DeepONet Generated Optimal Test Functions for the Finite Element Method

by

Titus Ex

to obtain the degree of Master of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Thursday December 2, 2021 at 11:00 AM.

Student number:	4672631	
Project duration:	September 1, 2020 – December 2, 2021	
Thesis committee:	Assistant Prof. dr. D. Toshniwal	TU Delft, supervisor
	Prof. dr. ir. C. Vuik	TU Delft
	Prof. dr. ir. A.W. Heemink	TU Delft

TU Delft

Preface

Over the past year, I have investigated whether machine learning can be used to improve the finite element method. Specifically, I have tried to find out whether Deep Operator Networks can be used to approximate optimal test functions that help stabilise the finite element method for advection dominated problems. Throughout my research I found that the finite element method solution can be very sensitive to the small approximation errors introduced by the networks. Although this has made implementing the networks difficult, I feel the results look promising as I have found at least one case where the DeepONets improved the finite element solution and were able to generate optimal test functions while using problem specific parameters like the diffusion coefficient as input variables.

Coming from a BSc in Econometrics, I have learned a great deal while working on this project. It has improved my understanding of numerical analysis, but it also taught me how to better navigate through a field with which I am relatively unfamiliar. I would like to thank all the members of my thesis committee for being involved in this project. Both prof. Heemink and Prof. Vuik have been very understanding when deadlines had to be postponed. Furthermore, Prof. Vuik has helped me by providing me with insightful remarks and questions during both my literature presentation and my green light meeting. Lastly, I would like to express special gratitude to my supervisor, Dr. Deepesh Toshniwal. Dr. Toshniwal has been extremely supportive and positive during the entire length of this thesis. He has spent a lot of time helping me understand and solve the problems that I faced and was always willing to explain even the most basic concepts. I have thoroughly enjoyed working with him and would not have been able to produce this report without him.

Abstract

The finite element method (FEM) is a numerical method that is used to approximate the solutions to partial differential equations when solutions in the classical sense do not exist or are very hard to find. The method is used to solve problems that are relevant for industries like the automotive industry, the petroleum industry, and the aviation industry. The finite element method uses a discretisation of the problem domain into sub domains and uses a variational form involving trial and test functions to arrive at a system of equations. For problems that are advection dominated (for problems with a large Péclet number), the FEM solutions start to oscillate and produce inaccurate results near boundary layers. In the past finite element schemes have been developed that use so called optimal test functions to reduce the oscillations of the solution and increase the accuracy of the method.

In this thesis an attempt was made to use Deep Operator Networks (DeepONets) to generate optimal test functions for the steady state advection-diffusion equation in 1D and 2D to improve the stability/accuracy of the finite element method. An advantage of using neural networks is that once trained they can take in problem specific parameters like the diffusion coefficient and produce optimal test functions for a wide range of problems almost instantaneously. It was found that the applicability of the DeepONets in this context varies and depends on the weak formulation for which the DeepONet generated optimal test functions are implemented, as the finite element method solution can be very sensitive to small perturbations in the optimal test functions. The approach does look promising as a DeepONet was able to improve the finite element method significantly in a 2D setting by generating optimal test functions, while using the problem specific diffusion coefficient as a variable.

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 problems like (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. Numerical methods like 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 help tackle solutions to non-linear advection-diffusion equations like the Navier-Stokes equations, which are much more widely applicable.

The structure of this thesis 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, a brief overview of how others have used optimal test functions to deal with this problem will be included. Once the concept of optimal test functions has been introduced, a summary of data-driven methods used to solve PDEs will be presented. After examining these different methods and naming their advantages and disadvantages, research questions will be formulated. Then, the research questions will be tested by using data-driven methods to generate and implement the optimal test functions into the finite element method for the steady state version of the advection-diffusion equation in 1D and 2D. The last section of this thesis includes the conclusions and a discussion of potential next steps.

1.1 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, an example from [39] will be used. Consider the following 1D steady-state version of the advection-diffusion equation from (2)

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

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

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

where h is the distance between grid points. 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 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

$$r = \frac{1 + \frac{vh}{2\epsilon}}{1 - \frac{vh}{2\epsilon}} \tag{7}$$

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, $vh/2\epsilon$, 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 is plotted and is compared to the exact solution in Figure 1.

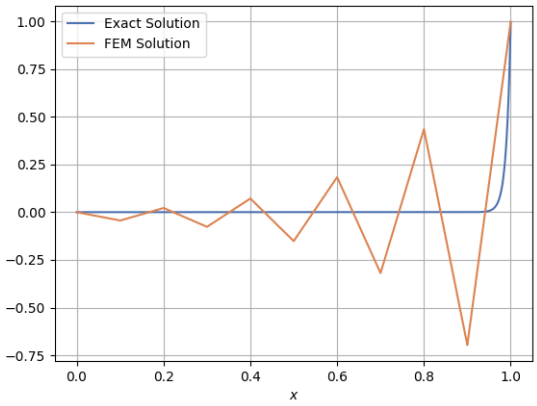


Figure 1: Galerkin FEM approximate solution vs. exact solution of (3), given $\epsilon = 0.01$ and $v = 1$.

This plot shows that the approximate solution is unstable and exhibits wild oscillations that do not occur in the exact solution. These oscillations get progressively worse as the Péclet number gets larger, and render the approximate solution useless.

2 Building Stable Numerical Schemes

To deal with the instability arising from the central-differences in the Galerkin method, several approaches have been used, of which a brief summary can be found in the introduction of [38]. 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 $vh/2\epsilon > 1$. This grid Péclet number gets smaller whenever h, 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). The 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 or 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 method 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.

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

2.1.1 Introduction

In [2] a method is laid out for building discontinuous Petrov-Galerkin (DPG) methods that can 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.
As the name would suggest, 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.

2.1.2 Optimal Test Functions

In [2], L. Demkowicz and J. Gopalakrishnan define the concept of optimal test functions using 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, with norms $\|\cdot\|_U$ and $\|\cdot\|_V$. Furthermore the right hand side l of (8) is a continuous linear form defined on the test space V, and b 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 1 *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

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

Using (14) the following proposition can be derived

Proposition 1.1. *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, what follows is evident from Hilbert space theory

Proposition 1.2. *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*

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

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 1.2. *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.*

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

Since the energy norm is an equivalent norm on U, this means that using optimal test functions leads to the best approximation error in the norm that is endowed on U. Using optimal test functions in a Petrov-Galerkin scheme therefore means that a method can be built in which the oscillations that arise due to a high peclet number are minimized.

2.1.3 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 to lay out the following method.

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 that has good approximation properties. **Theorem 1.2** shows that the error in the energy norm will be equal to the smallest possible error based on the trial functions available, therefore choosing a trial space that can approximate the solution well is key. 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 on an element-wise basis, i.e., $T_n : U_n \to \tilde{V}_n$ 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 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 $V_n = \text{span}\{t_j\}$ 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

$$\begin{aligned} b(e_j, t_i) &= (T_n e_j, t_i)_V && \text{by (21a)} \\ &= (T_n e_j, T_n e_i)_V && \text{as } t_i = T_n e_i \\ &= (T_n e_j, T_n e_i)_V \\ &= b(e_i, t_j) \end{aligned}$$

   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.

2.1.4 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

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

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

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

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

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

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.

2.1.4.1 1D Spectral Discretisation To showcase the next three steps of the scheme that was previously outlined, 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 $\Omega_0 = (x_1, x_2)$. The space H_β, which coincides with $H^1(\Omega_0)$, 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 $L^2(\Omega_0)$ 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

$$U_p = \mathcal{P}_p(\Omega_0) \times \mathbb{R} \tag{31}$$

which means that a function (u_p, q) in U_p is a combination of some $u \in \mathcal{P}_p(\Omega_0)$ 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 $u \in \mathcal{P}_p(\Omega_0)$ and one corresponding to the flux q.

The optimal test function v_u to the interior trial function $u \in \mathcal{P}_p(\Omega_0)$ is the function in $H^1(\Omega_0)$ such that the following equation holds

$$\int_{x_1}^{x_2} v_u'\delta_u' + v_u(x_2)\delta(x_2) = -\int_{x_1}^{x_2} u\delta_v', \quad \forall \delta_v \in H^1(\Omega_0) \tag{32}$$

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

$$\int_{x_1}^{x_2} v_q'\delta_v' + v_q(x_2)\delta_v(x_2) = \delta_v(x_2), \quad \forall \delta_v \in H^1(\Omega_0) \tag{34}$$

which means that

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

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

$$V_p = \text{span}\{v_u, v_q : u \in \mathcal{P}_p(\Omega_0), q \in \mathbb{R}\} \tag{36}$$

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.

2.1.4.2 A multi-element 1D discretisation The authors in [2] also provide a derivation of the multi-element version of section 2.1.4.1. 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 2.1.4.1, 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 moved to the right hand side and 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 2.1.4.1 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

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

2.2 Automatic Variationally Stable Analysis for FE Computations: An Introduction

In [35] an automatic variationally stable analysis for finite element computations of convection diffusion equations for non-constant and highly oscillatory coefficients is introduced, called the AVS-FE method. The idea of least squares finite element methods is used, introduced in [36], where second order PDEs are transformed into 1D systems by introducing the fluxes as auxiliary 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. By choosing element discontinuous test spaces, the element-wise computation of optimal test functions becomes possible. This is essentially the DPG method from [2] that was summarised previously. 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.

2.2.1 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. The corresponding Lipschitz boundary $\partial\Omega$, consists of two parts: Γ_D and Γ_N, with $\Gamma_D \cap \Gamma_N = \emptyset$ 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 conditions 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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.

2.2.2 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 in (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.

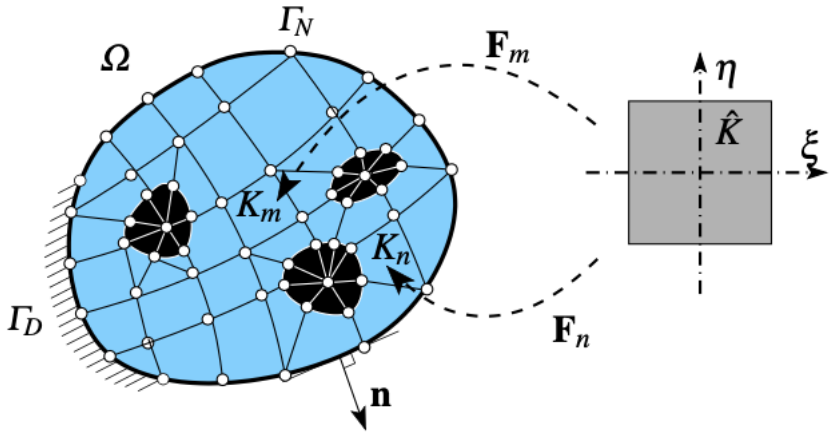


Figure 2: Discretisation of Ω, [35]

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

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

Here p_m is the degree of the polynomials used in the approximation on element K_m. The approximate solutions $(u^h, \mathbf{q}^h)$ 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

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

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

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

The optimal test functions can now 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)

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

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

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

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

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

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 $(u,\boldsymbol{q})$

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

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.

3 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 a lot of data is not readily available, 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 using custom loss functions that employ prior information about a PDE to train deep neural networks [4], while other methods use the weak formulation to rewrite the original equation into a min-max problem ([22], [1]).
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.

3.1 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 focus on solving the following variational problem

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

with

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

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. Essentially, the Deep Ritz method consists of three main ideas. The fact the trial functions are being approximated using a deep neural network, that numerical quadrature rules are used to approximate the functional, and that an algorithm is employed to solve the optimization problem.

3.1.1 Approximating the Trial Functions

As was mentioned briefly, approximating the trial functions is done using a deep neural network. 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 so-called blocks. Each block is made up out of two linear transformations, two linear activation functions, and a residual connection (residual meaning that the input of layers is added back to outputs of layers further down the network). The authors show that 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.

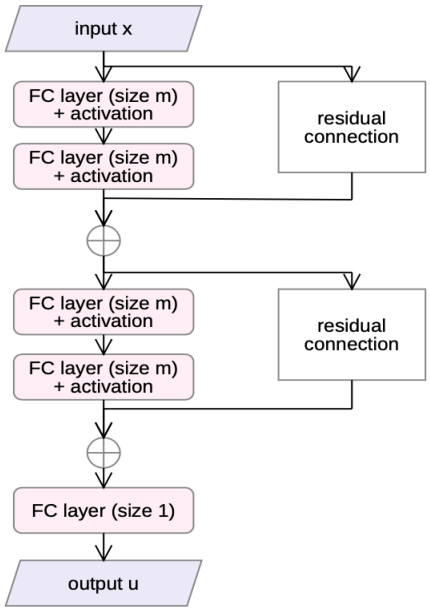


Figure 3: Neural network composed of two blocks and a linear output layer, [32]

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

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

By substituting this expression into the functional in (66), the latter can be rewritten to

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

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

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

3.1.2 Mini-Batch Gradient Descent and Discretisation

Now that approximation of the trial function 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 is 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

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

Note that in the context of the Deep Ritz method, the loss function is essentially $g(x;\theta)$, and the datapoints that are used to evaluate $g(x;\theta)$ are simply points on the domain. 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. The neural network that is trained during this process can be used to approximate the solution $u(x)$ at any point in the domain.

3.2 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 a matrix of functions in $L^2(\Omega)$ that is symmetric and has dimension $n \times n$, X is a differential operator that is linear and is of order of at most one, and $\mathbf{n}$ is the outward unit vector normal to the Γ_N. 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 $\mathbf{q} = -A\nabla u$:

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

with

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

Now, the authors let

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

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

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

Furthermore, they state that 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

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

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

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

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)

$$\mathcal{G}(\boldsymbol{\tau}, v; \boldsymbol{f}) = \|\nabla \cdot \boldsymbol{\tau} + Xv - f\|_{0,\Omega}^2 + \|A^{-1/2}\boldsymbol{\tau} + A^{1/2}\nabla v\|_{0,\Omega}^2 \tag{87}$$

$$+\alpha_D \|v - g_D\|_{1/2,\Gamma_D}^2 + \alpha_N \|\boldsymbol{n} \cdot \boldsymbol{\tau} - g_N\|_{-1/2,\Gamma_N}^2 \tag{88}$$

for all $(\boldsymbol{\tau}, v) \coloneqq H(\mathrm{div}; \Omega) \times H^1(\Omega)$. 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

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

It is possible to prove the functional $\mathcal{G}(\boldsymbol{\tau}, v; \mathbf{0})$ 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 and to train the network values for x can be sampled randomly from the domain.

