Improving hp-Variational Physics-Informed Neural Networks for Steady-State Convection-Dominated Problems

Thivin Anandh [1,†]   THIVINANANDH@IISC.AC.IN
Divij Ghose [1,†]   DIVIJGHOSE@IISC.AC.IN
Himanshu Jain [1]   HIMANSHUJ1@IISC.AC.IN
Pratham Sunkad [2]   PRATHAMSUNKAD@SMAIL.IITM.AC.IN
Sashikumaar Ganesan[1]   SASHI@IISC.AC.IN
Volker John [3,4]   JOHN@WIAS-BERLIN.DE

[1] *Department of Computational and Data Sciences*
Indian Institute of Science, Bangalore
Karnataka, India
[2] *Department of Mechanical Engineering*
Indian Institute of Technology, Madras
Tamil Nadu, India
[3] *Numerical Mathematics and Scientific Computing*
Weierstrass Institute for Applied Analysis and Stochastics
Berlin, Germany
[4] *Department of Mathematics and Computer Science*
Freie University
Berlin, Germany
[†] *denotes equal contribution*

Abstract

This paper proposes and studies two extensions of applying hp-variational physics-informed neural networks, more precisely the FastVPINNs framework, to convection-dominated convection-diffusion-reaction problems. First, a term in the spirit of a SUPG stabilization is included in the loss functional and a network architecture is proposed that predicts spatially varying stabilization parameters. Having observed that the selection of the indicator function in hard-constrained Dirichlet boundary conditions has a big impact on the accuracy of the computed solutions, the second novelty is the proposal of a network architecture that learns good parameters for a class of indicator functions. Numerical studies show that both proposals lead to noticeably more accurate results than approaches that can be found in the literature.

Keywords: steady-state convection-diffusion-reaction problems, FastVPINNs, SUPG stabilization, hard-constrained Dirichlet boundary conditions, learning of the indicator function

1 Introduction

Convection-diffusion-reaction (CDR) problems are fundamental models for simulating transport events. CDR problems capture the interaction between a fluid's bulk motion, or convection, its progressive spreading of properties by random molecular motion, or diffusion, and the impact from other quantities in coupled systems, which might be modelled as the

reaction term. They constitute a framework for modelling the transport of variables like temperature or concentration.

Let $\Omega \subset \mathbb{R}^2$ be a bounded domain with polygonal Lipschitz-continuous boundary $\partial\Omega$. The Lebesgue and Sobolev spaces on this domain are denoted by $L^p(\Omega)$ and $W^{k,p}(\Omega)$, respectively, where, $1 \leq p \leq \infty$, $k \geq 0$. The Hilbert space, equivalent to $W^{k,2}(\Omega)$, is denoted by $H^k(\Omega)$. Then, a linear CDR boundary value problem, already in nondimensionalized form, is given by

$$
\begin{aligned}
-\varepsilon \Delta u(\boldsymbol{x}) + \boldsymbol{b}(\boldsymbol{x}) \cdot \nabla u(\boldsymbol{x}) + c(\boldsymbol{x})u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \text{in } \Omega, \\
u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \text{on } \partial\Omega.
\end{aligned}
\tag{1}
$$

Here, $\boldsymbol{x} = (x, y) \in \overline{\Omega}$, $u(\boldsymbol{x})$ is the unknown scalar solution and $f(\boldsymbol{x}) \in L^2(\Omega)$ is a known source function. In addition, $\varepsilon \in \mathbb{R}^+$ is the diffusion coefficient, $\boldsymbol{b} \in (W^{1,\infty}(\Omega))^2$ is the convection field, and $c \in L^\infty(\Omega)$ is the reaction field. The Dirichlet boundary condition is prescribed by $g(\boldsymbol{x}) \in H^{1/2}(\partial\Omega)$.

Of particular interest in many applications is the situation that the convective term dominates the equation. In particular, it is often larger than the diffusive term by several orders of magnitude, i.e., the CDR problem is said to be convection-dominated when $\varepsilon \ll L\|\boldsymbol{b}\|_{L^\infty(\Omega)}$, where L is the characteristic length of the problem. With the Péclet number, Pe, defined as

$$
\text{Pe} = \frac{L\|\boldsymbol{b}\|_{L^\infty(\Omega)}}{\varepsilon},
$$

the convection-dominated regime is often characterized by $\text{Pe} \gg 1$, in many applications $\text{Pe} \gtrsim 10^6$. In the convection-dominated situation, which is also sometimes called singularly perturbed, typical solutions of CDR problems possess so-called layers, which are thin regions where the solution has very steep gradients, e.g., see Roos et al. (2008). It is known from asymptotic analysis that the layer width is $\mathcal{O}(\varepsilon)$ or $\mathcal{O}(\sqrt{\varepsilon})$, depending on the type of the layer.

Classical discretization techniques for CDR problems, based on finite element, finite volume, or finite difference methods, face in the convection-dominated regime the difficulty that the layer width is (much) smaller than the affordable mesh width. Consequently, the layers, which are the most important features of the solution, cannot be resolved. The situation that important features of a solution cannot be resolved is typical for multiscale problems. For convection-dominated CDR problems, the layers are subgrid scales. It is well known that one has to utilize so-called stabilized discretizations to cope with this difficulty. In the framework of finite element methods, one of the earliest but still one of the most popular proposals is the Streamline-Upwind Petrov–Galerkin (SUPG) method from Hughes and Brooks (1979); Brooks and Hughes (1982). The numerical analysis of this linear discretization is well understood, see Roos et al. (2008). However, it requires the choice of a function of stabilization parameters, which is often chosen to be a piecewise constant function on the given triangulation. For these parameters, usually proposals based on a one-dimensional problem with constant coefficients are applied, e.g., see John and Knobloch (2007). However, numerical solutions with such parameters often exhibit spurious oscillations in a vicinity of layers. It is shown in John et al. (2011, 2023) that (in some sense) optimized stabilization parameters might reduce the size of spurious oscillations considerably. A more recent approach for determining appropriate stabilization

parameters is presented in Yadav and Ganesan (2024), which utilizes artificial neural networks. Both techniques, the optimization and the use of neural networks, are inherently nonlinear. The recent review Barrenechea et al. (2024) studied methods that lead to numerical solutions which satisfy discrete maximum principles, i.e., which provide numerical solutions without unphysical values. It is emphasized in this paper that to achieve this property in combination with high accuracy, then one has to apply a nonlinear method, because different approaches have to be applied for the subgrid and for the large scales. As conclusion, convection-dominated CDR problems are a challenging class of problems from the numerical point of view and the accurate and physically consistent numerical solution requires to use nonlinear methods.

In recent times, deep learning based methods are increasingly being used in applied mathematics problems, such as solving boundary value problems with partial differential equations (PDEs). Collectively termed scientific machine learning (SciML), Cuomo et al. (2022); Baker et al. (2019); Psaros et al. (2023), such methods act in conjunction with or entirely replace classical approaches. The field of SciML has witnessed a surge in development, leading to a growing number of accessible SciML libraries, e.g., Lu et al. (2021a); NVIDIA Modulus. The application of such methods has grown exponentially since the introduction of physics-informed neural networks (PINNs) in Lagaris et al. (1998); Raissi et al. (2019). In addition to the typical data-driven loss functional used in neural networks, a PINN incorporates an additional loss term to minimize the residual of the underlying PDE and to enforce the physics-based constraints. The abilities to obtain gradients conveniently with automatic differentiation and to train the same neural network for both forward and inverse modelling are some characteristics that make PINNs an interesting alternative of classical methods, Abueidda et al. (2021); Lu et al. (2021b).

Variational PINNs (VPINNs) are an extension of PINNs that use a weak form of the PDE in the loss functional, see Kharazmi et al. (2019); Khodayi-Mehr and Zavlanos (2020). VPINNs use, as PINNs, a neural network for approximating the solution. The test functions belong to a polynomial function space, which is analogous to the Petrov–Galerkin framework in finite element methods. Usually, the domain Ω is decomposed into subdomains (elements) and the support of each test function is just one subdomain, defining a so-called hp-VPINN. Similar to finite element methods, the accuracy of hp-VPINNs has been shown, for certain problems, to increase by further domain decomposition (h-refinement) or by increasing the order of the polynomial space (p-refinement), see Kharazmi et al. (2021).

However, the standard use of hp-VPINNs shows some bottlenecks and limitations. A bottleneck is the high computational expense of the training, e.g., as reported in Frerichs-Mihov et al. (2024a), which increases in direct proportion to the number of elements. Furthermore, many current hp-VPINNs implementations face difficulties with complex geometries, which include skewed quadrilateral elements, which are prevalent in real-world scenarios. To overcome these issues, FastVPINNs, as introduced in Anandh et al. (2024b,a); Ghose et al. (2024), employs a tensor-based method for loss functional computations, coupled with the implementation on GPUs, leading to a remarkable reduction in training time of order $\mathcal{O}(100)$ compared to conventional hp-VPINNs. Additionally, FastVPINNs use bilinear transformations, allowing to simulate problems on complex geometries with skewed quadrilateral elements.

Despite all the positive aspects of PINNs and hp-VPINNs, it has been observed in Krishnapriyan et al. (2021) that these approaches face difficulties when applied to singularly perturbed problems, like convection-dominated CDR problems. Many contributions that study PINNs or (hp-)VPINNs for CD(R) problems consider either the (much more) simpler one-dimensional case or only a mildly convection-dominated regime, see the introduction of Frerichs-Mihov et al. (2024a) for a survey of corresponding works. The choice of examples for the current paper is guided by Frerichs-Mihov et al. (2024a,b); Matthaiou et al. (2024), where strongly convection-dominated problems are studied. The emphasis of these papers was on using loss functionals that are alternative to the standard residual loss, the selection of collocation points in PINNs, and on approaches that preserve given bounds of the solution. For two examples studied in Frerichs-Mihov et al. (2024a,b) the obtained results were characterized as nonsatisfactory.

