Literature Review

Titus Ex

January 2021

Contents

1 Introduction

Partial differential equations (PDEs) are equations that describe the relationship between a multivariable function and its partial derivatives. These equations are used extensively in scientific fields and help describe a lot of the natural processes in our world. One of these PDEs is called the *advection-diffusion* equation, which describes the transport of physical quantities through a system, due to two different types of transport called *advection* (or *convection*) and *diffusion.* The advection diffusion equation looks like this

$$\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u) - \nabla \cdot (\boldsymbol{v} u) \tag{1}$$

Here, u is some variable of interest, D is the diffusion coefficient, and $\boldsymbol{v}$ is the velocity field of u. The first part of this equation, $\nabla \cdot (D\nabla u)$, captures the transport that occurs due to diffusion. This is the effect of particles moving from high concentration to low concentration. The effect of convection is captured by the second term, $-\nabla \cdot (\boldsymbol{v}u)$. Convection describes what happens when larger packages of the quantity of interest move due to flow.
Natural phenomena are often described by equations like the advection-diffusion equation in (1) together with boundary conditions, resulting in what is called a *boundary value problem*

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla \cdot (D \nabla u) - \nabla \cdot (\boldsymbol{v} u) \\ u(t,0) = g \\ u(t,1) = h \end{cases} \tag{2}$$

Typically, one is interested in using numerical methods to approximate the solution of (2), especially when the problem does not have a solution in the classical sense. Industries to which solving these types of problems is very relevant include the automotive industry, the petroleum industry, and the aviation industry.
One of the most common numerical methods used to approximate u in (2) is called the finite element method. This method discretises the domain of the original problem and uses a variational formulation that consists of so-called trial and test functions, to arrive at a system of matrix equations. By solving this system an approximate solution to the original problem can be found. Advanced adaptations of the finite element method do not only say something about how to find an approximate solution to the linear advection-diffusion equation in (2), but can also approximate the solution to non-linear advection-diffusion equations like the Navier-Stokes equations, which are much more widely applicable.

The structure of this literature report is as follows. First, one of the challenges of using finite element methods for advection dominated equations will be examined, namely the instability of the numerical solution. Then, an overview of how others have dealt with this problem will be presented.

These efforts can be categorised into two different approaches: building stable numerical schemes and using data-driven methods based on deep learning. After examining these different methods and naming their advantages and disadvantages, research questions will be formulated. Then, since the feasibility of these research questions has been tested to a very basic degree, some preliminary results will be presented.

2 The Instability of Numerical Solutions

As was mentioned earlier, the finite element method approximates the solution by using the variational (weak) form, essentially a discretisation of the original problem. One of the most common ways of doing this is called the Galerkin method, which is typified by the fact that the trial functions and test functions used in the weak form come from the same class of functions. The Galerkin method has been very successful as in many applications it can be shown that it leads to the optimal approximation error with respect to a particular norm. One of the problems with the Galerkin finite element method is that the approximate solution becomes unstable for advection dominated PDEs, see [38]. To understand why, consider the following 1D steady-state version of the advection-diffusion equation from (2)

$$\begin{cases} -\epsilon\dfrac{d^2u}{dx^2} + v\dfrac{du}{dx} = 0 \text{ for } x \in (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases} \tag{3}$$

The exact solution to this problem is given by

$$u(x) = \frac{1 - e^{xv/\epsilon}}{1 - e^{v/\epsilon}} \tag{4}$$

To approximate this solution, consider the finite central difference method, which discretises the domain into N equidistant points and approximates the first and second order derivatives at the point x_i in the following way

$$\begin{aligned} u'(x_i) &= \frac{-u(x_{i-1}) + u(x_{i+1})}{2h} \\ u''(x_i) &= \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} \end{aligned} \tag{5}$$

These approximations are substituted into the N equations resulting from the discretisation of (3), to get to

$$-\epsilon\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + v\frac{-u_{i-1} + u_{i+1}}{2h} = 0 \tag{6}$$

If the solution is assumed to be of the form $u_i = r^k = r^{k-1}r$ such that $u_{i-1} = r^{k-1}$ and $u_{k+1} = r^{k-1}r^2$, then it follows from (6) that

$$r = \frac{1 + \frac{v\Delta x}{2\epsilon}}{1 - \frac{v\Delta x}{2\epsilon}} \tag{7}$$

The ratio between the transport through advection and through diffusion is called the Péclet number and is denoted by Pe, i.e., $Pe = v/\epsilon$. Looking at (7) it becomes clear that if the grid Péclet number,

$v\Delta x/2\epsilon$, is larger than one, $r < 0$ and the solution will oscillate. In [38] the point is made that Galerkin finite element methods used to discretise (3) result in the central-difference approximations in (5), and therefore suffer from the same problem.
To see this problem in an example, consider (3) with $\epsilon = 0.01$ and $v = 1$. After running a Galerkin finite element scheme using 10 elements and first order B-spline finite element functions, the approximate solution was plotted and is compared to the exact solution in Figure 1.

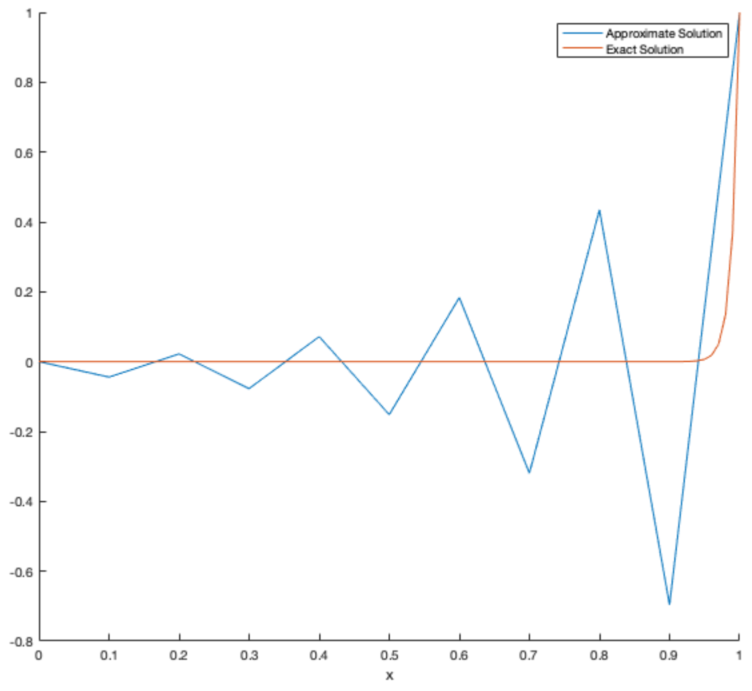

Figure 1: Galerkin FEM approximate solution vs. exact solution of (3), given $\epsilon = 0.01$ and $v = 1$.

This plot shows that the approximate solution is unstable and exhibits wild oscillations that do not happen in the exact solution. These oscillations get progressively worse as the Péclet number gets larger, and render the approximate solution useless. The next few sections will present approaches that have been used to deal with this problem.

3 Building Stable Numerical Schemes

To deal with the instability arising from the central-differences in the Galerkin method, several different approaches have been used. First, increasing the number of grid points used in the discretisation can help reduce the element to element wiggles. To see why, note that in (7) a problem arises only when the grid Péclet number is larger than 1, i.e., when $v\Delta x/2\epsilon > 1$. This grid Péclet

number gets smaller whenever Δx, the distance between grid points, get smaller. If the grid is refined enough, convection no longer dominates on an element level.
Secondly, "upwind" techniques have been used to stabilize the solution. In the case of finite difference techniques this means that the advective term will be approximated using solutions at upwind and central points only. While upwind techniques do stabilize the solution, they suffer from a loss in accuracy (upwind difference techniques are first-order accurate while central difference techniques are second-order accurate). This loss in accuracy leads the approximate solutions to become overly diffusive. This added artificial diffusion is one of the biggest flaws of the upwind method. As it turns out, using a combination of upwind and central difference techniques is better than using just upwind of central differences. In the finite element method the idea of using upwind convective terms was first achieved with the Petrov-Galerkin framework in which different trial and test functions are used. Here, the test functions were chosen to put more weight on the upstream element of a node than to the downwind element. These early methods performed better than the regular Galerkin method when applied to simple convection dominated problems, but unfortunately performed much worse for more complicated cases.
Finally, the streamline upwind/Petrov-Galerkin was introduced in [38]. This method did not suffer from the artificial diffusion criticism, and is based on the idea of adding diffusion or viscosity in the direction of the flow only. This method was able to deal with advection dominated equations, and achieved good results on complicated problems.
The approaches in the papers that will be covered in the remainder of this section are based on the Petrov-Galerkin method. The novelty of these approaches comes from the fact that they use discontinuous test spaces, which make it easy to compute so called optimal test functions on a local basis.

3.1 A Class of Discontinuous Petrov-Galerkin Methods. Part II: Optimal Test Functions

3.1.1 Introduction

In [2] a method is laid out for building discontinuous Petrov-Galerkin (DPG) methods that are automatically computable to guarantee stability for a particular boundary value problem. Specifically, given the weak form of a boundary value problem and a space consisting of numerical solutions (trial functions), a space of test functions that can be used to guarantee the stability of a numerical scheme is constructed. This new approach differs from the way Galerkin methods have traditionally been constructed, where finite element methods are built by setting test and trial spaces for each mesh element simultaneously.
An important piece of the DPG method proposed in [2] is that is uses discontinuous approximation spaces. This allows for the test function spaces to be calculated locally (on each element separately). In contrast to standard discontinuous Galerkin methods however, the DPG method proposed in [2] uses the Petrov-Galerkin version, meaning that different functions for the trial functions spaces and the test function spaces are chosen. In [30], the advantage of this design choice is explained: although the trial spaces need to have good approximation properties, the test spaces can be chosen to obtain a stable scheme. In this literature review, the two most important parts of [2] are covered: the introduction of optimal test functions and the derivation of practical schemes.

3.1.2 Optimal Test Functions

In [2], L. Demkowicz and J. Gopalakrishnan define ”optimal test functions” in the setting of a variational boundary value problem

$$\text{Find } u \in U : b(u,v) = l(v) \quad \forall v \in V \tag{8}$$

where U and V are real Hilbert spaces, normed by $\|\cdot\|_U$ and $\|\cdot\|_V$ respectively. Furthermore the right hand side l of (8) is a continuous linear form defined on the test space V, and $b(\cdot,\cdot)$ denotes a bilinear form that is defined on $U \times V$ that is continuous

$$|b(u,v)| \leq M\|u\|_U\|v\|_V \tag{9}$$

and that satisfies the inf-sup condition

$$\inf_{\|u\|_U=1} \sup_{\|v\|_V=1} b(u,v) \geq \gamma \tag{10}$$

with $\gamma > 0$. Additionally, L. Demkowicz and J. Gopalakrishnan assume that

$$\{v \in V : b(u,v) = 0 \quad \forall u \in U\} = \{0\} \tag{11}$$

The authors mention that given these conditions, [31] showed that problem (8) has a unique solution for all $l \in V'$, where the prime refers the dual space. Furthermore, if (8) is approximated by the following Galerkin method

$$\begin{cases} \text{Find } u_n \in U_n \text{ satisfying} \\ \quad b(u_n, v_n) = l(v_n) \ \forall v \in V_n \end{cases} \tag{12}$$

with $U_n \subseteq U$, $V_n \subseteq V$, $\dim U_n = \dim V_n$, and equation (10) holds for the subspaces U_n and V_n as well, then the following theorem from [31] holds

Theorem 2.1 (Babuška). *Under the above assumptions, the exact and the discrete problems* (8) *and* (12) *are uniquely solvable. Furthermore,*

$$\|u - u_n\|_U \leq \frac{M}{\gamma_n} \inf_{w_n \in U_n} \|u - w_n\|_U \tag{13}$$

To explain the idea ”optimal test functions”, a new norm called the *energy norm* is defined next

$$\|u\|_E \overset{\text{def}}{=} \sup_{\|v\|_v=1} b(u,v) \tag{14}$$

Using (14) the following proposition follows

Proposition 2.1. *The energy norm $\|\cdot\|_E$ is an equivalent norm on U, specifically,*

$$\gamma\|u\|_U \leq \|u\|_E \leq M\|u\|_U, \quad \forall u \in U, \tag{15}$$

if and only if (9) *and* (10) *hold.*

Next, the authors define the map from trial space to test space T. For every $u \in U$, define Tu in V as the unique solution of

$$(Tu, v)_V = b(u, v), \quad \forall v \in V, \tag{16}$$

Here, the notation $(\cdot, \cdot)_V$ refers to the inner product on V. Furthermore, the authors note that by the Riesz representation theorem, T is a well defined map. Then, the following is evident from Hilbert space theory

Proposition 2.2. *For any u in U, the supremum in* (14) *is attained by $v = Tu \in V$. The norm $\|u\|_E$ is generated by the inner product*

$$(u, u)_E \overset{\text{def}}{=} (Tu, Tu)_V \tag{17}$$

L. Demkowicz and J. Gopalakrishnan then consider a Petrov-Galerkin method of the form (12), with the following finite dimensional trial subspace

$$U_n = \text{span}\{e_j : j = 1, \ldots, n\} \tag{18}$$

for a set of linearly independent set of functions e_j in U. The following definition can now be formulated

Definition 2.1. Every trial subspace U_n, as in (18), has its corresponding **optimal test space**, defined by

$$V_n = \text{span}\{Te_j : j = 1, \ldots, n\} \tag{19}$$

L. Demkowicz and J. Gopalakrishnan explain that test spaces defined as above are "optimal" in that they result in the optimal ratio of continuity constant to stability constant when U is endowed with the energy norm. More specifically, the following theorem is presented and proven

Theorem 2.2. *Let V_n be the optimal test space corresponding to a finite dimensional trial space U_n. Then the error in the Petrov-Galerkin scheme* (12) *using $U_n \times V_n$ equals the best approximation error in the energy norm, i.e.*

$$\|u - u_n\|_E = \inf_{w_n \in U_n} \|u - w_n\|_E \tag{20}$$

3.1.3 Derivation of Practical Schemes

To construct a scheme using the optimal test functions, the operator T in (17) needs to be approximated. L. Demkowicz and J. Gopalakrishnan proceed by laying out a method to derive practical schemes

1. Given a boundary value problem, develop mesh dependent variational formulations $b(\cdot, \cdot)$ with an underlying space V that allows *inter-element* discontinuities (this choice is the reason that theses schemes are called "discontinuous" Petrov-Galerkin methods).

2. Choose a trial subspace U_n. It follows from Theorem 2.2 that trial spaces must be chosen with *good approximation* properties. This means that they are most often standard piecewise polynomial spaces, where the degree of the polymomials depends on the local order of accuracy that is needed.

3. Next, the optimal test functions need to be approximated. Because of the inter-element discontinuities in V in step 1, T can be approximated by a *local, element-by-element* computable approximation $T_n : U_n \to \tilde{V}_n$ such that

$$(T_n u_n, \tilde{v}_n)_V = b(u_n, \tilde{v}_n), \quad \forall \tilde{v}_n \in \tilde{V}_n \tag{21a}$$

and

$$T_n \text{ is injective on } U_n \tag{21b}$$

where $\tilde{V}_n \subseteq V$ is a computationally convenient space of discontinuous functions, that can be used to represent the approximate optimal test space. If e_j forms a basis for U_n as in (18), then the trial space is set to $V_n = \operatorname{span}\{t_j\}$ where $t_j = T_n e_j$. It follows that t_j forms a basis for V_n due to (21b).

