

Deep learning-based schemes for singularly perturbed convection-diffusion problems

Adrien Beguinet, Virginie Ehrlacher, Roberta Flenghi, Maria Fuente-Ruiz, Olga Mula, Agustin Somacal

▶ **To cite this version:**

Adrien Beguinet, Virginie Ehrlacher, Roberta Flenghi, Maria Fuente-Ruiz, Olga Mula, et al.. Deep learning-based schemes for singularly perturbed convection-diffusion problems. 2022. hal-03664049

HAL Id: hal-03664049
https://inria.hal.science/hal-03664049

Preprint submitted on 10 May 2022

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

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

ESAIM: PROCEEDINGS AND SURVEYS, Vol. ?, 2022, 1-10
Editors: Will be set by the publisher

DEEP LEARNING-BASED SCHEMES FOR SINGULARLY PERTURBED CONVECTION-DIFFUSION PROBLEMS *

A. Beguinet[1], V. Ehrlacher[2], R. Flenghi[3], M. Fuente[4], O. Mula[5] and A. Somacal[6]

Abstract. Deep learning-based numerical schemes such as Physically Informed Neural Networks (PINNs) have recently emerged as an alternative to classical numerical schemes for solving Partial Differential Equations (PDEs). They are very appealing at first sight because implementing vanilla versions of PINNs based on strong residual forms is easy, and neural networks offer very high approximation capabilities. However, when the PDE solutions are low regular, an expert insight is required to build deep learning formulations that do not incur in variational crimes. Optimization solvers are also significantly challenged, and can potentially spoil the final quality of the approximated solution due to the convergence to bad local minima, and bad generalization capabilities. In this paper, we present an exhaustive numerical study of the merits and limitations of these schemes when solutions exhibit low-regularity, and compare performance with respect to more benign cases when solutions are very smooth. As a support for our study, we consider singularly perturbed convection-diffusion problems where the regularity of solutions typically degrades as certain multiscale parameters go to zero.

1. Introduction

1.1. Scientific context and goals

Singularly perturbed differential equations are typically characterized by a small parameter $\varepsilon > 0$ multiplying some of the highest order terms in the differential equation. In general, the solutions to such equations exhibit multiscale phenomena, and this raises significant challenges to classical numerical methods such as finite elements or finite volumes. To build accurate and robust approximations with these methods as ε decreases, it is necessary to develop elaborate numerical discretizations. In addition to the mathematical difficulties of the formulation, the resulting numerical schemes are often not entirely trivial to implement: they often require mesh adaptation, and working on complicated geometries is challenging. These difficulties motivate the search for new discretization schemes, hopefully mesh-free, with potential to deliver good quality approximations with easier implementation

* *This project was partly funded by the Emergence project grant of the Paris City Council "Models and Measures". Funding from I-Site FUTURE is also acknowledged. This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme (Grant Agreement no 810367).*

[1] Paris-Saclay University.
[2] Ecole des Ponts ParisTech & MATHERIALS INRIA team-project, Marne-la-Vallée, France
[3] Cermics, École des Ponts, INRIA, Marne-la-Vallée, France
[4] COMMEDIA, INRIA Paris, France & LJLL, Sorbonne Université.
[5] Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB, Eindhoven, Netherlands.
[6] LJLL, Sorbonne Université, Paris, France

© EDP Sciences, SMAI 2022

techniques. In this work, we explore this research direction, and consider strategies based on deep learning techniques. Our main goal is to test various neural network-based schemes, so as to design a strategy which should be robust when $\varepsilon \to 0$, easily implementable even for complicated geometries, and with potential to scale in high dimension.

The idea of working with neural network functions to solve PDEs is by far not novel, and countless contributions have been proposed on this front in recent years. The strategies can be roughly classified into two categories:

(1) In the first category, deep neural networks are employed to assist classical numerical methods by improving some limitations, or accelerating certain steps (see, e.g., [1–3]).
(2) In the second category, neural networks are used to directly approximate the solution of PDEs. The solution schemes become in this case an optimization problem where it is crucial to design appropriate loss functions. The loss functions are mostly based on residuals of the equations, and yield to different methods depending on the specific choice:
   (a) Physics-informed neural networks (PINNs, [4]) is a collocation-based method. One finds the coefficients of the neural network solution by minimizing a discretized version of the L^2 norm of the strong form of the residual of the PDE. This method is very easily implementable but it implicitly assumes that solutions are very regular.
   (b) Other strategies leverage weak variational formulations where less regular solutions are allowed. On this front, most of the classical methods originally formulated for piecewise polynomial functions have by now been tested with trial and test spaces of neural network functions. In this respect, the deep Galerkin method (DGM, [5]) is based on a least-squares formulation, and the variational PINNS (VPINNs, [6, 7]) is based on the Galerkin method. The main drawback of this approach is that the approximation quality depends on the architecture of both the trial *and* the test neural network classes. In addition, numerous evaluations for multiple test functions need to be performed. Also, strategies involving the minimization of weak-form residuals are usually not trivial to implement because they involve the computation of norms in very weak spaces which necessitate extra discretization steps.
   (c) Another approach based on weak variational formulations is the so-called deep Ritz method (DRM, [8]). It leverages the fact that the solution of certain PDEs are the unique minimizer to a certain energy functional. When possible, this approach seems the most appealing: the loss functions is naturally given by the problem, it can accommodate low regular solutions, and the computational cost is moderate in the sense that it only requires to handle test functions (no trial functions). It also carries potential to address high dimensional problem as illustrated in [8–10].

1.2. **Contribution**

The goal of this work is to compare and develop several neural network schemes for singularly perturbed problems when $\varepsilon \to 0$. We focus more particularly on convection-diffusion (or stationnary Fokker-Planck) problems with vanishing diffusion for which we explore schemes from the second category according to the above distinction. In other words, we approximate solutions of singular PDEs with feedforward neural network functions. When $\varepsilon \to 0$, the regularity of the solutions is deteriorated because of local or boundary thin layers. Therefore the vanilla PINNs method is expected to perform poorly for small values of ε because it commits a variational crime (and this is actually confirmed in our numerical experiments). Methods based on weak variational formulations seem better adapted, and on that front, it is desirable to work with the deep Ritz method. However, finding energy formulations is not straightforward due to the non-symmetric nature of convective effects. We show how this method can be applied in this context thanks to a change of variable. We compare its numerical robustness with respect to the PINNs method, and a naive finite element discretization with a uniform grid. In the present study, our tests are performed on a 1D example. Despite its simplicity, the example exhibits all the features that are challenging for numerical schemes. For our purposes, the example also presents the important advantage of having analytic solutions which we can leverage in our error analysis, and

our validations. Higher-dimensional tests involving also more elaborate sampling strategies are left for future work.

The paper is organized as follows. In Section 2, various formulations of the convection-diffusion problem we are interested in are introduced. In Section 3, we introduce various neural networks-based schemes which are inspired from the various formulations introduced in Section 2. The reader is encouraged to observe that an expert mathematical insight is required in order to build formulations that do not incur in variational crimes. Lastly, in Section 4, these various schemes are compared for one-dimensional problem. We comment on their respective merits and limitations as ε goes to 0.

2. A singularly perturbed convection-diffusion equation

The aim of this section is to introduce the singularly perturbed convection-diffusion equation we consider in this work, and various formulations of the problem which will be used in Section 3 so as to design various neural networks-based schemes for its numerical solution.

2.1. Problem definition

As a prototypical example, we consider the following singularly perturbed convection-diffusion equation on a given domain $\Omega \subset \mathbb{R}^d$, with $d \in \mathbb{N}^*$. Let $F : \Omega \to \mathbb{R}^d$ be a given force field, $0 < \varepsilon \ll 1$ a small parameter, and $f : \Omega \to \mathbb{R}$ is a given right-hand side function. Our goal is to find a solution $u : \Omega \to \mathbb{R}$ to

$$-\varepsilon(\Delta u)(x) + \nabla \cdot (Fu)(x) = f(x), \quad \forall x \in \Omega, \tag{1}$$

together with Robin boundary conditions

$$\alpha(\nabla u \cdot n)(x) + \kappa u(x) = g(x) \quad \forall x \in \partial\Omega, \tag{2}$$

where n refers to the outward unit vector of $\partial\Omega$, α, $\kappa \geq 0$ and g is a real-valued function defined on $\partial\Omega$. In the following, we assume that the force field F derives from a potential function $V : \Omega \to \mathbb{R}$, in the sense that

$$F(x) = -\nabla V(x), \quad \forall x \in \Omega.$$

Under appropriate assumptions on F (or V), f and g, which are assumed to be smooth functions for the sake of simplicity, problem (1)-(2) can be proved to have a unique solution [11–13]. Note that, more generally, α and κ could also be given as real-valued functions defined on $\partial\Omega$, instead of constants, and our subsequent developments could be easily adapted.

The equation represents the change in the concentration u of a quantity in a given medium, and in presence of convective and diffusive effects. The force field F represents the drag force while the singular perturbation parameter ε represents the diffusivity of the medium. In the limit of an inviscid medium as $\varepsilon \to 0$, the equation changes from elliptic to hyperbolic nature, and from second to first order. For Dirichlet boundary conditions $u = 0$ on $\partial\Omega$, the solution can develop sharp boundary layers of width ε near the outflow. We refer the reader to [14] for general references on this equation regarding its analysis and numerical methods.