3.3 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.

3.3.1 Training the Weak Adversarial Networks

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

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

Here, $a_{ij}, b_i, c : \Omega \to \mathbb{R}$ for $i, j \in [d] \coloneqq \{1, \dots, d\}$, $f : \Omega \to \mathbb{R}$ and $g : \partial\Omega \to \mathbb{R}$ are given, and $(\partial u / \partial \overrightarrow{n})(x)$ denotes the normal derivative of u at the boundary point $x \in \partial\Omega$. The paper also discusses solving PDEs that introduce time as another variable, 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 PDE in (90) 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

Find $u \in H^1(\Omega; \mathbb{R})$, such that

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

Here $H_0^1(\Omega; \mathbb{R})$ denotes the Sobolev space, and 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 $\mathcal{A}[u](\phi) \coloneqq \langle \mathcal{A}[u], \phi \rangle$, defined in (91). In that case the operator norm of $\mathcal{A}[u]$ that is derived from the L^2 is defined by

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

Here, $\|\phi\|_2 = \left(\int_\Omega |\phi(x)|^2 \, dx \right)^{1/2}$. 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 problems

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

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

Theorem. *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 $\|\mathcal{A}[u^*]\|_{op} = 0$.*

Using the result in (93) it is possible to use what the authors of [1] call an "adversarial approach" to find the weak solution corresponding to (91). 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 on Ω, can be reformulated into

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

Lastly, u_θ also needs to satisfy $\mathcal{B}[u] = 0$ on $\partial\Omega$ and it is not necessarily true that the neural network that is trained to minimize (94) will automatically learn to satisfy the boundary condition too. If $\{x_b^{(j)}\}_{j=1}^{N_b}$ are a set of N_b points on the boundary $\partial\Omega$, then the mean squared error of u_θ for the Dirichlet boundary condition on $\partial\Omega$ is as follows

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

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

$$n_i(x_b^{(j)}) \partial_i u_\theta(x_b^{(j)})$$

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 and is chosen by the user.

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 any 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.

3.4 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 for which the equation should be solved, $\mathcal{N}[\cdot; \lambda]$ is an operator that is nonlinear and is parameterised by λ, and $\Omega \subseteq \mathbb{R}^D$. 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. They explain that this assumption together with equation (98) results in what is a so called physics-informed neural network $f(t, x)$. While this network has the same parameters as the network used to approximate $u(t, x)$ it has different activation functions due to the impact of the differential operator $\mathcal{N}$. The parameters that are used by both $u(t, x)$ 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

$$\text{MSE}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u(t_u^i, x_u^i) - u^i \right|, \tag{100}$$

and

$$\text{MSE}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f(t_f^i, x_f^i) \right|^2 \tag{101}$$

In this formulation, the points $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ represent the initial and boundary points on $u(t, x)$, while the collocation points for $f(t, x)$ are denoted by $\{t_f^i, x_f^i\}_{i=1}^{N_f}$. 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 to solve PDEs [9–21]. Those approaches use machine learning algorithms as what the authors of [4] call *black-box* models. The approach in [4] goes one step further though, by directly implementing the underlying differential operator into the custom loss functions. To see the difference, it makes sense to think about another way in which a neural network could be trained to approximate the solution to equation (97). One could approximate the solutions at a random set of points on the domain using any type of algorithm, and then train a neural network to approximate the solution at those points by using the mean squared error loss function. In doing just that, one might end up with a neural network that can produce very accurate approximations, but the way in which the network learns to approximate the solution is different. Updating the network's weights based on the approximate solution to (97) solely, means that the network only learns about the PDE through its solution. When instead the network is provided with direct information about the PDE as in (101), it can learn to approximate the solution using a much simpler architecture and using less training data.

3.5 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.
Firstly, using integration by parts to reduce the order of the differential operators, reduces the required regularity in the solution space. One of the results of this strategy is that the VPINNs will be less computationally expensive compared to PINNs. Besides being less computationally expensive, the VPINNs use a loss function that can be expressed analytically. This means that numerical analyses can be run. Furthermore, VPINNs use a small number of quadrature points compared to the number of penalising points used in PINNs, which means that the network leads less data to train. Lastly, the authors mention that the integrals in the weak formulation allow for the possible benefit a decomposition of the domain. By 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. In the next section the VPINN network will be introduced using a steady state problem.

3.5.1 Training VPINNs

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

$$\mathcal{L}^q u(\mathbf{x}) = f(\mathbf{x}),\ \mathbf{x} \in \Omega \tag{102a}$$

$$u(\mathbf{x}) = h(\mathbf{x}),\ \mathbf{x} \in \partial\Omega \tag{102b}$$

Here $\Omega \subset \mathbb{R}^d$ with dimensionality d and boundaries $\partial\Omega$. The function $u(\mathbf{x}) : \Omega \to \mathbb{R}$ is used to represent the underlying physics, the forcing term $f(\mathbf{x})$ is some external excitation, and lastly, the authors mention that $\mathcal{L}$ usually contains differential and/or integro-differential operators with the parameters $\mathbf{q}$.

The approximate solution $\tilde{u}(\mathbf{x})$ is given by the weights and the biases of the neural network, u_{NN}. Alternatively, $\tilde{u}(\mathbf{x}) = u_{\text{NN}}(\mathbf{x}; \mathbf{w}, \mathbf{b})$. The equation (102a) is multiplied by a test function $v(\mathbf{x})$ and integrated over the whole domain to obtain the variational form

$$(\mathcal{L}^q u_{\text{NN}}(\mathbf{x}), v(\mathbf{x}))_\Omega = (f(\mathbf{x}), v(\mathbf{x}))_\Omega \tag{103a}$$

$$u(\mathbf{x}) = h(\mathbf{x}),\ \mathbf{x} \in \partial\Omega \tag{103b}$$

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

$$\begin{aligned} \text{Residual}^v &= \mathcal{R} - F - r^b, \\ \mathcal{R} = (\mathcal{L}^q u_{\text{NN}}, v)_\Omega,\ F &= (f, v)_\Omega \end{aligned} \tag{104}$$

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

$$r^b(\mathbf{x}) = u_{\text{NN}}(\mathbf{x}) - h(\mathbf{x}),\ \forall \mathbf{x} \in \partial\Omega \tag{105}$$

Then, the authors define a discrete finite dimensional space V_k using a finite set of admissible test functions

$$V_k = \text{span}\{v_k, k = 1, 2, \ldots, K\} \tag{106}$$

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

$$L^v = L_R^v + L_u \tag{107}$$
$$L_R^v = \frac{1}{K}\sum_{k=1}^{K} |\mathcal{R}_k - F_k|^2, \; L_u = \tau \frac{1}{N_u}\sum_{i=1}^{N_u} \left|r^b(x_{u_i})\right|^2$$

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 the residual. Using these new definitions the problem of solving (103a) and (103b) can be reformulated as

$$\text{find } \tilde{u}(\mathbf{x}) = u_{\text{NN}}(\mathbf{x}; \mathbf{w}^*, \mathbf{b}^*) \text{ such that } \{\mathbf{w}^*, \mathbf{b}^*\} = \arg\min(\mathcal{L}^v(\mathbf{w}, \mathbf{b})) \tag{108}$$

Given the objective function above, and the earlier defined variational loss function (107) the neural network $u_{\text{NN}}(\mathbf{x}; \mathbf{w}, \mathbf{b})$ 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.

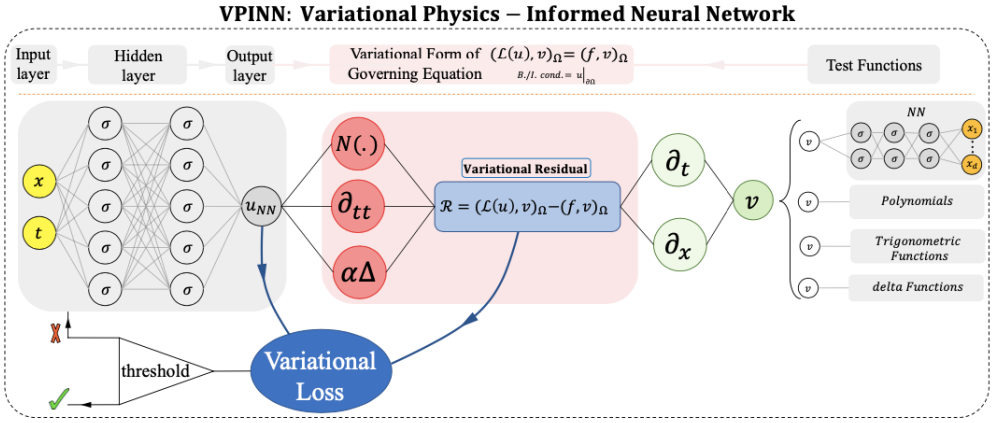


Figure 4: VPINN approach, shown in [22]. The authors note that in the Petrov-Galerkin approach the neural networks represent the trial functions and that the test functions can either be from another neural network (as in the WAN approach) or other functions spaces. The green parts in this Figure represent the test functions whereas the red parts represent the differential operators in the trial function space.

3.6 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). In the papers that have been

covered so far, neural networks were trained to approximate the solutions to a particular PDE. Instead, the authors in [3] lay out a method that uses deep neural networks to approximate the operator G that takes in the input function u. The value that the neural networks predict is $G(u)(y)$, the function value of the function that is $G(u)$. Instead of taking in just a set of points on a domain, the DeepONet takes in u and y. What makes this network so powerful for the application in this thesis (especially compared to other networks), is that it can take in a trial function as an input variable. The DeepONets will be able to generate the optimal test functions corresponding to multiple different trial functions, while with the other network architectures a single neural network per optimal test function would have been needed.

3.7 Training DeepONets

The authors base their approach on a theorem that 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: (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, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{\text{branch}} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{\text{trunk}} \right| < \epsilon \tag{109}$$

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.

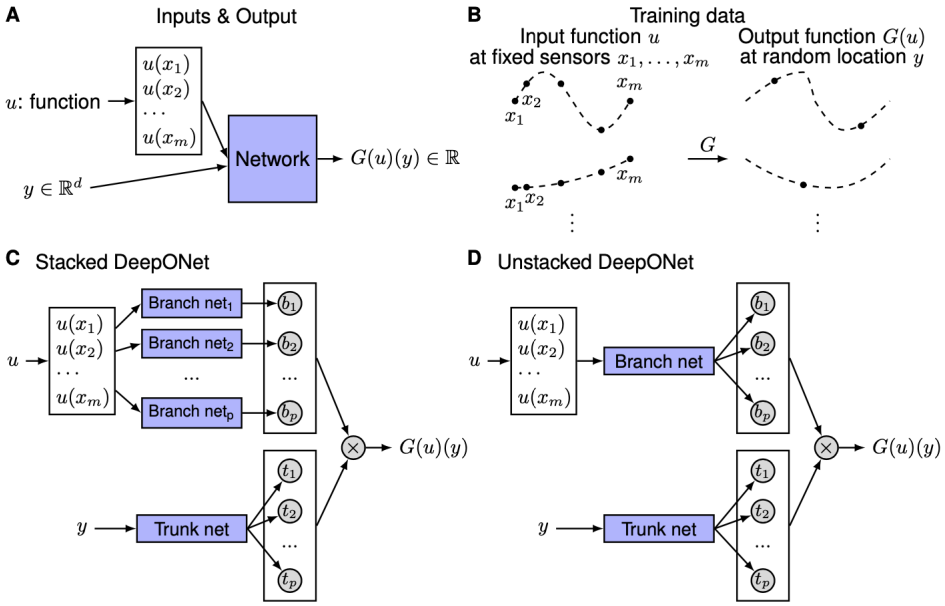


Figure 5: Deep Operator Network Approach, [3]

In **(A)** of Figure 5, the inputs and outputs of the network are shown. The network is fed an array $[u(x_1), \ldots, u(x_m)]$ consisting of the values of an input function u at a finite number of fixed points, called "sensors" and a point on the domain $y \in \mathbb{R}^d$. **(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 $G(u)(y)$. The DeepONet can be trained using any type of gradient descent based algorithm by iterating over the training data.
(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 $\{x_1, \ldots, x_m\}$ for the input functions. The inputs consists of two parts: $[u(x_1), \ldots, u(x_m)]^T$ 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, $[u(x_1), \ldots, u(x_m), y]$, 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 $i = 1, 2, \ldots, m$. 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 $[u(x_1), \ldots, u(x_m)]^T$ 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 $[t_1, \ldots, t_p]^T \in \mathbb{R}^p$. Secondly, the p branch networks each take in $[u(x_1), \ldots, u(x_m)]^T$ as the input, and output a scalar $b_k \in \mathbb{R}$ for $k = 1, 2, \ldots, p$. Therefore, the approximation in equation (109) can be written in terms of the outputs of the branches and the trunk

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k \tag{110}$$

Lu Lu et al. use an activation function in the final layer of the trunk network, i.e., $t_k = \sigma(\cdot)$ for $k = 1, 2, \ldots, p$. If this fact is combined with equation (110), 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.
In [3] it is mentioned 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 $[b_1, \ldots, b_p]^T \in \mathbb{R}^p$.

3.7.1 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

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0, 1] \tag{111}$$

The authors mention that solving this problem can essentially be seen as learning the following operator

$$G : u(x) \to s(x) = \int_0^x u(\tau) d\tau \tag{112}$$

To compare with, a regular FNN is trained to learn the operator in (112). 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.