The first contribution of this paper consists in proposing alternative loss functionals, compared with the standard residual loss, for convection-dominated problems. A term in the spirit of the SUPG stabilization is introduced, which contains a stabilization parameter. In addition, the effect of including a regularization with respect to the weights of the neural network is studied. A first main novelty is the proposal of a FastVPINNs architecture that learns spatially varying stabilization parameters. It is shown that this approach leads to noticeable improvements of the accuracy of the numerical solutions. The second main contribution is on the realization of hard-constrained Dirichlet boundary conditions, more precisely, on the definition of the indicator function that separates the extension of the boundary conditions and the function that is learnt by the network. Also for this task, a FastVPINNs architecture is proposed that learns the parameters for a class of indicator functions. The adaptively chosen parameters lead to a considerable increase of the accuracy compared with manually chosen parameters.

The paper is organized as follows. Section 2 describes the general setup of the FastVPINNs framework. The loss functionals are introduced in Section 3. Numerical studies with constant stabilization parameters and with manually chosen indicator functions are presented in Section 4. The two main novelties, the FastVPINNs architectures for spatially varying stabilization parameters and for the adaptive choice of the parameters in the indicator function, are introduced and studied in Section 5. Finally, Section 6 summarizes the main results of this paper.

2 Setup of the FastVPINNs Framework

This section starts by introducing the variational form of the CDR problem (1). Then, the fast hp-VPINNs are described.

Let

$$V := \left\{ v \in H^1(\Omega) \ : \ v = 0 \text{ on } \partial\Omega \right\},$$

then the variational form is obtained in the usual way by multiplying equation (1) with an arbitrary test function $v \in V$, integrating over Ω, and then utilizing integration by parts on the second order derivative term, e.g., see Ganesan and Tobiska (2017). Given an extension $u_{\text{ext}} \in H^1(\Omega)$ of the boundary condition into Ω. Then, the variational form of (1) reads as

4

follows: Find $u \in H^1(\Omega)$ such that $u - u_{\text{ext}} \in V$ and

$$a(u, v) = f(v) \quad \forall \, v \in V,$$

where the bilinear form $a(\cdot, \cdot) \; H^1(\Omega) \times V \to \mathbb{R}$ is defined as

$$a(u, v) := \int_\Omega \varepsilon \nabla u \cdot \nabla v \; \mathrm{d}\boldsymbol{x} + \int_\Omega \boldsymbol{b} \cdot \nabla u \; v \; \mathrm{d}\boldsymbol{x} + \int_\Omega c u \, v \; \mathrm{d}\boldsymbol{x},$$
$$f(v) := \int_\Omega f \, v \; \mathrm{d}\boldsymbol{x}. \tag{2}$$

For defining a hp-VPINN, the domain Ω is then divided into an array of non-overlapping cells, labeled as K_k, where $k = 1, 2, \ldots, N_{\text{elem}}$, such that their union $\bigcup_{k=1}^{N_{\text{elem}}} K_k = \overline{\Omega}$. Let V_h be a finite-dimensional subspace of V, spanned by N_{test} basis functions. As a result, the discretized form of (2) can be written as follows: Find u_h such that $u_h - u_{\text{ext},h} \in V_h$ and

$$a_h(u_h, v_h) = f_h(v_h) \quad \forall \, v_h \in V_h, \tag{3}$$

where $u_{\text{ext},h}$ is an appropriate discrete extension of the Dirichlet boundary condition and

$$a_h(u_h, v_h) := \sum_{k=1}^{N_{\text{elem}}} \int_{K_k} \varepsilon \nabla u_h \cdot \nabla v_h \; \mathrm{d}\boldsymbol{x} + \sum_{k=1}^{N_{\text{elem}}} \int_{K_k} \boldsymbol{b} \cdot \nabla u_h \; v_h \; \mathrm{d}\boldsymbol{x}$$
$$+ \sum_{k=1}^{N_{\text{elem}}} \int_{K_k} c u_h \; v_h \; \mathrm{d}\boldsymbol{x},$$
$$f_h(v_h) := \sum_{k=1}^{N_{\text{elem}}} \int_{K_k} f \; v_h \; \mathrm{d}\boldsymbol{x}.$$

These integrals are approximated by employing numerical quadrature:

$$\int_{K_k} \varepsilon \nabla u_h \cdot \nabla v_h \; \mathrm{d}\boldsymbol{x} \approx \sum_{q=1}^{N_{\text{quad}}} w_q \; \varepsilon \nabla u_h(\boldsymbol{x}_q) \cdot \nabla v_h(\boldsymbol{x}_q) \; ,$$

$$\int_{K_k} \boldsymbol{b} \cdot \nabla u_h \; v_h \; \mathrm{d}\boldsymbol{x} \approx \sum_{q=1}^{N_{\text{quad}}} w_q \; \boldsymbol{b} \cdot \nabla u_h(\boldsymbol{x}_q) \; v_h(\boldsymbol{x}_q) \; ,$$

$$\int_{K_k} c u_h \; v_h \; \mathrm{d}\boldsymbol{x} \approx \sum_{q=1}^{N_{\text{quad}}} w_q \; c u_h(\boldsymbol{x}_q) \; v_h(\boldsymbol{x}_q) \; ,$$

$$\int_{K_k} f \; v_h \; \mathrm{d}\boldsymbol{x} \approx \sum_{q=1}^{N_{\text{quad}}} w_q \; f(\boldsymbol{x}_q) \; v_h(\boldsymbol{x}_q) \; .$$

Here, N_{quad} is the number of quadrature points in a cell, $\{\boldsymbol{x}_q\}$ is the set of quadrature nodes, and $\{w_q\}$ is the set of weights (which include the area of K_k).

The hp-VPINN framework (Kharazmi et al., 2021) utilizes specific test functions v_k, where k ranges from 1 to N_{elem}, which are localized and defined within individual non-overlapping cells

$$v_k = \begin{cases} v^p \neq 0, & \text{in } K_k, \\ 0, & \text{elsewhere.} \end{cases}$$

Here, v^p represents a polynomial function of degree p. This selection of test and solution spaces resembles a Petrov–Galerkin finite element method.

By utilizing these functions, we define the cell-wise residual of the variational form (3) with $u_{\text{NN}}(\boldsymbol{x}; \theta_W, \theta_b)$ by

$$\mathcal{W}_k(\theta_W, \theta_b) = \sum_{q=1}^{N_{\text{quad}}} w_q \big[\varepsilon \nabla u_{\text{NN}}(\boldsymbol{x}_{kq}) \cdot \nabla v_k(\boldsymbol{x}_{kq}) + \boldsymbol{b} \cdot \nabla u_{\text{NN}}(\boldsymbol{x}_{kq}) \, v_k(\boldsymbol{x}_{kq}) \tag{4}$$
$$+ c \, u_{\text{NN}}(\boldsymbol{x}_{kq}) \, v_k(\boldsymbol{x}_{kq}) - f(\boldsymbol{x}_{kq}) \, v_k(\boldsymbol{x}_{kq}) \big] \, .$$

Now, the variational loss is given by

$$\mathcal{L}_{\text{var}}(\theta_W, \theta_b) = \frac{1}{N_{\text{elem}}} \sum_{k=1}^{N_{\text{elem}}} |\mathcal{W}_k(\theta_W, \theta_b)|^2 \, . \tag{5}$$

There are two conceptually different ways of incorporating Dirichlet boundary conditions in the framework of hp-VPINNs. First, these boundary conditions can be learnt. To this end, a term is included in the loss functional that measures the difference between the prescribed boundary values and the values of the neural network solution $u_{\text{NN}}(\boldsymbol{x}; \theta_W, \theta_b)$ taken at the boundary. In this approach, the Dirichlet boundary conditions are satisfied usually only approximately. This is, in our opinion, not appropriate in the case of convection-dominated CDR problems. First, conditions posed on the inlet boundary $(\boldsymbol{b} \cdot \boldsymbol{n})(\boldsymbol{x}) < 0$, $\boldsymbol{x} \in \partial\Omega$, where $\boldsymbol{n}$ is the outward pointing unit normal vector at $\partial\Omega$, are transported in the domain and thus are essential for determining the correct form of the solution inside the domain. And second, boundary layers, which are an essential feature of the solution, might not be present in $u_{\text{NN}}(\boldsymbol{x}; \theta_W, \theta_b)$ if the boundary condition is only approximately satisfied. In addition, one has to choose a weight for the boundary term that relates the importance of this term to other terms in the loss functional. For these reasons, we decided to pursue the alternative approach of using a hard-constrained imposition of the boundary conditions in the hp-VPINNs. With this approach, the boundary conditions, which are given data of the problem, are satisfied exactly.

The ansatz function for using hard-constrained boundary conditions, see Lu et al. (2021b), is

$$u_{\text{NN}}^{\text{hard}}(\boldsymbol{x}; \theta_W, \theta_b) = j(\boldsymbol{x}) + h(\boldsymbol{x})u_{\text{NN}}(\boldsymbol{x}; \theta_W, \theta_b), \tag{6}$$

Here, $j(\boldsymbol{x})$ is a continuous function in Ω that satisfies $j(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$ and $h(\boldsymbol{x})$ is an indicator function satisfying

$$h(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in \partial\Omega, \\ > 0, & \text{elsewhere.} \end{cases} \tag{7}$$
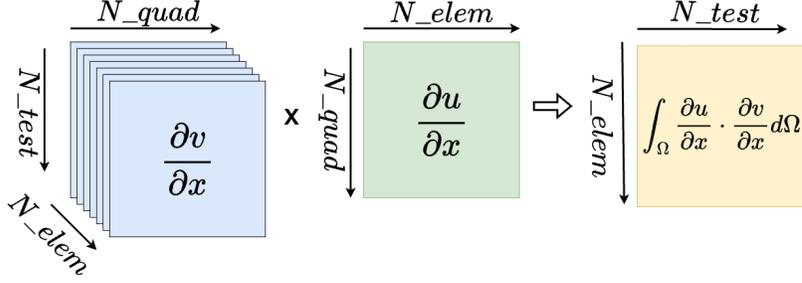
Figure 1: Tensor-based loss computation schematic for FastVPINNs