4. The last step involves solving a *symmetric positive definite* matrix system. Regardless of the assymetry of the bilinear form, the result is always a symmetric linear system, because the (i, j)th entry of the matrix of (12) is

$$\begin{aligned} b(e_j, t_i) &= (T_n e_j, t_i)_V && \text{by } (\textbf{??}) \\ &= (T_n e_j, T_n e_i)_V && \text{as } t_i = T_n e_i \\ &= (T_n e_j, T_n e_i)_V \\ &= b(e_i, t_j) \end{aligned}$$

thus coinciding with the (j, i)th entry. The positive definiteness follows from (21b).

After describing this method to construct an approximation scheme L. Demkowicz and J. Gopalakrishnan mention that they do not have a universal prescription for selecting the space $\tilde{V}_n$, but that its dimension must be at least of $\dim(U_n)$ to satisfy (21b). Their motivation is that as $\tilde{V}_n$ gets richer, the discrete energy norm $\|T_n u_n\|_V$ may be expected to converge to $\|T u_n\|_V$, so that the discrete approximation should (more and more) inherit the stability properties of the original problem.

3.1.4 Example: Pure Convection

In [2] the use of the DPG method is presented for the transport equation, to illustrate how one can go through steps 1.-4. from the previous subsection. Consider the following pure convection problem

$$\begin{cases} \boldsymbol{\beta} \cdot \nabla u &= f \quad \text{in } \Omega \\ u &= u_0 \quad \text{on } \Gamma_{\text{in}} \end{cases} \tag{22}$$

where $\Omega \subset \mathbb{R}^n, n = 1, 2$, and Γ_{in} is the inflow boundary that looks like this

$$\Gamma_{\text{in}} = \{x \in \partial\Omega : \boldsymbol{\beta} \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\} \tag{23}$$

Here $\boldsymbol{\beta}$ is either a scalar or a vector, depending on the dimension of Ω, and $\boldsymbol{n}$ is the outward normal unit vector to the boundary. Suppose that Ω is partitioned into a set of finite elements. To get to the corresponding weak formulation of (22) the convection equation is multiplied with a test function that is supported on element K, and then the results is integrated by parts on element K to get to

$$-\int_K u \partial_\beta v + \int_{\partial K} \beta_n u v = \int_K f v \tag{24}$$

where $\partial_\beta v = \boldsymbol{\beta} \cdot \nabla v$ denotes the directional derivative in the direction of $\boldsymbol{\beta}$, and $\beta_n = \boldsymbol{\beta} \cdot \boldsymbol{n}$. Next, the authors introduce the flux as an auxiliary variable

$$q = |\beta_n|\, u \tag{25}$$

Using this new variable together with (24) gives the following variational formulation

$$\begin{cases} \text{Find } u \in L^2(\Omega),\ q \in L^2(\Gamma_h)\text{: such that} \\ b((u,q),v) = l(v), \quad \forall v \in H_\beta(K), \forall K \end{cases} \tag{26}$$

with

$$b((u,q),v) = \sum_K \int_K -u\partial_\beta v + \int_{\partial K \setminus \Gamma_{\text{in}}} \operatorname{sgn}(\beta_n) q v, \tag{27a}$$

$$l(v) = \sum_K \int_K fv + \int_{\partial K \cap \Gamma_{\text{in}}} \beta_n u_0 v, \tag{27b}$$

$$H_\beta(K) = \{v \in L^2(K) : \partial_\beta v \in L^2(K)\} \tag{27c}$$

With the formulation of (26) the first step of the DPG scheme is completed. In this example, the trial and test spaces are defined as follows

$$U = L^2(\Omega) \times L^2(\Gamma_h), \tag{28a}$$

$$V = \{v : v \in H_\beta(K), \quad \forall \text{elements } K\} \tag{28b}$$

so that both are inter-element discontinuous.

3.1.4.1 1D Spectral Discretisation

To showcase the next three steps, the authors use a 1D, single element discretisation. Set $\beta = 1$. Now, consider the discretisation of the domain $\Omega = (x_1, x_2)$ using a single element $K = (x_1, x_2)$. The space H_β, which coincides with $H^1(\Omega)$, is endowed with a Hilbert structure through the following inner product

$$(v,w)_V = \int_{x_1}^{x_2} v'w' + qv(x_2) \tag{29}$$

With this inner product defined on V and the $L^2(\Omega)$ defined on U the bilinear form

$$b((u,q),v) = \int_{x_1}^{x_2} uv' + qv(x_2) \tag{30}$$

satisfies (9). The next steps involves setting the trial subspace U_h. In this example it is defined as

$$U_p = \mathcal{P}_p(K) \times \mathbb{R} \tag{31}$$

which means that a function (u_p, q) in U_p is a combination of some $u \in \mathcal{P}_p(K)$ and a one point value q that represents the flux. In this example the optimal test functions can be calculated exactly. Two optimal test functions are found: one corresponding to the solution on the interior trial function $u \in \mathcal{P}_p(K)$ and one corresponding to the flux q.

The optimal test function v_u to the interior trial function $u \in \mathcal{P}_p(K)$ is the function in $H^1(K)$ such that the following equation holds

$$\int_{x_1}^{x_2} v_u' \delta_u' + v_u(x_2)\delta(x_2) = -\int_{x_1}^{x_2} u\delta_v', \quad \forall \delta_v \in H^1(K) \tag{32}$$

The solution to the above equation is given by

$$v_u(x) = \int_x^{x_2} u(s)ds \tag{33}$$

Similarly, the optimal test function corresponding to the flux q is the function v_q that satisfies

$$\int_{x_1}^{x_2} v_q' \delta_v' + v_q(x_2)\delta_v(x_2) = \delta_v(x_2), \quad \forall \delta_v \in H^1(K) \tag{34}$$

which means that

$$v_q = 1 \tag{35}$$

Using the span of these two types of functions, the optimal test space can be defined

$$V_p = \text{span}\{v_u, v_q : u \in \mathcal{P}_p(K), q \in \mathbb{R}\} \tag{36}$$

The authors conclude this example by pointing out that a different inner product would have let to a different optimal test space. For example, choosing

$$(v, \delta_v)_V = \int_{x_1}^{x_2} (v'\delta_v' + v\delta_v) \tag{37}$$

instead of (29) would have led to non-polynomial optimal test functions.

3.1.4.2 A multi-element 1D discretisation

The authors in [2] go one step further and also provide a derivation of the multi-element version of section 3.1.4.1. Instead of using a single element, consider the multi-element discretisation of $\Omega = (x_0, x_n)$ into elements (x_i, x_{i+1}). The same inner product from the single element case is used

$$(u, w)_V = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} v'w' + \alpha_i v^{\text{up}}(x_i) w^{\text{up}}(x_i) \tag{38}$$

Here, α_i are scaling factors and $v^{\text{up}}(x_i)$ is used to denote the limit of $v(x)$ as x approaches x_i from the left. Similarly $v^{\text{dn}}(x_i)$ is used to denote the limit of $v(x)$ as x approaches x_i from the right. The authors use the following trial space

$$U_h = \{(w_h, q_1, \ldots, q_n) : w_{h|(x_i, x_{i+1})} \in \mathcal{P}_p(x_i, x_{i+1})\} \tag{39}$$

Here, like was done in section 3.1.4.1, w_h is used to approximate u and $q_1, \ldots, q_n$ are used to approximate the rightward fluxes of each element. The bilinear form looks like this

$$b((u, q), v) = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} -uv' + q_i v^{\text{up}}(x_i) - q_{i-1} v^{\text{dn}}(x_{i-1}) \tag{40}$$

Here, $q_0 = 0$ because this part is included in the linear form as in (27b). Again, the optimal test function can be calculated exactly. For a trial function w defined on element (x_i, x_{i+1}) the optimal test function $v_w(x)$ is given by

$$v_w(x) = \int_x^{x_{i+1}} w(s)ds \tag{41}$$

For the fluxes q_i, the optimal test functions v_{q_i} are different compared to the section 3.1.4.1 case, in that they are non-zero on two elements instead of one element. The optimal test functions v_{q_i} can be found by solving the following two equations

$$\int_{x_{i-1}}^{x_i} v_i'\delta_v' + \alpha_i v_i^{\text{up}}(x_i)\delta_v^{\text{up}}(x_i) = \delta_v^{\text{up}}(x_i), \quad \forall\delta_v \in H^1(x_{i-1}, x_i)$$

and

$$\int_{x_i}^{x_{i+1}} v_i'\delta_v' + \alpha_{i+1} v_i^{\text{up}}(x_{i+1})\delta_v^{\text{up}}(x_{i+1}) = -\delta_v^{\text{dn}}(x_i), \quad \forall\delta_v \in H^1(x_i, x_{i+1})$$

The function v_{q_i} that satisfies both these equations is given by

$$v_{q_i}(x) = \begin{cases} \dfrac{1}{\alpha_i} & \text{if } x \in (x_{i-1}, x_i) \\ x - \dfrac{1 + \alpha_{i+1}x_{i+1}}{\alpha_{i+1}} & \text{if } x \in (x_i, x_{i+1}) \\ 0 & \text{elsewhere} \end{cases} \tag{42}$$

Therefore, the optimal test space V_h corresponding to the trial space is given by

$$V_h = \text{span}\{v_w : \forall w \in \mathcal{P}_p(K), \text{ and } v_{q_i} : \forall i = 1, \ldots, n\} \tag{43}$$

3.2 Automatic Variationally Stable Analysis for FE Computations: An Introduction

In [35] an automatic variationally stable analysis (AVS) for finite element (FE) computations of convection diffusion equations for non-constant and highly oscillatory coefficients is introduced, called the AVS-FE method. The idea of least squares FEM is used, introduced in [36], that transforms second-order PDEs into a system of first-order PDEs, by using the fluxes as variables. In the derivation of the weak formulation a Petrov-Galerkin method is used, where the trial functions are global C^0 functions and the test functions come from discontinuous Hilbert spaces. This choice allows for the use of the DPG method [2] that was summarized previously. Using the DPG method, optimal test functions corresponding to the global C^0 trial functions can be constructed. Setting up the test functions in this way guarantees the unconditional stability of the equations governing the FE approximation.

The authors motivate their choice for C^0 trial functions by stating this enables them to enforce the continuity of all variables strongly and in a simple manner. This continuity of the variables is useful for the analysis of fluxes with highly oscillatory diffusion coefficients. (!) Before going deeper into the advantages of using continuous function spaces, make sure you understand the point that is being made about not having to introduce the edge fluxes as auxiliary variables.

3.2.1 Derivation of Integral Statement and FE Discretisation

To describe their method, the authors introduce a convection-diffusion equation with homogeneous Dirichlet boundary conditions and non-homogeneous Neumann boundary conditions

$$\begin{aligned} &\text{Find } u \text{ such that} \\ -\nabla\cdot(\mathbf{D}\nabla u)+\mathbf{b}\cdot u &= f \quad \text{in } \Omega \\ u &= 0, \quad \text{on } \Gamma_D \\ \mathbf{D}\nabla u\cdot\mathbf{n} &= g, \quad \text{on } \Gamma_N \end{aligned} \tag{44}$$

Here, $\Omega \subset \mathbb{R}^2$ is an open bounded domain with. The corresponding Lipschitz boundary $\partial\Omega$, consists of two parts: Γ_D and Γ_N, such that $\Gamma_D \cap \Gamma_N = \emptyset$ and $\partial\Omega = \overline{\Gamma_D \cup \Gamma_N}$. Furthermore, $\mathbf{D}$ denotes the second order diffusion tensor that has symmetric, bounded and positive definite coefficients $D_{i,j} \in L^\infty(\Omega)$. The convection coefficient is denoted by $\mathbf{b} \in [L^2(\Omega)]^2$, the source function by $f \in L^2(\Omega)$ and the neumann boundary data by $g \in H^{-1/2}(\Gamma_N)$. Lastly, $\mathbf{n}$ denotes the outward unit normal vector to the boundary.
As was done in [36], a new auxiliary variable is introduced for the flux: $\mathbf{q} = \{q_x, q_y\}^T = \mathbf{D}\nabla u$. Using the new variable, the problem (44) can be rewritten as a first-order system of PDEs

$$\begin{aligned} &\text{Find } (u,\mathbf{q}) \in H^1(\Omega)\times H(\text{div},\Omega) \text{ such that:} \\ \mathbf{q}-\mathbf{D}\nabla u &= 0, \quad \text{in } \Omega \\ -\nabla\cdot\mathbf{q}+\mathbf{b}\cdot\nabla u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \Gamma_D, \\ \mathbf{q}\cdot\mathbf{n} &= g, \quad \text{on } \Gamma_N \end{aligned} \tag{45}$$

Next, the DPG version of (45) is constructed. First, a partition of subdomains of Ω is created, denoted by P_h. This partition consists of elements K_m with diameter h_m such that

$$\Omega = \text{int}\left(\bigcup_{K_m\in P_h} \overline{K_m}\right) \tag{46}$$

Then, the authors enforce the PDE weakly on each element in the partition and combine the first two equations of (45) multiplied with test functions $\mathbf{w}_m$ and v_m to get to

$$\begin{aligned} \int_{K_m}\left\{[\mathbf{q}_m-\mathbf{D}\nabla u_m]\cdot\mathbf{w}_m+[-\nabla\cdot\mathbf{q}_m+\mathbf{b}\cdot\nabla u_m]\,v_m\right\}dx = \int_{K_m} f v_m dx \\ \forall(v_m,\mathbf{w}_m) \in L^2(K_m)\times[L^2(K_m)]^2 \end{aligned} \tag{47}$$

Here, u_m and $\mathbf{q}_m$ are restrictions of u and $\mathbf{q}$ respectively. Summing the equations of the form (47) for the all the elements in the partition results in

$$\begin{aligned} \sum_{K_m\in P_h}\int_{K_m}\left\{[\mathbf{q}_m-\mathbf{D}\nabla u_m]\cdot\mathbf{w}_m+[-\nabla\cdot\mathbf{q}_m+\mathbf{b}\cdot\nabla u_m]\,v_m\right\}dx \\ = \sum_{K_m\in P_h}\int_{K_m} f v_m dx, \quad \forall(v,\mathbf{w}) \in L^2(\Omega)\times\left[L^2(\Omega)\right]^2 \end{aligned} \tag{48}$$