Classical numerical methods are challenged by problem (1) when ε is small. In the case of the Galerkin finite element method, the poor performance for this problem is reflected in the bound on the error in the finite element solution. For $\Omega = (0, 1)$ and Dirichlet boundary conditions, a standard Galerkin method with a uniform grid of size h delivers a solution u_h on a finite element space $\mathbb{P}_h$ that satisfies

$$\|u - u_h\|_{H^1(0,1)} \leq C(\varepsilon) \inf_{w_h \in \mathbb{P}_h} \|u - w_h\|_{H^1(0,1)}, \tag{3}$$

where $C(\varepsilon) \sim \varepsilon^{-1}$, so that the constant blows up as $\varepsilon \to 0$ (see [14, Theorem 2.49]). The dependence of C on ε is usually referred to as a *loss of robustness* in the sense that, as ε decreases, the Galerkin method is bounded more and more loosely by the best approximation error. As a consequence, on a coarse mesh and for

small values ε, the Galerkin approximation develops spurious oscillations everywhere in the domain. This very well-known behavior will actually be observed later on in our numerical tests.

Numerous methods have been proposed in order to address this loss of robustness in finite element methods. An important family of methods is based on using residual-based stabilization techniques. Given some variational form, the problem is modified by adding to the bilinear form the strong form of the residual, weighted by a test function and scaled by a stabilization constant τ. The most well-known example of this technique is the streamline upwind Petrov-Galerkin (SUPG) method (see [15]). The addition of the residual-based stabilization term, can be interpreted as a modification of the test functions which means that these methods seek stabilization by changing the test space, and motivates to search for optimal test spaces in the spirit of [16, 17].

Other classical discretization methods such as finite volumes suffer from similar issues, and strategies involving layer-adaptive grids such as Shishkin meshes have been proposed (see, e.g., [18]).

The aim of this work is to explore the potential of approximating solutions of such problems with neural network functions, and the next section presents several options for this, with a discussion on their merits and limitations.

2.2. General formulation

Any neural-network based numerical scheme for the solution of (1)-(2) relies on the use of a variational formulation of this problem which enables to write u (or another function defined from u) as a minimizer of a problem of the form

$$\min_{v\in\mathcal{V}} \mathcal{J}(v), \tag{4}$$

where $\mathcal{V}$ is a particular set of real-valued functions defined on Ω. The loss function $\mathcal{J} : \mathcal{V} \to \mathbb{R}$ is usually of the form

$$\mathcal{J}(v) := \int_\Omega \mathcal{R}(v)(x)\mathrm{d}\rho(x) + \int_{\partial\Omega} \mathcal{S}(v)(r)\mathrm{d}\tau(r), \quad \forall v \in \mathcal{V}, \tag{5}$$

where for every $v \in \mathcal{V}$, $\mathcal{R}(v)$ and $\mathcal{S}(v)$ are real-valued functions defined on Ω and $\partial\Omega$ respectively. They are assumed to be measurable with respect to the measures ρ and τ, which are defined on Ω and $\partial\Omega$ respectively.

The aim of the next sections is to introduce various formulations of problem (1)-(2) under the form (4)-(5). This requires to define appropriate definitions of the set $\mathcal{V}$, the functions $\mathcal{R}(v)$ and $\mathcal{S}(v)$ for any $v \in \mathcal{V}$ and the unknown function solution of (4). Unless otherwise stated, the measures ρ and τ will be defined as the Lebesgue and Lebesgue surfacic measures respectively.

2.3. Vanilla (V) formulation

We begin by introducing the most classical formulation used in neural network-based numerical schemes such as PINNs. For the reasons that we outline next, different aspects of this formulation can be improved, therefore we refer to it as *vanilla* (V) formulation in the following.

The formulation consists in interpreting the solution u of (1)-(2) as the unique solution of a minimization problem of the form (4) with $\mathcal{V} = H^2(\Omega)$ and to define for all $v \in \mathcal{V}$,

$$\begin{cases} \mathcal{R}(v)(x) := \lambda \left|-\varepsilon(\Delta v)(x) + \nabla\cdot(Fv)(x) - f(x)\right|^2, & \text{for all } x \in \Omega, \\ \mathcal{S}(v)(x) := (1-\lambda)\left|\alpha(\nabla v\cdot n)(x) + \kappa v(x) - g(x)\right|^2, & \text{for all } x \in \partial\Omega, \end{cases} \tag{6}$$

for some $\lambda \in (0, 1)$. In this approach, the parameter λ enables to tune the respective weight of the contributions of the bulk and boundary terms in the total functional $\mathcal{J}$ to be minimized. In practice, in the numerical tests presented in Section 4, λ will always be chosen to be equal to $\frac{1}{2}$.

Note that such an approach requires the solution u to belong to $H^2(\Omega)$, which implies that the solution has to be sufficiently regular. When $\varepsilon \to 0$, this assumption becomes less and less realistic due to the formation of boundary layers. This raises the question as to whether it is possible to introduce another formulation of problem (1)-(2) which would allow for less regular solutions. The goal of the next section is to introduce such an alternative formulation.

2.4. **Weak variational (W) formulation**

In this section we develop an avenue based on an energy minimization approach which requires less regularity in the solutions than the vanilla formulation. To this aim, we introduce the change of variable

$$u(x) = e^{cV(x)} z(x), \tag{7}$$

where $c \in \mathbb{R}$ is a constant yet to be determined. Taking first and second derivatives in (7) yield that for all $x \in \Omega$,

$$\begin{aligned}
\nabla u(x) &= e^{cV(x)} \left(c\nabla V(x) z(x) + \nabla z(x) \right) \\
\Delta u(x) &= e^{cV(x)} \left(c\Delta V(x) z(x) + |c\nabla V(x)|^2 z(x) + 2c\nabla V(x) \cdot \nabla z(x) + \Delta z(x) \right).
\end{aligned}$$

Now, setting the value of c to be

$$c = \frac{1}{2\varepsilon},$$

and inserting the change of variable into (1), we conclude that u is a solution to (1) if and only if z is a solution to the elliptic problem

$$-\Delta z(x) + \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2} \right) z(x) = f(x) \frac{e^{-\frac{V(x)}{2\varepsilon}}}{\varepsilon}, \quad \forall x \in \Omega, \tag{8}$$

with Robin boundary conditions

$$\alpha(\nabla z(x) \cdot n(x)) + \left(\kappa + \frac{\alpha}{2\varepsilon} \nabla V(x) \cdot n(x) \right) z(x) = e^{\frac{-V(x)}{2\varepsilon}} g(x), \quad \forall x \in \partial\Omega. \tag{9}$$

At this stage, one could of course apply the vanilla formulation to solve (8)-(9) and compute z solution of a minimization problem of the form (4) with $\mathcal{V} = H^2(\Omega)$ and the functionals $\mathcal{R}$ and $\mathcal{S}$ defined by

$$\begin{cases}
\mathcal{R}(v)(x) := \lambda \left| \Delta v(x) + \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2} \right) v(x) - f(x) \frac{e^{-\frac{V(x)}{2\varepsilon}}}{\varepsilon} \right|^2, & \text{for all } x \in \Omega, \\
\mathcal{S}(v)(x) := \left| (1-\lambda)\alpha(\nabla v(x) \cdot n(x)) + \left(\kappa + \frac{\alpha}{2\varepsilon} \nabla V(x) \cdot n(x) \right) v(x) - e^{\frac{-V(x)}{2\varepsilon}} g(x) \right|^2, & \text{for all } x \in \partial\Omega,
\end{cases} \tag{10}$$

for all $v \in \mathcal{V} = H^2(\Omega)$ and some $\lambda \in (0,1)$. The value of λ chosen in our numerical tests is $\lambda = 0.5$. We will refer to this approach as the *vanilla-z* (Vz) formulation.

Note that this method does not fully exploit the change of variables since the elliptic nature of problem (8) allows us to easily build a weak formulation of this equation. Testing against a smooth test function v and integrating by parts we obtain the weak formulation

$$\int_\Omega \nabla z \cdot \nabla v - \int_{\partial\Omega} v \nabla z \cdot n \mathrm{d}x + \int_\Omega \left(\frac{\Delta V}{2\varepsilon} + \frac{|\nabla V|^2}{4\varepsilon^2} \right) zv = \int_\Omega f \frac{e^{-\frac{V}{2\varepsilon}}}{\varepsilon} v.$$

Using equality (9), we get

$$\begin{aligned}
\int_\Omega \nabla z \cdot \nabla v + \int_\Omega \left(\frac{\Delta V}{2\varepsilon} + \frac{|\nabla V|^2}{4\varepsilon^2} \right) zv + \int_{\partial\Omega} \left(\frac{\kappa}{\alpha} + \frac{1}{2\varepsilon} \nabla V \cdot n \right) zv \\
= \int_\Omega f \frac{e^{-\frac{V}{2\varepsilon}}}{\varepsilon} v + \int_{\partial\Omega} \frac{1}{\alpha} e^{\frac{-V}{2\varepsilon}} gv.
\end{aligned}$$