For example, the indicator function used in Frerichs-Mihov et al. (2024a) has the form

$$h(\boldsymbol{x}) = \left(1 - \mathrm{e}^{-\kappa x}\right)\left(1 - \mathrm{e}^{-\kappa y}\right)\left(1 - \mathrm{e}^{-\kappa(1-x)}\right)\left(1 - \mathrm{e}^{-\kappa(1-y)}\right), \tag{8}$$

where κ depends on the diffusion coefficient ε, but not on the shape of the solution close to the boundary. The current paper employs different indicator functions for different examples, which are detailed in the respective numerical sections. In addition, we study a method where the parameters for a function of type (8) are obtained from the neural network, in Section 5. Now, the representation (6) is inserted in (4) to get the residual $\mathcal{W}_k^{\mathrm{hard}}(\theta_W, \theta_b)$, which then can be substituted in (5) to define the cost functional of the neural network by

$$\mathcal{L}_{\mathrm{var}}^{\mathrm{hard}}(\theta_W, \theta_b) = \frac{1}{N_{\mathrm{elem}}} \sum_{k=1}^{N_{\mathrm{elem}}} \left| \mathcal{W}_k^{\mathrm{hard}}(\theta_W, \theta_b) \right|^2. \tag{9}$$

FastVPINNs computes the variational loss (9) by stacking the test functions and their gradients into a three-dimensional tensor and reshaping the neural network gradients into a matrix, see Figure 1, thus enabling the calculation of the final residual vector for all elements in a single operation using BLAS routines available in TensorFlow Abadi et al. (2015). This approach has two main advantages: using BLAS operations on the GPU allows for efficient and fast calculations, and stacking the gradients and performing tensor-based operations enable the loss for the entire domain to be computed in a single calculation, reducing the training time's dependency on the number of elements, which is a major challenge in conventional hp-VPINNs frameworks. Additionally, FastVPINNs can handle complex geometries, making this method suitable for a wide range of problems. For further details on the implementation of the FastVPINNs framework, see Anandh et al. (2024a).

3 Loss Functional

The main topic of Frerichs-Mihov et al. (2024a) is the investigation of different loss functionals for PINNs and hp-VPINNs applied to the numerical solution of convection-dominated CDR problems. The motivation for studying different loss functionals stems from the observation that the standard residual is not an appropriate choice for performing parameter optimization in stabilized finite element methods, see John et al. (2011). It was observed

in Frerichs-Mihov et al. (2024a) that it was possible to obtain more accurate solutions with loss functionals that are different to the standard residual loss.

In the current paper, two alternative loss functionals will be studied.

3.1 SUPG Stabilization Loss

The SUPG finite element method, which is already mentioned in the introduction, adds an additional term to the standard Galerkin finite element discretization. This term essentially introduces numerical diffusion in streamline direction. The global SUPG stabilization term has the form

$$\mathcal{L}_{\mathcal{SUPG}} = \int_{\Omega} \tau(\boldsymbol{x}) \left(\boldsymbol{b} \cdot \nabla u(\boldsymbol{x}) + cu(\boldsymbol{x}) - f(\boldsymbol{x})\right) \left(\boldsymbol{b} \cdot \nabla v(\boldsymbol{x})\right) \ \mathrm{d}\boldsymbol{x}, \tag{10}$$

where $\tau(\boldsymbol{x})$ is called stabilization parameter and the diffusive term is neglected in the residual (the first factor), which is appropriate in the convection-dominated regime. In the framework of finite elements, (10) is localized by decomposing the integral in a sum of integrals over the mesh cells and then $\tau(\boldsymbol{x})$ is often defined to be a piecewise constant function on the given triangulation. In the framework of neural networks, the SUPG stabilization loss functional is given by

$$\mathcal{L}_{\tau}^{\mathcal{SUPG}} = \mathcal{L}_{\mathrm{var}}^{\mathrm{hard}} + \mathcal{L}_{\mathcal{SUPG}}, \tag{11}$$

where $\mathcal{L}_{\mathrm{var}}^{\mathrm{hard}}$ is the variational loss defined in (9). Minimizing the loss functional (11) penalizes large streamline derivatives. Hence, the additional SUPG loss can be also interpreted as a physics-informed regularization term, since functions with small residuals but large streamline derivatives are excluded from the set of possible solutions. It is well known that regularization techniques might help to prevent overfitting. An architecture of the network for the loss functional including the SUPG term with prescribed stabilization parameter is presented in Figure 2.

3.2 L^2 Weight Regularization Loss

The training of a neural network requires the solution of a large-scale non-convex optimization problem. For such problems, usually a regularization term is included in the loss functional. In addition, this term might counteract overfitting and thus it enhances the generalization capacity of the network. A standard approach consists in adding a L^2-type regularization, so that the loss functional becomes

$$\mathcal{L}_{\lambda}^{\mathrm{reg}} = \mathcal{L}_{\mathrm{var}}^{\mathrm{hard}} + \frac{\lambda}{N} \sum_{j} w_j^2, \tag{12}$$

where N is the total number of entries in the weight matrices of the network and $\{w_j\}_{j=1}^{N}$ is the set of all weights in the network. The L^2 weight decay regularization parameter λ needs to be tuned.

4 Numerical Studies with Constant Parameters

In this section, we first validate the FastVPINNs code by solving the Eriksson–Johnson problem, as presented in Sikora et al. (2023), by comparing accuracy and computing times.
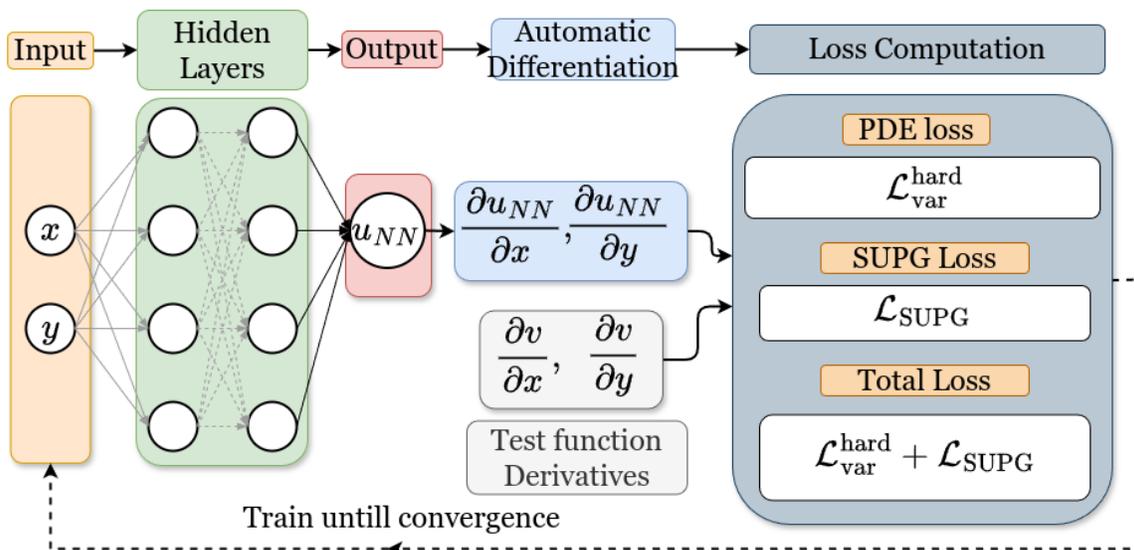
Figure 2: hp-VPINNs architecture for convection-dominated problems with SUPG stabilization, prescribed stabilization parameter, and hard constraints

Subsequently, we address two convection-dominated problems, assessing the performance of the proposed loss functionals (11) and (12) with constant stabilization and regularization parameter, respectively, within the framework of hp-VPINNs and we compare the results and efficiency with those reported in the literature.

All simulations were conducted on a system equipped with an AMD Threadripper CPU and an NVIDIA A6000 GPU. For neural network training, we utilized TensorFlow 2.0, specifically employing the `tf.float64` datatype to ensure high precision. The network architecture incorporates `tanh` activation functions, and we used the Adam optimizer for the training iterations. We adopted test functions similar to those described in Kharazmi et al. (2021). These functions are defined as $v_k = P_{k+1} - P_{k-1}$, where P_k represents the kth order Legendre polynomial. For numerical integration, we implemented Gauss–Lobatto–Legendre quadrature routines as used in Kharazmi et al. (2021).

4.1 Studied Problems

THE ERIKSSON–JOHNSON PROBLEM (P_{EJ})

This problem, originally proposed in Eriksson and Johnson (1991), was used in Sikora et al. (2023) for assessing a PINN for (mildly) convection-dominated problems. It is defined by $\Omega = (0,1)^2$, $\boldsymbol{b} = (1,0)^T$, $c = 0$, and the right-hand side of (1) is chosen such that the solution has the form

$$u(\boldsymbol{x}) = \frac{e^{r_1(x-1)} - e^{r_2(x-1)}}{e^{-r_1} - e^{-r_2}} \sin(\pi y) \quad \text{with}$$

$$r_1 = \frac{1 + \sqrt{1 + 4\varepsilon^2\pi^2}}{2\varepsilon}, \quad r_2 = \frac{1 - \sqrt{1 + 4\varepsilon^2\pi^2}}{2\varepsilon}.$$
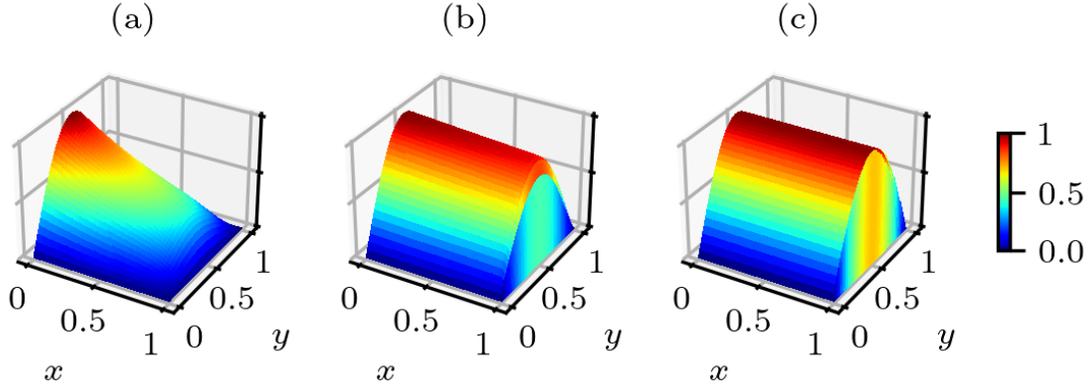
9

Figure 3: Exact Solution of P_{EJ} for (a) $\varepsilon = 0.1$, (b) $\varepsilon = 0.01$ and (c) $\varepsilon = 0.001$

Dirichlet conditions were imposed on $\partial\Omega$. Solutions for various values of the diffusion coefficient are displayed in Figure 3. The smallest value of the diffusion coefficient considered in Sikora et al. (2023) is $\varepsilon = 0.001$. For small values of the diffusion coefficient, the solution exhibits a boundary layer at $x = 1$.