To simplify (48) Green's theorem is applied to the $(\nabla \cdot \mathbf{q}_m)v_m$ terms. Using Green's theorem does require that each $v_m \in H^1$ for the corresponding element K_m. The result looks like this

$$\begin{aligned} &\text{Find } (u,\mathbf{q}) \in H^1(\Omega) \times H(\text{div},\Omega): \\ &\sum_{K_M \in P_h} \Bigg\{ \Bigg[(\mathbf{q}_m - \mathbf{D}\nabla u_m)\cdot \mathbf{w}_m + \mathbf{q}_m \cdot \nabla v_m + (\mathbf{b}\cdot\nabla u_m) v_m \Bigg] dx \\ &\quad - \oint_{\partial K_m} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\gamma_0^m(v_m) ds \Bigg\} = \sum_{K_m \in P_h} \int_{K_m} f v_m dx, \\ &\quad \forall (v,\mathbf{w}) \in H^1(P_h) \times [L^2(\Omega)]^2 \end{aligned} \tag{49}$$

here $\gamma_0 : H^1(K_m) :\to H^{1/2}(\partial K_m)$ and $\gamma_{\mathbf{n}}^m : H(\text{div}, K_m) \to H^{-1/2}(\partial K_m)$ denote the trace and normal trace operators on K_m. Furthermore, H^1 is defined on the partition P_h as follows

$$H^1(P_h) \overset{\text{def}}{=} \Bigg\{ v \in L^2(\Omega) : v_m \in H^1(K_m), \forall K_m \in P_h \Bigg\} \tag{50}$$

Equation (49) can be rewritten by splitting up ∂K_m into $\partial K_m \backslash \overline{\Gamma_D \cup \Gamma_N}$, $\partial K_m \cap \Gamma_D$, and $\partial K_m \cap \Gamma_N$ as follows

$$\begin{aligned} &\text{Find } (u,\mathbf{q}) \in H^1(\Omega)\times H(\text{div},\Omega): \\ &\sum_{K_m \in P_h} \Bigg\{ \int_{K_m} \Bigg[(\mathbf{q}_m - \mathbf{D}\nabla u_m)\cdot \mathbf{w}_m + \mathbf{q}_m \cdot \nabla v_m + (\mathbf{b}\cdot\nabla u_m) v_m \Bigg] dx \\ &\quad - \int_{\partial K_m \backslash \overline{\Gamma_D \cup \Gamma_N}} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\gamma_0^m(v_m) ds - \int_{\partial K_m \cap \Gamma_D} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\gamma_0^m(v_m) ds \\ &\quad - \int_{\partial K_m \cap \Gamma_N} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\gamma_0^m(v_m) ds \Bigg\} = \sum_{K_m \in P_h} \int_{K_m} f v_m dx, \\ &\quad \forall (v,\mathbf{w}) \in H^1(P_h) \times [L^2(\Omega)]^2 \end{aligned} \tag{51}$$

To arrive at the final variational form, the authors enforce the Neumann boundary condition on the trace $\mathbf{q}$ and restrict the traces of the test function v_m on the Dirichlet boundary condition

$$\begin{aligned} &\text{Find } (u,\mathbf{q}) \in U(\Omega): \\ &\sum_{K_m \in P_h} \Bigg\{ \int_{K_m} \Bigg[(\mathbf{q}_m - \mathbf{D}\nabla u_m)\cdot \mathbf{w}_m + \mathbf{q}_m \cdot \nabla v_m + (\mathbf{b}\cdot\nabla u_m) v_m \Bigg] dx \\ &- \int_{\partial K_m \backslash \overline{\Gamma_D \cup \Gamma_N}} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\gamma_0^m(v_m) ds \Bigg\} = \sum_{K_m \in P_h} \Bigg\{ \int_{K_m} f v_m dx + \int_{\partial K_m \cap \Gamma_N} g \gamma_0^m(v_m) ds \Bigg\}, \\ &\quad \forall (v,\mathbf{w}) \in V(P_h) \end{aligned} \tag{52}$$

Here, the trial and test spaces, $U(\Omega)$ and $V(P_h)$ look like this

$$\begin{aligned} U(\Omega) &\overset{\text{def}}{=} \Bigg\{ (u,\mathbf{q}) \in H^1(\Omega) \times H^1(\text{div},\Omega) : \gamma_0^m(u_m)_{|\partial K_m \cap \Gamma_D} = 0, \forall K_m \in P_h \Bigg\}, \\ V(P_h) &\overset{\text{def}}{=} \Bigg\{ (v,\mathbf{w}) \in H^1(P_h) \times [L^2(\Omega)]^2 : \gamma_0^m(v_m)_{|\partial K_m \cap \Gamma_D} = 0, \forall K_m \in P_h \Bigg\} \end{aligned} \tag{53}$$

The norms $\|\cdot\|_{U(\Omega)} : U(\Omega) \to [0,\infty)$ and $\|\cdot\|_{V(P_h)} V(P_h) \to [0,\infty)$ are defined as follows:

$$\begin{aligned} \|(u,\mathbf{q})\|_{U(\Omega)} &\overset{\text{def}}{=} \sqrt{\int_\Omega \left[\nabla u \cdot \nabla u + u^2 + (\nabla \cdot \mathbf{q})^2 + \mathbf{q}\cdot\mathbf{q}\right] dx} \\ \|(v,\mathbf{w})\|_{V(P_h)} &\overset{\text{def}}{=} \sqrt{\sum_{K_m \in P_h} \int_{K_m} \left[h_m^2 \nabla v_m \cdot \nabla v_m + v_m^2 + \mathbf{w}_m \cdot \mathbf{w}_m\right] dx} \end{aligned} \tag{54}$$

The weak formulation in (52) can be written more compactly in the following way

$$\begin{aligned} &\text{Find } (u,\mathbf{q}) \in U(\Omega) \text{ such that:} \\ &\qquad B((u,\mathbf{q}),(v,\mathbf{w})) = F(v,\mathbf{w}), \quad \forall (v,\mathbf{w}) \in V(P_h) \end{aligned} \tag{55}$$

where $B((u,\mathbf{q}),(v,\mathbf{w}))$ and $F(v,\mathbf{w})$ denote the left and right hand side of (52) respectively.
The problem statement in (55) is a DPG formulation. The big difference with the approach from [2] however, is that the trial spaces are globally continuous, meaning that each trial function has support on multiple elements. The fact that in [2] each trial function is supported on a single element only introduces the requirement of using numerical traces and fluxes as auxiliary variables. The authors in [35] state that by introducing globally continuous trial functions they attempt to keep the formulation as close as possible to a standard FE discretisation.

3.2.2 AVS-FE Discretisation

The next step is to find the numerical approximations $(u^h, \mathbf{q}^h)$ of the solutions $(u, \mathbf{q})$ of the weak formulation (55). To do so, a finite element discretisation has to be derived. The authors proceed by introducing a family of invertible maps, $\{\mathbf{F}_m : \hat{K} \subset \mathbb{R}^2 \to \Omega\}$, such that every $K_m \in P_h$ is the image of element $\hat{K}$ and one of the mappings F_m, as shown in Figure 2.

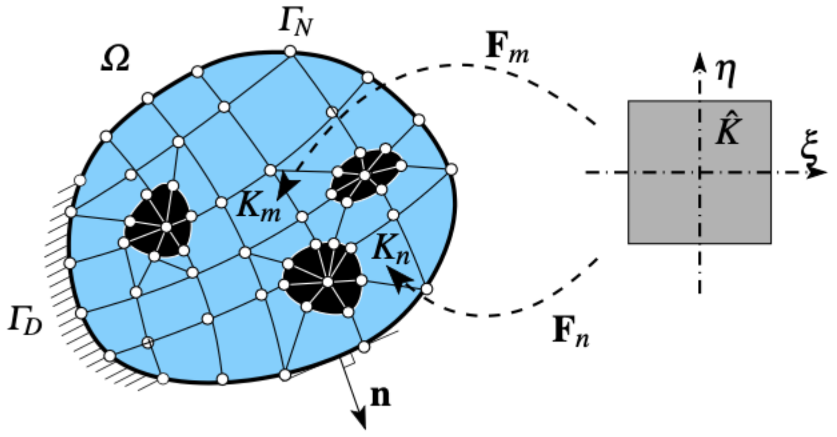

Figure 2: Discretisation of Ω, [35]

As was mentioned in the previous section, the authors of [35] use globally continuous trial functions in the approximations $(u^h, \mathbf{q}^h)$. The resulting space of trial functions, $U^h(\Omega) \subset U(\Omega)$, looks like this

$$U^h(\Omega) \overset{\text{def}}{=} \Big\{(\phi^h, \boldsymbol{\theta}^h) \in C^0(\Omega) \times [C^0(\Omega)]^2 : (\phi_{|K_M}, \boldsymbol{\theta}^h_{|K_m}) = (\hat{\phi}, \hat{\boldsymbol{\theta}}_{|K_m}) \circ \mathbf{F}_m, \\ \hat{\phi} \in \mathcal{P}^{p_m}(\hat{K}) \wedge \hat{\boldsymbol{\theta}} \in [\mathcal{P}^{p_m}(\hat{K})]^2, \quad \forall K_m \in P_h \Big\} \tag{56}$$

Here p_m denotes the degree of the polynomial approximation on element K_m. In line with the classical finite element method, $(u^h, \mathbf{q}^h)$ are approximated using linear combinations of trial functions $(e^i(\mathbf{x}), (E^j_x(\mathbf{x}), E^k_y(\mathbf{x}))) \in U^h(\Omega)$ with constants $\{u^h_i \in \mathbb{R}, i = 1, 2, \ldots, N\}, \{q^{h,j}_x \in \mathbb{R}, j = 1, 2, \ldots, N\}$ and $\{q^{h,k}_y \in \mathbb{R}, k = 1, 2, \ldots, N\}$, such that

$$u^h(\mathbf{x}) = \sum_{i=1}^{N} u^h_i e^i(\mathbf{x}), \quad q^h_x(\mathbf{x}) = \sum_{j=1}^{N} q^{h,j}_x E^j_x(\mathbf{x}), \quad q^h_y(\mathbf{x}) = \sum_{k=1}^{N} q^{h,k}_y E^k_y(\mathbf{x}) \tag{57}$$

The test functions that will be used are allowed to be piecewise continuous and can be found using the DPG method [2] that was described earlier. In [35] however, each trial function will be paired with a vector valued test function (versus a scalar valued one in the original DPG method). More specifically, $e^i(\mathbf{x})$ is paired with $(\tilde{e}^i, \tilde{\mathbf{E}}^{\mathbf{i}}) \in V(P_h)$, $E^j_x(\mathbf{x})$ with $(\tilde{e}^j_x, \tilde{\mathbf{E}}^{\mathbf{j}}_{\mathbf{x}}) \in V(P_h)$ and $\tilde{E}^k_y(\mathbf{x})$ with $(\tilde{e}^k_y, \tilde{\mathbf{E}}^{\mathbf{k}}_{\mathbf{y}}) \in V(P_h)$. Pairing the trial and test functions in this way results in the following variational problems

$$\begin{aligned}
\Big((r, \mathbf{z}), (e^i, \tilde{\mathbf{E}}^{\mathbf{i}})\Big)_{V(P_h)} &= B((e^i, \mathbf{0}), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(P_h), i = 1, \ldots, N, \\
\Big((r, \mathbf{z}), (\tilde{e}^j_x, \tilde{\mathbf{E}}^{\mathbf{j}}_{\mathbf{x}})\Big)_{V(P_h)} &= B((0, (E^j_x, 0)), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(P_h), j = 1, \ldots, N, \\
\Big((r, \mathbf{z}), (\tilde{e}^k_y, \tilde{\mathbf{E}}^{\mathbf{k}}_{\mathbf{y}})\Big)_{V(P_h)} &= B((0, (0, E^k_y)), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(P_h), k = 1, \ldots, N
\end{aligned} \tag{58}$$

Here, $(\cdot, \cdot)_{V(P_h)} : V(P_h) \times V(P_h) \to \mathbb{R}$ is the inner product that is defined as follows

$$((r, \mathbf{z}), (v, \mathbf{w}))_{V(P_h)} \overset{\text{def}}{=} \sum_{K_m \in P_h} \int_{K_m} \Big[h^2_m \nabla r_m \cdot \nabla v_m + r_m v_m + \mathbf{z}_m \cdot \mathbf{w}_m\Big] dx \tag{59}$$

The test functions that are optimal in terms of the error in the energy norm can be found by solving the variational problems in (58). Lastly, the authors note that by restricting the functions $(r, \mathbf{z}) \in V(P_h)$ so that they vanish outside a given element K_m, the local restriction of the test functions on K_m can be computed by solving the following, restricted version of (58)

$$\begin{aligned}
\Big((r, \mathbf{z}), (e^i, \tilde{\mathbf{E}}^{\mathbf{i}})\Big)_{V(K_m)} &= B_{|K_m}((e^i, \mathbf{0}), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(K_m), i = 1, \ldots, N, \\
\Big((r, \mathbf{z}), (\tilde{e}^j_x, \tilde{\mathbf{E}}^{\mathbf{j}}_{\mathbf{x}})\Big)_{V(K_m)} &= B_{|K_m}((0, (E^j_x, 0)), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(K_m), j = 1, \ldots, N, \\
\Big((r, \mathbf{z}), (\tilde{e}^k_y, \tilde{\mathbf{E}}^{\mathbf{k}}_{\mathbf{y}})\Big)_{V(K_m)} &= B_{|K_m}((0, (0, E^k_y)), (r, \mathbf{z})), && \forall (r, \mathbf{z}) \in V(K_m), k = 1, \ldots, N
\end{aligned} \tag{60}$$

Here, the authors use $B_{|K_m}(\cdot,\cdot)$ as the restriction of $B(\cdot,\cdot)$ to the element K_m and define

$$
\begin{aligned}
V(K_m) &\overset{\text{def}}{=} \Big\{(v,\boldsymbol{w}) \in H^1(K_m) \times [L^2(K_m)]^2 : \gamma_0^m(v_m)_{|\partial K_m \cap \Gamma_D} = 0\Big\}, \\
&(\cdot,\cdot)_{V(K_m)} : V(K_m) \times V(K_m) \to \mathbb{R}, \\
((r,\boldsymbol{z}),(v,\boldsymbol{w}))_{V(K_m)} &\overset{\text{def}}{=} \int_{K_m} \Big[h_m^2 \nabla r \cdot \nabla v + rv + \boldsymbol{z}\cdot\boldsymbol{w}\Big] dx
\end{aligned}
\tag{61}
$$

Consider the restriction of $B(\cdot,\cdot)$ for K_m on the functions $(\phi,\boldsymbol{\theta})$ with the same regularity as the globally continuous trial functions and test functions $(r,\mathbf{z}) \in V(P_h)$ that are non-zero only on K_m. From the left hand side of (52) it follows that

$$
\begin{aligned}
B_{|K_m}((\phi,\boldsymbol{\theta}),(r,\boldsymbol{z})) = \int_{K_m} &\Big[(\boldsymbol{\theta}_m - \boldsymbol{D}\nabla\phi_m)\cdot \boldsymbol{z}_m + \boldsymbol{\theta}_m \cdot \nabla r_m + (\boldsymbol{b}\cdot\nabla\phi_m) r_m\Big] dx \\
&- \int_{\partial K_m \setminus \overline{\Gamma_D \cup \Gamma_N}} \gamma_{\boldsymbol{n}}^m(\boldsymbol{\theta}_m)\gamma_0^m(r_m) ds
\end{aligned}
\tag{62}
$$

The authors point out a very convenient result that stems from (62). Because the restricted bilinear form in (60) only has effect on the element K_m, a test function is non-zero only if the corresponding trial function is non-zero. This means that the support of each test function is identical to the support of the equivalent trial function.
Finally, the authors give the FE discretisation of (52), that governs the AVS-FE approximation $(u^h,\mathbf{q}^h) \in U^h(\Omega)$ of $(u,\boldsymbol{q})$

$$
\begin{gathered}
\text{Find } (u^h,\boldsymbol{q}^h) \in U^h(\Omega) \text{ such that:} \\
B((u^h,\boldsymbol{q}^h),(v^*,\boldsymbol{w}^*)) = F((v^*,\boldsymbol{w}^*)), \quad \forall (v^*,\boldsymbol{w}^*) \in V^*(P_h)
\end{gathered}
\tag{63}
$$

Here $V^*(P_h) \subset V(P_h)$ is spanned by the approximations of the test functions

$$
\{(\tilde{e}_h^i, \tilde{\boldsymbol{E}}_{\boldsymbol{h}}^{\boldsymbol{i}})\}_{i=1}^N, \{(\tilde{e}_{x_h}^j, \tilde{\boldsymbol{E}}_{\boldsymbol{x_h}}^{\boldsymbol{j}})\}_{j=1}^N, \text{ and } \{(\tilde{e}_{y_h}^k, \tilde{\boldsymbol{E}}_{\boldsymbol{y_h}}^{\boldsymbol{k}})\}_{k=1}^N. \tag{64}
$$

that can be computed using (60). As was mentioned a few times, the fact that the philosophy of the DPG method is used means that the finite element discretisation in (63) is unconditionally stable, which removes the need for additional mesh dependent stabilization.