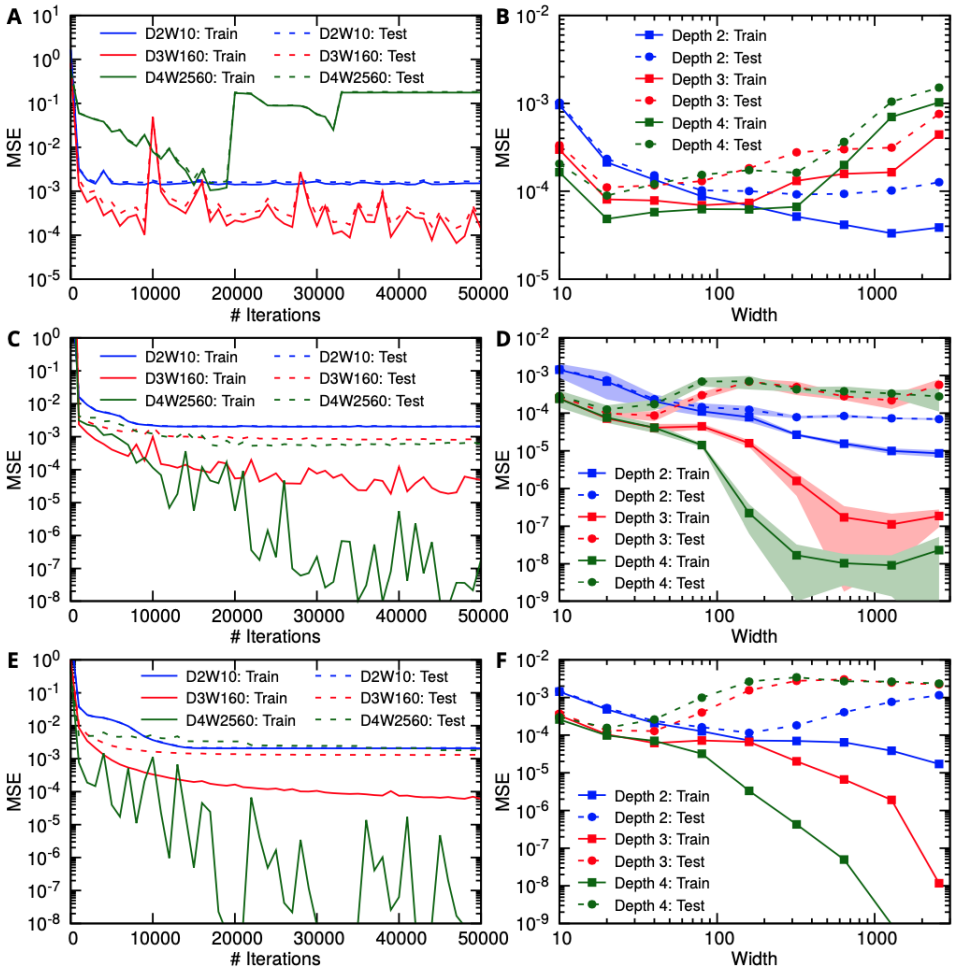


Figure 6: Errors of FNNs trained to learn the antiderivative operator, [3].

To compare with the regular neural network, a stacked and an unstacked DeepONet (see **C** and **D** of Figure 5) are trained to learn (112). 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 (110), the networks output

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0 \tag{113}$$

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. One interesting point to make here

is that the DeepONets have a much smaller generalisation error compared to the regular FNN, even without a grid search.

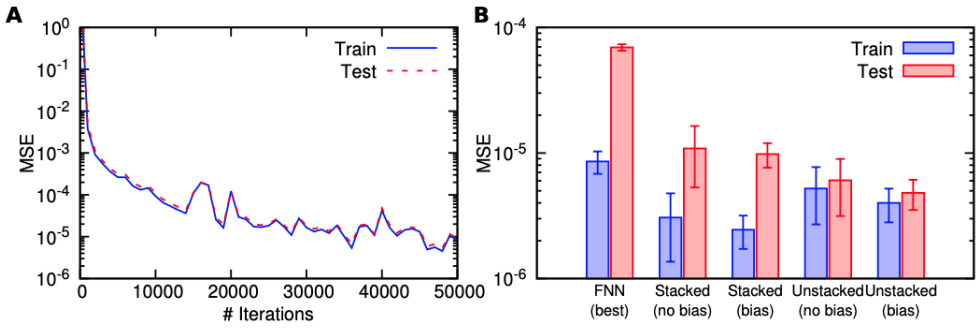


Figure 7: Errors of DeepONets trained to learn the antiderivative operator, [3].

3.8 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 function. Using a neural network architecture that can deal with discontinuities could be very useful for this thesis. It was shown in sections 2.1.4.2 and 2.2.2 that the optimal test functions with multi-element support exhibit jumps between elements. As will be covered in section 4, 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.

3.8.1 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

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{114}$$

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

$$\phi_k(x) = \begin{cases} 0, & \text{for } x < x_J \\ \left(\dfrac{1-e^{-x}}{1+e^{-x}}\right)^k, & \text{for } x \geq x_J \end{cases} \tag{115}$$

and

$$\phi_k(x) = \begin{cases} 0, & \text{for } x < x_J \\ \left(\dfrac{e^x - e^{-x}}{e^x + e^{-x}}\right)^k, & \text{for } x \geq x_J \end{cases} \tag{116}$$

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.

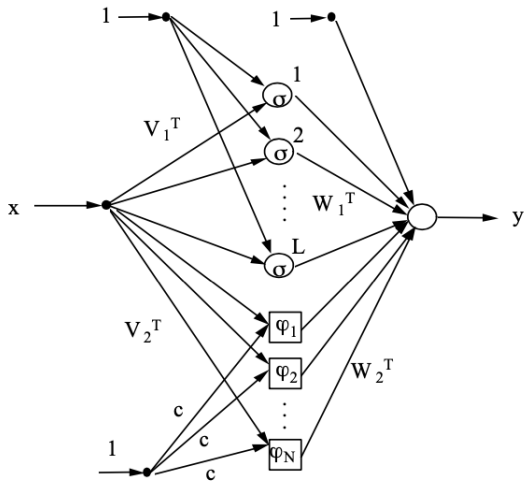


Figure 8: Augmented Neural Network, [37].

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

$$y = \sum_{l=1}^{L} w_{1,l}\, \sigma(v_{1,l}x + c_{1,l}) + \sum_{k=1}^{N} w_{2,k}\varphi_k(v_{2,k}x + c_{2,k}) + b \tag{117}$$

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

$$y = \begin{cases} \sin(x) & x < -1 \\ \sin(x) + 1 & x \geq -1 \end{cases} \tag{118a}$$

$$y = \begin{cases} x & x < 0 \\ 0.5x + 1 & x \geq 0 \end{cases} \tag{118b}$$

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.

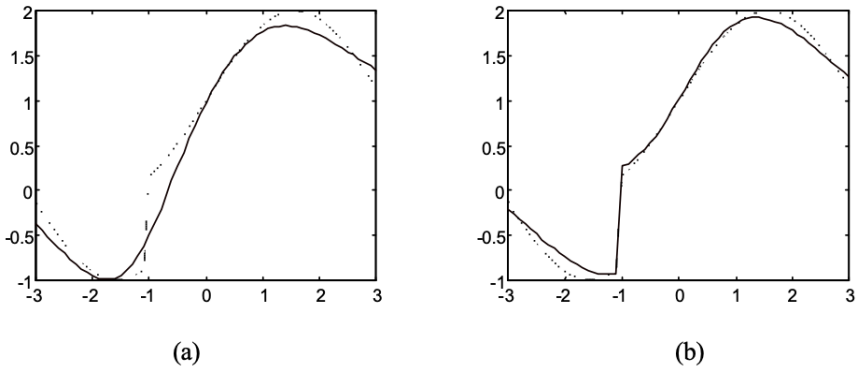


Figure 9: Approximation of function (118a), by a regular neural network (a), and by a neural network with jump activation functions (b); neural network (full line), true function (dashed line), [37].

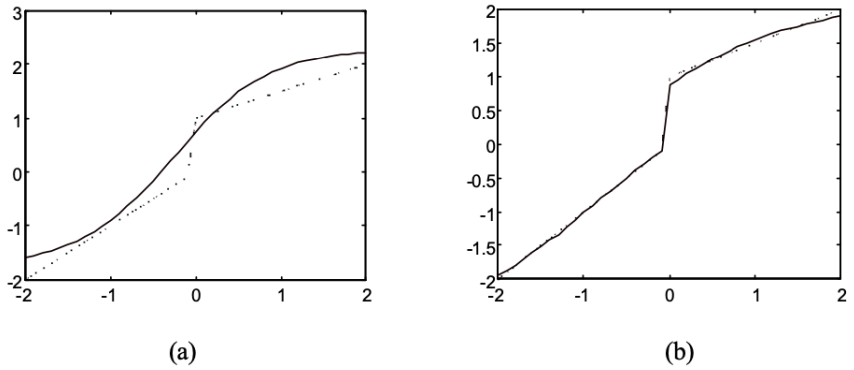


Figure 10: Approximation of function (118b) by a regular neural network (a), and by a neural network with jump activation functions (b); neural network (full line), true function (dashed line) [37].

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.

4 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.

5 Implementing DeepONet Generated Optimal Test Functions Into the Finite Element Method

In this section, DeepONets will be used to approximate and implement optimal test functions into the finite element method, with the goal of improving the stability and the accuracy of the approximate solution. The previous section covered many different neural network architectures that can all be used to solve partial differential equations. In this thesis the only neural network architecture that will be used is the DeepONet as it has a few deciding advantages over the other network architectures.
The most distinctive difference between the DeepONet and the rest of the neural networks that were summarised in the previous section is that the DeepONet approximates operators (a mapping from one space of functions to another space of functions), using input functions as variables. This is extremely useful in the context this thesis, as it means that a single DeepONet can approximate multiple optimal test functions, by passing the trial functions as input variables to the network. In the setting of a 2D weak formulation that does not employ a mixed strategy, there already exist 14 unique optimal test functions corresponding to trial functions that are the tensor product of C^0 piecewise linear, C^0 quadratic, and C^1 quadratic B-splines. Using the VPINN architecture would have meant that 14 different networks would have to be trained. In the case of a mixed weak formulation that uses discontinuous test spaces in the spirit of the DPG method, the number of neural network that would have had to been trained could be much higher.
Another advantage of the DeepONet over some of the other architectures is that it uses supervised learning. This means that it is relatively easy to treat problem specific parameters like the diffusion coefficient as a variable.

5.1 1D Non-mixed Weak Formulation

As was mentioned previously, the advection-diffusion equation will be used to test the implementation of the optimal test functions. More specifically, the following boundary value problem will be considered

$$\begin{cases} -\epsilon \dfrac{d^2u}{dx^2} + c\dfrac{du}{dx} = f, \quad x \in (0,1) \\ u(0) = 0,\ u(1) = 1 \end{cases} \tag{119}$$

where ϵ and c are scalars that denote the diffusion coefficient and advection velocity. In this thesis $c = 1$ will be used for all problems, and only ϵ will be adjusted to experiment with different Péclet numbers. This is a nice problem to test the use of optimal test functions with, as the exact solution is known and the Galerkin method will produce oscillatory behaviour for large Péclet numbers.

5.1.1 Weak Form

Using a discretisation of the domain into open sub-intervals $\mathcal{P} = \sum_{k=1}^{n} \Omega_k$, with $\Omega_k = (x_{k-1}, x_k)$, and $x_0 = 0, x_k = 1$, the first weak form that will be used to find a weak solution to (119) is the following

Find $u \in U$ such that

$$\epsilon \sum_{k=1}^{n} \int_{\Omega_k} u'\, w' - \sum_{k=1}^{n} \int_{\Omega_k} uw' = \sum_{k=1}^{n} \int_{\Omega_k} fv, \quad \forall v \in V$$

Here, the $H^1([0,1])$ space is used for the trial and test spaces U and V, i.e, U and V are defined as follows

$$\begin{aligned} U &= \{u : u \in L^2([0,1]),\, u' \in L^2([0,1])\} \\ V &= \{v : v \in L^2([0,1]),\, v' \in L^2([0,1])\} \end{aligned} \tag{120}$$

By introducing the notation

$$\begin{aligned} b(u,v) &= \epsilon \sum_{k=1}^{n} \int_{\Omega_k} u'w' - \sum_{k=1}^{n} \int_{\Omega_k} uw' \\ l(v) &= \sum_{k=1}^{n} \int_{\Omega_k} fv \end{aligned} \tag{121}$$

the weak formulation can be written in compact form as follows

$$\begin{aligned} &\text{Find } u \in U \text{ such that} \\ &\qquad b(u,v) = l(v), \quad \forall v \in V \end{aligned} \tag{122}$$

That this weak form leads to a well defined finite element scheme can be seen in Figure 11. The instability that has been show earlier can be seen here as well. As the number of elements in the discretisation increases, the instability disappears as the local Péclet number becomes smaller.