SOLUTION WITH OUTFLOW BOUNDARY LAYERS (P_{OUT})

This example, proposed in John et al. (1997), features two layers at the outflow boundary, which, according to an asymptotic analysis, have a width of $\mathcal{O}(\varepsilon)$. The data for problem (1) are given by $\Omega = (0,1)^2$, $\varepsilon = 10^{-8}$, $\mathbf{b} = (2,3)^T$, $c = 1$, and the prescribed solution has the form:

$$
\begin{aligned}
u(x,y) := xy^2 - y^2 \exp\left(\frac{2(x-1)}{\varepsilon}\right) - x \exp\left(\frac{3(y-1)}{\varepsilon}\right) \\
+ \exp\left(\frac{2(x-1) + 3(y-1)}{\varepsilon}\right).
\end{aligned}
\tag{13}
$$

Dirichlet boundary conditions are imposed on $\partial\Omega$. By construction, the outflow boundary layers are situated at $x = 1$ and at $y = 1$, and there is even a so-called corner singularity at the right upper corner of the domain. Figure 4(a) depicts the prescribed solution (13).

PARABOLIC LAYER PROBLEM (P_{PARA})

The data for problem (1) are given by $\Omega = (0,1)^2$, $\varepsilon = 10^{-8}$, $\boldsymbol{b} = (1,0)^T$, $c = 0$, and $f = 1$. Homogeneous Dirichlet conditions are prescribed at $\partial\Omega$. The solution exhibits an exponential layer at the outflow boundary $x = 1$ and two parabolic boundary layers at the characteristic boundaries $y = 0$ and $y = 1$, as shown in Figure 4 (b).

To obtain a high accuracy solution for this problem to compare with, we employed the Monotone Upwind-type Algebraically Stabilized (MUAS) method. The MUAS method, formulated in John and Knobloch (2022), is an algebraically stabilized scheme. It has been shown in the survey Barrenechea et al. (2024) that algebraically stabilized scheme are currently the most promising finite element approaches for computing solutions of convection-
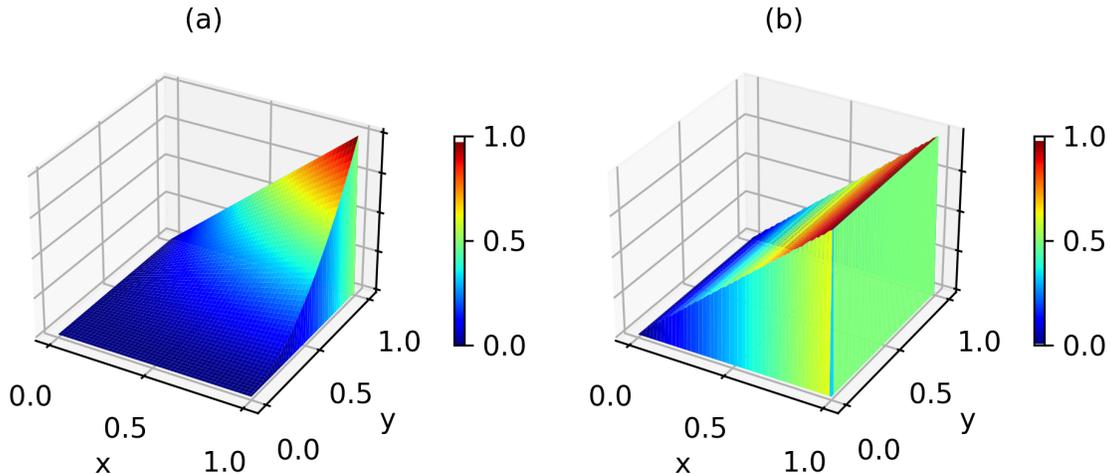
Figure 4: Exact solution for (a) P_{out} and (b) P_{para}

dominated convection-diffusion-reaction problems that satisfy Discrete Maximum Principles.

4.2 Validation of the FastVPINNs Implementation and Comparison with the Literature

In this section, we consider the Eriksson–Johnson problem P_{EJ} as studied in Sikora et al. (2023) and compare the available data with our results. The approach in Sikora et al. (2023) utilizes a single element with 6400 quadrature points and a weak boundary constraint for the Dirichlet boundaries. In contrast, our hp-VPINNs method employs an 8×8 element domain with 9 test functions and 100 quadrature points per element, i.e., in total also 6400 quadrature points, and it uses a hard boundary constraint for imposing Dirichlet conditions. Despite these differences, both approaches share common features in their neural network architecture and training process. Specifically, both use a neural network with 4 hidden layers and 20 neurons per hidden layer, employ the Adam optimizer for training with a learning rate of 0.00125, and run for $40,000$ training epochs. In this study, we have used $\varepsilon \in \{0.1, 0.01, 0.001\}$, as it was done in Sikora et al. (2023).

The indicator functions for the hard constraints (6) were chosen to be

$$
\begin{aligned}
j(\boldsymbol{x}) &= \sin(\pi y)\cos\left(\frac{\pi}{2}x\right), \\
h(\boldsymbol{x}) &= \left(1 - e^{-\kappa_1 x}\right)\left(1 - e^{-\kappa_1 y}\right)\left(1 - e^{-\kappa_2(1-x)}\right)\left(1 - e^{-\kappa_1(1-y)}\right)
\end{aligned}
$$

with $\kappa_1 = 30$ and $\kappa_2 = 10/\varepsilon$. That means, for the indicator function $h(\boldsymbol{x})$ we prescribed a sharp boundary layer at the boundary $x = 1$ to account for the corresponding layer of the solution. This is not necessary at the other parts of the boundary. The standard loss functional $\mathcal{L}_{var}^{hard}$ defined in (9), which takes into account the hard constraint boundary conditions, was used.

11

Table 1: Eriksson–Johnson problem P_{EJ}, L^2_{err} and comparison with the literature

quadrature points	ε	FastVPINNs	Sikora et al. (2023)
6400	0.1	$1.400 \cdot 10^{-3}$	0.265
6400	0.01	$1.440 \cdot 10^{-2}$	0.346
6400	0.001	$7.000 \cdot 10^{-4}$	0.332

Table 2: Eriksson–Johnson problem P_{EJ}, average training time per epoch in s and comparison with the literature

FastVPINNs (CPU)	FastVPINNs (GPU)	(Sikora et al., 2023, Fig. 14)
0.011	0.0027	0.0375

Tables 1 and 2 present the obtained results. It can be seen that, on the one hand, the application of FastVPINNs gives results that are more accurate by several orders of magnitude than the results from Sikora et al. (2023). Here, the approximation L^2_{err} of the error in $L^2(\Omega)$ is defined by

$$L^2_{err} = \left(\frac{1}{N_T} \sum_{t=1}^{N_T} \left(\left(u - u_{NN}^{hard} \right) (\boldsymbol{x}_t) \right)^2 \right)^{1/2}, \tag{14}$$

where N_T is the number of test points at which the error is calculated. In all of the examples, the test points $\boldsymbol{x}_t$ are given by a 100×100 grid of equidistant points in $\overline{\Omega}$. And on the other hand, the training times are considerably smaller, by a factor of about 3 on CPUs and by a factor of 14 if GPUs are used.

The obtained results validate our implementation of FastVPINNs. In our opinion, the improved accuracy compared with the literature can be attributed to two factors. First, the use of hard constraints for the boundary conditions, which seems to be essential for boundary layer problems, and second the h-refinement approach of hp-VPINNs.

4.3 Studies with the SUPG Stabilization and the Regularization with Constant Parameters

In this section, we analyze the performance of the SUPG-based loss functional and loss functional with regularization applied to problems P_{out} and P_{para}. Concretely, our studies explore the SUPG stabilization $\mathcal{L}_\tau^{\mathcal{SUPG}}$ defined in (11) and PDE loss with regularization $\mathcal{L}_\lambda^{reg}$ given in (12). For the simulations, we employed a neural network architecture consisting of seven hidden layers with 30 neurons each. The domain was discretized into 64 cells, with 100 quadrature points and 36 test functions per cell, resulting in a total of 6400 quadrature points. To conduct the experiments, learning rates of $0.01 \cdot 3^{-2}$ and $0.01 \cdot 3^{-3}$ have been used. The L^2_{err} is calculated as given in (14). The network is trained for 100,000 epochs, and the smallest error obtained during this training, i.e., after each epoch and not necessarily after the final epoch, is reported below. The used parameters are similar to those used in the studies described in Frerichs-Mihov et al. (2024a). For both problems, we applied an

Table 3: Problem P_{out}, results for using constant parameters in the SUPG loss or regularization loss; best results reported in Frerichs-Mihov et al. (2024a): $3.419 \cdot 10^{-2}$ (PINN) and $3.619 \cdot 10^{-2}$ (hp-VPINN)

loss	search range	optimal value	L_{err}^2
$\mathcal{L}_\lambda^{\text{reg}}$	$[10^{-5}, 10^{-3}]$	$2.4 \cdot 10^{-4}$ (λ)	$3.202 \cdot 10^{-2}$
$\mathcal{L}_\tau^{\mathcal{SUPG}}$	$[10^{-5}, 10^{-1}]$	$3.5 \cdot 10^{-2}$ (τ)	$\mathbf{1.973 \cdot 10^{-2}}$

Table 4: Problem P_{para}, results for using constant parameters in the SUPG loss or regularization loss

loss	search range	optimal value	L_{err}^2
$\mathcal{L}_\lambda^{\text{reg}}$	$[10^{-8}, 10^{-2}]$	$4 \cdot 10^{-6}$ (λ)	$4.888 \cdot 10^{-1}$
$\mathcal{L}_\tau^{\mathcal{SUPG}}$	$[10^{-5}, 5]$	1.2 (τ)	$\mathbf{5.384 \cdot 10^{-2}}$

indicator function as presented in Frerichs-Mihov et al. (2024a):

$$h(x, y) = \left(1 - e^{-\kappa x}\right)\left(1 - e^{-\kappa y}\right)\left(1 - e^{-\kappa(1-x)}\right)\left(1 - e^{-\kappa(1-y)}\right),$$
$$\kappa = 10/10^{-8} = 10^9. \tag{15}$$

Our numerical studies were conducted in two phases. First, we performed an initial broad-range hyperparameter search, computing L_{err}^2 for a wide range of hyperparameters, as detailed in Tables 3 and 4. Following this, we conducted a refined hyperparameter search based on the best performing values from the initial search. The optimal hyperparameters for each loss functional are presented in aforementioned tables and the corresponding solutions and point-wise errors in Figure 5. For both problems, it can be seen that the SUPG stabilization ($\mathcal{L}_\tau^{\mathcal{SUPG}}$) outperformed the approach that uses only regularization ($\mathcal{L}_\lambda^{\text{reg}}$). In particular, for problem P_{out} our methods achieved lower errors compared to the PINNs and hp-VPINNs simulations performed in Frerichs-Mihov et al. (2024a). However, on the other hand, both results are not yet completely satisfactory. There is a comparatively big error in the right lower corner of the domain for the numerical solution of P_{out} and the contour lines of the numerical solution of P_{para} are not parallel to the y-axis. But all in all, we think that the obtained results demonstrate that including the SUPG stabilization term is an appropriate way for enhancing the accuracy of variational physics-informed neural networks.