4 Data-Driven Approaches

The goal of this section is to provide a comprehensive overview of data-driven approaches for solving PDEs. More specifically, how deep learning has been used to solve PDEs. Deep learning has been one of the most revolutionary disciplines in machine learning of the past decade. Due to advances in computational resources, deep learning has been extremely successful in fields like computer vision, natural language processing, and speech recognition. To work well however, deep learning approaches generally need a lot data. This restriction has lead to a slower adoption rate in fields in which the cost of data is expensive, like in the case of solving complex engineering systems. In the past couple of years, the use of deep learning to solve partial differential equations and boundary

value problems has become a more common practice. Some approaches get around the lack of data by incorporating prior knowledge about a PDE into the loss function used to train deep neural networks, while other methods use the weak formulation to rewrite the original equation into a min-max problem.
Neural networks have several advantages over numerical approaches like the finite element method. When neural networks convergence correctly, there is no issue of instability in the solution. Numerical methods that discretise a domain into a grid are limited in their application to higher-dimensional problems, due to the curse of dimensionality; neural networks do not have this problem. Furthermore, neural networks can produce solutions almost instantaneously once trained. This property becomes especially powerful when combined with new architectures like the Deep Operator Network [3] that can learn to approximate the operators (functions that map from a space of functions into a space of functions) that govern a PDE.

4.1 Deep Ritz Method: a Deep Learning-Based Numerical Algorithm for Solving Variational Problems

In [32] a deep learning method is proposed called the Deep Ritz Method, for numerically solving variational problems. The name of the method comes from the way neural networks represent functions in the Ritz method.

The authors are interested in solving the following variational problem

$$\min_{u \in H} I(u) \tag{65}$$

with

$$I(u) = \int_{\Omega} \left(\frac{1}{2} |\nabla u(x)|^2 - f(x)u(x) \right) dx \tag{66}$$

and where H is a set of so-called trial functions, denoted by u. Furthermore, f is a function that is given, that represents external forcing to the system. The following ideas form the underpinnings of the Deep Ritz Method

1. The approximation of the trial functions by neural networks.
2. A numerical quadrature rule for the functional.
3. An algorithm for solving the final optimization problem.

4.1.1 Approximating the Trial Functions

Approximating the trial functions by a neural network is the basic part of the Deep Ritz method. This approximation is essentially a nonlinear transformation

$$x \to z_\theta(x) \in \mathbb{R}^m \tag{67}$$

where z_θ denotes a neural network with parameters represented by θ. The layers of the network used in [32] consist of stacks of so-called blocks. Each block is made up out of two linear transformation, two nonlinear activation functions, and a residual connection (residual meaning that the input of layers is added back to outputs of layers further down the network). Block i can be written as a function of its input

$$f_i(s) = \phi(W_{i,2} \cdot \phi(W_{i,1}s + b_{i,1}) + b_{i,2}) + s \tag{68}$$

Here, $W_{i,1}$, $W_{i,2}$ are $m \times m$ matrices of block i, and ϕ is an activation function. By adding the term s in (68), i.e., the residual connection, the vanishing gradient is avoided making the network easier to train. Figure 3 shows what the neural network architecture, composed of two blocks and an output layers, looks like.

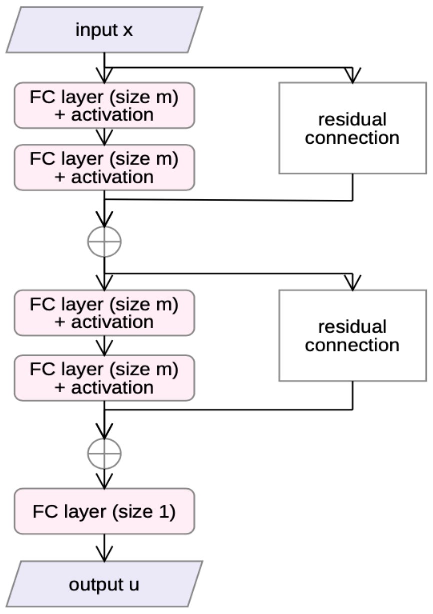
input x
FC layer (size m) + activation
FC layer (size m) + activation
residual connection
FC layer (size m) + activation
FC layer (size m) + activation
residual connection
FC layer (size 1)
output u

Figure 3: Neural network composed of two blocks and a linear output layer, [32]

Combining these separate blocks, the full network can be represented as follows

$$z_\theta(x) = f_n \circ \ldots \circ f_1(x) \tag{69}$$

where, again, θ represents the parameters corresponding to the network. Now that the neural network architecture has been laid out, the approximation of the trial function can be specified

$$u(x;\theta) = a \cdot z_\theta(x) + b \tag{70}$$

If the functional in (66) is rewritten to

$$g(x;\theta) = \frac{1}{2} \left|\nabla_x u(x;\theta)\right|^2 - f(x)u(x;\theta) \tag{71}$$

then the minimization problem from (65) can be rewritten to

$$\min_\theta L(\theta), \quad L(\theta) = \int_\Omega g(x;\theta)dx \tag{72}$$

4.1.2 Mini-Batch Gradient Descent and Discretisation

Now that approximation of the trial functions has been explained, there are two more parts left. The optimization algorithm that will be used to train the neural network and the discretisation of the integral in (72). This discretisation of (72) is needed because integrals with functions as the one defined in (71) cannot be calculated explicitly.
To optimize the neural network the *mini-batch gradient descent* algorithm will be used, which is an algorithm that is very similar to the *stochastic gradient descent* algorithm. In stochastic gradient descent the parameters in the network are updated by calculating the loss function corresponding to a randomly chosen point in the training dataset. One iteration of the algorithm looks like this

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(x;\theta) \tag{73}$$

Here, θ represents the network parameters, η is parameter that is user-chosen and $\mathcal{L}(x;\theta)$ is the loss function evaluated at the point x with network parameters θ. The mini-batch gradient descent algorithm differs from the stochastic gradient descent algorithm in that it calculates the loss function corresponding to a randomly chosen *batch* of points in the training dataset (instead of just a single random point). One iteration of the mini-batch gradient descent algorithm looks like this

$$\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(x_i;\theta) \tag{74}$$

To use this algorithm to minimize (72) a method for computing the integral is needed. The authors in [32] proceed in the following way: for each iteration k of the training algorithm, they uniformly sample a mini-batch $\{x_{j,k}\}_{j=1}^{N}$ from the domain Ω, and use the mean of $g(x;\theta)$ evaluated at those N points to represent the integral. A single iteration in the resulting approach looks like this

$$\theta^{k+1} \leftarrow \theta^{k} - \eta \nabla_\theta \frac{1}{N}\sum_{j=1}^{N} g(x_{j,k};\theta) \tag{75}$$

The fact that the points used to evaluate the integral are sampled randomly is important here. If fixed points were used the algorithm would only minimize the integral at those points and not necessarily over the entire domain.

4.2 Deep Least-Squares Methods: An Unsupervised Learning-Based Numerical Method for Solving Elliptic PDEs

In [33] an unsupervised deep learning based approach for solving PDEs is proposed. The method uses least-squares functionals as loss functions that can be minimized by deep neural networks. The authors consider the following problem. Let Ω be a bounded subset of $\mathbb{R}^d$ with Lipschitz boundary $\partial\Omega = \bar{\Gamma}_D \cup \bar{\Gamma}_N$. The authors consider the following second-order scalar elliptic pde

$$-\nabla\cdot(A\nabla u) + Xu = f, \quad \text{in } \Omega \tag{76}$$

with

$$u = g_D, \text{ on } \Gamma_D \text{ and } -\mathbf{n}\cdot A\nabla u = g_N, \text{ on } \Gamma_N \tag{77}$$

with $f \in L^2(\Omega)$, $g_D \in H^{1/2}(\Gamma_D), g_N \in H^{-1/2}(\Gamma_N)$. Furthermore, $A(x)$ is an $n\times n$ symmetric matrix of functions in $L^2(\Omega)$, X is a linear differential operator of order at most one, and $\mathbf{n}$ is the outward unit vector normal to the boundary. Lastly, the authors assume that A is uniformly positive definite.

Since problem (76)-(77) is generally non-symmetric, it does not have an underlying minimization principle that can be used to solve it. Since neural networks essentially solve minimization problems, (76)-(77) needs to be rewritten so that it does lend itself to minimization. This is where the least squares formulation comes in. The authors use the so called first-order system least squares (FOSLS) formulation that was introduced in [34]. It is possible to rewrite (76)-(77), a second order problem, into a first-order system, by introducing a flux variable $\mathbf{q} = -A\nabla u$ and setting $\mathbf{q} + A\nabla u = 0$:

$$\begin{cases} \nabla \cdot \mathbf{q} + Xu & = f, \text{ in } \Omega \\ \mathbf{q} + A\nabla & = 0, \text{ in } \Omega \end{cases} \tag{78}$$

with

$$u = g_D, \text{ on } \Gamma_D \text{ and } \mathbf{n}\cdot\mathbf{q} = g_N, \text{ on } \Gamma_N \tag{79}$$

Now, let

$$H(\text{div};\Omega) := \{\mathbf{v} \in L^2(\Omega)^d : \text{ div } \mathbf{v} \in L^2(\Omega)\} \tag{80}$$

and denote the subsets of $H^1(\Omega)$ and $H(\text{div};\Omega)$ that satisfy the non-homogeneous boundary conditions by

$$H^1_{D,g}(\Omega) = \{v \in H^1(\Omega) : v|_{\Gamma_D} = g_D\} \text{ and } H_{N,g} = \{\boldsymbol{\tau} \in H(\text{div};\Omega) : \boldsymbol{\tau}\cdot\mathbf{n}|_{\Gamma_N} = g_N\} \tag{81}$$

respectively. If $g_D = 0$ and $g_N = 0$, then the subsets in (81) become subspaces and will be denoted by $H^1_D(\Omega)$ and $H_N(\text{div};\Omega)$. Let

$$\mathcal{V}_g = H_{N,g}(\text{div};\Omega) \times H^1_{D,g}(\Omega) \text{ and } \mathcal{V}_0 = H_n(\text{div};\Omega) \times H^1_D(\Omega), \tag{82}$$

Now, given (82) the FOSLS formulation is as follows: find $(\mathbf{q}, u) \in \mathcal{V}_g$ such that

$$\tilde{\mathcal{G}}(\mathbf{q}, u; \mathbf{f}) = \min_{(\boldsymbol{\tau},v)\in\mathcal{V}_g} \tilde{\mathcal{G}}(\boldsymbol{\tau}, v; \mathbf{f}) \tag{83}$$

Here, $\mathbf{f} = (f, g_D, g_N)$ and $\tilde{\mathcal{G}}(\boldsymbol{\tau}, u; \mathbf{f})$ is defined as

$$\tilde{\mathcal{G}}(\boldsymbol{\tau}, v; \mathbf{f}) = \|\nabla\cdot\boldsymbol{\tau} + Xv - f\|^2_{0,\Omega} + \|A^{-1/2}\boldsymbol{\tau} + A^{1/2}\nabla v\|^2_{0,\Omega} \tag{84}$$

The authors mention that it has been proven in [34] that the homogeneous functional $\bar{\mathcal{G}}(\boldsymbol{\tau}, v; \mathbf{0})$ is coercive and bounded in $\mathcal{V}_0$, i.e., positive constants c_1 and c_2 exist such that

$$c_1 |||(\boldsymbol{\tau}, v)|||^2 \le \tilde{\mathcal{G}}(\boldsymbol{\tau}, v; \mathbf{0}) \le c_2 |||(\boldsymbol{\tau}, v)|||^2, \quad \forall(\boldsymbol{\tau}, v) \in \mathcal{V}_0 \tag{85}$$

Here, the notation $|||\cdot|||$ is used to for the FOSLS energy norm given by

$$|||(\boldsymbol{\tau}, v)||| = (\|\boldsymbol{\tau}\|^2_{0,\Omega} + \|\nabla\cdot\boldsymbol{\tau}\|^2_{0,\Omega} + \|v\|^2_{1,\Omega})^{1/2} \tag{86}$$

This result (the coercivity and boundedness of $\tilde{\mathcal{G}}(\boldsymbol{\tau}, v; \mathbf{0})$) is very important as it implies that (83) is well-posed, i.e., that it has a unique solution.

The neural network that will be used to approximate the solution to (76) does need to satisfy the boundary conditions (77). However, in [32] it was observed that in general it is not easy to make a neural network satisfy prescribed boundary conditions. Therefore, the authors of [33] proceed by adding both the Dirichlet and Neumann boundary conditions to the functional in (84)

$$
\begin{aligned}
\mathcal{G}(\boldsymbol{\tau}, v; \boldsymbol{f}) = \|\nabla \cdot \boldsymbol{\tau} + Xv - f\|^2_{0,\Omega} + \|A^{-1/2}\boldsymbol{\tau} + A^{1/2}\nabla v\|^2_{0,\Omega} \qquad (87)\\
+\alpha_D \|v - g_D\|^2_{1/2,\Gamma_D} + \alpha_N \|\boldsymbol{n} \cdot \boldsymbol{\tau} - g_N\|^2_{-1/2,\Gamma_N} \qquad (88)
\end{aligned}
$$

for all $(\boldsymbol{\tau}, v) \coloneqq H(\text{div}; \Omega) \times H^1(\Omega)$. Here α_D and α_N are constants that need to be added to deal with the difference in scale between the interior norm and the boundary norm (by the Sobolev trace theorem). The new FOSLS formulation becomes: find $(\boldsymbol{q}, u) \in \mathcal{V}$ such that

$$
\mathcal{G}(\boldsymbol{q}, u; \boldsymbol{f}) = \min_{(\boldsymbol{\tau}, v) \in \mathcal{V}} \mathcal{G}(\boldsymbol{\tau}, v; \boldsymbol{f}) \qquad (89)
$$

It is possible to prove the functional $\mathcal{G}(\boldsymbol{\tau}, v; \boldsymbol{0})$ is coercive and bounded in $\mathcal{V}$, which again means that the problem (89) has a unique solution.