Therefore the weak formulation of problem (8) is to find $z \in H^1(\Omega)$ such that

$$a(z,v) = \ell(v), \quad \forall v \in H^1(\Omega) \tag{11}$$

with

$$\begin{aligned} a(z,v) &:= \int_\Omega \nabla z \cdot \nabla v + \int_\Omega \left(\frac{\Delta V}{2\varepsilon} + \frac{|\nabla V|^2}{4\varepsilon^2} \right) zv \mathrm{d}x + \int_{\partial\Omega} \left(\frac{\kappa}{\alpha} + \frac{1}{2\varepsilon} \nabla V \cdot n \right) zv \\ l(v) &:= \int_\Omega f \frac{e^{-\frac{V}{2\varepsilon}}}{\varepsilon} v + \int_{\partial\Omega} \frac{1}{\alpha} e^{\frac{-V}{2\varepsilon}} gv \end{aligned}$$

To ensure that the symmetric bilinear form a is continuous and coercive, we assume in the sequel that the following conditions are satisfied:

$$\begin{cases} \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2} \right) \geq a_0 > 0, & \forall x \in \Omega \\ \left(\frac{\kappa}{\alpha} + \frac{1}{2\varepsilon} \nabla V(x) \cdot n(x) \right) \geq 0, & \forall x \in \partial\Omega \\ f \in L^2(\Omega),\ g \in L^2(\partial\Omega) \end{cases} \tag{12}$$

In that case, all the hypothesis of the Lax-Milgram theorem are satisfied for a and ℓ. Thus, z can be equivalently rewritten as the unique solution of a minimization problem of the form

$$z = \underset{v \in H^1(\Omega)}{\operatorname{argmin}} \frac{1}{2} a(v,v) - \ell(v). \tag{13}$$

This implies that z can equivalently be recast as the unique solution of a minimization problem of the form (4) with $\mathcal{V} = H^1(\Omega)$ and

$$\begin{cases} \mathcal{R}(v)(x) & := \frac{1}{2} \left[|\nabla v(x)|^2 + \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2} \right) |v(x)|^2 \right] - f(x) \frac{e^{-\frac{V(x)}{2\varepsilon}}}{\varepsilon} v(x), \quad \forall x \in \Omega, \\ \mathcal{S}(v)(x) & := \frac{1}{2} \left[\left(\frac{\kappa}{\alpha} + \frac{1}{2\varepsilon} \nabla V(x) \cdot n(x) \right) |v(x)|^2 \right] - \frac{1}{\alpha} e^{\frac{-V(x)}{2\varepsilon}} g(x) v(x), \quad \forall x \in \partial\Omega. \end{cases} \tag{14}$$

We will refer to this approach as the *weak-z* (Wz) formulation.

Moreover, using (7), we can equivalently express u as a solution of a minimization problem of the form (4) with

$$\mathcal{V} := \left\{ v = e^{\frac{V}{2\varepsilon}} \overline{v},\ \overline{v} \in H^1(\Omega) \right\}, \tag{15}$$

and

$$\begin{cases} \mathcal{R}(v)(x) & := \frac{1}{2} \left[|\nabla \overline{v}(x)|^2 + \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2} \right) |\overline{v}(x)|^2 \right] - f(x) \frac{e^{-\frac{V(x)}{2\varepsilon}}}{\varepsilon} \overline{v}(x), \quad \forall x \in \Omega, \\ \mathcal{S}(v)(x) & := \frac{1}{2} \left[\left(\frac{\kappa}{\alpha} + \frac{1}{2\varepsilon} \nabla V(x) \cdot n(x) \right) |\overline{v}(x)|^2 \right] - \frac{1}{\alpha} e^{\frac{-V(x)}{2\varepsilon}} g(x) \overline{v}(x), \quad \forall x \in \partial\Omega, \end{cases} \tag{16}$$

for all $v \in \mathcal{V}$ with $\overline{v} := ve^{-\frac{V}{2\varepsilon}}$. We will refer to this formulation as the *weak* (W) formulation.

2.5. **Rescaled formulation**

In this section, we introduce another formulation based on a change of scale in the original problem. More precisely, introducing $\Omega_\varepsilon := \frac{1}{\varepsilon}\Omega$, we introduce auxiliary functions $\widetilde{u} : \Omega_\varepsilon \to \mathbb{R}$, $\widetilde{z} : \Omega_\varepsilon \to \mathbb{R}$ and $\widetilde{V} : \Omega_\varepsilon \to \mathbb{R}$ defined so that for all $x \in \Omega$,

$$u(x) = \varepsilon \widetilde{u}\left(\frac{x}{\varepsilon}\right), \quad z(x) = \varepsilon \widetilde{z}\left(\frac{x}{\varepsilon}\right), \quad V(x) = \varepsilon \widetilde{V}\left(\frac{x}{\varepsilon}\right). \tag{17}$$

Notice that if u and z satisfy (7), then

$$\widetilde{z}(y) = \widetilde{u}(y)e^{\frac{1}{2}\widetilde{V}(y)}, \quad \forall y \in \Omega_\varepsilon.$$

Denoting by $\widetilde{F}(y) := -\nabla\widetilde{V}(y) = F(\varepsilon y)$ for all $y \in \Omega_\varepsilon$, it holds that u is solution to (1)-(2) if and only if $\widetilde{u}$ is solution to

$$-\Delta\widetilde{u}(y) + \nabla\cdot\left(\widetilde{F}\widetilde{u}\right)(y) = \widetilde{f}(y), \quad \forall y \in \Omega_\varepsilon \tag{18}$$

where $\widetilde{f}(y) := f(\varepsilon y)$ for all $y \in \Omega_\varepsilon$ with boundary conditions

$$\alpha(\nabla\widetilde{u}\cdot n)(y) + \varepsilon\kappa\widetilde{u}(y) = \widetilde{g}(y), \quad \forall y \in \partial\Omega_\varepsilon, \tag{19}$$

with $\widetilde{g}(y) := g(\varepsilon y)$ for all $y \in \Omega_\varepsilon$.

Using similar calculations to the ones done in Section 2.4, the Lax-Milgram theorem guarantees that $\widetilde{z}$ is the unique solution in $H^1(\Omega_\varepsilon)$ of the following variational problem: for all $\widetilde{v} \in H^1(\Omega_\varepsilon)$,

$$\begin{aligned}\int_{\Omega_\varepsilon}\nabla\widetilde{z}(y)\cdot\nabla\widetilde{v}(y)\,dy + \int_{\partial\Omega_\varepsilon}\left(\frac{\varepsilon\kappa}{\alpha} + \frac{1}{2}\nabla\tilde{V}(y)\cdot n(y)\right)\widetilde{z}(y)\widetilde{v}(y)\,dy + \int_{\Omega_\varepsilon}\left(\frac{\Delta\tilde{V}(y)}{2} + \frac{|\nabla\tilde{V}(y)|^2}{4}\right)\widetilde{z}(y)\,\widetilde{v}(y)\,dy \\ = \int_{\Omega_\varepsilon}\widetilde{f}(y)e^{-\frac{1}{2}\tilde{V}(y)}\widetilde{v}(y)\,dy + \int_{\partial\Omega_\varepsilon}\frac{1}{\alpha}e^{-\frac{1}{2}\tilde{V}(y)}\widetilde{g}(y)\widetilde{v}(y)\,dy.\end{aligned} \tag{20}$$

The result is valid provided that the following assumptions on the coefficients hold

$$\begin{cases}\Delta\tilde{V}(y) + |\nabla\tilde{V}|^2(y) \geq 0, \ \forall y \in \Omega_\varepsilon, \\ \frac{\kappa\varepsilon}{\alpha} + \frac{1}{2}\nabla\tilde{V}(y)\cdot n(y) \geq 0, \ \forall y \in \partial\Omega_\varepsilon, \\ \widetilde{f} \in L^2(\Omega_\varepsilon), \ \widetilde{g} \in L^2(\partial\Omega_\varepsilon).\end{cases}$$

The above conditions are equivalent to the assumptions (12) stated in Section 2.4.

As in the previous section, the function $\widetilde{z}$ is then the unique solution of a minimization problem of the form (4) with $\mathcal{V} = H^1(\Omega_\varepsilon)$ with

$$\begin{cases}\mathcal{R}(v)(y) & := \frac{1}{2}\left[|\nabla v(y)|^2 + \left(\frac{\Delta\widetilde{V}(y)}{2} + \frac{|\nabla\widetilde{V}(y)|^2}{4}\right)|v(y)|^2\right] - \widetilde{f}(y)e^{-\frac{\widetilde{V}(y)}{2}}v(y), \quad \forall y \in \Omega_\varepsilon, \\ \mathcal{S}(v)(y) & := \frac{1}{2}\left[\left(\frac{\varepsilon\kappa}{\alpha} + \frac{1}{2}\nabla\widetilde{V}(y)\cdot n(y)\right)|v(y)|^2\right] - \frac{1}{\alpha}e^{\frac{-\widetilde{V}(y)}{2}}\widetilde{g}(y)v(y), \quad \forall y \in \partial\Omega_\varepsilon.\end{cases} \tag{21}$$

We will refer to this approach as the *rescaled-weak-z* (RWz) formulation.

2.6. **Summary of the methods**

For the sake of clarity, we summarize here the main features of each method.