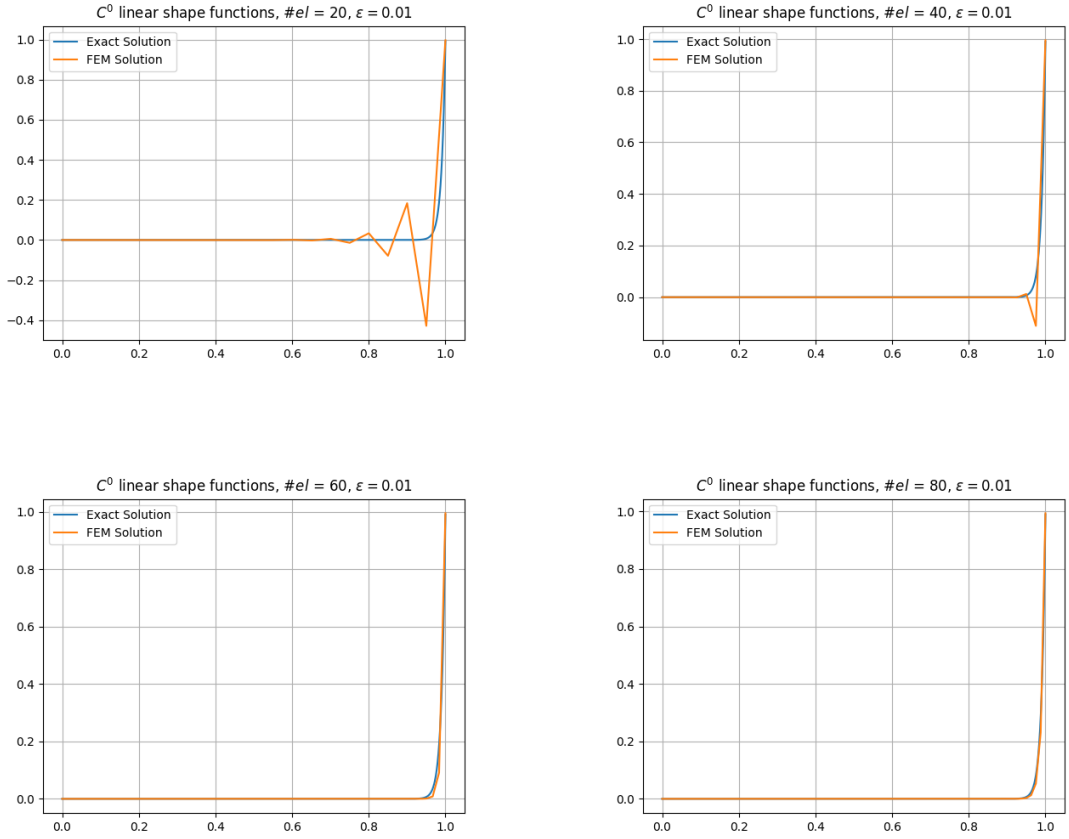


Figure 11: Galerkin method using C^0 linear shape functions corresponding to (120), $\epsilon = 0.01$.

5.1.2 Optimal Test Functions

To find the optimal test function corresponding to the trial functions u in (122), the map from trial space to test space $T : U \to V$ is defined. As was done in [2], Tu in V is defined as the solution of

$$(Tu, v)_V = b(u, v), \quad \forall v \in V \tag{123}$$

where $(\cdot, \cdot)_V$ denotes the inner product of V, and where $b(u, v)$ denotes the bilinear form as defined in (121). To derive the optimal test functions in, the inner product that will be used is induced by the mesh dependent $H^1(\mathcal{P})$ norm which is defined as follows

$$\|u\|^2_{H^1(\mathcal{P})} = \sum_{k=1}^{n} \int_{\Omega_k} (h^2\, u'^2 + u^2) \tag{124}$$

where h is equal to the element length used in the finite element discretisation. The corresponding inner product looks like this

$$(v, w)_{V(\mathcal{P})} = \sum_{k=1}^{n} \int_{\Omega_k} (h^2\, v'\, w' + v\, w) \tag{125}$$

Now, to find the optimal test function v_u corresponding to trial function u, the following problem has be solved

$$\epsilon \sum_{k=1}^{n} \int_{\Omega_k} u' \delta_v' - \sum_{k=1}^{n} \int_{\Omega_k} u \delta_v' = \sum_{k=1}^{n} \int_{\Omega_k} (h^2\, v_u'\, \delta_v' + v_u\, \delta_v), \quad \forall \delta_v \in H^1([0,1]) \tag{126}$$

In all the problems in [2], the optimal test functions could be approximated on a element-wise basis, by choosing a weak formulation that allowed the test functions to be in $L^2([0,1])$. If the space of test functions in (120) would have been $L^2([0,1])$, the local restriction of the optimal test function v_u to element Ω_k in (126), $v_{u,k}$, could have been computed on a local basis as well by solving

$$\epsilon \int_{\Omega_k} u' \delta_v' - \int_{\Omega_k} u \delta_v' = \int_{\Omega_k} (h^2\, v_{u,k}'\, \delta_v' + v_{u,k}\, \delta_v), \quad \forall \delta_v \in L^2([0,1]) \tag{127}$$

However, since the trial space in (120) is an $H^1([0,1])$ space, the optimal test functions have to be in $H^1([0,1])$ as well. Therefore, to find the optimal test function v_{u_i} corresponding to trial function u_i the following boundary value problem has to be solved

$$\begin{cases} \text{Find } v_{u_i}, \text{ such that} \\ \epsilon \displaystyle\int_{S_{u_i}} u_i' \delta_v' - \int_{S_{u_i}} u_i \delta_v' = \int_{S_{u_i}} (h^2\, v_{u_i}'\, \delta_v' + v_{u_i}\, \delta_v), \quad \text{in } S_{u_i} \\ v_{u_i} = 0, \quad \text{on } \partial S_{u_i} \end{cases} \tag{128}$$

where S_{u_i} denotes the union of elements on which u_i has non-zero support. Here, the boundary conditions on v_{u_i} are enforced to ensure that $v_{u_i} \in H^1([0,1])$. In Figure 12 a FEM approximation is shown of the optimal test functions corresponding to (120) and inner product (125), for different values of ϵ and different element sizes, for a piecewise linear trial function. It is interesting to see that the optimal test function show a clear upwind effect (i.e. they place more weight to the upwind part of their support).
In this section the use of optimal test functions will be tested for implementations that use piecewise linear, $C^0(\Omega)$ quadratic and $C^1(\Omega)$ quadratic trial functions, shown in Figure 13.

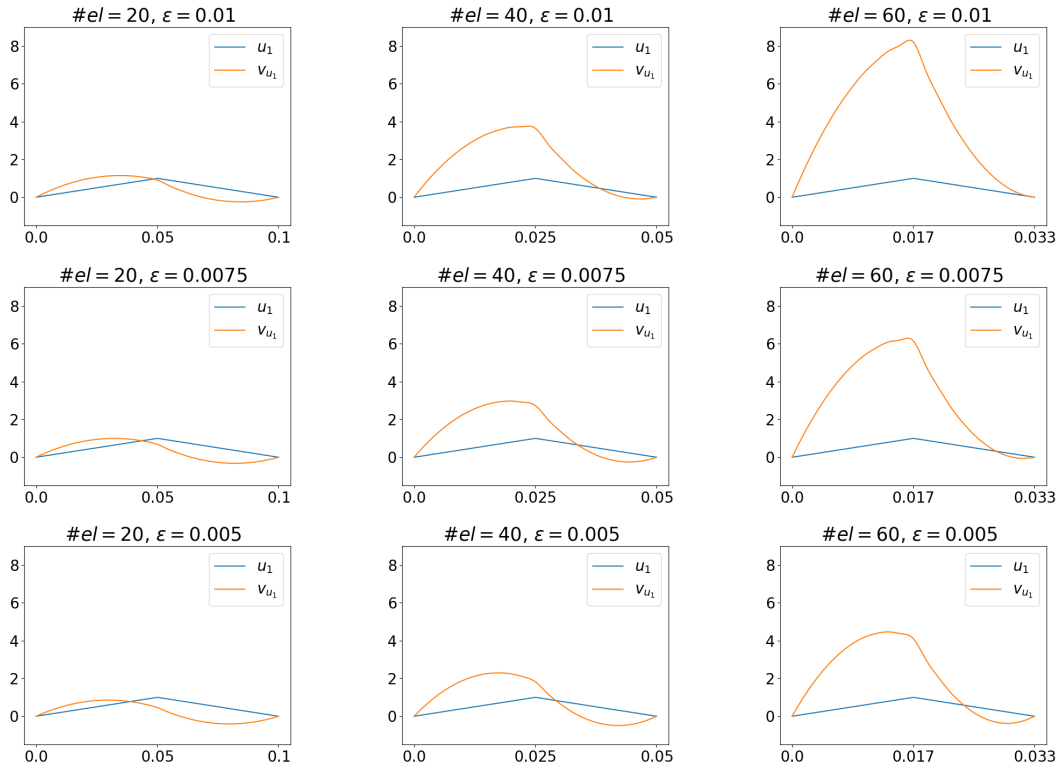
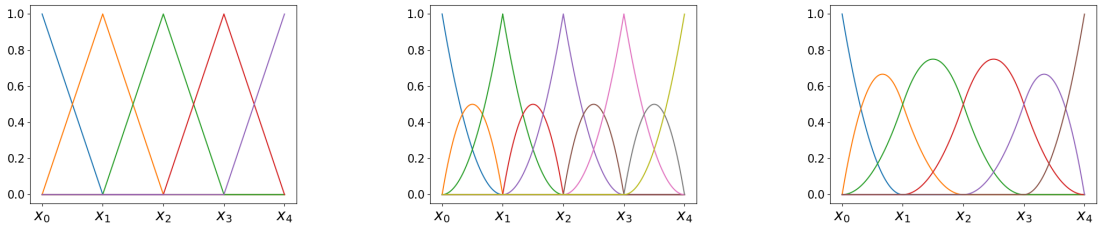


Figure 12: Optimal test function v_{u_1} corresponding to piecewise linear trial function u_1 for different values of ϵ and different discretisations.



(a) C^0 piecewise linear trial functions.

(b) C^0 quadratic trial functions.

(c) C^1 quadratic trial functions.

Figure 13: Different types of trial functions used that will be used throughout this section.

5.1.3 Implementing the Optimal Test Functions

The optimal test functions were implemented into a Petrov-Galerkin scheme based on the following weak that was introduced earlier

$$\text{Find } u \in U \text{ such that}$$
$$\epsilon \sum_{k=1}^{n} \int_{\Omega_k} u'\, w' - \sum_{k=1}^{n} \int_{\Omega_k} uw' = \sum_{k=1}^{n} \int_{\Omega_k} fv, \quad \forall v \in V \tag{129}$$

The results are shown in Figure 14 for implementations that use piecewise linear trial functions corresponding to (129) with $f = 0$. The boundary conditions corresponding to the strong form (119) were enforced strongly. Implementations using optimal test functions lead to a much more accurate solution in these cases when compared to Galerkin implementations that do not use optimal test functions.

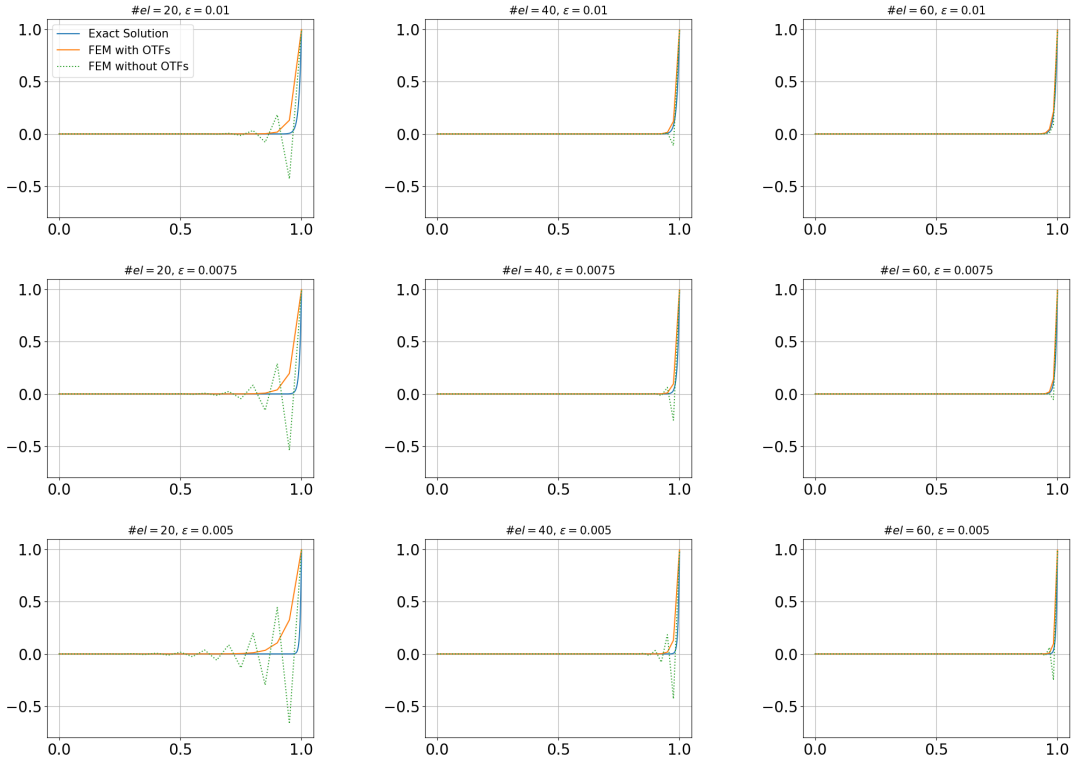
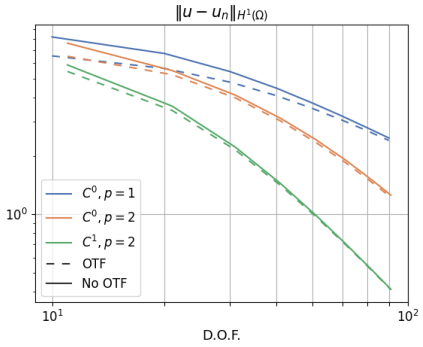
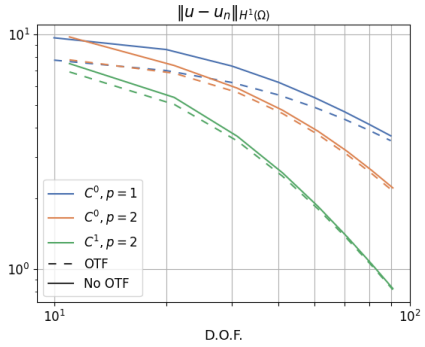


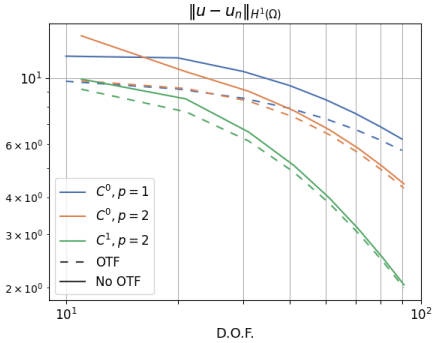
Figure 14: Comparison of Galerkin method and Petrov-Galerkin method with linear trial functions and optimal test functions corresponding to (129) with $f = 0$, for several finite element discretisations and values for ϵ. As ϵ gets bigger the oscillations near the boundary layer get worse. Increasing the number of elements used in the finite element discretisation reduces the osciallations. The implementations that use optimal test functions perform much better than those that do not.