To approximate the solution $(\hat{\boldsymbol{q}}(x, \theta), \hat{u}(x, \theta))$ a neural network is used, where θ again is used to denote all the parameters in the neural network. The parameters θ can be optimized using any kind of gradient descent algorithm.

4.3 Weak Adversarial Networks

In [1] an new approach is proposed for solving PDEs. The usefulness of this approach lies in the fact that it is very applicable to high-dimensional problems on arbitrary domains. Most of the conventional approaches that are used to solve PDEs, like finite difference and finite element methods, discretise the domain Ω. While these methods are very capable at solving highly complex problems, they suffer from the curse of dimensionality. As the dimension d of the PDE increases, the number of grid points used in the discretisation increase exponentially. The method proposed in [1], using so called *weak adversarial networks* (WAN) does not suffer from the curve of dimensionality because it avoids using any kind of discretisation.

Instead of discretising the domain, the new method uses so-called weak adversarial networks to solve the weak formulation of a PDE. In the weak formulation of the PDE, the weak solution and the test function are parameterised using two neural networks. By rewriting the weak form into a min max problem, these two networks can be trained in an unsupervised way. The first network, representing the weak solution, will learn to minimize the loss function corresponding to this problem, while the second network will maximise the same loss function.

The PDE that is used as an example is the second-order elliptic PDE with Dirichlet's or Neumann's boundary conditions on a domain $\Omega \subset \mathbb{R}^d$,

$$
\begin{cases}
-\sum_{i=1}^d \partial_i(\sum_{j=1}^d a_{ij}\partial_j u) + \sum_{i=1}^d b_i \partial_i u + cu - f = 0, \text{ in } \Omega \\
u(x) - g(x) = 0 \quad \text{ or } \quad (\partial u / \partial \overrightarrow{n})(x) - g(x) = 0 \text{ on } \partial\Omega
\end{cases} \qquad (90)
$$

Here, $a_{ij}, b_i, c : \Omega \to \mathbb{R}$ for $i, j \in [d] \coloneqq \{1, \ldots, d\}$, $f : \Omega \to \mathbb{R}$ and $g : \partial\Omega \to \mathbb{R}$ are all given, and $(\partial u / \partial \overrightarrow{n})(x)$ denotes the directional derivative of u along the outer normal direction at the

boundary point $x \in \partial\Omega$. The authors of the paper also assume that there exists a constant $\theta > 0$ such that $\xi^T A(x)\xi \geq \theta\,|\xi|^2$ for any $\xi = (\xi_1, \ldots, \xi_d) \in \mathbb{R}^d$ with $|\xi|^2$ and $x \in \Omega$, where $a_{ij} = a_{ji}$ for all $i, j \in [d]$ and $A(x) := [a_{ij}(x)] \in \mathbb{R}^{d\times d}$.
The paper also discusses solving PDEs involving time, but those will not be examined in this literature review.

The weak formulation can be found by multiplying the left and right hand side of the equation with a test function $\phi \in H_0^1(\Omega;\mathbb{R})$ and integrating by parts. The weak form corresponding to (90) looks like this

$$\begin{cases} \langle \mathcal{A}[u], \phi\rangle := \int_\Omega \Big(\sum_{j=1}^d \sum_{i=1}^d a_{ij}\partial_j u \partial_i \phi + \sum_{i=1}^d b_i \phi \partial_i u + cu\phi - f\phi \Big) dx = 0 \\ \mathcal{B}[u] = 0, \text{ on } \partial\Omega \end{cases} \tag{91}$$

where $H_0^1(\Omega;\mathbb{R})$ denotes the Sobolev space. Here u is called the weak solution of (91).

The weak formulation (91) can be reformulated as a min max problem in the following way. First, think of $\mathcal{A}[u] : H_0^1(\Omega) \to \mathbb{R}$ as a linear functional operator so that $\mathcal{A}[u](\phi) := \langle \mathcal{A}[u], \phi\rangle$, defined in (91). In that case the operator norm of $\mathcal{A}[u]$ that is derived from the L^2 is defined by

$$\|\mathcal{A}[u]\|_{\text{op}} := \max\left\{\langle \mathcal{A}[u], \phi\rangle / \|\phi\|_2 \,|\, \phi \in H_0^1, \phi \neq 0\right\}, \tag{92}$$

Here, $\|\phi\|_2 = \big(\int_\Omega |\phi(x)|^2\, dx\big)^{1/2}$. With this definition of the operator norm of $\mathcal{A}[u]$ it becomes evident that the weak solution u of (90) satisfies $\|\mathcal{A}[u]\|_{\text{op}} = 0$ and $\mathcal{B}[u] = 0$ on $\partial\Omega$. Since the operator norm of $\mathcal{A}[u]$, $\|\mathcal{A}[u]\|_{\text{op}}$, is zero or positive, the weak solution to (90) thus solves the following two equivalent problem

$$\min_{u\in H^1} \|\mathcal{A}[u]\|_{\text{op}}^2 \Leftrightarrow \min_{u\in H^1} \max_{\phi \in H_0^1} |\langle \mathcal{A}[u], \phi\rangle|^2 / \|\phi\|_2^2 \tag{93}$$

In [1] these results are summarised in the following theorem

Theorem 1. *Suppose that u^* satisfies the boundary condition $\mathcal{B}[u^*] = 0$, then u^* is a weak solution of the BVP* (90) *if and only if u^* solves the problems* (93) *and* $\|\mathcal{A}[u^*]\|_{op} = 0$.

With (93) it is possible to use what the authors of [1] call an "adversarial approach" to find the weak solution corresponding to (90). The goal of the approach is to train a neural network with parameter θ to learn the function $u_\theta : \mathbb{R}^d \to \mathbb{R}$, such that $\mathcal{A}[u]$ minimizes the operator norm (93). At the same time a neural network with parameter η is trained to model the test function ϕ, that challenges u_θ by maximising $\langle \mathcal{A}[u_\theta], \phi_\eta\rangle$ modulus $\|\phi_\eta\|_2$. Given the strictly monotone nature of the logarithm function, equation (93) which defines the loss function for u_θ and ϕ_η in the interior of Ω, can be reformulated into

$$L_{\text{int}}(\theta, \eta) := |\langle \mathcal{A}[u], \phi\rangle|^2 - \log\|\phi_\eta\|_2^2 \tag{94}$$

Lastly, u_θ also needs to satisfy the boundary condition $\mathcal{B}[u] = 0$ on $\partial\Omega$. If $\{x_b^{(j)}\}_{j=1}^{N_b}$ are a set of N_b points on the boundary $\partial\Omega$, then the squared error of u_θ for the Dirichlet boundary condition on $\partial\Omega$ is given by

$$L_{\text{bdry}}(\theta) := (1/N_b) \cdot \sum_{j=1}^{N_b} \Big| u_\theta(x_b^{(j)}) - g(x_b^{(j)}) \Big|^2 \tag{95}$$

If there is a Neumann boundary condition imposed instead, then $u_\theta(x_b^{(j)})$ is replaced with

$$\sum_{i=1}^{d} n_i(x_b^{(j)})\partial_i u_\theta(x_b^{(j)})$$

i.e., the outward normal derivative of u_θ. Using a weighted sum of L_{int} and L_{bdry} the final objective function for u_θ and ϕ_η is defined as a min max problem

$$\min_\theta \max_\eta L(\theta,\eta), \quad \text{where} \quad L(\theta,\eta) \coloneqq L_{\text{int}}(\theta,\eta) + \alpha L_{\text{bdry}}(\theta) \tag{96}$$

where $\alpha > 0$ is a parameter that can be set to speed up convergence.

Evaluating the objective function (96), can then be done through evaluation of L_{int} by sampling points on the interior of Ω, and through evaluation of L_{bdry} by sampling points on $\partial\Omega$. Training the networks then comes down to finding the gradients of $L(\theta,\eta)$ with respect to the network parameters θ and η. With the gradients, θ and η can be optimized using a gradient descent algorithm. Depending on the configuration used, each iteration consists of performing K_u steps of gradient descent on θ, and then performing K_ϕ steps of gradient descent on η.

4.4 Physics Informed Neural Networks

Where weak adversarial networks use the weak formulation of a PDE to solve the strong form, Physics Informed Neural Networks (PINNs), introduced in [4], use the strong form directly. PINNs are deep neural networks that are used together with automatic differentiation. As these neural networks will be differentiated with respect to their parameters and input data, they automatically respect symmetries, invariances, and conservation principles that come from the physical laws governing the data, as they are modeled from the PDEs.
[4] is divided into two parts that focus on two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. In this literature review, the focus will be on the first type of problem, the data-driven solutions of partial differential equations.

The authors consider parameterised and nonlinear partial differential equation of the form

$$u_t + \mathcal{N}[u;\lambda] = 0,\ x \in \Omega,\ t \in [0,T], \tag{97}$$

where $u(t,x)$ denotes the unknown solution, $\mathcal{N}[\cdot;\lambda]$ is a nonlinear operator parameterised by λ, and the domain Ω is a subset of $\mathbb{R}^D$. In [4], two types of algorithms are considered, continuous and discrete time models. In this literature review only continuous time models will be covered.
The authors define $f(t,x)$ to be the left hand side of equation (97), i.e., they define $f(t,x)$ such that

$$f \coloneqq u_t + \mathcal{N}[u] \tag{98}$$

and proceed by approximating $u(t,x)$ by a deep neural network. This assumption together with equation (98) results in a physics-informed neural network $f(t,x)$. This network has the same parameters as the network used to approximate $u(t,x)$, but it has different activation functions due to the impact of the differential operator $\mathcal{N}$. The shared parameters between the networks $u(t,x)$ and $f(t,x)$ can be optimized by minimizing the mean squared error loss

$$\text{MSE} = \text{MSE}_u + \text{MSE}_f \tag{99}$$

with

$$\text{MSE}_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u(t_u^i, x_u^i) - u^i\right|,$$

and

$$\text{MSE}_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f(t_f^i, x_f^i)\right|^2$$

In this formulation, the points $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary points on $u(t,x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocation points for $f(t,x)$. In this loss function, the part MSE_u relates to the initial and boundary data and the part MSE_f makes sure that equation (97) is enforced at the collocation points.
The approach in [4] differs from existing approaches in the literature that use machine learning in computational physics [9–21]. Those approaches use machine learning algorithms as *black-box* models. The approach in [4] goes one step further though, by directly implementing the underlying differential operator into the custom loss functions. The authors of [4] mention that this allows them to open the black-box by understanding and appreciating the role played by automatic differentiation in the deep learning field.

4.5 VPINNs: Variational Physics-Informed Neural Networks

In [22] a variational physics-informed neural network (VPINN) is developed. The approach in this paper operates within the Petrov-Galerkin framework. In this framework the solution is approximated by a neural network, while the test functions come from linear function spaces. The approach differs from the approach in [4] in that it evaluates the weak formulation of the problem, instead of directly incorporating the strong form. It does so by constructing a *variational loss function.* The authors of [22] list several advantages to this approach:

- Using integration by parts to reduce the order of the differential operators, reduces the required regularity in the solution space. This means that the VPINNs will be less computationally expensive compared to PINNs.
- For shallow networks and particular activation and test functions the loss function can be expressed analytically, opening the door to numerical analysis of such formulations.
- VPINNs use a relatively small number of quadrature points (compared to the penalising points used in PINNs) that are used to compute the integrals in the variational formulation.
- The weak formulation allows the possible benefit of domain decompositions. Dividing the domain up into sub-domains it is possible to use a separate number of test functions based on the local regularity of the solution. This yields a local and more flexible learning approach.

The following formulation of the governing equation of a physical problem in steady state is used

$$\mathcal{L}^q u(x) = f(x),\ x \in \Omega \tag{100a}$$

$$u(x) = h(x),\ x \in \partial\Omega \tag{100b}$$

Here $\Omega \subset \mathbb{R}^d$ with dimensionality d and boundaries $\partial\Omega$. The function $u(x) : \Omega \to \mathbb{R}$ is used to represent the underlying physics, the forcing term $f(x)$ is some external excitation, and lastly, $\mathcal{L}$

usually contains differential and/or integro-differential operators with parameters $\mathbf{q}$. A numerical approximation of the solution to this problem should obtain $\tilde{u}(x) \approx u(x)$. To ensure stability and convergence of the method, the approximation should satisfy the boundary/initial conditions.

The approximate solution $\tilde{u}(x)$ is given by the weights and the biases of the neural network, u_{NN}. Alternatively, $\tilde{u}(x) = u_{\mathrm{NN}}(x; w, b)$. The equation (100a) is multiplied by a test function $v(x)$ and integrated over the whole domain to obtain the variational form

$$(\mathcal{L}^q u_{\mathrm{NN}}(x), v(x))_\Omega = (f(x), v(x))_\Omega \tag{101a}$$

$$u(x) = h(x),\, x \in \partial\Omega \tag{101b}$$

Here, $(\cdot,\cdot)$ represents the inner product. Now that the variational form is defined, the *variational residual* is defined as

$$\begin{gathered}\text{Residual}^v = \mathcal{R} - F - r^b, \\ \mathcal{R} = (\mathcal{L}^q u_{\mathrm{NN}}, v)_\Omega,\, F = (f, v)_\Omega\end{gathered} \tag{102}$$

Here, the variational residual is enforced for all admissible test functions v_k, $k = 1, 2, \ldots$ and r^b is defined as follows

$$r^b(x) = u_{\mathrm{NN}}(x) - h(x),\, \forall x \in \partial\Omega \tag{103}$$

Then, a discrete finite dimensional space V_k is constructed by choosing a finite set of admissible test functions

$$V_k = \text{span}\{v_k, k = 1, 2, \ldots, K\} \tag{104}$$

Using the definition of the residual and of r^b the *variational loss function* can be defined

$$\begin{aligned}L^v &= L^v_R + L_u \\ L^v_R &= \frac{1}{K}\sum_{k=1}^{K} |\mathcal{R}_k - F_k|^2,\, L_u = \tau \frac{1}{N_u}\sum_{i=1}^{N_u} \left|r^b(x_{u_i})\right|^2\end{aligned} \tag{105}$$

Here, τ is a user chosen parameter and the superscript v is used to refer to the loss function corresponding to the variational form of residual. Using these new definitions the problem of solving (101a) and (101b) can be reformulated as

$$\text{find } \tilde{u}(x) = u_{\mathrm{NN}}(x; w^*, b^*) \text{ such that } \{w^*, b^*\} = \arg\min(\mathcal{L}^v(w, b)) \tag{106}$$

Given the objective function above, and the earlier defined variational loss function (105) the neural network $u_{\mathrm{NN}}(x; w, b)$ can be trained using any type of gradient descent algorithm, by evaluating the loss function at the collocation points and adjusting the parameters so that the result is minimised. Figure 4 shows the VPINN approach, consisting of the network layers, the variational residual and the test functions.