Next, results of a sensitivity analysis concerning the accuracy of the numerical solutions for variations of the parameter τ in $\mathcal{L}_\tau^{\mathcal{SUPG}}$ will be presented. It is important to note that even when all parameters are kept constant, the minimum error obtained varies noticeably across multiple runs of the simulations. To obtain a more representative picture, we repeated the simulations 5 times and used the average of the minimum errors to plot the values shown in Figure 6. It is important to clarify that the errors mentioned in Tables 3 and 4 are the result of the best run only. For P_{out}, it can be seen that there is a certain, but relatively small, interval for the stabilization parameter where the obtained errors are close to the
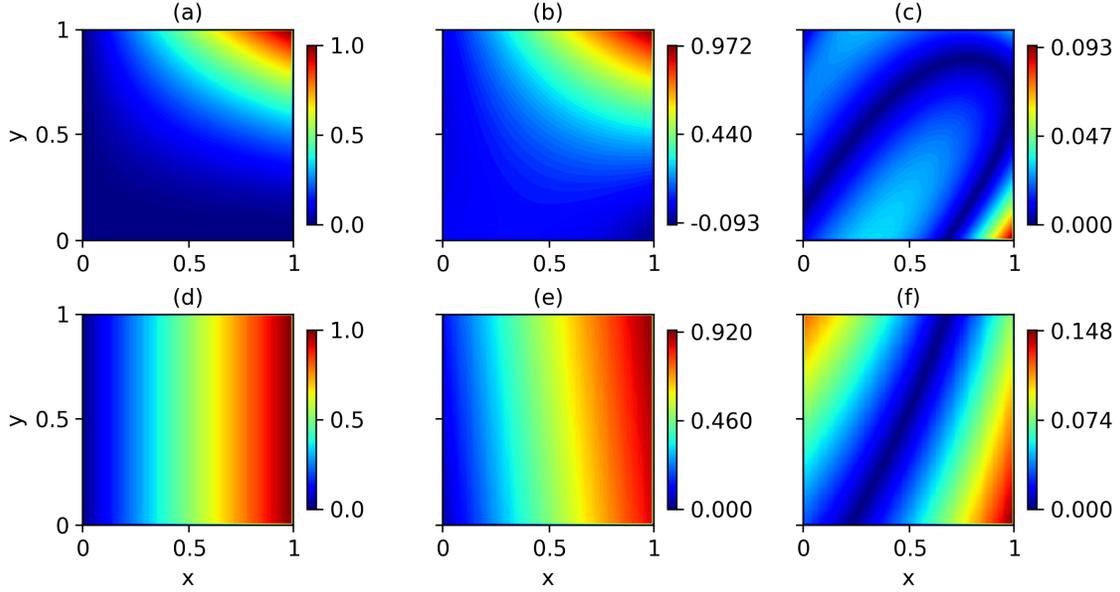
Figure 5: Best results for P_{out} (top) and P_{para} (bottom). The exact solutions for P_{out} and P_{para} are shown in (a) and (d). The predicted solutions are shown in (b) and (e) and the point-wise errors in (c) and (f)
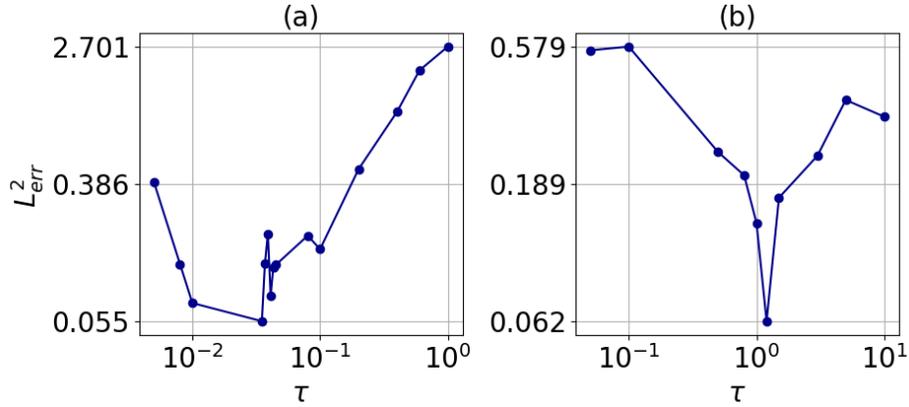

Figure 6: Sensitivity analysis on the τ parameter for problems P_{out} and P_{para} using $\mathcal{L}_\tau^{\mathcal{SUPG}}$ as loss functional

best one. In the case of P_{para}, there is even just a single stabilization parameter that gives the by far most accurate result. The best values from Figure 6 correspond with those from Tables 3 and 4. This study shows that a refined search is needed to find the optimal value of the stabilization parameter to obtain accurate results. This search is usually not feasible in practice since the solution of the problem is usually not known.
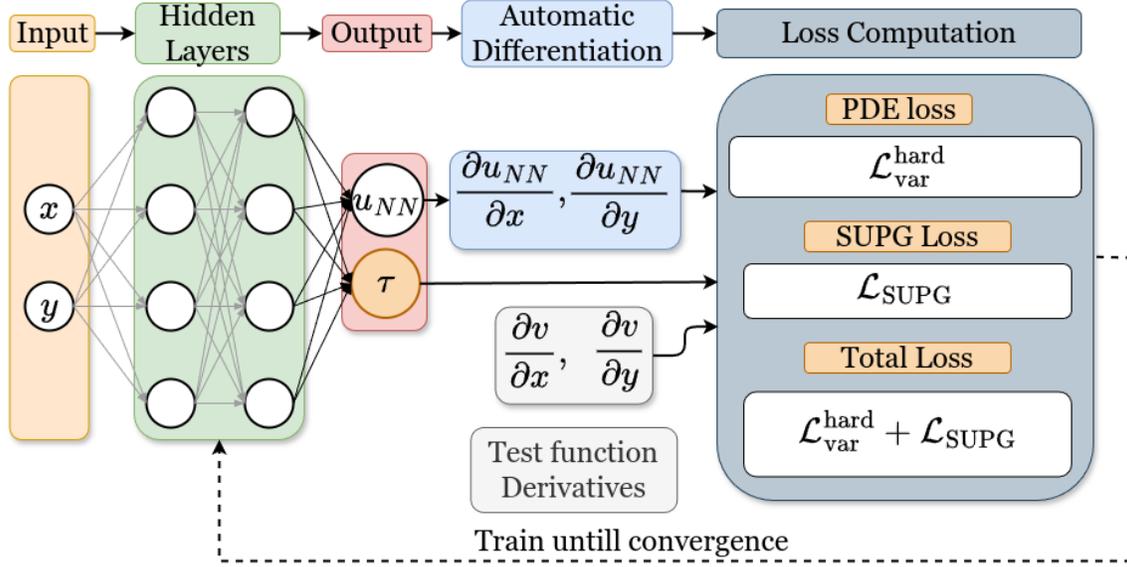
Figure 7: hp-VPINNs architecture for convection-dominated problems with SUPG stabilization that predicts the stabilization parameter τ and imposes hard constraints

5 Learning the Parameters in the SUPG Loss and the Indicator Function

The studies from Section 4 and further preliminary numerical simulations have identified two main challenges in addressing convection-dominated CDR problems with the hp-VPINNs as studied in this paper. The first challenge consists in determining the optimal, or at least a good, SUPG stabilization parameter τ. The second challenge involves the choice of a suitable indicator function (7) for enforcing Dirichlet boundary conditions, see also Matthaiou et al. (2024) for a discussion of this challenge. This second challenge is particularly important because the indicator function must not only match the Dirichlet boundary values at the boundary, but its gradient near the boundary should also fit to the solution's gradient. To address these challenges, we propose two enhancements to hp-VPINNs, and with that in particular to FastVPINNs. Firstly, we introduce a modified neural network that simultaneously predicts a spatially varying stabilization parameter $\tau(x, y)$ along with the solution as shown in Figure 7. Secondly, we also propose an adaptive indicator function whose slope near the boundaries is controlled by a learnable parameter, which is learnt during the training process.

5.1 Neural Network Predicting the Stabilization Parameter $\tau(x, y)$

The stabilization parameter that controls the artificial streamline diffusion can be expressed as a function of the coefficients of the CDR problem and the local mesh width, as it is done in finite element methods. However, the difficulty of this approach is that the optimal stabilization parameter is not known for problems defined in two- and higher-dimensional domains. In a neural network context, the stabilization parameter τ can be considered as

an additional hyperparameter that has to be found by performing hyperparameter tuning, as it was done in Section 4. However, this approach is time-consuming and it might be infeasible in practice.