(a) $\epsilon = 0.01$.



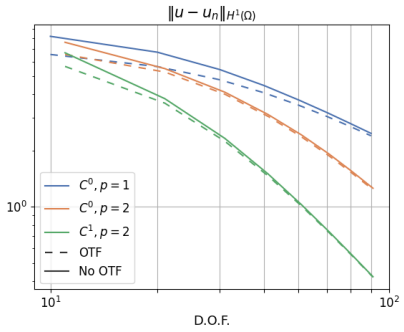
(b) $\epsilon = 0.0075$.



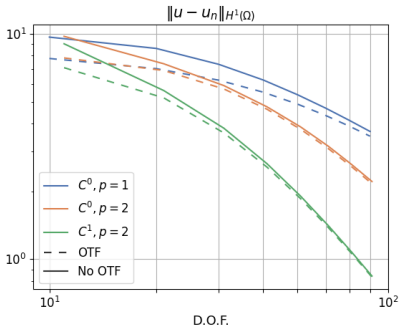
(c) $\epsilon = 0.005$.

Figure 15: Comparison of the convergence of the error $\|u - u_n\|_{H^1(\Omega)}$ of the approximate solution of (129) with $f = 0$, for the different trial functions shown in Figure 13 between implementations that do and do not use optimal test functions.

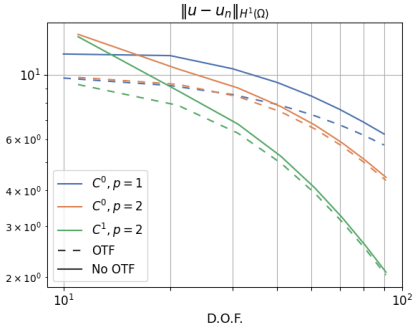
Figure 15 and Figure 16 show a comparison between the $\|u - u_n\|_{H^1(\Omega)}$ error of finite element implementations that use and do not use optimal test functions, for $f = 0$ and $f = 1$ respectively. The solutions that use optimal test functions are clearly an improvement over their counterparts that do not use optimal test functions. One thing that is interesting to see is that the Galerkin methods that do no use optimal test functions appear to catch up with the methods that do use optimal test functions rather quickly. As the discretisation is made finer, the relative difference in the H^1 error between the Galerkin methods and the optimal test functions methods becomes smaller. While the optimal test functions still do better than the regular test functions (as they should), it appears that as the number of elements used in the discretisation increases and the local Péclet number decreases the need for stabilisation is reduced.



(a) $\epsilon = 0.01$.



(b) $\epsilon = 0.0075$.



(c) $\epsilon = 0.005$.

Figure 16: Comparison of the convergence of the error $\|u - u_n\|_{H^1(\Omega)}$ of the approximate solution of (129) with $f = 1$, for the different trial functions shown in Figure 13 between implementations that do and do not use optimal test functions.

5.1.4 Training *DeepONet-1D-1*

The first DeepONet that will be trained to approximate the optimal test functions is called *DeepONet-1D-1*. Specifically, the network is trained to approximate the approximate solutions to the boundary value problem in (128), for a finite element discretisation with 20 elements, and with the diffusion coefficient set to $\epsilon = 0.01$. Note that without parameterising the diffusion coefficient and the interval length of the discretisation, the DeepONet can only generate the optimal test functions for fixed values of those parameters. A DeepONet trained for a discretisation using 80 elements and a diffusion coefficient of $\epsilon = 0.01$ cannot be used to generate the optimal test functions corresponding to a discretisation of 40 elements and a diffusion coefficient of $\epsilon = 0.005$, as the optimal test functions are different (see Figure 12).

The training dataset for *DeepONet-1D-1* is generated by repeatedly approximating the solution $v_{u_i}(x)$ in boundary value problem (128) for trial function u_i at a random point x for all the unique trial functions shown in Figure 13, except for the ones that are non-zero on the boundary. Before the points on the domain x are passed to the network they are scaled to the interval $[0, 1]$. In this

section the boundary conditions on u will be enforced strongly which means that the first and last B-spline of each class of functions will not be used. The solutions are approximated using the finite element method using a discretisation of 200 elements and second order polynomial trial and shape functions. Using this approach each point in the training dataset is a triplet that looks like this

$$(\boldsymbol{u}_i, x, \hat{v}_{u_i}(x)) \tag{130}$$

Here $\boldsymbol{u}_i$ is vector that contains the values of trial function u_i evaluated at a set of fixed sensors, x a random point from the interval $[0, 1]$ that corresponds to a point on the original domain on which the trial function is non-zero, and $\hat{v}_{u_i}(x)$ is the approximate solution of (128) corresponding to trial function u_i at random point x. *DeepONet-1D-1* uses the ReLU activation function, has two layers in the branch network consisting of 100 neurons each, and three layers in the trunk network consisting of 100 neurons each.
To train the neural network the mean squared error (MSE) loss function is used. The MSE function looks like this:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{131}$$

Here y_i is the correct solution that the model needs to predict, $\hat{y}_i$ is the value that the model predicts, and N is the size of the batch for which the MSE is evaluated. Figure 17 shows the result of training *DeepONet-1D-1* for 100 epochs.

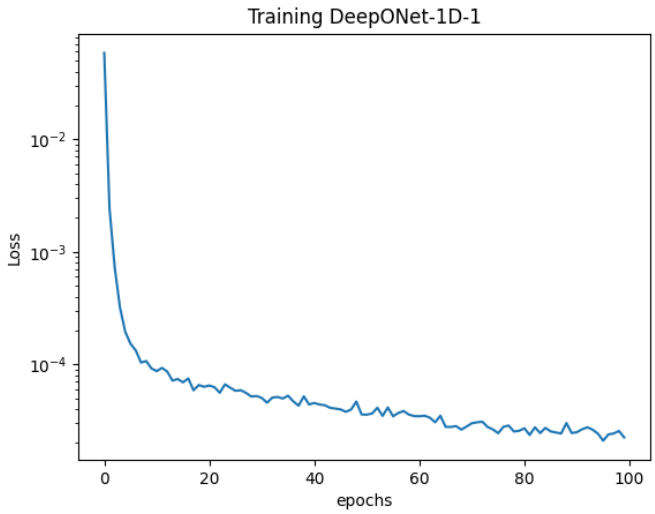


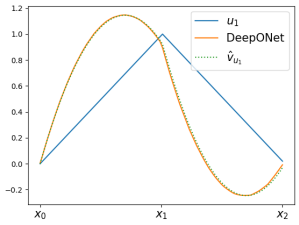
Figure 17: Convergence of MSE during training of *DeepONet-1D-1*.

The the MSE loss during training of *DeepONet-1D-1* is shown in Figure. The model appears to convergence well, which is confirmed by Figure 18 that shows a comparison between the finite element optimal test function approximations, and the approximations by *DeepONet-1D-1*. The difference between the finite element approximation and the *DeepONet-1D-1* approximation is very small.

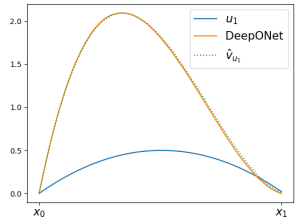
It's interesting to see that the network is able to deal with the different interval lengths on which the optimal test functions have non-zero support, without being passed that information as a variable. It means that *DeepONet-1D-1* is able to learn the solutions to three different variations of (128) based on the shape of the trial functions, namely

$$\begin{aligned}
\epsilon \int_{x_0}^{x_1} u_i' \delta_v' - \int_{x_0}^{x_1} u_i \delta_v' &= \int_{x_0}^{x_1} (h_0^2\, v_{u_i}'\, \delta_v' + v_{u_i}\, \delta_v) \\
\epsilon \int_{x_0}^{x_2} u_i' \delta_v' - \int_{x_0}^{x_2} u_i \delta_v' &= \int_{x_0}^{x_2} (h_1^2\, v_{u_i}'\, \delta_v' + v_{u_i}\, \delta_v) \\
\epsilon \int_{x_0}^{x_3} u_i' \delta_v' - \int_{x_0}^{x_3} u_i \delta_v' &= \int_{x_0}^{x_3} (h_2^2\, v_{u_i}'\, \delta_v' + v_{u_i}\, \delta_v)
\end{aligned} \tag{132}$$

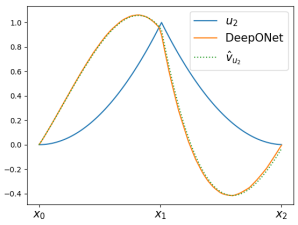
Essentially, the network *DeepONet-1D-1* has learned to remember that *separate operators* have to be used on the different trial functions, to approximate the correct solution.



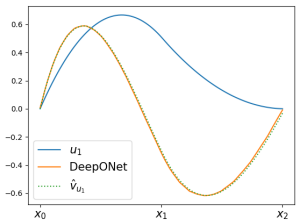
(a) Piecewise linear trial function u_1 and *DeepONet-1D-1*.



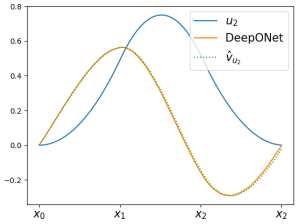
(b) C^0 quadratic trial function u_1 and *DeepONet-1D-1*.



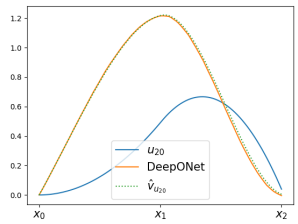
(c) C^0 quadratic trial function u_2 and *DeepONet-1D-1*.



(d) C^1 quadratic trial function u_1 and *DeepONet-1D-1*.



(e) C^1 quadratic trial function u_2 and *DeepONet-1D-1*.

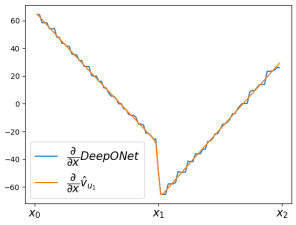


(f) C^1 quadratic trial function u_{20} and *DeepONet-1D-1*.

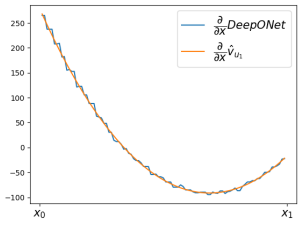
Figure 18: Comparison of *DeepONet-1D-1* vs FEM approximation of globally continuous optimal test functions corresponding to (128).

To approximate the derivatives of the optimal test functions, the gradient of the network is used. This gradient is computed using automatic differentiation, which is the standard in TensorFlow. Since the input variable x was scaled to the interval $[0, 11$, the gradient has to multiplied by $1/(x_1 - x_0)$, $1/(x_2 - x_0)$, or $1/(x_3 - x_0)$ depending on the length of the domain on which the trial

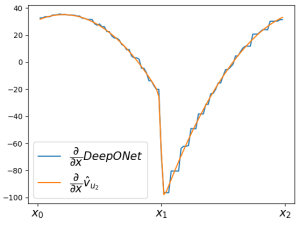
function has non-zero support. Figure 19 shows a comparison between the FEM approximation of the derivative and the *DeepONet-1D-1* gradient aproximation for the optimal test functions corresponding to the trial functions shown in Figure 13.



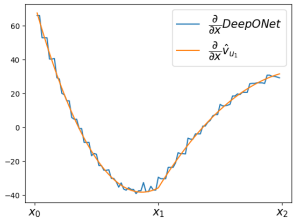
(a) Optimal test function derivative corresponding to piecewise linear trial function u_1.



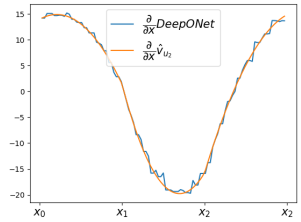
(b) Optimal test function derivative corresponding to C^0 quadratic trial function u_1.



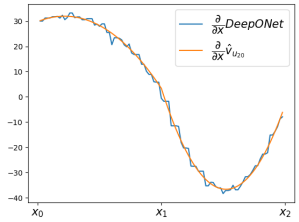
(c) Optimal test function derivative corresponding to C^0 quadratic trial function u_2.



(d) Optimal test function derivative corresponding to C^1 quadratic trial function u_1.



(e) Optimal test function derivative corresponding to C^1 quadratic trial function u_2.



(f) Optimal test function derivative corresponding to C^1 quadratic trial function u_{20}.

Figure 19: Comparison betweem FEM and *DeepONet-1D-1* gradient approximation of $\frac{\partial}{\partial x}\hat{v_{u_1}}$ corresponding to (128) for $d = 0.01$, for different trial functions.

The *DeepONet-1D-1* gradient approximations shown in Figure 19 resemble a step function. This is a result of the activation function that is used. To see why, consider the ReLU activation function and its derivative

$$\begin{aligned} f(x) &= \max(0, x) \\ f'(x) &= \begin{cases} 0, & \text{if} \quad x < 0 \\ 1, & \text{if} \quad x > 0 \end{cases} \end{aligned} \tag{133}$$

The gradient of *DeepONet-1D-1* is a sum of the derivatives of its activation functions with respect to x. Instead using the *DeepONet-1D-1* gradient approximation shown in Figure 19, it is possible to train a separate network to approximate the optimal test functions' derivatives (this approach is employed in a later section). In the next section, the *DeepONet-1D-1* network and its gradient will be implemented into finite element method.