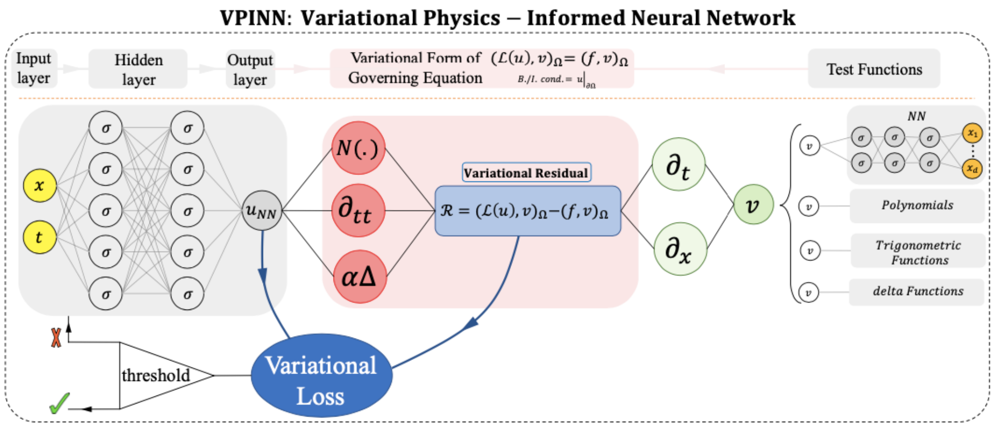
VPINN: Variational Physics – Informed Neural Network
Input layer
Hidden layer
Output layer
Variational Form of Governing Equation $(\mathcal{L}(u),v)_\Omega = (f,v)_\Omega$
Test Functions
Variational Residual
$\mathcal{R} = (\mathcal{L}(u),v)_\Omega - (f,v)_\Omega$
Polynomials
Trigonometric Functions
delta Functions
threshold
Variational Loss

Figure 4: VPINN approach, [22]

A major challenge in VPINNs, especially when compared to PINNs, is the need to compute the integrals in the variational loss function. To deal with this challenge, deeper networks are required.

4.6 Deep Operator Networks

In [3] an approach is introduced that approximates nonlinear operators using a new type of neural network architecture, called a Deep Operator Network (DeepONet). The famous universal approximation theorem states that a neural network can approximate any continuous function to an arbitrary accuracy, if it is allowed to have enough layers and neurons [23, 24]. Another approximation result, which is less well known, goes further and states that a neural network can accurately approximate any functional [25–27], which is a mapping from a function space to the real numbers, or nonlinear operator [28, 29], which is a mapping of a space of functions into a space of functions. Based on this second result from [28], the authors present the following theorem

Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{\text{branch}} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{\text{trunk}} \right| < \epsilon \tag{107}$$

holds for all $u \in V$ and $y \in K_2$ and where $G(u)(y)$ denotes the function value of the output function corresponding to the operator G and the input function u at the point y in the domain.

This theorem forms the basis for the DeepONet approach that is shown in Figure 5.

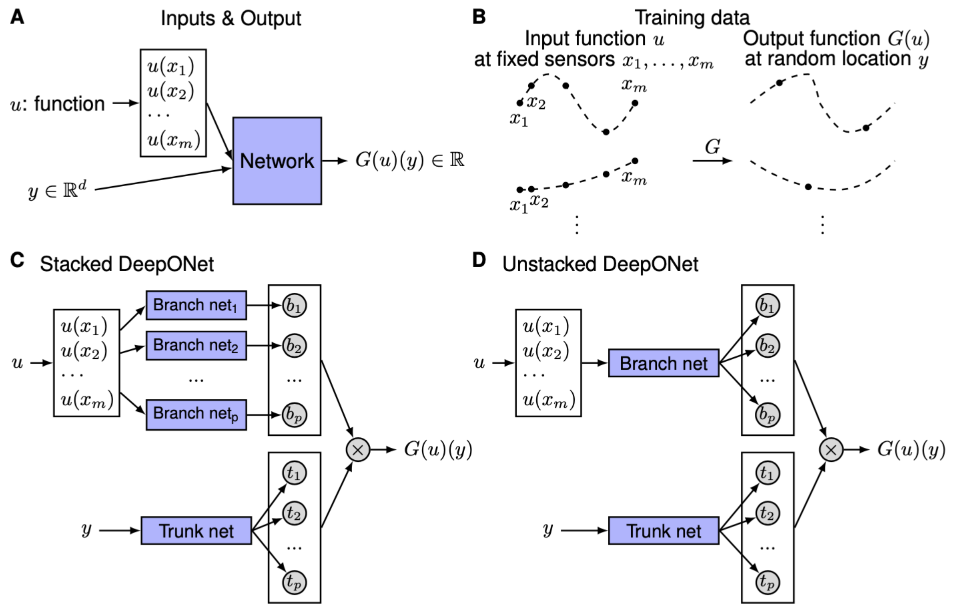

Figure 5: Deep Operator Network Approach, [3]

In **(A)** of Figure 5, the inputs and outputs of the network are shown. The network is fed an array $[u(x_1), \dots, u(x_m)]$ consisting of the values of the input function u at a finite number of fixed points, called "sensors". **(B)** shows what the training data looks like: input functions u evaluated at the fixed sensors, points in the domain y, and the corresponding solutions $G(u)(y)$.
(C) and **(D)** show two different variations of the actual network architecture. The architecture in **(C)** is what the authors call a *stacked* DeepONet, that consists of multiple *branches* and a single *trunk*. Each branch is a neural network and is used to process the information that is given by the input function u, while the trunk, also a neural network, operates on the point of the domain y. **(D)** shows what the authors call an *unstacked* DeepONet. It differs from the stacked network in that it uses a single branch network to process the information that lies in the input function u.

The focus lies on being able to learn operators in a general setting, with the single requirement for the training data being the fixed location of the sensors $\{x_1, \dots, x_m\}$ for the input functions. The inputs consists of two parts: $[u(x_1), \dots, u(x_m)]^T$ and y, and the goal is to reach high performance by using a suitable architecture. Lu Lu et al. note that one approach would be to use a regular neural network and concatenate the inputs, $[u(x_1), \dots, u(x_m), y]$, but explain that the drawback of this idea would be that as the dimension d of y gets bigger, it would no longer match the dimension of $u(x_i)$ for $i = 1, 2, \dots, m$. This in turn would mean that it would no longer be possible to treat $u(x_i)$ and y equally. Therefore two separate networks are needed to deal with $[u(x_1), \dots, u(x_m)]^T$ and y.

When Lu Lu et al. define the DeepONet they mention that the branch and the trunk networks can be any types of neural networks. There are some general points that can be made about them though. First, the trunk network takes in y as its input and outputs $[t_1, \dots, t_p]^T \in \mathbb{R}^p$. Secondly, the p branch networks each take in $[u(x_1), \dots, u(x_m)]^T$ as the input, and output a scalar $b_k \in \mathbb{R}$ for $k = 1, 2, \dots, p$. Therefore, the approximation in equation (107) can be written in terms of the outputs of the branches and the trunk

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k \tag{108}$$

Lu Lu et al. use an activation function in the final layer of the trunk network, i.e., $t_k = \sigma(\cdot)$ for $k = 1, 2, \dots, p$. If this fact is combined with equation (108), the trunk-branch network can be seen as a trunk network where each weight in the last layers, i.e. b_k, is being parameterised by another network instead of being treated as a classical parameter.
Lastly, Lu Lu et al. mention that p is at least of the order of 10, meaning that the stacked version of the DeepONet is computationally expensive, see **(C)** of Figure 5. This is why the unstacked version is created, see **(D)** of Figure 5. Here all the branch networks are merged into a single network that outputs the vector $[b_1, \dots, b_p]^T \in \mathbb{R}^p$.

4.6.1 Example: a 1D Dynamic System

The authors demonstrate that the DeepONets, due to their superior ability to generalise well, achieve better results for solving ODEs and PDEs than regular fully connected neural networks (FNNs). In one of their examples they consider the problem of finding $s(x)$ over the domain $[0, 1]$ for any $u(x)$ in the following 1D dynamic system

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0, 1] \tag{109}$$

Solving this problem is equivalent to learning the antiderivative operator

$$G : u(x) \to s(x) = \int_0^x u(\tau) d\tau \tag{110}$$

As a baseline, a regular FNN is trained to learn the operator in (110). A grid search is performed, using a depth from 2 to 4, width from 10 to 2560, and learning rates 0.01, 0.001, and 0.001 together with the Adam optimizer. Figure 6 shows the mean squared error of the resulting networks on a test dataset.

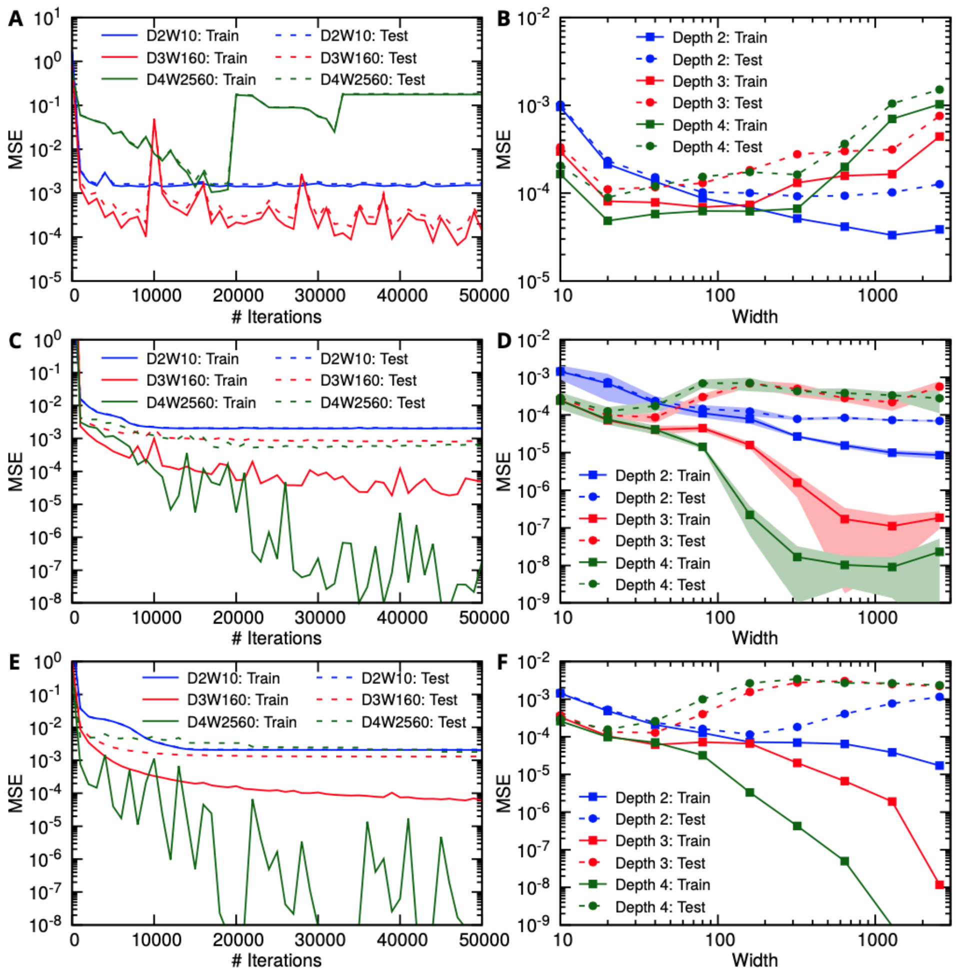

Figure 6: Errors of FNNs trained to learn the antiderivative operator, [3].

To compare with the regular neural network, a stacked and an unstacked DeepONet (see **C** and **D** of Figure 5) are trained to learn (110). Of both types, two versions are trained: one with and one without bias (resulting in four networks in total). The addition of bias means that instead of outputting (108), the networks output

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0 \tag{111}$$

where b_0 is a parameter that is learned by the network. For all the DeepONets branches of depth 2, width 40, and trunks of depth 3, width 40 are used (the authors note that they did not try to find the optimal configuration and did not use a grid search). The results of testing the DeepOnets, together with the best performing FNN, are shown in Figure 7. There are a couple of interesting points to be made.

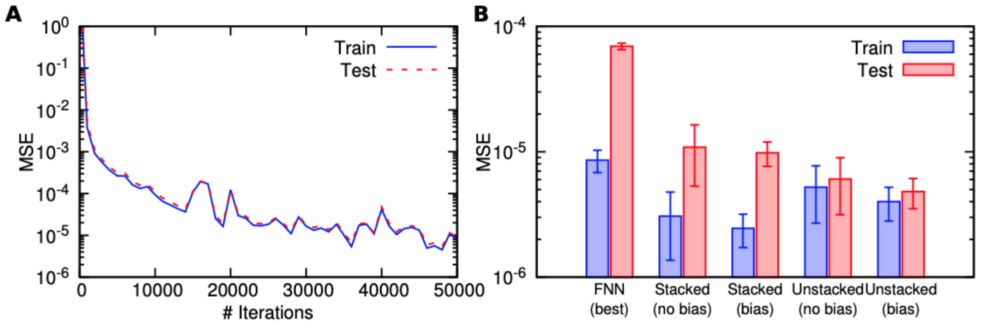

Figure 7: Errors of DeepONets trained to learn the antiderivative operator, [3].

First, Figure 7 shows that even without a grid search, the DeepONets performed better than the best FNN. The DeepONets also exhibit a much smaller generalisation error (the difference between the training error and test error).
Secondly, it turns out that adding bias to the DeepONets decreases both the training error and test error. Furthermore, while the unstacked versions have a higher training error, they achieved a much lower testing error and generalisation error (using less parameters than the stacked versions).

4.7 Neural Network Approximation of Piecewise Continuous Functions: Application to Friction Compensation

The universal approximation theorem that was introduced in [23] holds for continuous functions only. In [37] an approach for approximating functions with discontinuities is proposed. If the locations of the discontinuities of the function that is to be approximated are known, the authors show that they can be dealt with using a specific type of activation functions. Using a neural network architecture that can deal with discontinuities could be very useful in this research. It was shown in sections 3.1.4.2 and 3.2.2 that the optimal test functions with multi-element support exhibit jumps between elements. As will be covered in section 5, a major goal of this thesis will be to generate optimal test functions using neural networks. If the neural network of choice is not able to deal with the inter-element jumps in the test functions, multiple networks will be needed to approximate the continuous parts of the optimal test functions, which will reduce the efficacy of the overall approach.

4.7.1 Augmented Neural Network for Jump Function Approximation

One of the most commonly used activation functions in deep learning is the so called sigmoid function, that is defined as follows

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{112}$$

This activation function is used in the universal approximation theorem [23] for neural networks. This theorem states that a neural network with a single layer together with a sigmoid activation function can approximate any continuous function arbitrarily close, given enough neurons. The universal approximation theorem does not make any statements about discontinuous functions however, which is why the authors of [37] introduce a new type of activation called *sigmoidal jump approximation functions*. Several choices for these discontinuous functions include

$$\phi_k(x) = \begin{cases} 0, & \text{for } x < x_J \\ \left(\dfrac{1-e^{-x}}{1+e^{-x}}\right)^k, & \text{for } x \geq x_J \end{cases} \tag{113}$$

and

$$\phi_k(x) = \begin{cases} 0, & \text{for } x < x_J \\ \left(\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right)^k, & \text{for } x \geq x_J \end{cases} \tag{114}$$

Here x_J denotes the value of x for which the jump in the function occurs. These discontinuous jump functions can be used in tandem with the sigmoid activation functions. The authors of [37] propose an augmented neural network that is shown in Figure 8.