Method	Acronym	Unknown	$\mathcal{V}$	$\mathcal{R}$ and $\mathcal{S}$
vanilla	V	u	$H^2(\Omega)$	(6)
vanilla-z	Vz	z	$H^2(\Omega)$	(10)
weak	W	u	(15)	(16)
weak-z	Wz	z	$H^1(\Omega)$	(14)
rescaled-weak-z	RWz	$\widetilde{z}$	$H^1(\Omega_\varepsilon)$	(21)

3. Neural networks based numerical schemes

In this section we describe the numerical approach used in order to compute an approximation of the solution of a minimization problem of the form (4) by means of a neural-network based method. We first present in Section 3.1 the general principle of such approaches. The main ingredients to design a neural-network based method consist in the choice of a class of neural network functions and of sampling schemes in order to approximate the integrals involved in the definition of the loss function $\mathcal{J}$ defined by (5). These two ingredients are detailed respectively in Section 3.2 and Section 3.3 respectively. Finally, some details on the numerical implementation are given in Section 4.2.

3.1. General principle

The numerical solution of a minimization problem of the form (4) usually requires to consider alternatives to $\mathcal{V}$ and $\mathcal{J}$ that are amenable for practical implementation. The strategy thus consists in formulating a related problem of the form

$$\min_{v\in\mathcal{K}} \widehat{\mathcal{J}}(v), \tag{22}$$

where

- $\mathcal{K}\subset\mathcal{V}$ is a set of functions parametrized by a finite number of scalar coefficients. A classical class of functions are finite elements. Here, we consider neural networks (see Section 3.2 below);
- $\widehat{\mathcal{J}}$ is an approximation of the loss function $\mathcal{J}$ where the integrals are approximated using some particular quadrature or sampling schemes.

More precisely, for given integers $K, M\in\mathbb{N}^*$, given sets of points $(x_k)_{1\le k\le K}\subset\Omega$, $(y_m)_{1\le m\le M}\subset\partial\Omega$, and given sets of weights $(\rho_k)_{1\le k\le K}\subset\mathbb{R}_+$ and $(\tau_m)_{1\le m\le M}\subset\mathbb{R}_+$, for all $v\in\mathcal{K}$, the functional $\widehat{\mathcal{J}}(v)$ is defined by

$$\widehat{\mathcal{J}}(v) := \sum_{k=1}^{K}\rho_k\mathcal{R}(v)(x_k) + \sum_{m=1}^{M}\tau_m\mathcal{S}(v)(y_m). \tag{23}$$

As a consequence, the definition of a neural-network based numerical scheme for the approximation of a problem of the form (4) requires the definition of two ingredients:

- the class $\mathcal{K}\subset\mathcal{V}$ of neural network functions;
- the sampling scheme, i.e. the choice of K, M, $(x_k)_{1\le k\le K}$, $(y_m)_{1\le m\le M}$, $(\rho_k)_{1\le k\le K}$ and $(\tau_m)_{1\le m\le M}$ in order to define the approximate functional $\widehat{\mathcal{J}}$ given by (23).

The set of neural network functions $\mathcal{K}$ we consider in our numerical experiments is presented in Section 3.2. The various sampling schemes tested here are given in Section 3.3.

3.2. Neural Network classes of functions

In this work, we only consider classes of functions defined by means of feedforward neural networks whose definition we recall next (see [19] for general references).

Let $\mathcal{X}\subset\mathbb{R}^{d_\mathcal{X}}$ and $\mathcal{Y}\subset\mathbb{R}^{d_\mathcal{Y}}$ be some input and output sets of finite dimensions $d_\mathcal{X}, d_\mathcal{Y}\in\mathbb{N}^*$. A feedforward neural network is a function

$$\psi : \mathcal{X}\to\mathcal{Y}$$

which reads as

$$\psi(x) = T_L(\sigma(T_{L-1}(\sigma(\dots\sigma(T_0(x)))))), \quad \forall x\in\mathcal{X}. \tag{24}$$

For every $\ell\in\{0,\dots,L\}$,

$$T_\ell : \left\{\begin{array}{ccc}\mathbb{R}^{p_\ell} & \to & \mathbb{R}^{p_{\ell+1}}\\ x_\ell & \mapsto & T_\ell(x_\ell) := A_\ell x_\ell + b_\ell\end{array}\right. \tag{25}$$

is an affine function which can be expressed through a matrix $A_\ell\in\mathbb{R}^{p_{\ell+1}\times p_\ell}$, and an offset vector $b_\ell\in\mathbb{R}^{p_{\ell+1}}$, and $\sigma:\mathbb{R}\to\mathbb{R}$ is called the (nonlinear) activation function. By a slight abuse of notation, for all $p\in\mathbb{N}^*$ and for

any vector $w := (w_i)_{1\leq i\leq p} \subset \mathbb{R}^p$, the notation $\sigma(w)$ actually denotes the vector of $\mathbb{R}^p$ with entries $\sigma(w_i)$, that is, $\sigma(w) = (\sigma(w_i))_{i=1}^p$. Note that since ψ maps $\mathcal{X}$ onto $\mathcal{Y}$, it is necessary that $p_0 = d_{\mathcal{X}}$ and $p_{L+1} = d_{\mathcal{Y}}$. The layers numbered from 1 to L are usually called the hidden layers of the neural network.

To define a class of feedforward neural networks, we fix an architecture by prescribing a given activation function σ, depth $L \in \mathbb{N}$, and layer widths $\boldsymbol{p} = (p_0, \ldots, p_{L+1}) \in (\mathbb{N}^*)^{L+2}$. Once the values of σ, L and $\boldsymbol{p}$ have been chosen, we view the coefficients $(A_\ell, b_\ell)_{0\leq\ell\leq L}$ of the affine mappings $T_0, \cdots, T_L$ as parameters. We gather these coefficients in the vector of parameters

$$\theta := \{(A_\ell, b_\ell)\}_{\ell=0}^L,$$

and assume that θ takes values in a set

$$\Theta \subseteq \bigtimes_{\ell=0}^{L} \left(\mathbb{R}^{p_\ell \times p_{\ell+1}} \times \mathbb{R}^{p_{\ell+1}}\right).$$

For any $\theta \in \Theta$, we define by $\psi_\theta : \mathcal{X} \to \mathcal{Y}$ the function ψ defined by (24) with $\theta = \{(A_\ell, b_\ell)\}_{\ell=0}^L \in \Theta$.

The class of neural network functions with architecture $(\sigma, L, \boldsymbol{p})$ and coefficient sets Θ is then defined as

$$\mathcal{N}(\sigma, L, \boldsymbol{p}, \Theta) := \{\psi_\theta \ : \ \theta \in \Theta\}.$$

In our context, the input and output sets $\mathcal{X}$ and $\mathcal{Y}$ are respectively given by

$$\mathcal{X} = \Omega \text{ (or } \Omega_\varepsilon) \quad \text{ and } \quad \mathcal{Y} = \mathbb{R},$$

so that $d_{\mathcal{X}} = d$ and $d_{\mathcal{Y}} = 1$. In all the numerical tests presented below, the class $\mathcal{K}$ is chosen as

$$\mathcal{K} := \mathcal{N}(\sigma, L, \boldsymbol{p}, \Theta),$$

with

$$\sigma = \tanh,\ L = 2,\ \boldsymbol{p} = (10, 10) \text{ and } \Theta = \bigtimes_{\ell=0}^{L} \left(\mathbb{R}^{p_\ell \times p_{\ell+1}} \times \mathbb{R}^{p_{\ell+1}}\right).$$

Note that the set $\mathcal{K}$ is then a subset of $\mathcal{V}$ for all the formulations of the convection-diffusion problem we introduced in Section 2. Moreover, the solution of the approximate problem (22) is equivalent to finding a minimizer $\theta^* \in \Theta$ solution to

$$\min_{\theta\in\Theta} \widehat{\mathcal{J}}(\psi_\theta). \tag{26}$$

Remark: In many machine learning applications, the choice of relu activation functions is very common due to its low computational cost when performing evaluation or first order differentiation. However, in our problem, second order derivatives are needed to calculate the loss function. If relu activation functions were used, then the second order derivative terms would be 0, and no good approximation could be learned. This reason motivates our choice of tanh as the activation function.

3.3. **Sampling schemes**

We detail in this section the various sampling schemes we considered in our numerical tests in order to define the approximate loss function $\widehat{\mathcal{J}}$.

Since we work with one-dimensional examples, we carry the discussion for dimension one. In fact, we consider problem (1)-(2) with $\Omega = (0, 1)$ so that $\partial\Omega = \{0\} \cup \{1\}$ (and $\partial\Omega_\varepsilon = \{0\} \cup \{1/\varepsilon\}$). Thus, for all our tests, the domain boundary has $M = 2$ points $y_1 = 0$ and $y_2 = 1$ (or $y_2 = \frac{1}{\varepsilon}$ for the RWz method). Taking $\tau_1 = \tau_2 = 1$ for the surface weights, the surface term in (23) takes the simple form

$$\sum_{m=1}^{M=2} \tau_m \mathcal{S}(v)(y_m) = \int_{\partial\Omega} \mathcal{S}(v)\, d\tau \quad \forall v \in \mathcal{V} \text{ (or } \int_{\partial\Omega_\varepsilon} \mathcal{S}(v)\, d\tau \text{ for the RWz formulation)}.$$

We consider three different sampling schemes for the approximation of the bulk term $\int_\Omega \mathcal{R}(v)\, d\rho$:

(1) The first choice is a simple *uniform* sampling scheme (labeled $-u$ in our tests). For a given $K \in \mathbb{N}^*$, we set $\rho_k = \frac{1}{K}$ and $(x_k)_{1\leq k\leq K}$ as the centers of the intervals given by a uniform discretization grid of the interval $(0,1)$.