5.1.5 Implementing *DeepONet-1D-1*

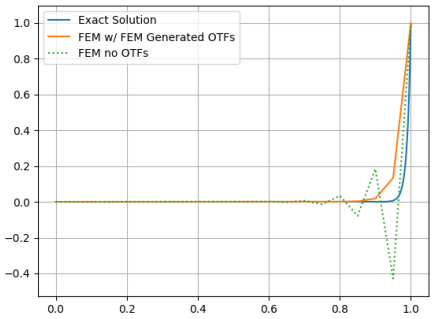
Figure 21 shows a comparison between a Galerkin finite element implementation, two implementations that use the *DeepONet-1D-1* generated and FEM generated optimal test functions respectively. The methods are implementations resulting from weak form (129) for $\epsilon = 0.01$ and $f = 0$, using a finite element discretisation of 20 elements. Using the optimal test functions leads to significant increase in the stability and accuracy of the finite element. That the optimal test functions improve the finite element method for this case was shown previously in Figure 15, but Figure 21 shows that the *DeepONet-1D-1* generated optimal test functions can do the same. In fact, the *DeepONet-1D-1* implementation is indistinguishable from the implementation that uses FEM generated optimal test functions, based on the plots that are shown. Moreover, it is promising to see that the *DeepONet-1D-1* gradient appears to be accurate enough to approximate the optimal test functions' derivatives (in fact, by setting $f = 0$ in (129) the optimal test functions disappear and only their derivatives, and therefore the *DeepONet-1D-1* gradients, are used).
The approximation errors of the implementations shown in Figure 21 are measured in the H^1 norm and displayed in Figure 20. This table confirms that the optimal test functions indeed improve the finite element method and lead to much smaller $\|u - u_n\|_{H^1(\Omega)}$ errors. It also shows how well *DeepONet-1D-1* can be used in this context. The errors corresponding to the finite elements implementations that used *DeepONet-1D-1* generated optimal test functions are hardly smaller than the methods using FEM generated optimal test functions.

Trial functions	Galerkin Method	FEM w/ DeepONet OTFs	FEM w/ FEM generated OTFs
Piecewise linear	6.59	5.55	5.55
$C^0(\Omega)$ quadratic	3.12	3.02	3.05
$C^1(\Omega)$ quadratic	3.60	3.42	3.42

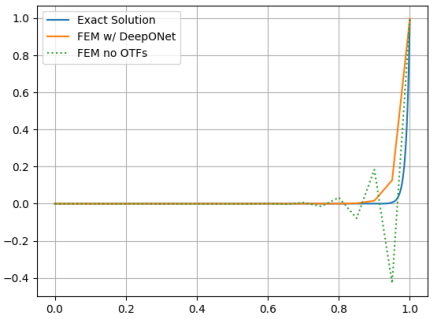
Figure 20: $\|u - u_n\|_{H^1(\Omega)}$ error comparison, corresponding to (129) with $f = 0$ and $\epsilon = 0.01$.

The errors that are reported in Figure 20 show that the implementations using the $C^0(\Omega)$ trial functions has a higher accuracy than the implementations using $C^1(\Omega)$ trial functions. This is a result of the fact that the same discretisation is being used for all the different implementations, meaning that ones with $C^0(\Omega)$ quadratic trial functions have a lot more degrees of freedom. The reason that all the implementations use the same discretisation has to do with the fact that *DeepONet-1D-1* was trained for a discretisation of 20 elements and cannot be used with other discretisations.
It can't go unnoticed that the results for the $C^0(\Omega)$ and $C^1(\Omega)$ quadratic trial functions in Figure 21 show a weird looking bubble in the approximate solution near the right boundary of the domain. As the solution approaches the boundary layer, it first goes up and then dips down. Although this behaviour might not be what one would expect from a method using optimal test functions, the implementations have been debugged and are correct. Furthermore, while the solution using optimal test functions may look weird, Figure 20 show that there is a significant improvement in the H^1 errors when compared with the Galerkin method.

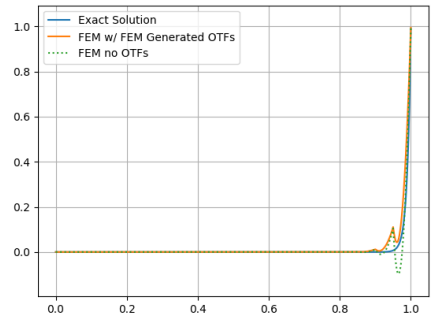
The optimal test functions were also implemented for (129) using a discretisation of 20 elements an for $\epsilon = 0.01$, but this time for $f = 1$. The resulting approximate solutions are plotted in Figure 22 and the corresponding errors are shown in Figure 23.



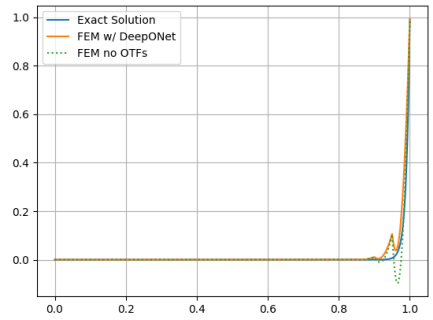
(a) Piecewise linear trial functions and FEM generated optimal test function derivatives.



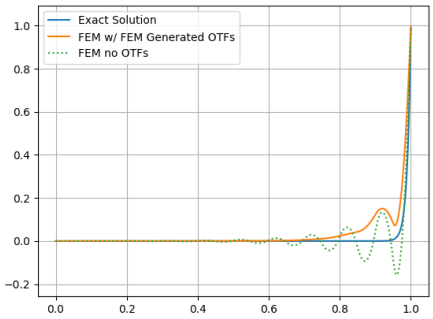
(b) Piecewise linear trial functions w/ *DeepONet-1D-1* gradients.



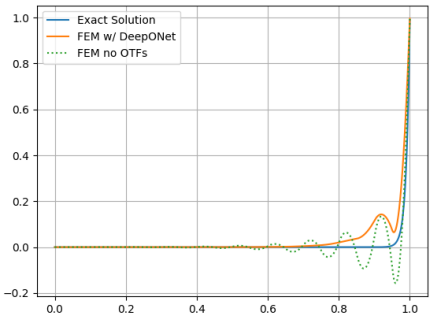
(c) C^0 quadratic trial functions w/ FEM generated optimal test function derivatives.



(d) C^0 quadratic trial functions w/ *DeepONet-1D-1* gradients.

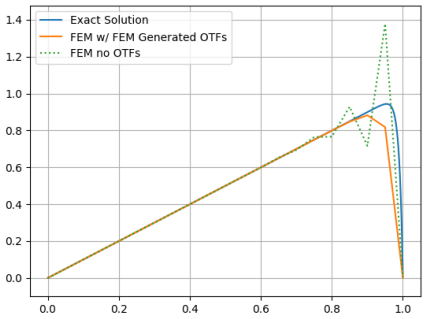


(e) C^1 quadratic trial functions w/ FEM generated optimal test function derivatives.

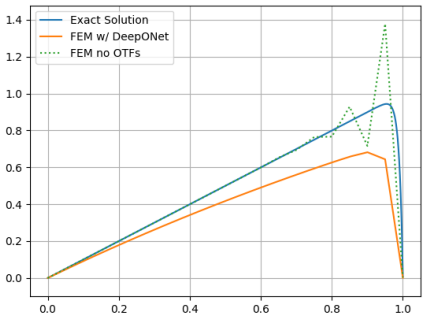


(f) C^1 quadratic trial functions w/ *DeepONet-1D-1* gradients.

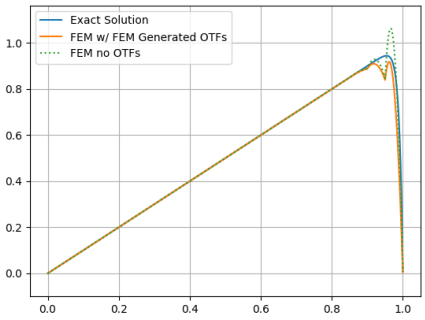
Figure 21: Comparison between Galerkin method, and methods that use FEM or *DeepONet-1D-1* generated optimal test function derivatives for problem (129) with $f = 0$ and $\epsilon = 0.01$.



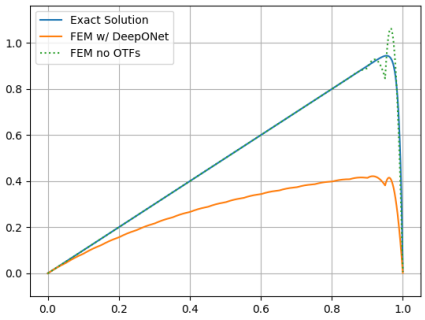
(a) Using piecewise linear trial functions FEM generated optimal test functions.



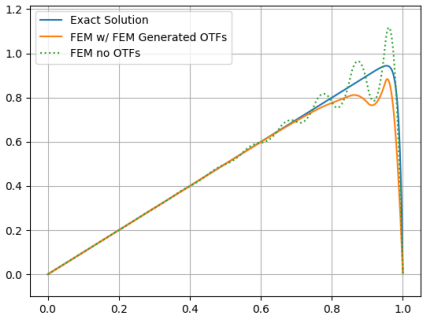
(b) Using piecewise linear trial functions w/ *DeepONet-1D-1* optimal test functions.



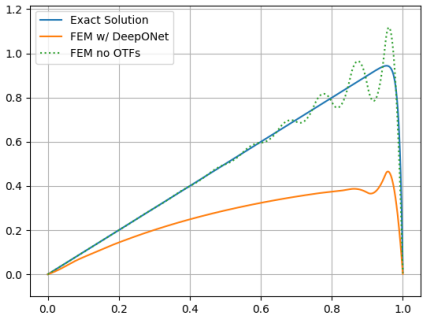
(c) Using C^0 quadratic trial functions w/ FEM generated optimal test functions.



(d) Using C^0 quadratic trial functions w/ *DeepONet-1D-1* optimal test functions.



(e) Using C^1 quadratic trial functions w/ FEM generated optimal test functions.



(f) Using C^1 quadratic trial functions w/ *DeepONet-1D-1* optimal test functions.

Figure 22: Comparison between Galerkin method, and methods that use FEM or *DeepONet-1D-1* generated optimal test functions for problem (129) with $f = 1$ and $\epsilon = 0.01$.

Except for the C^0 piecewise linear case, the *DeepONet-1D-1* generated optimal test functions do not improve the stability/accuracy of the finite element method.

Trial functions	Galerkin	FEM w/ *DeepONet-1D-1* OTFs	FEM w/ FEM generated OTFs
Piecewise linear	6.59	5.68	5.55
C^0 quadratic	3.12	4.85	3.02
C^1 quadratic	3.61	4.76	3.42

Figure 23: $\|u - u_n\|_{H^1(\Omega)}$ error comparison, corresponding to (129) with $f = 1$.

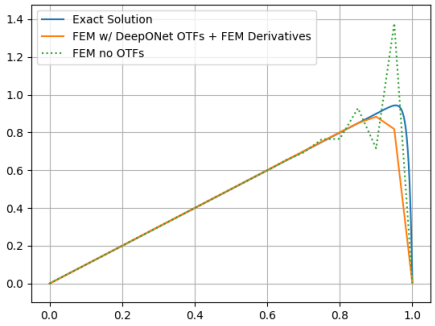
Since the *DeepONet-1D-1* generated optimal test functions did improve the finite element solution for the $f = 0$ case and performed similarly to the FEM generated optimal test functions, the poor results are somewhat surprising. Looking at Figure 18 and Figure 19, it makes sense to suspect that the problems are caused by the (relatively) imprecise approximation of the derivatives of the optimal test functions. What is puzzling is that in the previous implementation those same gradients were used too and the accuracy appeared to be high enough. Somehow the approximation errors that were introduced there did not affect the solution when the RHS was zero.

To better understand how the *DeepONet-1D-1* generated optimal test functions are causing the H^1 error to increase, the implementation of weak form (129) is run again using *DeepONet-1D-1* generated optimal test functions. This time however, the optimal test function derivatives are approximated using the finite element method. By implementing *DeepONet-1D-1* together with FEM in such a fashion, the hypothesis that the disappointing results in Figure 23 are caused by the *DeepONet-1D-1* approximation error can be tested. The results of implementing this approach are shown in Figure 24 and in Figure 25.

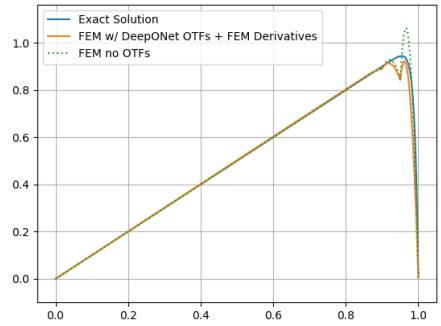
Trial functions	Galerkin	FEM w/ *DeepONet-1D-1* & FEM combo	FEM w/ FEM generated OTFs
Piecewise linear	6.59	5.55	5.55
C^0 quadratic	3.12	3.02	3.02
C^1 quadratic	3.61	3.42	3.42

Figure 24: $\|u - u_n\|_{H^1(\Omega)}$ error comparison, corresponding to weak form (129) with $f = 1$, for implementations that use *DeepONet-1D-1* to generate the optimal test functions and use the finite element method to generate the optimal test function derivatives.

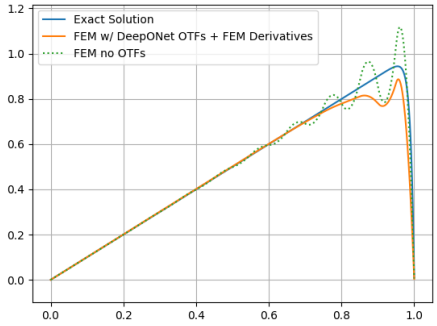
Using the combination of *DeepONet-1D-1* generated optimal test functions and FEM generated optimal test function derivatives leads to a big improvement in the error, which seems to confirm the suspicion that the issue does lie with the approximation of the derivatives by the network gradient. The table in Figure 24 shows that the *DeepONet-1D-1* generated optimal test functions are so accurate that using them produces results that are on par with using FEM generated optimal test functions.
Besides the fact that the $f = 0$ and $f = 1$ exhibit somewhat contradictory results, it is interesting to see that the optimal test functions and their derivatives have to be approximated very accurately before they start to improve the stability and accuracy of the finite element method. Figure 26 shows a FEM and *DeepONet-1D-1* approximation of the optimal function and its derivative corresponding to the piecewise linear trial function, evaluated at the quadrature points used to approximate the



(a) Using piecewise linear trial functions.