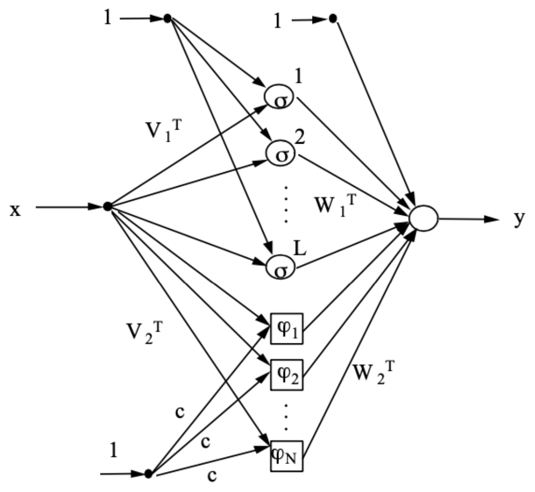

Figure 8: Augmented Neural Network, [37].

With this type of architecture, the neural network output looks like this

$$y = \sum_{l=1}^{L} w_{1,l}\, \sigma(v_{1,l}x + c_{1,l}) + \sum_{k=1}^{N} w_{2,k}\varphi_k(v_{2,k}x + c_{2,k}) + b \tag{115}$$

Here, $v_{1,l}$ and $v_{2,k}$ correspond to the elements of V_1^T and V_2^T respectively, and $w_{1,l}$ and $w_{2,k}$ correspond to the elements of W_1^T and W_2^T, respectively. Furthermore, $c_{1,l}$ and $c_{2,k}$ correspond to the biases of the sigmoid functions and sigmoidal jump approximation functions respectively, and b is the bias associated with the last layer, i.e., with the output y.

To test the performance of the new network, it was used to approximate the following two discontinuous functions

$$y = \begin{cases} \sin(x) & x < -1 \\ \sin(x) + 1 & x \geq -1 \end{cases} \tag{116a}$$

$$y = \begin{cases} x & x < 0 \\ 0.5x + 1 & x \geq 0 \end{cases} \tag{116b}$$

The results were compared to approximating the same functions with a regular neural network without jump approximation functions, as shown in Figures 9 and 10.

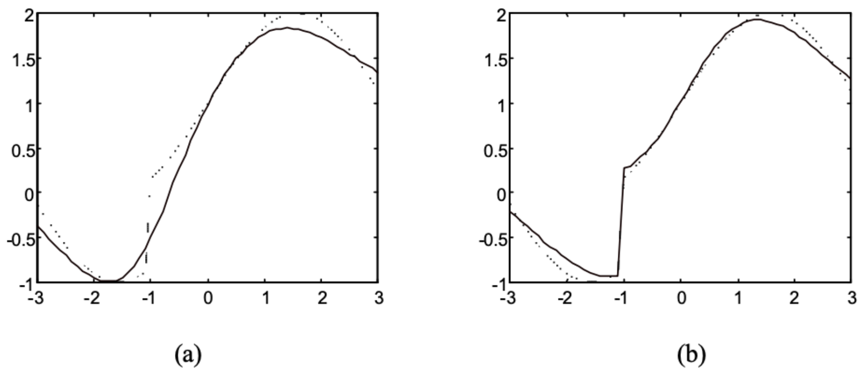

Figure 9: Approximation of function (116a), by a regular neural network (a), and by a neural network with jump activation functions (b); neural network (full line), true function (dashed line), [37].

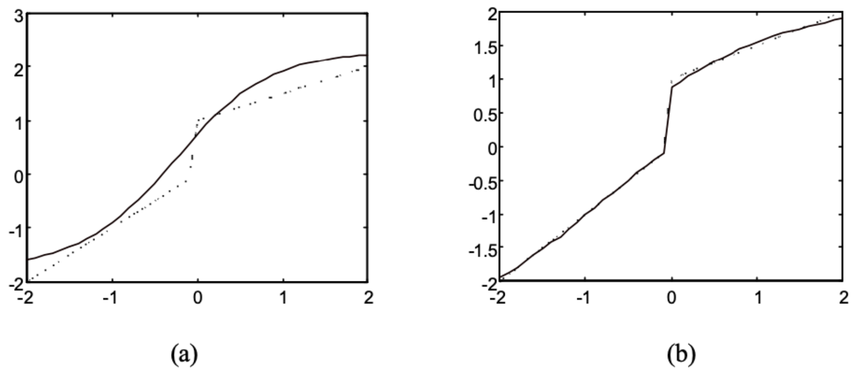

Figure 10: Approximation of function (116b) by a regular neural network (a), and by a neural network with jump activation functions (b); neural network (full line), true function (dashed line) [37].

In both cases, the regular neural network and the jump approximation network had 20 sigmoid nodes, and the jump network had two additional jump nodes. Both types of networks were trained with the same number of iterations. The Figures show clearly that the augmented neural networks are much better suited to the regular neural networks. The jump networks are able to determine the height of the jump, while the regular neural networks appear to average out the function values around the jump.

5 Research Questions

The goal of this research is to find out out whether it's possible to use the numerical approximation methods together with the data-driven approaches that have been covered in the previous chapter and combine the best of both worlds. The numerical methods like the DPG scheme [2] and variations on that scheme [35] are more accurate than data-driven approaches like the Deep Ritz Method [32], the Physics-Informed Neural Networks [4], the Variational Physics Informed Neural Networks [22], the Weak Adversarial Networks [1], and the Deep Operator Networks [3]. Furthermore, these schemes are an improvement over regular finite element methods, since they use optimal test functions to stabilize the approximate solution. The data-driven approaches however, while less accurate, have advantages over the numerical approaches as well. One of the biggest being that they require less computation and can produce an approximate solution almost instantaneously once trained. New network architectures like the deep operator networks, can be trained to learn operators that govern partial differential equations, and can thereafter be used to generate solutions for different variations of a particular problem.

Research Question 1: *Can data-driven approaches be used to generate optimal test functions corresponding to particular trial functions, that improve the stability/accuracy of finite element methods?*

The data-driven approaches will be used to generate optimal test functions for the original DPG scheme [2] and other comparable approaches that go one step further, like the AVS-FE method [35]. A neural network will be trained to approximate the optimal test function corresponding to a particular PDE and a particular trial function.
More specifically, the goal is to first generate the optimal test functions for the 1D pure convection equation and 1D advection-diffusion equation using the deep learning approaches that were covered previously, using trial functions that are non-zero on a single element, and allowing for inter-element discontinuities in the optimal test functions, like was done in the original DPG scheme [2]. Once the optimal test functions corresponding to these problems have successfully been generated, it is possible to test how using them in finite element schemes improves the stability of the approximate solutions, by comparing the results to other FEM schemes. If the 1D cases prove to be a success, the 2D versions of these problems will be the next step.
The next step is to use globally continuous trial functions that are non-zero on multiple elements, as was done in the AVS-FE scheme [35]. Approximating optimal test functions in the case of globally continuous trial functions could be harder, as the support of each test function is identical to that of the corresponding trial function, resulting in a piecewise continuous function that is non-zero on multiple elements. Ideally, this piecewise continuous test function should be approximated by a single neural network, but if this proves to be infeasible each continuous part could be approximated

with a single neural network. The paper on approximating piecewise continuous functions [37] could prove very useful here. Once the optimal test functions for these globally continuous trial functions have successfully been approximated, they will be implemented using finite element methods to see if they improve the stability of the approximate solution.

Research Question 2: *Can data-driven approaches be used to generate optimal test functions, using trial functions as variables, that improve the stability/accuracy of finite element methods?*

This is the most logical next step after the first research question has been proven to be a success. Instead of training a network on a specific choice of trial functions, it will be interesting to see whether it is possible to train a network to generate the optimal test function given using the trial function as a variable. This would make it possible to try out several trial functions and generate multiple optimal test functions without having to retrain the neural network. Again, once the optimal test functions have been generated by the neural network, they will be implemented in finite element schemes to see whether they improve the results.

Research Question 3: *Can neural networks be trained to generate optimal test functions, using problem specific parameters like the diffusion coefficient as variables, to improve the stability/accuracy of finite element methods?*

Once the optimal test functions have been successfully approximated using deep learning architectures and their impact on the stability of the approximate solution has been tested, it is time to go one step further. Consider the case of generating optimal test functions for the advection-diffusion equation. What if instead of training a neural network for a specific value of the diffusion coefficient, it would be possible to train a network that uses the diffusion coefficient as a variable? This would greatly increase the overall applicability of the approach, especially when combined with research question 2. After training a network for the advection-diffusion equation, it would be possible to provide the network with a trial function, the problem's parameters, and the network would output the optimal test function.

Research Question 4: *Can data-driven methods be used to estimate the finite element integrals?*

In the Galerkin method, the weak formulation is used to construct a system of equations that can be used to determine the approximate solution of the original problem. Let's consider this system of equations, $Kd = f$, where K is a matrix whose elements consist of integrals over the trial and test functions, d are the constants used in the approximate solution, and f is a vector whose elements consist of integrals over the test functions and the function f from the original problem. Instead of approximating the optimal test functions, which was the approach in the previous three research questions, it might be possible to approximate the elements of K, i.e., the coefficients of the finite element equations. This could make more sense, as the point-wise values of the optimal test functions are not used in the finite element method. Instead of approximating the optimal test functions and then using sampling to approximate their integrals, it would be possible to approximate the integrals directly.

Research Question 5: *Can data-driven methods be used to estimate the perturbation introduced* *to the finite element integrals, caused by using optimal test functions?*

Suppose that v is the usual test function used and w is the optimal test function. The effect of using an optimal test function can be seen as a perturbation to the finite element integrals. If the integral over $(v-w)u$ is close to zero (with trial function u), then no additional stabilization would be needed. If this integral is not close to zero, it means that using the optimal test function would change the coefficients used in the finite element equations.

6 Preliminary Results

To test the feasibility of some of first research question in this literature report, an initial test was performed. The WANs, VPINNs, and DeepONets were used to try to generate the optimal test functions corresponding to the pure convection problem in 1D, by learning the approximate operator T_n from (21a). While the results from the DNNs were not used with FEM yet, looking at how well they performed on this basic problem can help decide what the next few steps of this research should look like.

The neural network architectures were tested on the single element 1D pure convection problem that was introduced in section 3.1.4, where $\Omega = (0,1)$. The goal is to find the optimal test function corresponding to the trial function u. This is the function v_u that satisfies

$$\int_0^1 v_u'\delta' + v(1)\delta(1) + \int_0^1 u\delta' = 0 \tag{117}$$

In this case v_u is given by

$$v_u(x) = \int_x^1 u(s)ds \tag{118}$$

6.1 Weak Adversarial Networks (WANs)

To use the weak adversarial networks to approximate (118) a loss function of the form (96) needs to be defined. As trial function $u(x) = (1-x)/2$ is chosen. Based on (117) the following equation needs to hold

$$\int_0^1 v_u'\delta' + v_u(1)\delta(1) + \int_0^1 \frac{1-x}{2}\delta' = 0 \tag{119}$$

Using (96), the loss function is defined as follows

$$\min_\theta \max_\eta L(\theta,\eta), \text{ where } L(\theta,\eta) = \left|\int_0^1 v_\theta'\delta_\eta' + v_\theta(1)\delta_\eta(1) + \int_0^1 \frac{1-x}{2}\delta_\eta'\right|^2 / \|\delta_\eta^2\| \tag{120}$$

Here, v_θ and δ_η are neural networks with parameters θ and η respectively.
To test the useability of the WANs, four networks with identical architectures were trained. For each network $10,000$ training iterations were used. In each iteration $1,000$ uniformly sampled points from the interval $[0,1]$ were used to evaluate the integrals in the loss function. The results of training these networks are shown in Figure 11 and in Table 1.

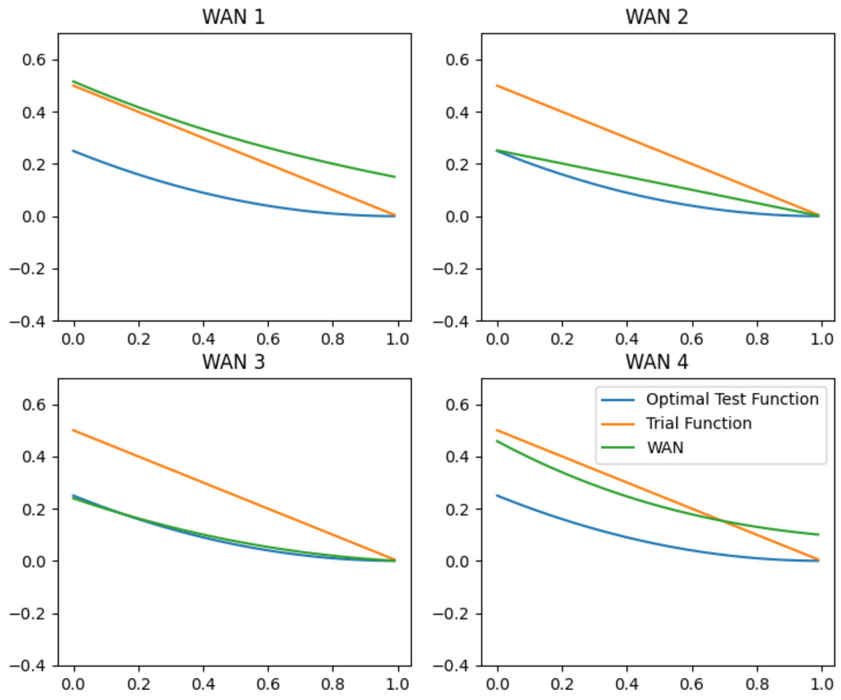

Figure 11: Four WANs after they have been trained to approximate the optimal test functions.

The reason that four networks were trained was to showcase the instability of the convergence of WANs. Only one out of the four networks trained appears to converge to the optimal test function. During the testing of the WANs for this literature report, it turned out that even when a network appeared to converge to the optimal test function (like is the case for networks WAN 3 in Figure 11) it could still suddenly diverge from the correct solution after many iterations.

Network	MSE
WAN 1	0.05204
WAN 2	0.00218
WAN 3	0.00008
WAN 4	0.02338

Table 1: Mean Squared Error (MSE) of the four WANs evaluated at 100 equispaced points on the domain.

6.2 Variational Physics Informed Neural Networks (VPINNs)

To find the solution v_u in (118) with VPINNs equation (117) is used. The VPINN approach differs from that of the WAN in that it does not use a neural network to represent δ. Instead several δ_k are chosen and a neural network representing v in (117) is optimized to minimize the mean of the loss functions corresponding to the choices for δ_k. More specifically, the objective function can be defined as follows

$$\min_{\theta} L(\theta), \text{ where } L(\theta) = \sum_{k=1}^{K} \left| \int_0^1 v'_\theta \delta'_k + v_\theta(1)\delta_k(1) + \int_0^1 u\delta'_k \right|^2 \tag{121}$$

As was the case with the WAN testing, the trial function $u(x) = (1 - x)/2$ was chosen. To compare the VPINNs with the WANs from the previous subsection, four VPINNs with the same architecture were trained. For each model, $5,000$ iterations were used. For each iteration, 256 points were uniformly sampled on the domain to evaluate the integrals in the loss function in (121). The results can are shown in Table 2 and in Figure 12.

