



(c) The relative L^2 and maximum error curve with respect to the iteration number.

FIG. 5.4. *Numerical results of Equation* (3.1) *when the exact solution is chosen as* (5.9).

5.4. High-Dimensional Advection-Reaction Equation. We consider a 10D advection equation with discontinuity in the domain $[0,1]^{10}$. In particular, we find $u = u(\boldsymbol{x})$ such that

$$2\big(1+\exp(-(\sum_{i=3}^{10} x_i)^2)\big)u_{x_1} + \exp(2x_1)u_{x_2} = 0, \tag{5.10}$$

ε	mean
0.2	3.47e−2
0.1	4.43e−2
0.01	8.16e−2
0.001	9.39e−2

TABLE 5.7
The relative L^2 errors by Friedrichs learning in the region of $\Omega\backslash\mathcal{B}(\mathbf{0},\varepsilon)$ for the Green's function experiment in Section 5.3.

Parameters	n	m_s	m_t	N	N_b	η_s^p
Value	$50,000$	150	150	$45,000$	$5,000$	3×10^{-4}
Parameters	η_t^p	$\eta_s^{(0)}$	$\eta_t^{(0)}$	ν_s	ν_t	parameter number
Value	3×10^{-3}	5×10^{-5}	5×10^{-4}	$20,000$	$20,000$	$115,050$

TABLE 5.8
The parameters for the Friedrichs learning solver of the experiment in Section 5.4.

where $u_{x_1}=\frac{\partial u}{\partial x_1}$ and $u_{x_2}=\frac{\partial u}{\partial x_2}$. The exact solution is

$$u^*(\boldsymbol{x})=g\Big(\exp(2x_1)-4\big(1+\exp(-(\sum_{i=3}^{10}x_i)^2)\big)x_2\Big), \tag{5.11}$$

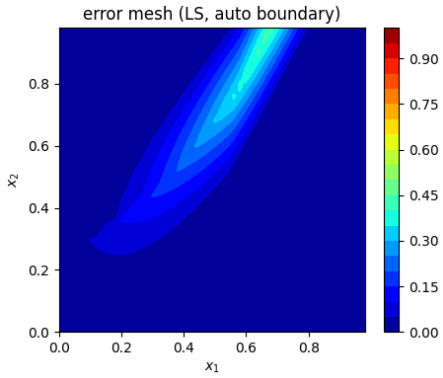
where

$$g(x)=\begin{cases}1, & x>0\\ 0, & x\le 0\end{cases}.$$

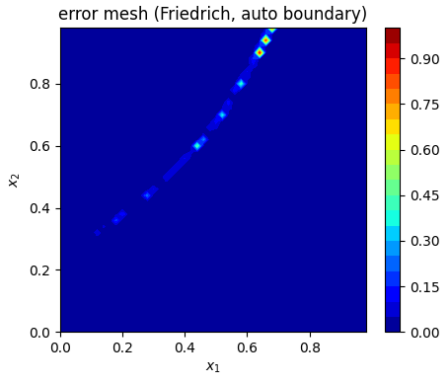
The Dirichlet boundary condition is given on the inflow boundary $\{\boldsymbol{x}|x_1=0 \text{ or } x_2=0\}$.

Figure 5.5(a) and Figure 5.5(c) show that Friedrichs learning can identify the location of low regularization by test DNNs in this high-dimensional problem. After 50,000 outer iterations, we obtain an approximate solution with a relative L^2 error 4.034e−2. As a comparison, the DNN-based least square is also applied to solve the same problem and the relative L^2 error is 1.015e−1, which is much larger than the one by Friedrichs learning. In Figure 5.5(d), we observe that DNN-based least square is not stable in optimization due to the curved discontinuity, and stops ultimately at a solution with a large error.