(b) Using C^0 quadratic trial functions.



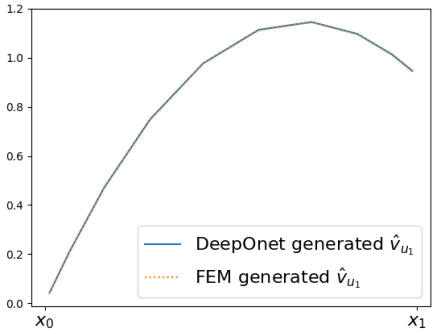
(c) Using C^1 quadratic trial functions.

Figure 25: Results of implementing weak form (129) for $f = 1$, $\epsilon = 0.01$ with a 20 element discretisation, using *DeepONet-1D-1* to generate optimal test functions and using FEM to generate the optimal test functions' derivatives.

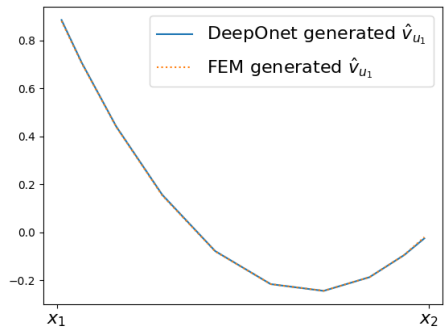
finite element integrals. Previously it was mentioned and shown that the *DeepONet-1D-1* gradient is a sum of discontinuous functions and might not be well suited to approximate the optimal test function derivative (recall the step function like appearance shown in Figure 19). However, by looking at Figure 26 it appears that it is not doing such a bad job after all. Although there is clearly room for improvement, the *DeepONet-1D-1* gradient sits quite close to the FEM approximation of the derivative. It would have been possible that using test functions that are closer to the optimal test function than the regularly used test functions, would already have improved the stability/accuracy of the finite element method. It turns that at least for the weak form in this section that is not the case.

Based on the results of the previous experiment that used FEM generated derivatives together with *DeepONet-1D-1* generated optimal test functions it makes sense to try and improve the ap-

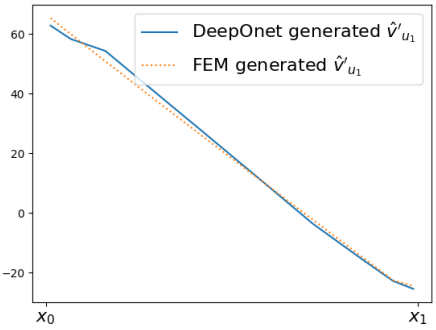
proximation of the derivatives of the optimal test functions. To achieve this goal several approaches could be used. Instead of using the ReLU activation function, a function with a continuous derivative could be used in an attempt to increase the smoothness of the network gradient. However, since the optimal test function derivative is already being approximated quite accurately (see Figure 26 below), it is questionable whether this would make a big difference. Another approach would be to train a second network that would be dedicated solely to approximating the optimal test functions' derivatives. If the DeepONets do as good a job approximating the derivatives of the optimal test functions as they do the optimal test functions themselves, this seems like the best option.



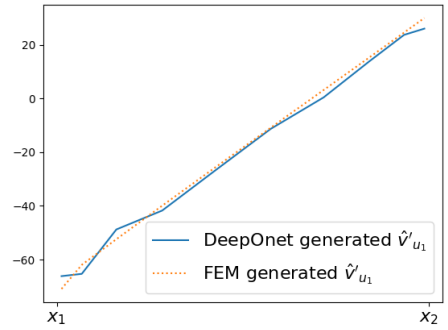
(a) Comparison of left side of $\hat{v}_{u_1}$ evaluated at quadrature points, approximated by *DeepONet-1D-1* and FEM.



(b) Comparison of right side of $\hat{v}_{u_1}$ evaluated at quadrature points, approximated by *DeepONet-1D-1* and FEM.



(c) Comparison of left side of $\hat{v}'_{u_1}$ evaluated at quadrature points, approximated by *DeepONet-1D-1* and FEM.



(d) Comparison of right side of $\hat{v}'_{u_1}$ evaluated at quadrature points, approximated by *DeepONet-1D-1* and FEM.

Figure 26: FEM vs *DeepONet-1D-1* generated optimal test function and its derivative evaluated at the quadrature points corresponding to the piecewise linear trial function for (128) with $\epsilon = 0.01$.

5.1.6 Training and Implementing *DeepONet-1D-1-grad*

A new DeepONet, called *DeepONet-1D-1-grad*, is trained to approximate the optimal test functions' derivatives. The training dataset is generated by repeatedly approximating the derivative of the solution $v_{u_i}(x)$, e.g. $v'_{u_i}(x)$, to (128) for trial function u_i at a random point x for every unique trial function that is zero on the domain boundary. The solutions are approximated using the finite element method using a discretisation of 200 elements and second order polynomial trial and shape functions. Using this approach each point in the training dataset is a triplet that looks like this

$$(\boldsymbol{u}_i, x, \hat{v'}_{u_i}(x)) \tag{134}$$

Similarly to the *DeepONet-1D-1* model, the *DeepONet-1D-1-grad* model is trained using the MSE loss function, shown in (131). Figure 27 shows the results of training *DeepONet-1D-1-grad* for 100 epochs. As was the case with the *DeepONet-1D-1* network, the loss decreases rapidly for the first few iterations after which the convergence seems to taper off. Figure 28 shows that the new network is able to approximate the optimal test function derivatives very well. One thing that is worth mentioning here is how well the network deals with the discontinuous nature of the gradients corresponding to the trial functions that have $C^0(\Omega)$ regularity. Instead of averaging out the jumps in the functions, *DeepONet-1D-grad* has learned that for some functions it needs to change its predictions very rapidly around the element boundaries.

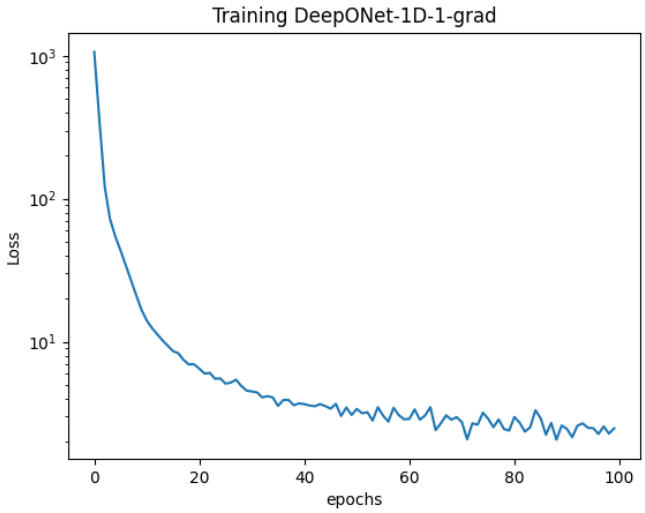
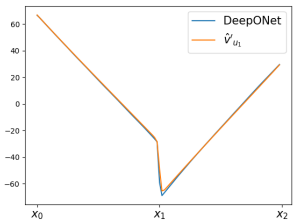


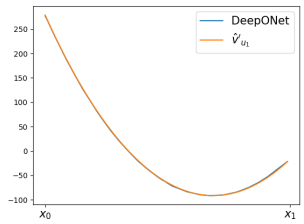
Figure 27: Convergence of MSE loss of *DeepONet-1D-1-grad* during training.

Figure 29 shows a comparison between FEM and *DeepONet-1D-grad* generated approximations of the optimal test functions' derivatives evaluated at the quadrature points used in the finite element method, for $C^0(\Omega)$ trial functions. Figure 30 shows the same comparison but for $C^1(\Omega)$ trial functions. Both figures confirm what was shown in Figure 28; that the *DeepONet-1D-grad* network is able to approximate the optimal test function derivatives very accurately. Although there still exist differences between the FEM approximations and those of the neural network,

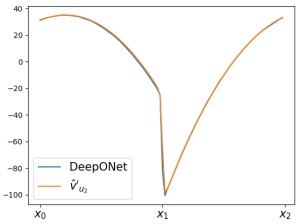
using a dedicated network is definitely much more precise than using the *DeepONet-1D-1* network gradient. The approximation no longer exhibits the discontinuous steps that were associated with the ReLU activation derivative.



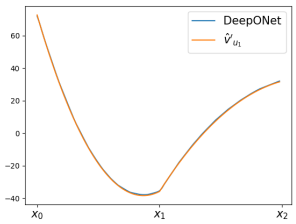
(a) Piecewise linear trial function u_1 and *DeepONet-1D-1-grad*



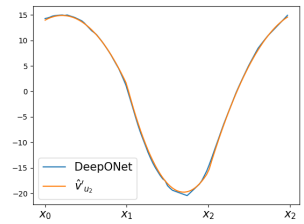
(b) C^0 quadratic trial function u_1 and *DeepONet-1D-1-grad*.



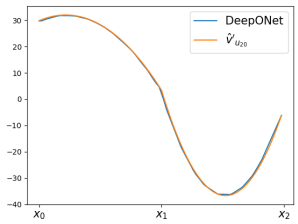
(c) C^0 quadratic trial function u_2 and *DeepONet-1D-1-grad*.



(d) C^1 quadratic trial function u_1 and *DeepONet-1D-1-grad*.



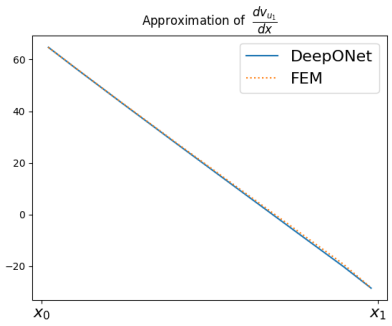
(e) C^1 quadratic trial function u_2 and *DeepONet-1D-1-grad*.



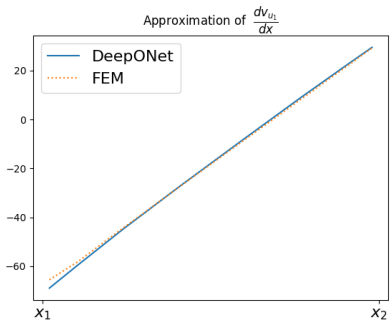
(f) C^1 quadratic trial function u_{20} and *DeepONet-1D-1-grad*.

Figure 28: Comparison of *DeepONet-1D-1-grad* vs FEM approximation of optimal test functions' derivates $\hat{v}'_{u_i}$ corresponding to (128), for a 20 element discretisation and using $\epsilon = 0.01$.

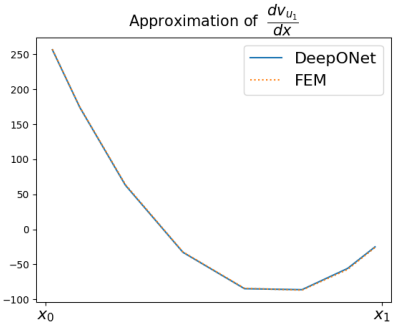
While the approximation looks a lot better, implementing the *DeepONet-1D-grad* network alongside the previously trained *DeepONet-1D-1* is not a big success. Figure 31 and Figure 32 show the results of implementing *DeepONet-1D-1-grad* together with *DeepONet-1D-1*. Although the results are an improvement over using just the *DeepONet-1D-1* network, using a Galerkin method still outperforms the *DeepONet-1D-1* + *DeepONet-1D-1-grad* combination for both the C^0 quadratic and C^1 quadratic trial functions. Since *DeepONet-1D-1-grad* appears to do a pretty good job of approximating the test function derivatives, this is a bit disappointing.
During experiments neural networks with activation with continuous derivatives were trained for longer periods of time (up to 250 epochs). Although the optimal test function approximation got more accurate it did not improve the finite element solution by much. This makes it seem that using deep learning to generate the optimal test functions for weak form (129) might not be a suitable approach.



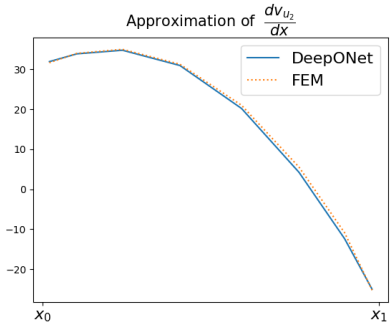
(a) Approximation at quadrature points of dv_{u_1}/dx for left side of C^0 piecewise linear trial function u_1.



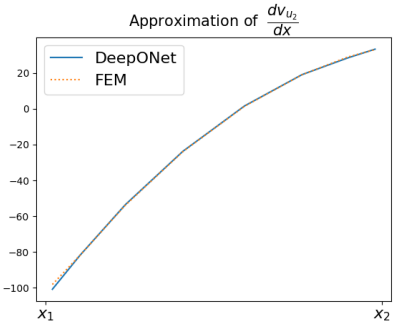
(b) Approximation at quadrature points of dv_{u_1}/dx for right side of C^0 piecewise linear trial function u_1.



(c) Approximation at quadrature points of dv_{u_1}/dx for C^0 quadratic trial function u_1.

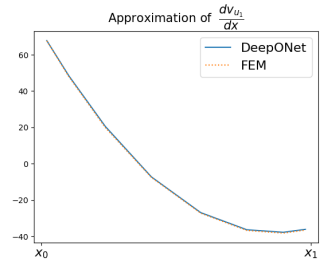


(d) Approximation at quadrature points of dv_{u_2}/dx on left side of C^0 quadratic trial function u_2.

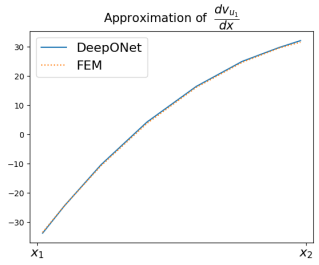


(e) Approximation at quadrature points of dv_{u_2}/dx on right side of C^0 quadratic trial function u_2.