(2) The second sampling scheme, called *random* $(-r)$ scheme, consists in choosing $\rho_k = \frac{1}{K}$ and the points $(x_k)_{1\leq k\leq K}$ as a collection of random points, identically independently distributed according to the uniform distribution on $(0,1)$.

(3) We lastly consider a third sampling scheme, called *exponential* (-e) scheme, which is specific to the Wz formulation. Recall that in this case, for all $v \in \mathcal{K}$, the expression of $\mathcal{R}(v)$ is given by (14), namely

$$\mathcal{R}(v)(x) = \mathcal{R}^{(1)}(v)(x) + \mathcal{R}^{(2)}(v)(x), \quad \forall x \in \Omega$$

with

$$\begin{cases} \mathcal{R}^{(1)}(v)(x) & := \frac{1}{2}\left[|\nabla v(x)|^2 + \left(\frac{\Delta V(x)}{2\varepsilon} + \frac{|\nabla V(x)|^2}{4\varepsilon^2}\right)|v(x)|^2\right] \\ \mathcal{R}^{(2)}(v)(x) & := -f(x)\frac{e^{-\frac{V(x)}{2\varepsilon}}}{\varepsilon} v(x), \quad \forall x \in \Omega. \end{cases}$$

The thus view the bulk integral term as

$$\int_\Omega \mathcal{R}(v)(x)\mathrm{d}\rho(x) = \int_\Omega \mathcal{R}^{(1)}(v)(x)\mathrm{d}\rho(x) + \int_\Omega \mathcal{R}^{(2)}(v)(x)\mathrm{d}\rho(x),$$

and we approximate each component separately as follows. For the first term, we draw a collection of $K_1 \in \mathbb{N}^*$ iid. random points $(x_k^{(1)})_{1\leq k\leq K_1}$ from the uniform distribution on $(0,1)$ and for all $1 \leq k \leq K_1$, the weights $\rho_k^{(1)}$ are chosen to be equal to $\frac{1}{K_1}$. For the second term, we draw $K_2 \in \mathbb{N}^*$ iid random points $(x_k^{(2)})_{1\leq k\leq K_2}$ following the probability density

$$\rho^{(2)}(x) := \frac{e^{-\frac{V(x)}{2\varepsilon}}}{Z_\varepsilon}, \quad x \in \Omega,$$

with

$$Z_\varepsilon := \int_\Omega e^{-\frac{V(x)}{2\varepsilon}}\, dx$$

Setting now $\rho_k^{(2)} = \frac{Z_\varepsilon}{K_2}$ for all $1 \leq k \leq K_2$, the integral $\int_\Omega \mathcal{R}(v)$ is the approximated by

$$\sum_{k=1}^{K_1} \rho_k^{(1)} \left(\frac{1}{2}\left[|\nabla v(x_k^{(1)})|^2 + \left(\frac{\Delta V(x_k^1)}{2\varepsilon} + \frac{|\nabla V(x_k^{(1)})|^2}{4\varepsilon^2}\right)|v(x_k^{(1)})|^2\right]\right) - \sum_{k=1}^{K_2} \rho_k^{(2)} \left(f(x_k^{(2)})\frac{1}{\varepsilon} v(x_k^{(2)})\right).$$

In the following, we use the notation $-u$ (respectively $-r$ and $-e$), after the name of a formulation, in order to refer to the numerical method obtained by using this formulation, together with a uniform (respectively random or exponential) sampling scheme. For instance, the $V-u$ method refers to the vanilla formulation used in conjunction with a uniform sampling scheme.

3.4. **Comparison with finite element schemes**

One important point in the investigation of the merits and limitations of deep learning-based numerical schemes is to understand how they compare with respect to other existing schemes. In our tests, we provide a numerical comparison with a vanilla finite element Galerkin scheme involving a uniform mesh. For the sake of completeness, we briefly recall the main steps of our finite element Galerkin approach.

Integrating the original equation (1) against a sufficiently smooth function $v \in \mathcal{C}^\infty(\Omega)$, and integrating by parts, it follows that a weak formulation of problem (1) is to find $u \in H^1(\Omega)$ such that

$$a(u,v) = l(v), \quad \forall v \in H^1(\Omega)$$

with

$$a(u,v) = \int_\Omega \nabla u \nabla v \mathrm{d}x + \varepsilon^{-1} \int_\Omega \nabla \cdot (Fu) v \mathrm{d}x + \frac{\kappa}{\alpha} \int_{\partial\Omega} u(x)v(x)\mathrm{d}x \tag{27}$$

$$l(v) = \varepsilon^{-1} \int_\Omega f(x)v(x)\mathrm{d}x - \int_{\partial\Omega} g(x)v(x)\mathrm{d}x. \tag{28}$$

We numerically solve this problem by Galerkin projection. For this, we consider a mesh $(T_n)_{n=1}^N$ of Ω and define the associated $\mathbb{P}_1$ finite element space

$$\mathcal{V}_N \coloneqq \{v \in \mathscr{C}^0(\mathbb{R}) \ : \ \forall 0 \leq s \leq N-1, \ v_{[x_s, x_{s+1})} \in \mathbb{P}^1([x_s, x_{s+1}))\}$$

with

$$\mathbb{P}^1([x_s, x_{s+1}] \coloneqq \{v : [x_s, \ x_{s+1}], \ v(x) = ax + b, \ (a,b) \in \mathbb{R}^2\}.$$

We then search for a solution $u_N \in \mathcal{V}_N \subset H^1(\Omega)$ by Galerkin projection, that is, we search for $u_N \in \mathcal{V}_N$ such that

$$a(u_N, v) = l(v), \quad \forall v \in \mathcal{V}_N.$$

We next take as a basis of $\mathcal{V}_N$ the set of tent functions defined as

$$\varphi_i(x_j) = \delta_{ij}, \text{ for } 1 \leq i,j \leq N.$$

and we express the solution as $u_N = \sum_{i=1}^N c_i \varphi_i$. Gathering the expansion coefficients in the vector $c = (c_i)_{i=1}^N$, and injecting the expansion of u_N in the variational formulation, we are led to the system of equations

$$Mc = q$$

where

$$M = (M_{i,j})_{1 \leq i,j \leq N}, \quad M_{i,j} \coloneqq a(\varphi_i, \varphi_j),$$
$$q = (q_i)_{i=1}^N, \quad q_i \coloneqq l(\varphi_i).$$

4. Numerical Results

4.1. Test case and comparison criteria

In this section we show the results obtained by approximating the exact solution of the problem described in equation (1) using the methods introduced above. We work on the one dimensional domain $\Omega = (0,1)$ with $F = 1$ and $f = 1$. We choose Robin boundary conditions that mimic Dirichlet conditions and we set $\alpha = 10^{-3}$, $\kappa = 1$, $g_0 = g_1 = 0$. Note that we cannot take $\alpha = 0$ since all variational methods are not well defined for pure Dirchlet boundary conditions. With these choices, the equation reads

$$\begin{cases} -\varepsilon u''(x) + u'(x) = 1, & \forall x \in (0,1), \\ -10^{-3}u'(0) + u(0) = 0, \\ 10^{-3}u'(1) + u(1) = 0. \end{cases} \tag{29}$$

Since we work in dimension 1, we can benefit from the fact that the exact solution u has an analytic form as shown in the Appendix A. We can thus easily compare the approximation quality of the output functions $\hat{u}$ from our methods by computing a discrete version of their $L^2(\Omega)$ error norm with respect to the exact solution.

$$e_{L^2}^2 = \|u - \hat{u}\|_{L^2(\Omega)}^2 \approx \frac{1}{\tilde{K}} \sum_{k=0}^{\tilde{K}-1} (u(x_k) - \hat{u}(x_k))^2 = e_{\ell^2}^2$$

The points x_k are sampled uniformly as defined in 3.3. We use 10 times more points than the ones used for training, so $\tilde{K} = 10K$. Similarly, we also compute the error with respect to the $H^1(\Omega)$ semi-norm.

$$e^2_{H^1} = \|u' - \hat{u}'\|^2_{L^2(\Omega)} \approx \frac{1}{\tilde{K}} \sum_{k=0}^{\tilde{K}-1} (u'(x_k) - \hat{u}'(x_k))^2 = e^2_{h^1}$$

Note that one can obtain the H^1 error by adding the above error components.

We study the impact on the errors of the following parameters:

- The values of ε. They range from 5.10^{-3} to 10.0 with a logarithmic spacing.
- The number K of training points. We consider $K = 10$, 10^2, 10^3, 10^4.
- The choice of the sampling method for the training points (uniformly spaced or uniformly random, labelled as $-u$ and $-r$).
- The impact of the machine precision (Float16, Float32, Float64).

Due to the randomness in the initialization of weights on the neural networks, for each combination of parameters (ε, K, sampling type, and machine precision), we perform 10 repetitions with different initializations. Since we didn't notice a big difference between the l^2 error and the h^1 error, we keep just the second one for clarity and put in the Appendix B the plots in l^2 error.