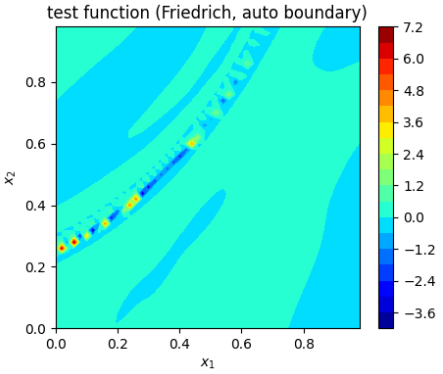
5.5. Maxwell Equations. In the last example, we consider Maxwell equations (3.8) defined in the domain $\Omega=[0,\pi]^3$. Let $\boldsymbol{H}$ and $\boldsymbol{E}$ be the solutions of the Maxwell equations (3.8) with $\mu=\sigma=1$. Let $\boldsymbol{f},\boldsymbol{g}\in[L^2(\Omega)]^3$ be $\boldsymbol{f}=(0,0,0)^\intercal$ and $\boldsymbol{g}=(3\sin y\sin z,3\sin z\sin x,3\sin x\sin y)^\intercal$. The boundary condition is set as $\boldsymbol{E}\times\boldsymbol{n}=0$, which is an ideal conductor boundary condition. The exact solutions to these equations are $\boldsymbol{H}^*=(\sin x(\cos z-\cos y),\sin y(\cos x-\cos z),\sin x(\cos y-\cos x))^\intercal$ and $\boldsymbol{E}^*=(\sin y\sin z,\sin z\sin x,\sin x\sin y)^\intercal$. Considering test functions $(\varphi_{\boldsymbol{H}}^\intercal,\varphi_{\boldsymbol{E}}^\intercal)^\intercal$ in the space $V^*=V$ mentioned in (3.4), we set up DNNs to satisfy the boundary conditions $\varphi_{\boldsymbol{E}}\cdot\boldsymbol{n}=0$ and $\varphi_{\boldsymbol{H}}\times\boldsymbol{n}=0$, where $\boldsymbol{n}$ is the unit outward normal direction to the boundary. Note that the domain is a cube, the normal vector is parallel to one of the unit vectors. The boundary condition above is indeed a Dirichlet boundary. For example, $S_1=\{x=\pi\}\cap\partial\Omega$ on the right surface, implies that $E_2|_{S_1}=E_3|_{S_1}=0$. It is worth pointing out that the Dirichlet boundary for $(E_i^\intercal,(\varphi_{\boldsymbol{H}})_i)^\intercal$ closes the faces of the cube as shown in Figure 5.6(a). Here, we denote by $E_i(i=1,2,3)$ the i-th component of the vector $\boldsymbol{E}$ and the same applies to other notations.



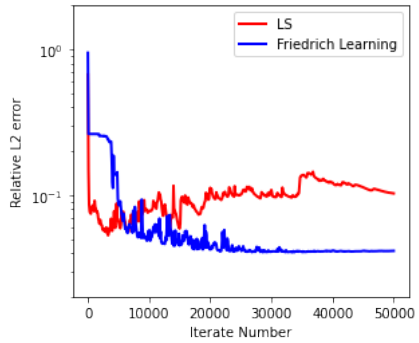
(a) The projected point-wise error by Friedrichs learning on the slice $x_i = \frac{1}{2}, i = 3, 4, \ldots, 10$.



(b) The projected point-wise error by DNN-based least square on the slice $x_i = \frac{1}{2}, i = 3, 4, \ldots, 10$.



(c) The projected point-wise test function value on the slice $x_i = \frac{1}{2}, i = 3, 4, \ldots, 10$.



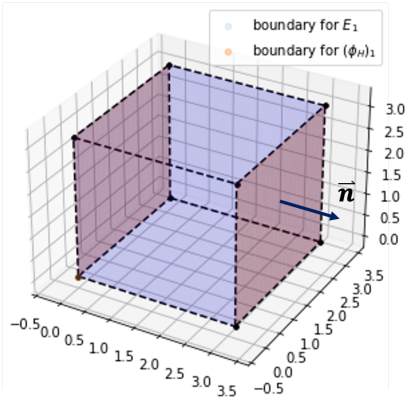
(d) The relative L^2 error curve with respect to the iteration number by DNN-based least square and Friedrichs learning.

FIG. 5.5. *Numerical results of Equation* (3.1) *when the exact solution is chosen as* (5.11).