To address this challenge, we propose a neural network that can predict the stabilization parameter τ along with the solution, as shown in Figure 7. With this approach, the neural network-predicted τ can be spatially varying, and this variation depends not only on the coefficients of the problem but also on the local shape of the predicted solution. This feature can also be observed for optimized SUPG stabilization parameters, e.g., see John et al. (2011, 2023). In this way, numerical stabilization is added only where needed.

The network-predicted τ is constrained using a sigmoid activation function to ensure that it remains positive. Preliminary numerical studies showed that it is of advantage, for obtaining consistent values of τ across different runs, to set τ to zero at the boundary of the domain. That means, the quadrature nodes on $\partial\Omega$ do not contribute to the SUPG stabilization loss $\mathcal{L}_{\mathcal{SUPG}}$ defined in (10). In fact, the solution on $\partial\Omega$ is known and applying a stabilization there is not mandatory. We used the indicator function

$$\tau_{\mathrm{NN}}^{\mathrm{hard}}\left(\boldsymbol{x};\theta_W,\theta_b\right) = w(\boldsymbol{x})\sigma\left(\tau_{\mathrm{NN}}(\boldsymbol{x};\theta_W,\theta_b)\right),$$

where, $\tau_{\mathrm{NN}}(\boldsymbol{x};\theta_W,\theta_b)$ is the stabilization parameter predicted by the neural network, σ is the sigmoid function, and $w(\boldsymbol{x})$ is an indicator function defined by

$$w(\boldsymbol{x}) = \tanh(50x)\tanh(50y)\tanh(50(1-x))\tanh(50(1-y)).$$

Then we set

$$\tau(\boldsymbol{x};\theta_W,\theta_b) = \tau_g\tau_{\mathrm{NN}}^{\mathrm{hard}}(\boldsymbol{x};\theta_W,\theta_b),$$

where τ_g, termed as 'τ growth', is controlling the strength of the stabilization factor. The hyperparameter τ_g needs to be fine-tuned. The predicted τ is then used in the calculation of $\mathcal{L}_{\tau}^{\mathcal{SUPG}}$ given in (11).

We will use problems $\mathrm{P}_{\mathrm{out}}$ and $\mathrm{P}_{\mathrm{para}}$ to demonstrate the effect of applying the learnt τ in the SUPG stabilization loss. Three approaches will be compared: PDE loss (9) only, PDE loss with SUPG stabilization (11) and constant τ, and PDE loss with SUPG stabilization (11) with learnt τ_g-based SUPG stabilization.

In our previous studies we observed that using a symmetric indicator function as given in (15) might generate large errors on the boundary when there is no steep gradient in the solution, compare Figure 5. Therefore, for this study, we applied a modified indicator function that has a steep slope at the outflow boundaries $x = 1$ and $y = 1$ and a low slope at the other two boundaries:

$$h(x,y) = \left(1 - \mathrm{e}^{-\kappa_1 x}\right)\left(1 - \mathrm{e}^{-\kappa_1 y}\right)\left(1 - \mathrm{e}^{-\kappa_2(1-x)}\right)\left(1 - \mathrm{e}^{-\kappa_2(1-y)}\right),$$
$$\kappa_1 = 30,\ \kappa_2 = 10/10^{-8}\ =\ 10^9. \tag{16}$$

Similarly for $\mathrm{P}_{\mathrm{para}}$, we applied a modified indicator function, which has a steep slope at boundaries $y = 0$, $y = 1$ and $x = 1$, and a low slope at the boundary $x = 0$:

$$h(x,y) = \left(1 - \mathrm{e}^{-\kappa_1 x}\right)\left(1 - \mathrm{e}^{-\kappa_2 y}\right)\left(1 - \mathrm{e}^{-\kappa_2(1-x)}\right)\left(1 - \mathrm{e}^{-\kappa_2(1-y)}\right),$$
$$\kappa_1 = 30,\ \kappa_2 = 10/10^{-8}\ =\ 10^9. \tag{17}$$

Table 5: L_{err}^2 for problem P_{out} using different loss functionals

loss	$\mathcal{L}_{\text{var}}^{\text{hard}}$	$\mathcal{L}_{\tau}^{\mathcal{SUPG}}$ constant τ	$\mathcal{L}_{\tau}^{\mathcal{SUPG}}$ learnt τ
search range	-	$\tau \in [10^{-5}, 10^{-1}]$	$\tau_g \in [5 \cdot 10^{-2}, 10]$
optimal parameter	-	$\tau = 10^{-5}$	$\tau_g = 1$
best L_{err}^2	$1.693 \cdot 10^{-4}$	$1.340 \cdot 10^{-4}$	$1.037 \cdot 10^{-4}$

A neural network with seven hidden layers and 30 neurons per layer was used. The network was trained for $50,000$ epochs with a learning rate of $0.01 \cdot 3^{-2}$. We used 8×8 cells with 100 quadrature points and 36 test functions per cell. To determine the optimal hyperparameters for the constant τ and neural network-predicted τ, we initially performed a single run for each hyperparameter within the broad search ranges specified in Tables 5 and 6 for the problems P_{out} and P_{para}, respectively. From these initial runs, we identified the hyperparameters that performed the best for each approach. Subsequently, we ran 10 simulations using these optimal hyperparameters for each study. The average L_{err}^2 values from these 10 runs are presented in Tables 5 and 6. It can be observed that the model with the learnt τ (with the hyperparameter τ_g) leads to notably more accurate results than both the model with constant τ (as a hyperparameter) and the model without any SUPG loss. Some representative solutions for the approach with the learnt stabilization parameter are presented in Figures 8 and 9.

Table 6: L_{err}^2 for problem P_{para} using different loss functionals

loss	$\mathcal{L}_{\text{var}}^{\text{hard}}$	$\mathcal{L}_{\tau}^{\mathcal{SUPG}}$ constant τ	$\mathcal{L}_{\tau}^{\mathcal{SUPG}}$ learnt τ
search range	-	$\tau \in [10^{-4}, 10^{-1}]$	$\tau_g \in [10^{-4}, 10]$
optimal parameter	-	$\tau = 5 \cdot 10^{-4}$	$\tau_g = 1$
best L_{err}^2	$1.544 \cdot 10^{-4}$	$1.192 \cdot 10^{-4}$	$9.043 \cdot 10^{-5}$

All errors presented in Tables 5 and 6 are considerably smaller than those in Tables 3 and 4, respectively. For P_{out}, in particular the error in the right lower corner of the domain almost disappeared, compare Figures 5 and 8. In the case of P_{para}, the contour lines of the computed solutions are parallel to the y-axis now. These observations show that a combination of a learnt stabilization parameter and a suitable indicator function, namely using (16) and (17) instead of (15), leads to a tremendous increase of accuracy.

5.2 Adaptive Indicator Function

In our previous set of studies, we used a fixed indicator function to impose hard-constrained boundary conditions and we observed in preliminary simulations that the choice of the indicator function might have a considerable impact on the accuracy of the computed solutions. To the best of our knowledge, there is no proposal available on how to select a priori a good indicator function. That's why, we pursue an approach to compute an adaptive indicator function. This function is still of the same principal form as used so far, but it is constructed
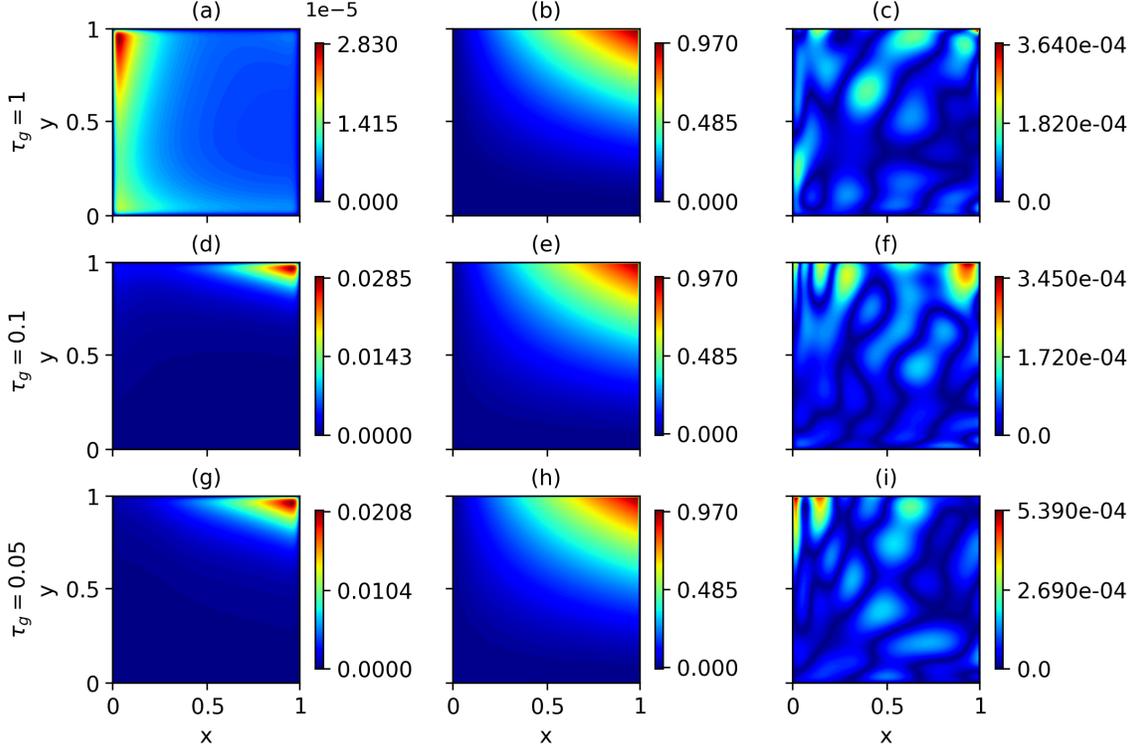
Figure 8: Results for problem P_{out} with $\mathcal{L}_\tau^{\mathcal{SUPG}}$ for $\tau_g = 1$, 0.1, 0.05. Note that the optimal solution was obtained for $\tau_g = 1$. The stabilization parameter $\tau(x, y)$ is shown in (a), (d) and (g). The predicted solution is depicted in (b), (e) and (h). The point-wise error is presented in (c), (f) and (i)

using trainable parameters that control the slope of the function near the boundaries. Our approach will enable the neural network to learn not only the solution but also the parameters for the indicator functions, resulting in a better fit for the specific parameters of the chosen problem.