Network	MSE
VPINN 1	2.96×10^{-5}
VPINN 2	2.49×10^{-4}
VPINN 3	3.91×10^{-5}
VPINN 4	5.77×10^{-6}

Table 2: Mean Squared Error (MSE) of the four VPINNs evaluated at 100 equispaced points on the domain.

Even though half as many iterations were used compared to the WAN networks, the VPINNs clearly performed much better. Every VPINN has a lower MSE compared to the best performing WAN (WAN 3). Furthermore, every VPINN appears to converge to the optimal test function. The fact that the VPINNs can be trusted to converge is important if they are to be used with approximation methods like FEM. Suppose that would not be certain that the neural network converges to the optimal test function. In that case you would need to compare the network's output with the actual optimal test function (which needs to be computed), every time you'd use it with FEM for a new problem.

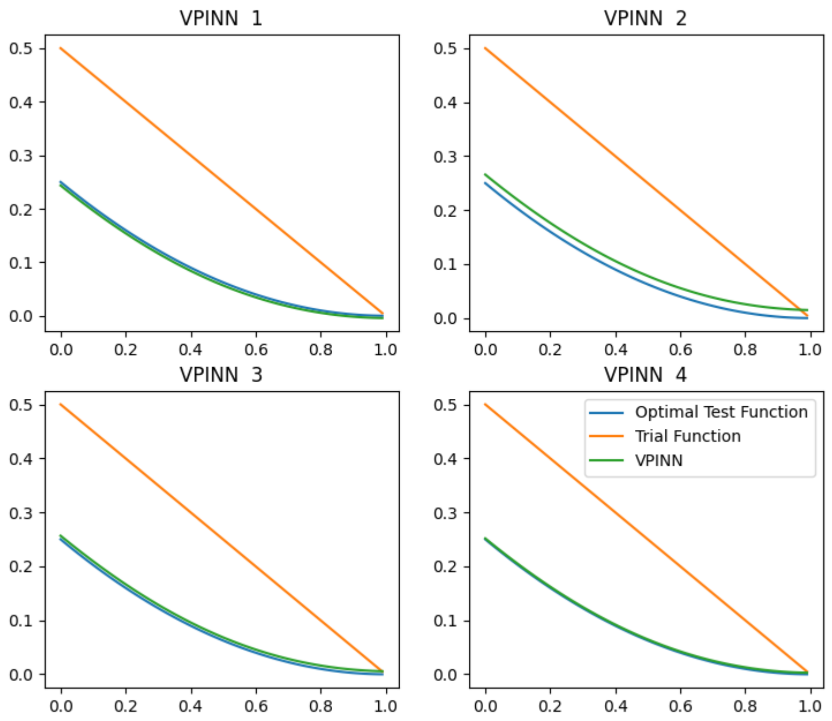

Figure 12: Four VPINNs after they have been trained to approximate the optimal test function.

6.3 Deep Operator Networks (DeepONets)

The DeepONet differs fundamentally from the WAN and the VPINN in that it does not approximate a function, but instead approximates an operator (a mapping from a space of functions into another space of functions). Unlike the WAN and the VPINN, the DeepONet is not trained using equation (117), but instead by comparing its output directly to the solution in (118).
As was mentioned in section 4.6, the DeepONet takes as inputs an function $u = [u(x_1), \dots, u(x_m)]$ evaluated at m fixed points, and a point y at which the output function $G(u)$ is supposed to be evaluated. This means that three things are needed to build a training set. First, a set of trial functions evaluated at the fixed sensors, secondly, a number of points at which the output functions are supposed to be evaluated, and thirdly, the value of the output function $G(u)(y)$.

To generate the input functions one of the in [3] proposed approaches was used: generating orthogonal (Chebyshev) polynomials. Let T_i be Chebyshev polynomials of the first kind. The orthogonal polynomials can be defined as

$$V_{\text{poly}} = \left\{ \sum_{i=0}^{N-1} a_i T_i(x) : |a_i| \leq M \right\} \tag{122}$$

An input function can be generated by uniformly sampling a_i from the interval $[-M, M]$. The authors in [3] do not specify to what value they set M, but after testing the approach on a problem that was mentioned in the original paper with, similar results were achieved with $M = 0.15$. Given the coefficients a_i the input function can be evaluated at the fixed points $x_1, \ldots, x_m$. For this literature report, 100 equispaced points on the domain $[0, 1]$ were used.
Once the input functions are sampled, the points on the domain are uniformly sampled for each input function. For each such point y the value of the output function $G(u)(y)$ needs to be computed. This is the value of the function v_u in (118), which can be calculated using Runge-Kutta. To see how, please note that if v_u is given by (118), then v_u' is given by

$$v_u'(x) = -u(x) \tag{123}$$

Since $-u(x)$ is known, v_u' is known, which can be used to find v_u.

Instead of training four models like was done in the case of the WANs and VPINNs, one large DeepONet was trained. The training set consisted of $200,000$ data points, each of them of the form $([u(x_1), \ldots, u(x_m)], y, G(u)(y))$. During training the model did two full epochs over the data. The resulting model was then tested to approximate v_u with the same trial function as in the previous two subsection, $u(x) = (1 - x)/2$, shown in Figure 13. The DeepONet reached an MSE of 3.0631×10^{-5} when evaluated on the same 100 equispaced points that were used to evaluate the MSEs of the WANs and VPINNs. Comparing the errors of the three network architectures shows that the DeepONet performs better than the WANs and about as good as the average of the four VPINNs, which is quite impressive.
Even though the VPINN needs less data to get to the same accuracy as the DeepONet, it is trained to approximate a function. The DeepONet on the other hand, is trained to approximate an operator. Once it has been trained, it can approximate v_u for any u that it is given. This flexibility that the DeepONet offers could be very useful once integrated with numerical methods like FEM.

Since the DeepONet is trained to approximate the operator from (118), and not to approximate the solution corresponding to a specific trial function u, it was also tested on a several other input functions. To do so, a test set was built in the same way as was done for the training set. The test set consisted of 10 trial functions each evaluated at 100 different points on the domain $[0, 1]$. Using the DeepONet to predict the value of the output function at those points resulted in an MSE of 6.1780×10^{-5}.

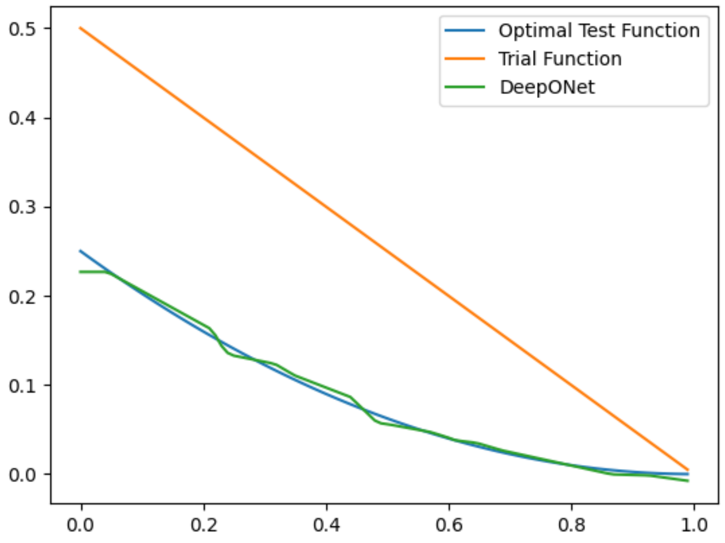

Figure 13: Approximating the optimal test function using the DeepONet architecture.

6.4 Preliminary Conclusions

These preliminary results show that neural networks can be used to generate optimal test functions corresponding to the 1D pure convection problem. While the problem that this idea was tested on is very simple, and while the effect of using these optimal test functions in finite element methods still has to be examined, these results do offer perspective and some guidance on how to proceed with the rest of this thesis. As it turned out, WANs are probably not the best choice for this use case, especially when the goal is to create a general approach that works without adjusting the network configuration to a specific problem too much. Furthermore, it became evident that VPINNs are more accurate than DeepONets, but that the latter offer more flexibility.
Moving on to harder problems like the 2D pure convection case and the advection-diffusion case, it makes sense to continue with VPINNs. The fact that this architecture has a lower MSE on the 1D convection problem probably means that it will be easier to fit to a problem in higher dimensions. Once the first research question has been completed, it makes sense to move over to the DeepONets to see whether good results can be achieved while taking in the trial function as a variable.

References

[1] Yaohua Zang, Gang Bao, Xiaojing Ye, Haomin Zhou. *Weak adversarial networks for high-dimensional partial differential equations*. Journal of Computational Physics, 2020.

[2] Leszek Demkowicz and Jay Gopalakrishnan. *A Class of Discontinuous Petrov-Galerkin Methods. II. Optimal Test Functions*. Portland State University, PDXScholar, 2010.

[3] Lu Lu, Pengzhan Jin, and George Em Karniadakis. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators.* Division of Applied Mathematics, Brown University, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 2020.

[4] M. Raissi, P. Perdikaris, 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, 2018.

[5] M. Raissi, P. Perdikaris, G.E. Karniadakis. *Inferring solutions of differential equations using noisy multi-fidelity data.* Journal of Computational Physics, 2017.

[6] M.Raissi, P. Perdikaris, G.E. Karniadakis. *Machine learning of linear differential equations using Gaussian processes.* Journal of Computational Physics, 2017.

[7] H. Owhadi. *Bayesian numerical homogenization.* Multiscale Model. Simul. 13, 2015.

[8] Ehsan Kharazmi, Zhongqiang Zhang, George EM Karniadakis. *VPINNs: Variational Physics-Informed Neural Networks For Solving Partial Differential Equations.*

[9] J.-X. Wang, J. Wu, J. Ling, G. Iaccarino, H. Xiao, *A comprehensive physics-informed machine learning framework for predictive turbulence modeling,* 2017, arXiv:1701.07102.

[10] Y. Zhu, N. Zabaras, *Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification,* 2018, arXiv:1801. 06879.

[11] T. Hagge, P. Stinis, E. Yeung, A.M. Tartakovsky, *Solving differential equations with unknown constitutive relations as recurrent neural networks*, 2017, arXiv:1710.02242.

[12] R. Tripathy, I. Bilionis, *Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification*, 2018, arXiv:1802. 00850.

[13] P.R. Vlachas, W. Byeon, Z.Y. Wan, T.P. Sapsis, P. Koumoutsakos, *Data-driven forecasting of high-dimensional chaotic systems with long-short term memory networks*, 2018, arXiv:1802.07486.

[14] E.J. Parish, K. Duraisamy, *A paradigm for data-driven predictive modeling using field inversion and machine learning*, J. Comput. Phys. 305 (2016) 758–774.

[15] K. Duraisamy, Z.J. Zhang, A.P. Singh, *New approaches in turbulence and transition modeling using data-driven techniques*, in: 53rd AIAA Aerospace Sciences Meeting, 2018, p. 1284.

[16] J. Ling, A. Kurzawski, J. Templeton, *Reynolds averaged turbulence modelling using deep neural networks with embedded invariance*, J. Fluid Mech. 807 (2016) 155–166.

[17] Z.J. Zhang, K. Duraisamy, *Machine learning methods for data-driven turbulence modeling*, in: 22nd AIAA Computational Fluid Dynamics Conference, 2015, p. 2460.

[18] M. Milano, P. Koumoutsakos, *Neural network modeling for near wall turbulent flow*, J. Comput. Phys. 182 (2002) 1–26.

[19] P. Perdikaris, D. Venturi, G.E. Karniadakis, *Multifidelity information fusion algorithms for high-dimensional systems and massive data sets*, SIAM J. Sci. Comput. 38 (2016) B521–B538.

[20] R. Rico-Martinez, J. Anderson, I. Kevrekidis, *Continuous-time nonlinear signal processing: a neural network based approach for gray box identification*, in: Neural Networks for Signal Processing IV. Proceedings of the 1994 IEEE Workshop, IEEE, 1994, pp. 596–605.

[21] J. Ling, J. Templeton, *Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty*, Phys. Fluids 27 (2015) 085103.

[22] Ehsan Kharazmi, Zhongqiang Zhang, George E.M. Karniadakis. *VPINNs: Variational Physics-Informed Neural Networks For Solving Partial Differential Equations.* 2019, arXiv:1912.00873.

[23] G. Cybenko. *Approximation by superpositions of a sigmoidal function.* Mathematics of Control, Signals and Systems, 2(4):303–314, 1989.

[24] K. Hornik, M. Stinchcombe, and H. White. *Multilayer feedforward networks are universal approximators.* Neural Networks, 2(5):359–366, 1989.

[25] T. Chen and H. Chen. *Approximations of continuous functionals by neural networks with application to dynamic systems.* IEEE Transactions on Neural Networks, 4(6):910–918, 1993.

[26] H. N. Mhaskar and N. Hahm. *Neural networks for functional approximation and system identification.* Neural Computation, 9(1):143–159, 1997.

[27] F. Rossi and B. Conan-Guez. *Functional multi-layer perceptron: A non-linear tool for functional data analysis.* Neural Networks, 18(1):45–60, 2005.

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

[29] T.Chen and H.Chen. *Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks.* IEEE Transactions on Neural Networks, 6(4):904–910, 1995.

[30] L. Demkowicz and J. Gopalakrishnan, *A class of discontinuous Petrov-Galerkin methods. part I: The transport equation*, Submitted, (2009). Available online as ICES Report 09-12.

[31] Babuška and A. K. Aziz, *Survey lectures on the mathematical foundations of the finite element method, in The mathematical foundations of the finite element method with applications to partial differential equations* (Proc. Sympos., Univ. Maryland, Baltimore, Md., 1972), Academic Press, New York, 1972, pp. 1–359. With the collaboration of G. Fix and R. B. Kellogg.

[32] W. E, B. Yu, *The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat. 6 (1) (2018) 1–12.

[33] Zhiqiang Cai, Jingshuang Chen, Min Liu, Xinyu Liu, *Deep Least-Squares Methods: An Unsupervised Learning-Based Numerical Method for Solving Elliptic PDEs*, 2019, arXiv:1911.02109.

[34] Z. Cai, R. Lazarov, T. A. Manteuffel, and S. F. McCormick. *First-order system least squares for second-order partial differential equations: Part i.* SIAM Journal on Numerical Analysis, 31(6):1785–1799, 1994.

[35] Victor M. Calo, Albert Romkes, Erik Valseth. *Automatic Variationally Stable Analysis for FE Computations: An Introduction.* 2019, arXiv:1808.01888v3.

[36] Bochev, P.B., Gunzburger, M.D.: *Least-Squares Finite Element Methods*, vol. 166. Springer Science & Business Media (2009)

[37] Rastko R. Šelmic, Frank L. Lewis. *Neural Network Approximation of Piecewise Continuous Functions: Application to Friction Compensation.* IEEE Transaction on Neural Networks, 1999.

[38] Alexander N. Brooks, Thomas J.R. 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, 1982.