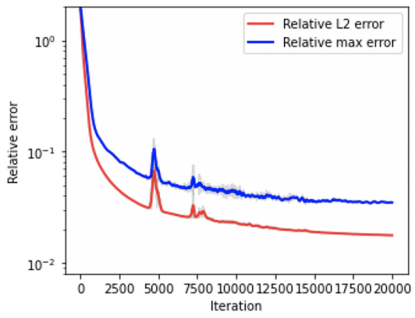
Parameters	n	N	$\eta_s^{(0)}$	ν_s	m_s
Value	$50,000$	$45,000$	1×10^{-3}	$20,000$	150

TABLE 5.9
The parameters of the comparative experiment in Section 5.4.

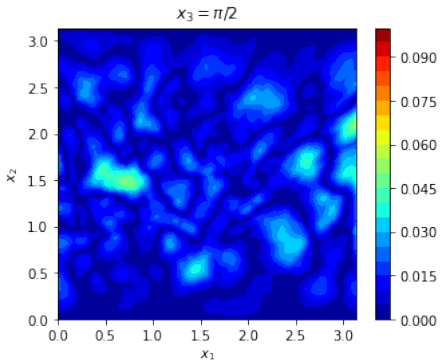
To solve the Maxwell equations by Friedrichs learning, we initialize sub-networks with width m_s for vector functions and each sub-network decides one output value of the vector function. The other test networks are set up similarly. We list all the parameters used in this experiment in Table 5.10. After $20,000$ outer iterations, we obtain an L^2 relative error $1.766e-2$ and an L^∞ relative error $3.467e-2$. Figure 5.6(c) and Figure 5.6(d) illustrate the absolute difference between E_1 and $(\phi_{\boldsymbol{E}})_1$ and the absolute difference between H_1 and $(\phi_{\boldsymbol{H}})_1$ after $20,000$ outer iterations.



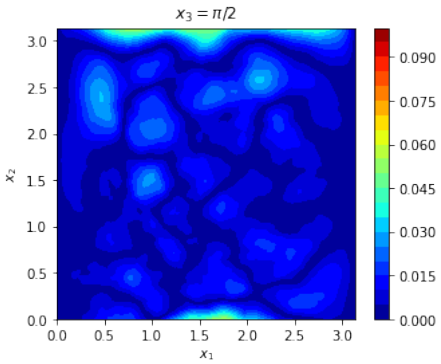
(a) The boundary conditions of $(E_1, (\phi_{\boldsymbol{H}})_1)$.



(b) The relative error versus iterations.



(c) The absolute difference between E_1 and $(\phi_{\boldsymbol{E}})_1$ after $20,000$ outer iterations.



(d) The absolute difference between H_1 and $(\phi_{\boldsymbol{H}})_1$ after $20,000$ outer iterations.

FIG. 5.6. *Numerical results of Maxwell equations in* (3.8).

Parameters	n	m_s	m_t	N
Value	$20,000$	250	50	$50,000$
Parameters	$\eta_s^{(0)}$	$\eta_t^{(0)}$	ν_s	ν_t
Value	3×10^{-6}	3×10^{-3}	$8,000$	$15,000$

TABLE 5.10
The parameters for Friedrichs learning solver of the experiment in Section 5.5.

6. Conclusion. Friedrichs learning was proposed as a new deep learning methodology to learn the weak solutions of PDEs via Friedrichs seminal minimax formulation. Extensive numerical results imply that our mesh-free method provides reasonably accurate solutions for a wide range of PDEs defined on regular and irregular domains in various dimensions, where classical numerical methods may be difficult to be employed. In particular, Friedrichs learning infers the solution without the knowledge of the location of discontinuity when the solution is discontinuous. Our numerical experiments show that Friedrichs learning can solve PDEs with a discontinuous solution to $O(10^{-2})$ accuracy, while the DNN-based least square method can typically only get $O(10^{-1})$ accuracy. This demonstrate the advantage of the loss function in Friedrichs learning over the naive least square loss function. Compared with traditional FEM methods, Friedrichs learning performs as well as DGFEM with adaptive mesh when no prior knowledge about the discontinuous location

is known. Friedrichs learning is better than LSFEM with adaptive mesh when no prior knowledge about the discontinuous location is known. In the future, it is interesting to develop adaptive Friedrichs learning to further reduce the error or the network size.

Acknowledgements. The work of J. H. was partially supported by NSFC (Grant No. 12071289) and the National Key Research and Development Project (2020YFA0709800). C. W. was partially supported by National Science Foundation Award DMS-2136380. H. Y. was partially supported by the US National Science Foundation under award DMS-1945029.