Concerning problem P_{out}, we propose an adaptive indicator function of form (18), more precisely,

$$h(x, y) = \left(1 - e^{-\kappa_1 x}\right) \left(1 - e^{-\kappa_1 y}\right) \left(1 - e^{-\kappa_2(1-x)}\right) \left(1 - e^{-\kappa_2(1-y)}\right),$$
$$\kappa_1 = 10^\alpha, \ \kappa_2 = 10^\beta, \tag{18}$$

where α and β are learnable parameters controlling the gradient near the boundaries. Specifically, α governs the behavior at the inlet boundaries, while β controls the behavior at the outflow boundaries.
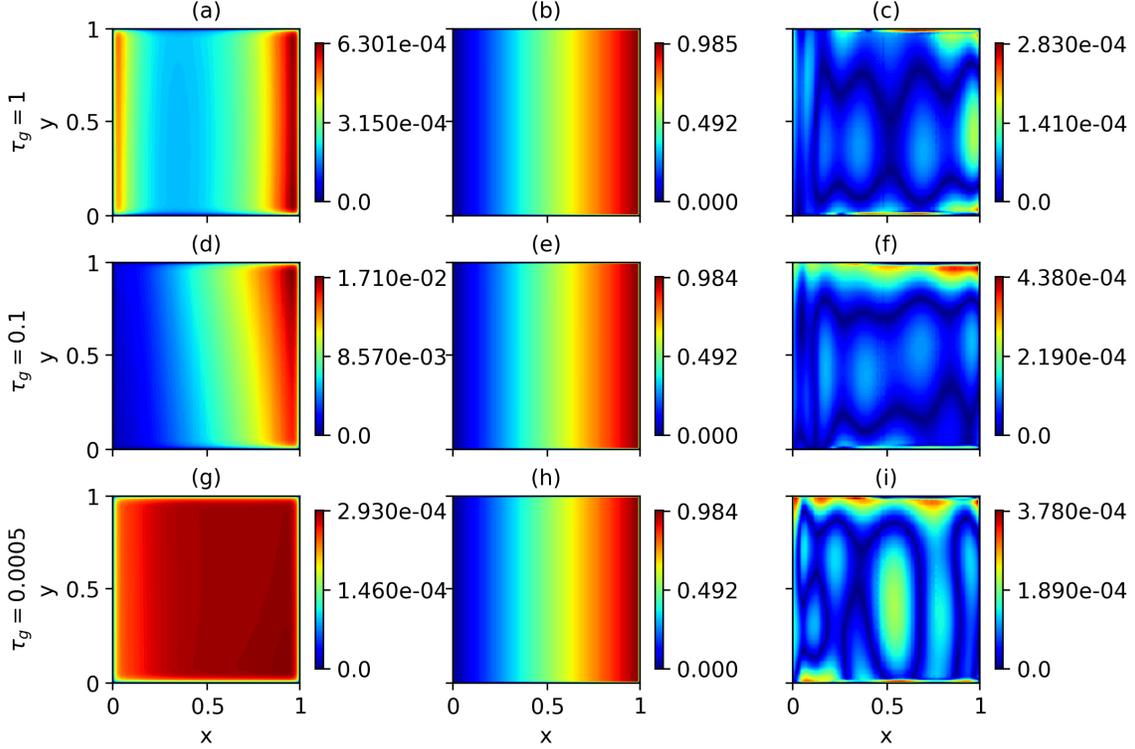
Figure 9: Results for problem P_{para} with $\mathcal{L}_\tau^{\mathcal{SUPG}}$ for $\tau_g = 1$, 0.1, 0.0005. Note that the optimal solution was obtained for $\tau_g = 1$. The stabilization parameter $\tau(x, y)$ is shown in (a), (d) and (g). The predicted solution is depicted in (b), (e) and (h). The point-wise error is presented in (c), (f) and (i)

Similarly, we introduce an adaptive indicator function for P_{para} with three learnable parameters: α, β, and γ, namely

$$h(x, y) = \left(1 - e^{-\kappa_1 x}\right) \left(1 - e^{-\kappa_2 y}\right) \left(1 - e^{-\kappa_3 (1-x)}\right) \left(1 - e^{-\kappa_2 (1-y)}\right),$$
$$\kappa_1 = 10^\alpha, \ \kappa_2 = 10^\beta, \ \kappa_3 = 10^\gamma. \tag{19}$$

In this formulation, α affects the inlet boundary, β the characteristic boundaries, and γ the outflow boundary.

The numerical studies with adaptive indicator functions were performed only with the loss functional $\mathcal{L}_{var}^{hard}$ from (9). Since the steepness of layers depends on the Péclet number, we considered different values of the diffusion coefficient to investigate how the network's prediction of the indicator function depends on the steepness of the layers.

Results for P_{out} are presented in Figure 10 and Table 7. The parameters for the indicator function were initialized with $\alpha = 1$ and $\beta = \phi$, where $\phi = -\log_{10}(\varepsilon)/2$. For instance, when $\varepsilon = 10^{-8}$, we set $\beta = -\log_{10}(10^{-8})/2 = 4$. It can be seen in Table 7 that in fact very different parameters are proposed by the neural network for the inlet and outflow
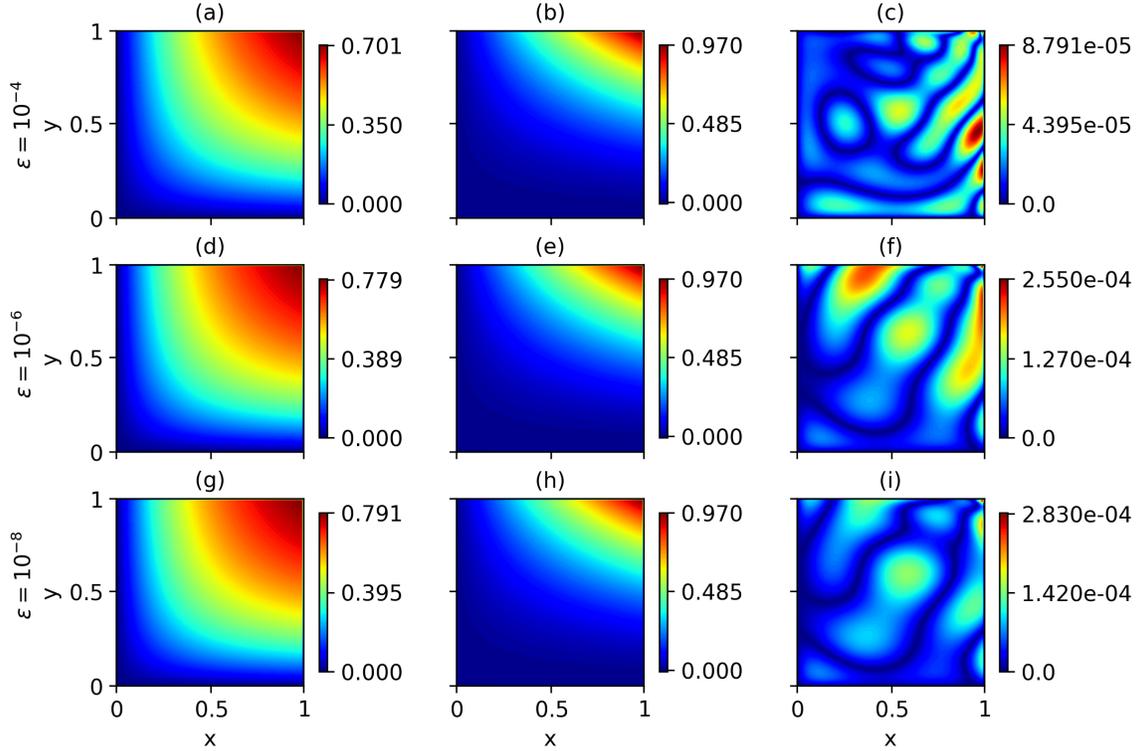
Figure 10: Results for problem P$_{\text{out}}$ for different ε. The indicator function $h(x, y)$ is shown in (a), (d) and (g), the predicted solution in (b), (e) and (h), and the point-wise error in (c), (f) and (i)

Table 7: Adaptive indicator function parameters and L^2_{err} for P$_{\text{out}}$

ε	initial α	final α	initial β	final β	initial L^2_{err}	final L^2_{err}
10^{-4}	1	0.281	2	3.998	$2.295 \cdot 10^{-1}$	$4.317 \cdot 10^{-5}$
10^{-6}	1	0.338	3	3.999	$1.034 \cdot 10^{-1}$	$5.518 \cdot 10^{-5}$
10^{-8}	1	0.360	4	4.051	$9.936 \cdot 10^{-5}$	$5.887 \cdot 10^{-5}$