4.2. Our code and practical implementation details

All our neural network based numerical tests were performed in Python 3.6 and using the TensorFlow 1.13.1 library [20]. The code provided in the original paper on PINNs [21] was used as the starting point for our own code developments, and we have followed similar guidelines to generalize and enlarge it where needed. In the same way, for each numerical method, derivatives of functions $v \in \mathcal{K}$ are computed using automatic differentiation. The numerical optimization procedure used in order to compute an approximation of θ^* a minimizer of problem (26) is given by the quasi-Newton L-BFGS algorithm [22]. The code used to generate the examples shown here is available at

https://github.com/agussomacal/ConDiPINN

The interested reader can reproduce our results and test the impact of the variations of certain parameters such as ε, K, the sampling method, and the machine precision.

4.3. Discussion

4.3.1. *Impact of the number K of training points*

In this section we discuss the impact of the number K of training points. We fix the machine precision to Float32, and the uniform sampling $-u$.

Figure 1 shows the best result obtained in the tests, i.e., the minimum value of the h^1 norm obtained in the 10 different simulations, plotted against the values of ε. In Figure 2, we fix $\epsilon = 10$, and plot statistics on the accuracy e_{h^1} (left plot) and computation runtimes for different K (right plot), and for the different methods.

From these figures, we first notice that the approximation of FEM degrades when ε decreases. However, the quality globally improves when the number of training point increases (see, e.g., Figure 1 - left plot). The rate of improvement is quadratic as we can see from the right plot in Figure 2. In addition, when looking at the runtimes (Figure 2 - right) we observe the expected quadratic increase with respect to the number K of training points.

We can next study the behavior of Vanilla PINN and compare to FEM. We observe that it performs at around constant accuracy for any number of training points until around $\varepsilon = 0.027$ where stops producing reliable approximations (see Figure 1). One remarkable observation is that the Vanilla PINN error for large values of ε and small number of training points $K = 10$ is comparable to the FEM errors with a much larger number of degrees of freedom $K > 10^3$ (see left plot in Figure 2). Regarding the runtime to fit the neural network, we see that it is roughly constant for all values of K and it is comparable to the runtime of the FEM method with $K = 100$ (right plot in Figure 2).

We next comment on the other PINN-based variational methods. For ε large enough, we observe that all the variational based methods follow the same error trend as FEM both with respect to ε and K and for $K < 10^4$ they even perform marginally better. With respect to the computing time, all the methods perform with almost constant time with respect to K and similarly to a FEM method with $K = 100$ degrees of freedom. However, for $\varepsilon < 0.63$, the methods $W - z$, $W - z - e$ and $V - z$ blow up and lose completely their approximation capabilities. We conjecture that this is due to the fact that the neural network is used to approximate the solution z from the transformed problem, and there is an exponential term to go back from z to u (see equation (7)). This may lead to machine precision overflows (in the exponential computation) and underflows (the neural network has to learn very small values of z which also are in the limits of precision). To address this issue, we have explored two possible strategies: one was by directly minimizing over u while maintaining the weak formulation which accounts for the method W. The second approach is to perform the re-scaling of the domain $RW - z$. In both cases the blow up caused by the exponential is solved although the re-scaling method $RW - z$ doesn't perform as good as others in the region with large ε values.

We finish this section by plotting in Figure 3 the best approximated solution for each model, and different values of ε. The interested reader may experiment other configurations in our provided code. The most striking observation is that only FEM and the vanilla PINN method recover the final shape of the exact solution when ε is small. The other variational PINN methods fail despite that some of them exhibit comparable values to FEM in the generalization errors as Figure 1 illustrates. This observations suggests that perhaps other types of error metrics should be introduced in order to be able to better distinguish between "good solution shapes" and "bad ones".

4.3.2. *Impact of Machine Precision*

Figure 4 shows the h^1-error of the different approximated solution by changing the machine precision in the parameters of the neural networks for the different values of ε: Float16, Float32 and Float64. There is an improvement when going from Float16 to Float32 in all methods. Interestingly, we did not obtain very satisfactory results when working with Float64 precision. This precision seems to difficult the convergence to good quality minima: even after 10 repetitions, we failed to find good results. However, as the plots show, when a good minimum is found, it delivers slightly better approximation than lower machine precisions. For these reasons we have performed our experiments using the Float32 which seemed the most stable choice.

4.3.3. *Impact of Sampling Strategy*

Figure 5 shows the h^1-error of the different approximated solution by changing the sampling strategy. For all models, the $uniform$ strategy is found to be either as good as the $random$ or slightly better. For this reason we performed all the experiments using the $uniform$ strategy.

4.4. **Conclusions from the numerical experiments**

The above numerical experiments depict a contrasted landscape concerning the merits and limitations of deep learning-based approaches when the solutions become low regular:

- For large values of ε when solutions are rather regular, some PINNs perform clearly better than FEM regarding the generalization errors. The superiority is particularly remarkable for very small number K of training points. However, the shapes of PINN solutions are sometimes not as satisfactory as the ones given by FEM.
- For the challenging case where ε becomes small and solutions become less regular (which was the main motivation of our study), the accuracy of the variational neural-network methods is essentially comparable or worse to the one given by FEM in terms of generalization errors. Some PINN variational approaches become too unstable and the errors blow up. Only FEM and the vanilla PINN approach seem to be able to recover the correct shape of the exact function. The latter one has however the risk of sometimes falling into local minima with bad shapes.
- The runtimes are clearly in favor to PINN methods as Figure 2 illustrates, and the simplicity of implementation is also in favor to all PINN methods.

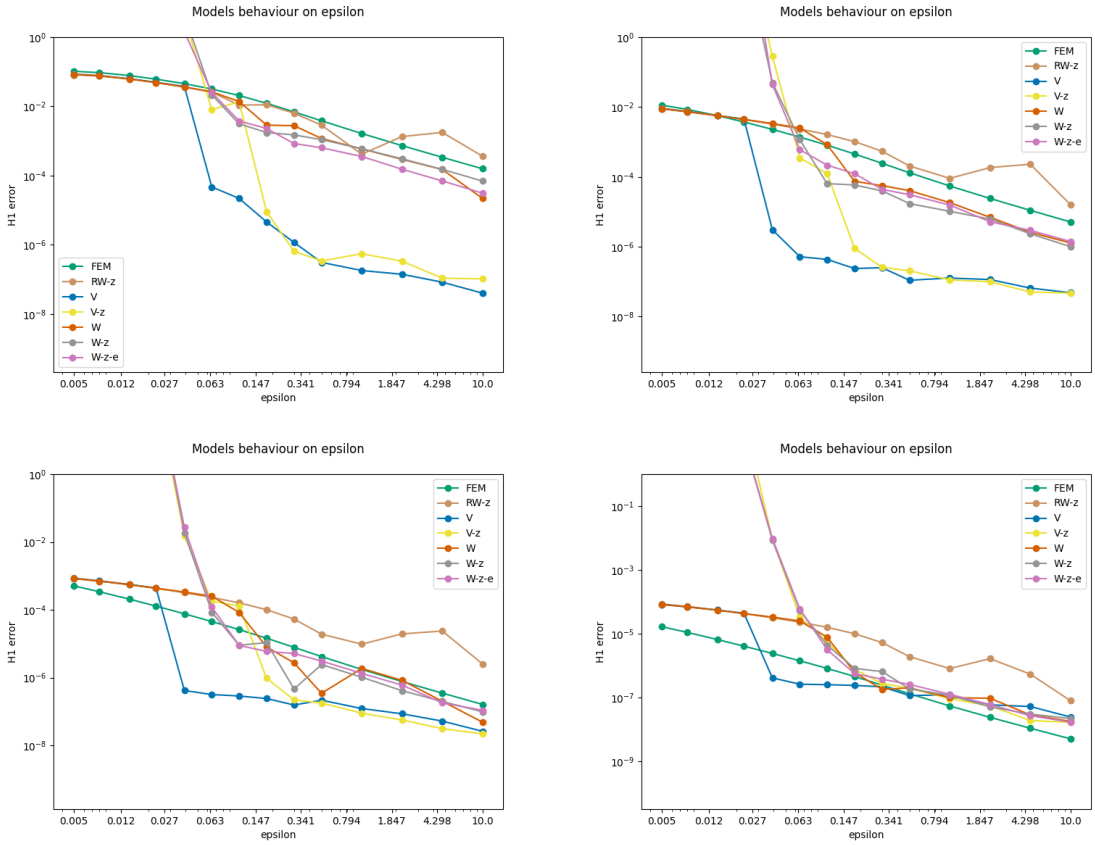


FIGURE 1. Comparison of the behavior of the h^1 error for the different methods and different number of sampling points in training. From top to bottom and from left to right, the first figure is produced for $K = 10$, the second for $K = 100$, the third for $K = 1000$ and the last one for $K = 10000$. The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been fixed to Float32 for all the tests.