REFERENCES

[1] J.-L. Guermond A. Ern and G. Caplain. An intrinsic criterion for the bijectivity of hilbert operators related to friedrichs systems. *Comm. Partial Differential Equations*, 32(05):317341, 2007.

[2] Abdullah Al-Dujaili, Shashank Srikant, Erik Hemberg, and Una-May OReilly. On the application of danskins theorem to derivative-free minimax problems. *AIP Conference Proceedings*, 2070(1):020026, 2019.

[3] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[4] N. Antonic and K. Burazin. Graph spaces of first-order linear partial differential operators. *Math. Commun.*, 14:135155, 2009.

[5] Nenad Antoni and Kreimir Burazin. Intrinsic boundary conditions for friedrichs systems. *Communications in Partial Differential Equations*, 35(9):1690–1715, 2010.

[6] Gang Bao, Xiaojing Ye, Yaohua Zang, and Haomin Zhou. Numerical solution of inverse problems by weak adversarial networks. *Inverse Problems*, 36(11):115003, nov 2020.

[7] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[8] Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *arXiv e-prints*, arXiv:1907.03452, Jul 2019.

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

[10] Tan Bui-Thanh, Leszek Demkowicz, and Omar Ghattas. A Unified Discontinuous PetrovGalerkin Method and its Analysis for Friedrichs Systems. *SIAM J. NUMER. ANAL.*, page 19331958, 2013.

[11] Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing*, 42(5):A3285–A3312, 2020.

[12] Constantinos Daskalakis and Ioannis Panageas. The limit points of (optimistic) gradient descent in min-max optimization. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9256–9266, USA, 2018. Curran Associates Inc.

[13] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[14] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.

[15] Weinan E, Chao Ma, and Lei Wu. Barron Spaces and the Compositional Function Spaces for Neural Network Models. *arXiv e-prints*, arXiv:1906.08039, Jun 2019.

[16] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1–12, 2018.

[17] Matthias Ehrhardt and Ronald E. Mickens. A fast, stable and accurate numerical method for the blackscholes equation of american options. *International Journal of Theoretical and Applied Finance*, 11(05):471–501, 2008.

[18] A. Ern and J. L. Guermond. Discontinuous Galerkin methods for Friedrichs systems. Part I. General theory. *SIAM J. Numer. Anal.*, page 753778, 2006.

[19] A. Ern and J. L. Guermond. Discontinuous Galerkin methods for Friedrichs systems. Part II. Second-order elliptic PDEs. *SIAM J. Numer. Anal.*, page 23632388, 2006.

[20] A. Ern and J. L. Guermond. Discontinuous Galerkin methods for Friedrichs systems. Part III. Multifield theories with partial coercivity. *SIAM J. Numer. Anal.*, page 776 804, 2008.

[21] Alexandre Ern, Jean-Luc Guermond, and Gilbert Caplain. An intrinsic criterion for the bijectivity of hilbert operators related to friedrich' systems. *Communications in Partial Differential Equations*, 32(2):317–341, 2007.

[22] K. O. Friedrichs. Symmetric positive linear differential equations. *Communications on Pure and Applied Mathematics*, 11(3):333–418, 1958.

[23] Abhijeet Gaikwad and Ioane Muni Toke. Gpu based sparse grid technique for solving multidimensional options pricing pdes. In *Proceedings of the 2Nd Workshop on High Performance Computational Finance*, WHPCF '09, pages 6:1–6:9, New York, NY, USA, 2009. ACM.

[24] D. Gobovic and M. E. Zaghloul. Analog cellular neural network with application to partial differential equations with variable mesh-size. In *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, volume 6, pages 359–362 vol.6, May 1994.

[25] Yiqi Gu, Chunmei Wang, and Haizhao Yang. Structure probing neural network deflation, 2020.

[26] Yiqi Gu, Haizhao Yang, and Chao Zhou. Selectnet: Self-paced learning for high-dimensional partial differential equations, 2020.

[27] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[28] Kenta Hanada, Takayuki Wada, and Yasumasa Fujisaki. *A Restart Strategy with Time Delay in Distributed Minimax Optimization*, pages 89–100.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[30] Geoffrey Hinton. Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent., 2012.