boundaries, respectively, with the value at the inlet boundaries being considerably smaller. The parameters do not depend very much on the diffusion coefficient, i.e., on the Péclet number. The indicator functions have a different shape than the function given in (16), which takes values very close to 1 in most parts of Ω. It can be seen that the results obtained with the adapted parameters are (much) more accurate than those with the initial parameters. Comparing with Table 5, one can notice also that the error in the case $\varepsilon = 10^{-8}$ is noticeably smaller if the adaptive indicator function is used.
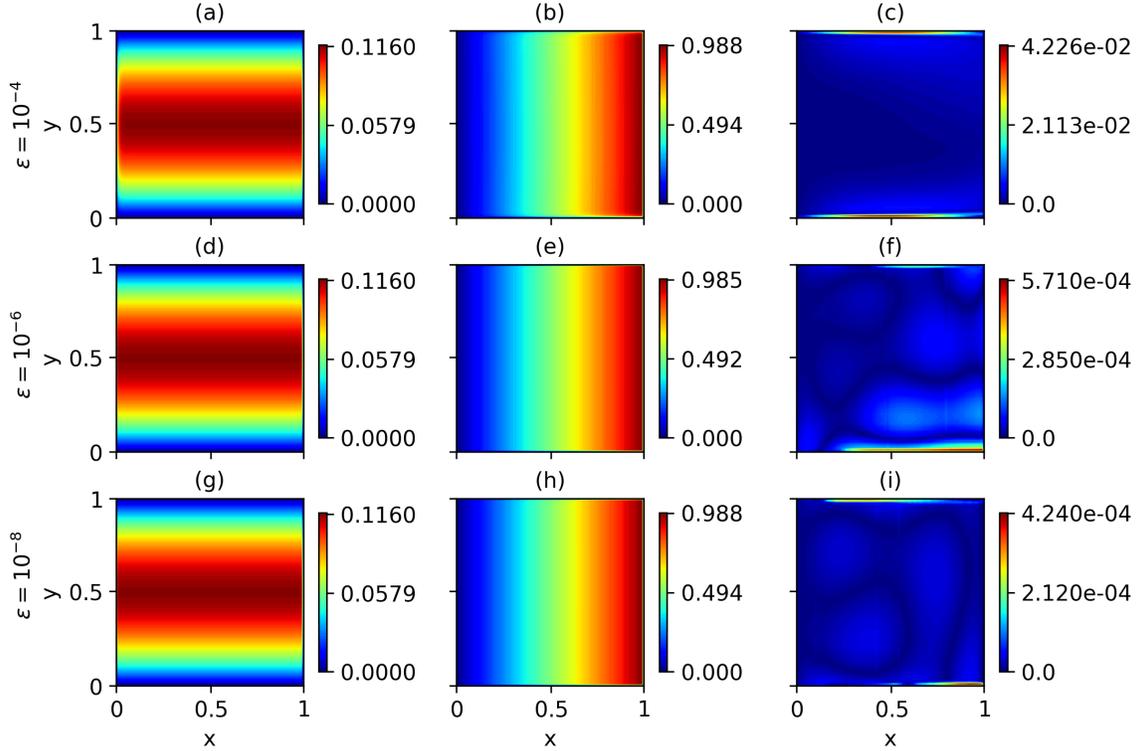
20

Figure 11: Results for problem P_{para} for different ε. The indicator function $h(x,y)$ is shown in (a), (d) and (g), the predicted solution in (b), (e) and (h), and the point-wise error in (c), (f) and (i)

Table 8: Adaptive indicator function parameters and L^2_{err} for P_{para}

ε	initial α	final α	initial β	final β	initial γ	final γ	initial L^2_{err}	final L^2_{err}
10^{-4}	1	2.129	0	-0.086	2	3.927	$5.124 \cdot 10^{-1}$	$1.075 \cdot 10^{-2}$
10^{-6}	1	3.295	0	-0.088	3	3.921	$2.706 \cdot 10^{-1}$	$1.066 \cdot 10^{-4}$
10^{-8}	2	3.380	0	-0.079	4	4.004	$1.534 \cdot 10^{-2}$	$6.007 \cdot 10^{-5}$

For problem P_{para}, whose solution possesses different types of layers and whose ansatz for the indicator function is given in (19), the obtained results are given in Figure 11 and Table 8. The evaluation comes to similar conclusions as for the other example. Again, the learnt parameters for the indicator function depend only slightly on the Péclet number. The visualizations of the indicator functions look very much alike. Comparing with Table 6, one can observe that the results for $\varepsilon = 10^{-8}$ is considerably more accurate for the approach with adaptive indicator function.

21

6 Conclusion

FastVPINNs are a very efficient approach for computing approximations of solutions of boundary value problems. In this paper, this approach was combined with two methods for improving the accuracy of computed solutions of convection-diffusion-reaction problems in the convection-dominated regime. First, the residual loss was augmented with a SUPG stabilization term and an architecture was proposed that computes a spatially varying stabilization parameter. And second, an architecture was developed that learns appropriate parameters for defining a good indicator function for imposing hard-constrained Dirichlet boundary conditions. Both extensions led to a considerable increase of the accuracy of the computed solutions. In contrast, the inclusion of a regularization term with the weights of the network was not helpful for obtaining solutions with higher accuracy.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Diab W. Abueidda, Qiyue Lu, and Seid Koric. Meshless physics-informed deep learning method for three-dimensional solid mechanics. *Internat. J. Numer. Methods Engrg.*, 122 (23):7182–7201, 2021. ISSN 0029-5981,1097-0207.

Thivin Anandh, Divij Ghose, and Sashikumaar Ganesan. FastVPINNs: An efficient tensor-based Python library for solving partial differential equations using hp-Variational Physics Informed Neural Networks. *Journal of Open Source Software*, 9(99):6764, 2024a.

Thivin Anandh, Divij Ghose, Himanshu Jain, and Sashikumaar Ganesan. FastVPINNs: Tensor-Driven Acceleration of VPINNs for Complex Geometries. *arXiv preprint arXiv:2404.12063*, 2024b.

Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, Karen Willcox, and Steven Lee. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. *USDOE Office of Science (SC), Washington, D.C. (United States)*, 2 2019. doi: 10.2172/1478744. URL https://www.osti.gov/biblio/1478744.

Gabriel R. Barrenechea, Volker John, and Petr Knobloch. Finite element methods respecting the discrete maximum principle for convection-diffusion equations. *SIAM Rev.*, 66 (1):3–88, 2024. ISSN 0036-1445,1095-7200.

Alexander N. Brooks and Thomas J. R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 32(1-3):199–259, 1982. ISSN 0045-7825. FENOMECH '81, Part I (Stuttgart, 1981).

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: where we are and what's next. *J. Sci. Comput.*, 92(3):Paper No. 88, 62 pages, 2022. ISSN 0885-7474,1573-7691.

Kenneth Eriksson and Claes Johnson. Adaptive finite element methods for parabolic problems. I: A linear model problem. *SIAM J. Numer. Anal.*, 28(1):43–77, 1991. ISSN 0036-1429.

Derk Frerichs-Mihov, Linus Henning, and Volker John. On loss functionals for physics-informed neural networks for steady-state convection-dominated convection-diffusion problems. *Comm. Appl. Math. Comp.*, page to appear, 2024a.

Derk Frerichs-Mihov, Volker John, and Marwa Zainelabdeen. On collocation points for physics-informed neural networks applied to convection-dominated convection-diffusion problems. In *Proceedings of ENUMATH 2023*. Springer, 2024b. accepted.

Sashikumaar Ganesan and Lutz Tobiska. *Finite elements*. Cambridge-IISc Series. Cambridge University Press, Cambridge, 2017. ISBN 978-1-108-41570-5. Theory and algorithms.

Divij Ghose, Thivin Anandh, and Sashikumaar Ganesan. FastVPINNs: A fast, versatile and robust Variational PINNs framework for forward and inverse problems in science. In *ICLR 2024 Workshop on Differential Equations in Science*, 2024.

T. J. R. Hughes and A. Brooks. A multidimensional upwind scheme with no crosswind diffusion. In *Finite element methods for convection dominated flows (Papers, Winter Ann. Meeting Amer. Soc. Mech. Engrs., New York, 1979)*, volume 34 of *AMD*, pages 19–35. Amer. Soc. Mech. Engrs. (ASME), New York, 1979.

V. John, J. M. Maubach, and L. Tobiska. Nonconforming streamline-diffusion-finite-element-methods for convection-diffusion problems. *Numer. Math.*, 78(2):165–188, 1997. ISSN 0029-599X.

Volker John and Petr Knobloch. On spurious oscillations at layers diminishing (SOLD) methods for convection-diffusion equations. I. A review. *Comput. Methods Appl. Mech. Engrg.*, 196(17-20):2197–2215, 2007. ISSN 0045-7825.

Volker John and Petr Knobloch. On algebraically stabilized schemes for convection-diffusion-reaction problems. *Numer. Math.*, 152(3):553–585, 2022. ISSN 0029-599X,0945-3245.

Volker John, Petr Knobloch, and Simona B. Savescu. A posteriori optimization of parameters in stabilized methods for convection-diffusion problems—Part I. *Comput. Methods Appl. Mech. Engrg.*, 200(41-44):2916–2929, 2011. ISSN 0045-7825.

Volker John, Petr Knobloch, and Ulrich Wilbrandt. A posteriori optimization of parameters in stabilized methods for convection-diffusion problems—Part II. *J. Comput. Appl. Math.*, 428:Paper No. 115167, 17 pages, 2023. ISSN 0377-0427,1879-1778.

Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

Ehsan Kharazmi, Zhongqiang Zhang, and George E. M. Karniadakis. *hp*-VPINNs: variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Engrg.*, 374:Paper No. 113547, 25 pages, 2021. ISSN 0045-7825,1879-2138.

Reza Khodayi-Mehr and Michael Zavlanos. Varnet: Variational neural networks for the solution of partial differential equations. In *Learning for Dynamics and Control*, pages 298–307. PMLR, 2020.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 26548–26560, Virtual, Online, 2021. Curran Associates, Inc.

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

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: a deep learning library for solving differential equations. *SIAM Rev.*, 63(1):208–228, 2021a. ISSN 0036-1445,1095-7200.

Lu Lu, Raphaël Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G. Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM J. Sci. Comput.*, 43(6):B1105–B1132, 2021b. ISSN 1064-8275,1095-7197.

Marina Matthaiou, Volker John, and Marwa Zainelabdeen. Bound-preserving PINNs for steady-state convection-diffusion-reaction problems. *WIAS Preprint No. 3134*, 2024.

NVIDIA Modulus, 2024. `https://developer.nvidia.com/modulus`. Last accessed Jan 01, 2024.

Apostolos F. Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. Uncertainty quantification in scientific machine learning: methods, metrics, and comparisons. *J. Comput. Phys.*, 477:Paper No. 111902, 83 pages, 2023. ISSN 0021-9991,1090-2716.

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. *J. Comput. Phys.*, 378:686–707, 2019. ISSN 0021-9991,1090-2716.

Hans-Görg Roos, Martin Stynes, and Lutz Tobiska. *Robust numerical methods for singularly perturbed differential equations*, volume 24 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2008. ISBN 978-3-540-34466-7. Convection-diffusion-reaction and flow problems.

Maciej Sikora, Patryk Krukowski, Anna Paszynska, and Maciej Paszynski. Physics informed neural networks with strong and weak residuals for advection-dominated diffusion problems, 2023.

Sangeeta Yadav and Sashikumaar Ganesan. Artificial neural network-augmented stabilized finite element method. *J. Comput. Phys.*, 499:Paper No. 112702, 15 pages, 2024. ISSN 0021-9991,1090-2716.