5. FUTURE RESEARCH DIRECTIONS AND EXTENSIONS

One important point to explore in future works concerns the choice of the loss function for the training, and also the metric to evaluate generalization errors. It will also be interesting to explore if adaptive sampling strategies during the training could help to recover good solutions in a more stable manner. Finally, the impact of the machine precision in some steps involving exponential transformations seems also to be an important obstacle to retrieving stable solutions. It would be interesting to develop strategies that circumvent this issue. All these developments will play a crucial role in order to address higher dimensional problems with similar characteristics as the one considered here.

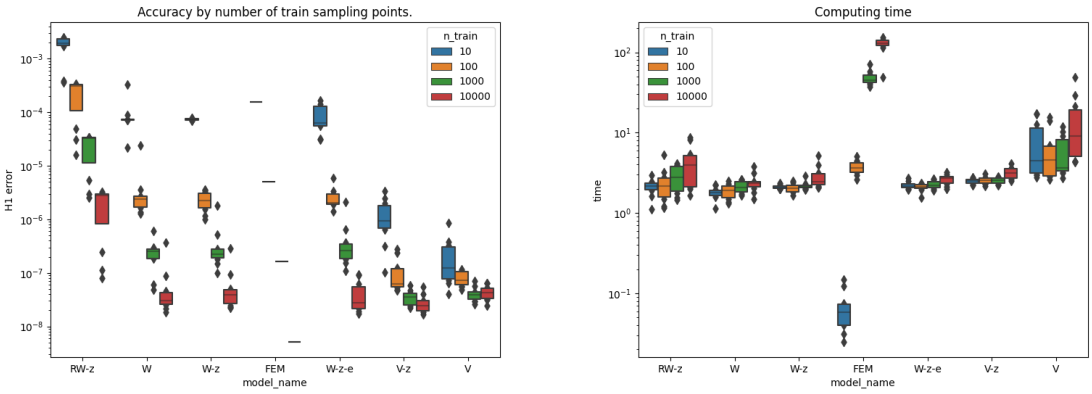


FIGURE 2. For $\varepsilon = 10$ (region where all methods work well), we look at the comparison between methods and the difference with respect to the number of training points K. The h^1 error (left) and the computation times (right). The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been chosen as Float32 for all the tests.

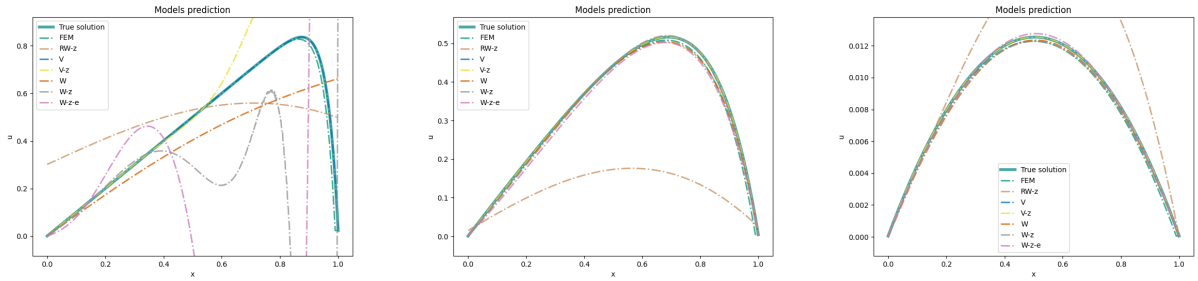


FIGURE 3. The best approximated solution out of 10 repetitions, for each model, and with $K = 100$ training samples. From left to right: $\varepsilon = 0.039$, 0.18, 10. The interested reader may experiment other configurations in our provided code.

A. Analytic solution in dimension 1

The aim of this section is to give the analytic expression of the solution of (1)-(2) in the case when $d = 1$, $\Omega = (0, 1)$, and F and f are assumed to be equal to some constant real numbers. Thus, in this section, using a slight abuse of notation, we assume that $F, f, g \in \mathbb{R}$. Let us also introduce $g_0, g_1 \in \mathbb{R}$ so that $g(0) = g_0$ and $g(1) = g_1$. The problem then reads as follows: find $u : (0, 1) \to \mathbb{R}$ solution to

$$\begin{cases} -\varepsilon u''(x) + Fu'(x) = f, & \forall x \in (0, 1), \\ -\alpha u'(0) + \kappa u(0) = g_0, \\ \alpha u'(1) + \kappa u(1) = g_1. \end{cases} \tag{30}$$

Then, it can be easily checked that the solution to this equation reads as

$$u(x) = C_1 + C_2 e^{\frac{Fx}{\varepsilon}} + \frac{f}{F}x$$

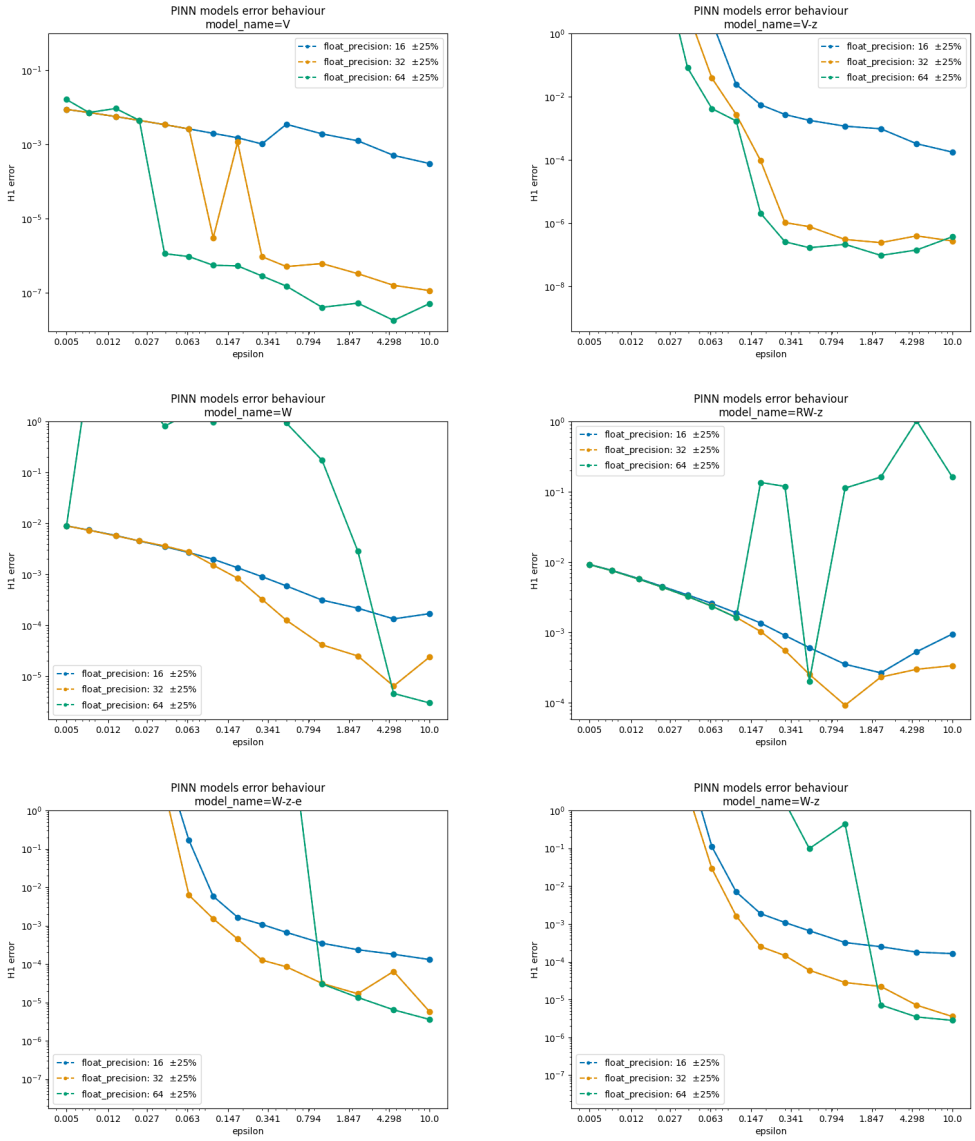


FIGURE 4. Here, the comparison of the behavior of the model for different float precision. The tests have been performed for $K = 100$ and uniform sampling.

where C_1 and C_2 are constants that are determined with the Robin boundary conditions. They satisfy the system

$$\begin{pmatrix} \kappa & \kappa - \frac{\alpha F}{\varepsilon} \\ \kappa & \kappa e^{\frac{F}{\varepsilon}} + \alpha \frac{F}{\varepsilon} e^{\frac{F}{\varepsilon}} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} g_0 + \alpha \frac{f}{F} \\ g_1 - \frac{f}{F}(\kappa + \alpha) \end{pmatrix}$$

which invertible except for

$$\kappa = 0, \quad \text{or} \quad \frac{\alpha}{\kappa} = \frac{\varepsilon(1 - e^{\frac{F}{\varepsilon}})}{F(1 + e^{\frac{F}{\varepsilon}})}.$$

In the following, we assume that the values of κ and α do not take these values, and the above system is invertible.