[31] Paul Houston, Christoph Schwab, and Endre Sli. Stabilized hp-finite element methods for first-order hyperbolic problems. *SIAM Journal on Numerical Analysis*, 37(5):1618–1643, 2000.

[32] X. Hu, R. Shonkwiler, and M. Spruill. Random restarts in global optimization. 2009.

[33] Jianguo Huang, Haoqin Wang, and Haizhao Yang. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, 2020.

[34] M. Hutzenthaler, A. Jentzen, Th. Kruse, and T. A. Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. Technical report, 2020.

[35] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *arXiv e-prints*, arXiv:1901.10854, Jan 2019.

[36] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, Tuan Anh Nguyen, and Philippe von Wurstemberger. Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. *arXiv e-prints*, arXiv:1807.01212, Jul 2018.

[37] Martin Hutzenthaler, Arnulf Jentzen, and von Wurstemberger Wurstemberger. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *Electron. J. Probab.*, 25:73 pp., 2020.

[38] 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.

[39] M. Jensen. Discontinuous Galerkin Methods for Friedrichs Systems with Irregular Solutions. *Ph.D. thesis, University of Oxford, Oxford*, 2004.

[40] Wayne Joubert. On the convergence behavior of the restarted gmres algorithm for solving nonsymmetric linear systems. *Numerical Linear Algebra with Applications*, 1(5):427–447, 1994.

[41] Ehsan Kharazmi, Zhongqiang Zhang, and George E.M. Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[42] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, page 115, 2020.

[43] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, 2014.

[44] 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, Sep. 1998.

[45] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110 – 131, 1990.

[46] T.T. Lee, F.Y. Wang, and R.B. Newell. Robust model-order reduction of complex biological processes. *Journal of Process Control*, 12(7):807 – 821, 2002.

[47] Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3M: A deep domain decomposition method for partial differential equations. *arXiv e-prints*, arXiv:1909.12236, Sep 2019.

[48] Qianxiao Li, Bo Lin, and Weiqing Ren. Computing committor functions for the study of rare events using deep learning. *The Journal of Chemical Physics*, 151(5):054112, 2019.

[49] Q. Liu and S. Zhang. Adaptive least-squares finite element methods for lin-ear transport equations based on an H(div) flux reformulation. *Computer Methods in Applied Mechanics and Engineering*, 366:113041, 2020.

[50] Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains, 2020.

[51] Hadrien Montanelli and Haizhao Yang. Error bounds for deep ReLU networks using the Kolmogorov–Arnold superposition theorem. *arXiv e-prints*, arXiv:1906.11945, Jun 2019.

[52] Hadrien Montanelli, Haizhao Yang, and Qiang Du. Deep ReLU networks overcome the curse of dimensionality for bandlimited functions. *arXiv e-prints*, arXiv:1903.00735, Mar 2019.

[53] Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D. Lee, and Meisam Razaviyayn. Solving a class of non-convex min-max games using iterative first order methods. pages 14905–14916, 2019.

[54] Hassan Rafique, Mingrui Liu, Qihang Lin, and Tianbao Yang. Non-convex min-max optimization: Provable algorithms and applications in machine learning. *ArXiv*, abs/1810.02060, 2018.

[55] M. Raissi, P. Perdikaris, and G.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.

[56] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network with approximation error being reciprocal of width to power of square root of depth. *arXiv:2006.12231*, 2020.

[57] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. *arxiv:2010.14075*, 2020.

[58] Jonathan W. Siegel and Jinchao Xu. Approximation rates for neural networks with general activation functions. *Neural Networks*, 128:313 – 321, 2020.

[59] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[60] Christopher Srinivasa, Inmar Givoni, Siamak Ravanbakhsh, and Brendan J Frey. Min-max propagation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5565–5573. Curran Associates, Inc., 2017.

[61] David J. Wales and Jonathan P. K. Doye. Stationary points and dynamics in high-dimensional systems. *The Journal of Chemical Physics*, 119(23):12409–12416, 2003.

[62] H. Yserentant. Sparse grid spaces for the numerical solution of the electronic schrödinger equation. *Numerische Mathematik*, 101(2):381–389, Aug 2005.

[63] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak Adversarial Networks for High-dimensional Partial Differential Equations. *arXiv e-prints*, arXiv:1907.08272, Jul 2019.