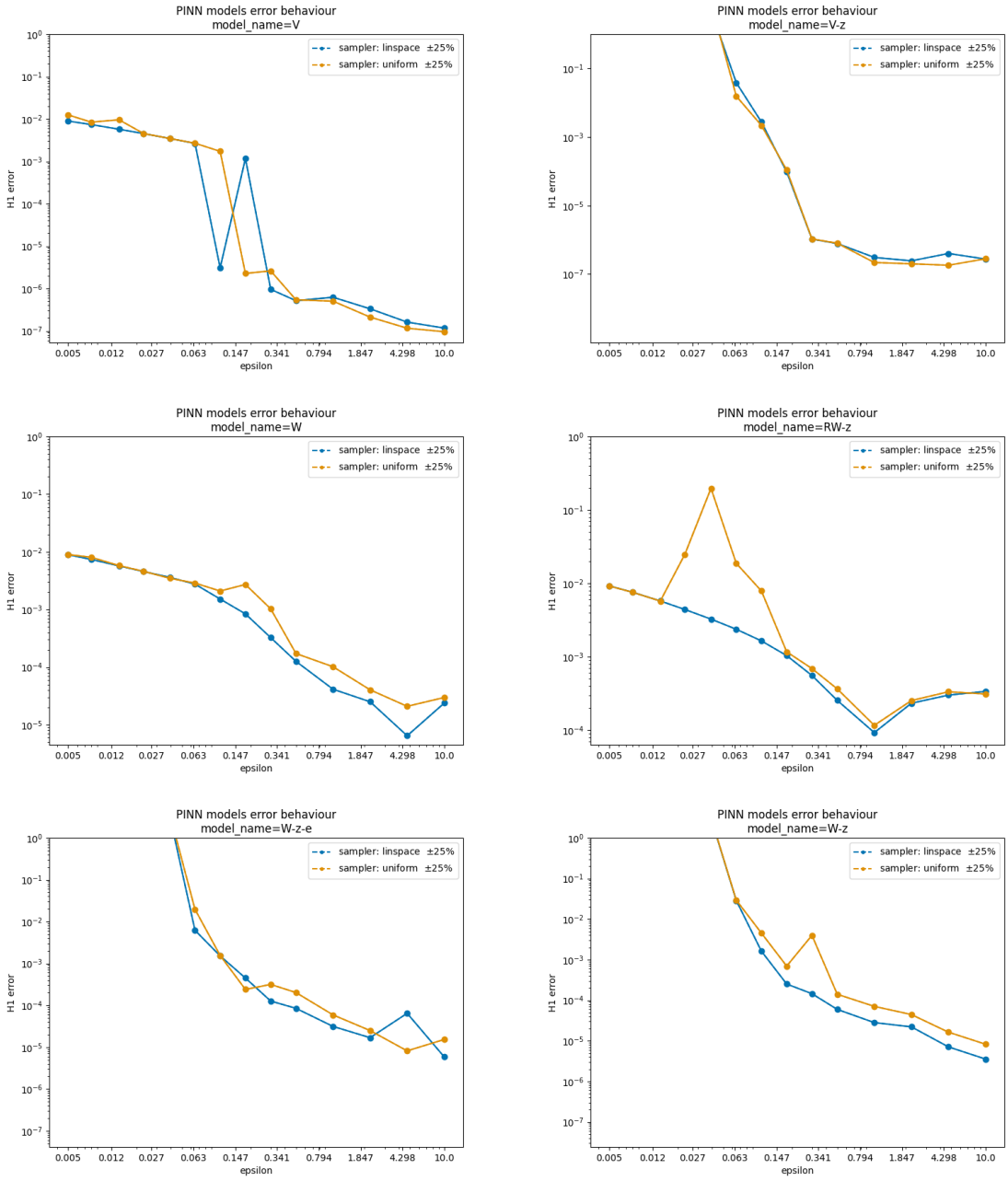


FIGURE 5. Here, the comparison of the behavior of the model for the two different sampling strategies. The tests have been performed for $K = 100$ and the float precision equal to Float32.

B. L2 ERROR PLOTS

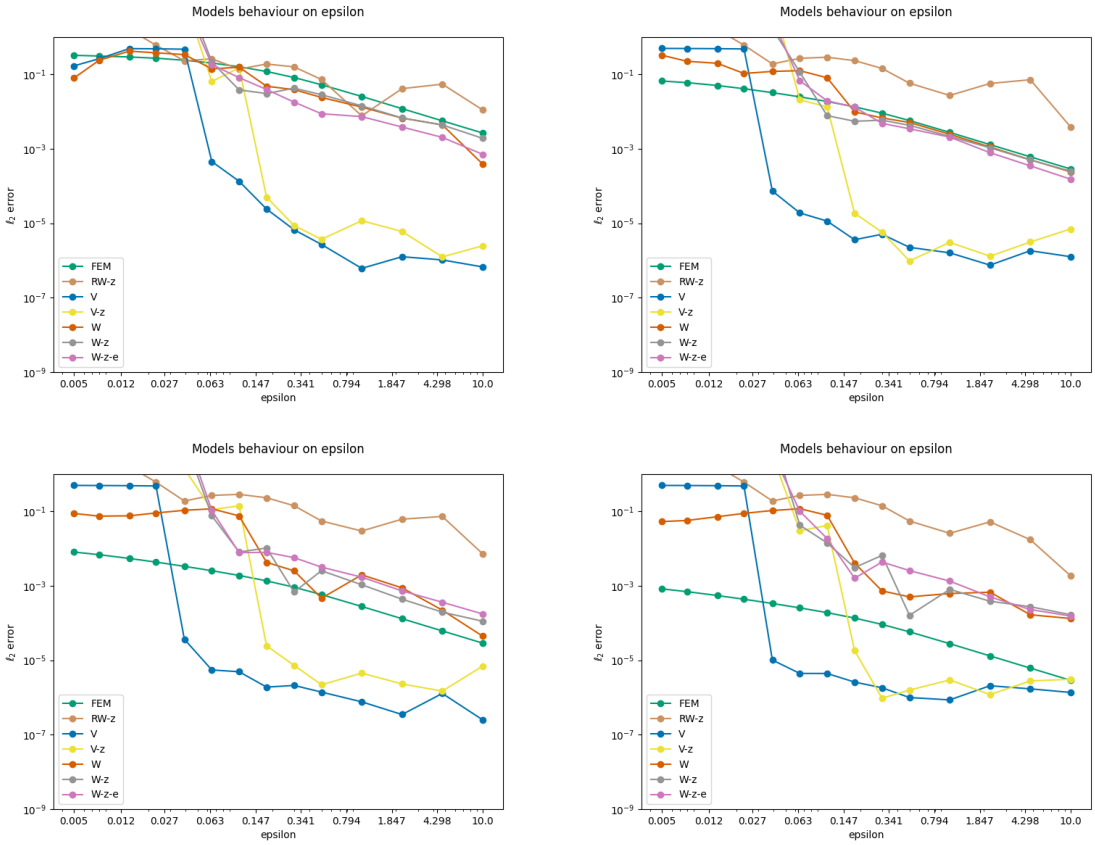


FIGURE 6. The comparison of the behavior of the l^2 error for the different methods and different number of sampling points in training. From up to down and from left to right, the first figure is produced for $K = 10$, the second for $K = 100$, the third for $K = 1000$ and the last one for $K = 10000$. The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been chosen as Float32 for all the tests.

REFERENCES

[1] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

[2] L. Wang and J. M. Mendel. Structured trainable networks for matrix algebra. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 125–132. IEEE, 1990.

[3] B. Després and H. Jourdren. Machine learning design of volume of fluid schemes for compressible flows. *Journal of Computational Physics*, 408:109275, 2020.

[4] 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.

[5] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[6] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

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

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

[9] W. E, J. Han, and A. 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, 2017.
[10] G. M. Rotskoff, A. R. Mitchell, and E. Vanden-Eijnden. Active importance sampling for variational objectives dominated by rare events: Consequences for optimization and generalization. *arXiv preprint arXiv:2008.06334*, 2020.
[11] Hannes Risken. Fokker-planck equation. In *The Fokker-Planck Equation*, pages 63–95. Springer, 1996.
[12] Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the fokker–planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17, 1998.
[13] Vladimir I Bogachev, Nicolai V Krylov, Michael Röckner, and Stanislav V Shaposhnikov. *Fokker-Planck-Kolmogorov Equations*, volume 207. American Mathematical Soc., 2015.
[14] H.-G. Roos, M. Stynes, and L. Tobiska. *Robust numerical methods for singularly perturbed differential equations: convection-diffusion-reaction and flow problems*, volume 24. Springer Science & Business Media, 2008.
[15] A. N Brooks and T. JR. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer methods in applied mechanics and engineering*, 32(1-3):199–259, 1982.
[16] Leszek Demkowicz and Norbert Heuer. Robust DPG method for convection-dominated diffusion problems. *SIAM Journal on Numerical Analysis*, 51(5):2514–2537, 2013.
[17] J. Chan, N. Heuer, T. Bui-Thanh, and L. Demkowicz. A robust dpg method for convection-dominated diffusion problems ii: Adjoint boundary conditions and mesh-dependent test norms. *Computers & Mathematics with Applications*, 67(4):771–795, 2014.
[18] N. Kopteva and E. O'Riordan. Shishkin meshes in the numerical solution of singularly perturbed differential equations. 2010.
[19] Chao Ma, Stephan Wojtowytsch, Lei Wu, et al. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. *arXiv preprint arXiv:2009.10713*, 2020.
[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 2016.
[21] 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.
[22] J. Nocedal D. C. Liu. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45:503–528, 1989.