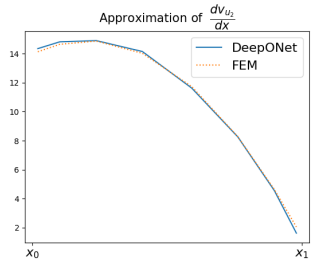
Figure 29: Comparison between *DeepONet-1D-1-grad* and FEM generate approximation of dv_{u_i}/dx evaluated at the quadrature points used in the implementation of (129).



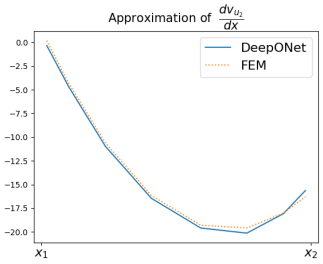
(a) Approximation at quadrature points of dv_{u_1}/dx for left side of C^1 quadratic trial function u_1.



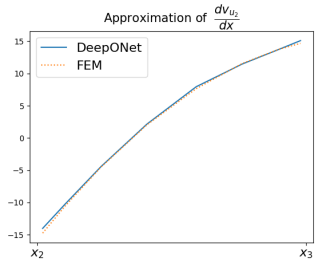
(b) Approximation at quadrature points of dv_{u_2}/dx on right side of C^1 quadratic trial function u_1.



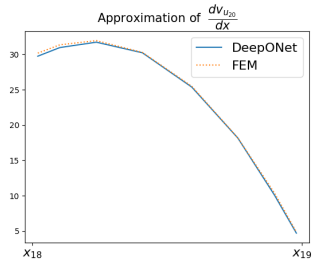
(c) Approximation at quadrature points of dv_{u_2}/dx on left side of C^1 quadratic trial function u_2.



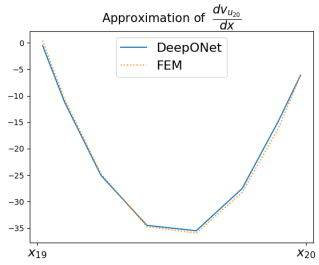
(d) Approximation at quadrature points of dv_{u_2}/dx on middle part of C^1 quadratic trial function u_2.



(e) Approximation at quadrature points of dv_{u_2}/dx on right side of C^1 quadratic trial function u_2.



(f) Approximation at quadrature points of $dv_{u_{20}}/dx$ on left side of C^0 quadratic trial function u_{20}.

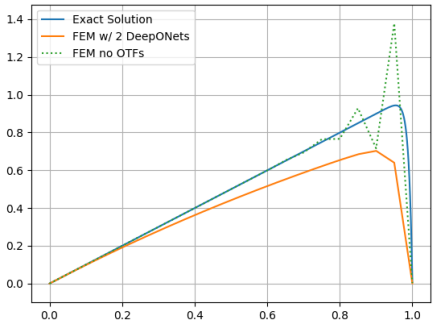


(g) Approximation at quadrature points of $dv_{u_{20}}/dx$ on right side of C^0 quadratic trial function u_{20}.

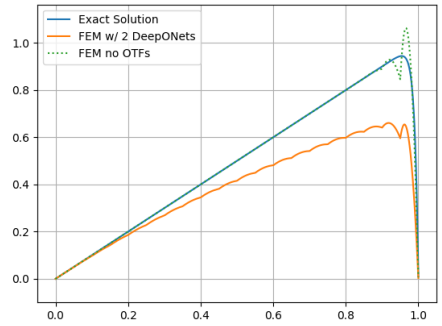
Figure 30: Comparison between *DeepONet-1D-1-grad* and FEM approximation of dv_{u_i}/dx evaluated at the quadrature points used in the implementation of (129).

Trial functions	Galerkin	w/ 2 DeepONets	FEM w/ FEM generated OTFs
Piecewise linear	6.59	5.7	5.55
C^0 quadratic	3.12	3.79	3.02
C^1 quadratic	3.61	3.65	3.42

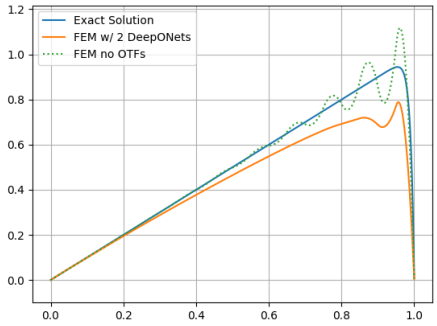
Figure 31: $\|u - u_n\|_{H^1(\Omega)}$ error comparison, corresponding to (129) with $f = 1$ and $\epsilon = 0.01$, using *DeepONet-1D-1* and the newly trained *DeepONet-1D-1-grad*.



(a) Using C^0 piecewise linear trial functions.



(b) Using C^0 quadratic trial functions.



(c) Using C^1 quadratic trial functions.

Figure 32: Results of implementing (129) using *DeepONet-1D-1* to approximate $v_{u_i}(x)$ and *DeepONet-1D-1-grad* to approximate $v'_{u_i}(x)$, with $f = 1$ and $\epsilon = 0.01$.

5.2 1D Mixed Weak Formulation

Again, consider the following boundary value problem

$$\begin{cases} -\epsilon \dfrac{d^2u}{dx^2} + \dfrac{du}{dx} = f, \quad x \in (0,1) \\ u(0) = u_0,\ u(1) = 0 \end{cases} \tag{135}$$

To use a mixed weak form implementation, a weak form that was proposed in [2] is used. First an auxiliary variable σ is introduced, where σ is set to be equal to $\sigma = \epsilon u'$. Using this new variable, the second order PDE in (135) can be reduced to a first order system as follows

$$\begin{cases} \sigma - \epsilon u' = 0 \\ -\sigma' + u' = f \\ u(0) = 0, u(1) = 1 \end{cases} \tag{136}$$

The original domain $\Omega = [0,1]$ is discretised using a partition of open subdomains (x_{k-1}, x_k), for $k = 1, \ldots, N$, such that

$$0 = x_0 < x_1 < \ldots < x_{k-1} < x_k < \ldots < x_N = 1 \tag{137}$$

The equations in (136) are multiplied by test functions v and w respectively and are weakly enforced on each subdomain, resulting in the following variational problem for each $k = 1, \ldots, N$

$$\begin{cases} \dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_k - \int_{x_{k-1}}^{x_k} u_k' w_k = 0 \\ \displaystyle\int_{x_{k-1}}^{x_k} -\sigma_k' v_k + \int_{x_{k-1}}^{x_k} u_k' v_k = \int_{x_{k-1}}^{x_k} f v_k \end{cases} \tag{138}$$

The equations in (140) can be rewritten using integration by parts into

$$\begin{cases} \dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_k + \int_{x_{k-1}}^{x_k} u_k w_k' + u_k(x_{k-1}) w_k(x_{k-1}) - u(x_k) w(x_k) = 0 \\ \displaystyle\int_{x_{k-1}}^{x_k} \sigma_k v_k' - \int_{x_{k-1}}^{x_k} u_k v_k' - u_k(x_{k-1}) v_k(x_{k-1}) + u(x_k) v_k(x_k) \\ \qquad + \sigma_k(x_{k-1}) v_k(x_{k-1}) - \sigma_k(x_k) v_k(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_k \end{cases} \tag{139}$$

By introducing the fluxes as unknowns, e.g., introducing $\tilde{\sigma}(x_k) = \sigma(x_k), \tilde{u}(x_k) = u(x_k), k = 0, \ldots, N$, the weak form can be stated as follows:

Find $\sigma_k, u_k \in L^2(x_{k-1}, x_k)$ and fluxes $\tilde{\sigma}(x_k), \tilde{u}(x_k), k = 0, \ldots, N$ such that for every $k = 1, \ldots, N$,

$$\begin{cases} \dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_k + \int_{x_{k-1}}^{x_k} u_k w_k' + \tilde{u}(x_{k-1}) w_k(x_{k-1}) - \tilde{u}(x_k) w_k(x_k) = 0 \\ \displaystyle\int_{x_{k-1}}^{x_k} \sigma_k v_k' - \int_{x_{k-1}}^{x_k} u_k v_k' - \tilde{u}(x_{k-1}) v_k(x_{k-1}) + \tilde{u}(x_k) v_k(x_k) \\ \qquad + \tilde{\sigma}(x_{k-1}) v(x_{k-1}) - \tilde{\sigma}(x_k) v(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_k \end{cases}$$

$$\forall (v_k, w_k) \in H^1(x_{k-1}, x_k) \tag{140}$$

Here $\tilde{u}(x_0) = 0$ and $\tilde{u}(x_N) = 1$ are known and are moved to the right hand side. Using the following notation

$$
\begin{aligned}
&b((u_k, \sigma_k, \tilde{u}(x_1), \ldots, \tilde{u}(x_{N-1}), \tilde{\sigma}(x_0), \ldots, \tilde{\sigma}(x_N)), (v_k, w_k)) = \\
&\sum_{k=1}^{N} \left\{ \frac{1}{\epsilon} \int_{x_{k-1}}^{x_k} \sigma_k w_k + \int_{x_{k-1}}^{x_k} u_k w_k' + \int_{x_{k-1}}^{x_k} \sigma_k v_k' - \int_{x_{k-1}}^{x_k} u_k v_k' \right\} + \sum_{k=0}^{N} \left(\tilde{\sigma}(x_{k-1}) v(x_{k-1}) - \tilde{\sigma}(x_k) v(x_k) \right) \\
&+ \sum_{k=2}^{N-1} \left(\tilde{u}(x_{k-1}) w_k(x_{k-1}) - \tilde{u}(x_k) w_k(x_k) - \tilde{u}(x_{k-1}) v_k(x_{k-1}) + \tilde{u}(x_k) v_k(x_k) \right) \\
&l(v_k, w_k) = \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} f v_k - u_0 w_k(x_0) + u_0 v_k(x_0)
\end{aligned}
\tag{141}
$$

the weak form can be written in its final compact form as

$$
\begin{aligned}
&\text{Find } \sigma_k, u_k \in L^2(x_{k-1}, x_k) \text{ and fluxes } \tilde{\sigma}(x_k), \tilde{u}(x_k), k = 0, \ldots, N \text{ such that} \\
&\qquad b((u_k, \sigma_k, \tilde{u}(x_1), \ldots, \tilde{u}(x_{N-1}), \tilde{\sigma}(x_0), \ldots, \tilde{\sigma}(x_N)), (v_k, w_k)) = l(v_k, w_k), \\
&\forall (v_k, w_k) \in H^1(x_{k-1}, x_k)
\end{aligned}
\tag{142}
$$

5.2.1 Optimal Test Functions

To derive the optimal test functions corresponding to (142) the H^1 norms will be used

$$
\begin{aligned}
\|(\mathbf{w}, \mathbf{v})\| &= \left(\sum_{k=1}^{N} \|w_k\|^2 + \|v_k\|^2 \right)^{1/2} \\
\|w_k\|^2 &= \int_{x_{k-1}}^{x_k} \left(|w_k'|^2 + |w_k|^2 \right) \\
\|v_k\|^2 &= \int_{x_{k-1}}^{x_k} \left(|v_k'|^2 + |v_k|^2 \right)
\end{aligned}
\tag{143}
$$

These norms induce the following inner product

$$
((w, v), (\delta_w, \delta_v))_V = \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} \left(w' \, \delta_w' + w \, \delta_w \right) + \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} \left(v' \, \delta_v' + v \, \delta_v \right) \tag{144}
$$

Using the bilinear form of (142) and using (144) the optimal test functions are defined by the following problem

$$
b((u_k, \sigma_k, \tilde{u}(x_1), \ldots, \tilde{u}(x_{N-1}), \tilde{\sigma}(x_0), \ldots, \tilde{\sigma}(x_N)), (v, w)) = ((w, v), (\delta_w, \delta_v))_V \tag{145}
$$

The weak form in (142) allows the optimal test functions to be discontinuous, meaning that they can be computed on an element-wise basis. The local variational problems for determining optimal

test functions are as follows:

$$\begin{aligned}\int_{x_{k-1}}^{x_k}(w'\delta_w'+w\delta_w)&=\frac{1}{\epsilon}\int_{x_{k-1}}^{x_k}\sigma_k\delta_w+\int_{x_{k-1}}^{x_k}u_k\delta_w'+\tilde{u}(x_{k-1})\delta_w(x_{k-1})-\tilde{u}(x_k)\delta_w(x_k)\\ \int_{x_{k-1}}^{x_k}(v'\delta_v'+v\delta_v)&=\int_{x_{k-1}}^{x_k}\sigma_k\delta_v'-\int_{x_{k-1}}^{x_k}u_k\delta_v'-\tilde{u}(x_{k-1})\delta_v(x_{k-1})+\tilde{u}(x_k)\delta_v(x_k)\\ &\quad+\tilde{\sigma}(x_{k-1})\delta_v(x_{k-1})-\tilde{\sigma}(x_k)\delta_v(x_k)\end{aligned}\tag{146}$$

for all $\delta_w\in H^1(x_{k-1},x_k),\delta_v\in H^1(x_{k-1},x_k)$. Using (146) the problems that define the local restrictions of the optimal test functions are:

Find $(w_{\sigma_k},v_{\sigma_k})$ such that

$$\begin{aligned}\int_{x_{k-1}}^{x_k}(w_{\sigma_k}'\delta_w'+w_{\sigma_k}\delta_w)&=\frac{1}{\epsilon}\int_{x_{k-1}}^{x_k}\sigma_k\delta_w,\quad\forall\delta_w\in H^1(x_{k-1},x_k)\\ \int_{x_{k-1}}^{x_k}(v_{\sigma_k}'\delta_v'+v_{\sigma_k}\delta_v)&=\int_{x_{k-1}}^{x_k}\sigma_k\delta_v',\quad\forall\delta_v\in H^1(x_{k-1},x_k)\end{aligned}\tag{147}$$

Find (w_{u_k},v_{u_k}) such that

$$\begin{aligned}\int_{x_{k-1}}^{x_k}(w_{u_k}'\delta_w'+w_{u_k}\delta_w)&=\int_{x_{k-1}}^{x_k}u_k\delta_w',\quad\forall\delta_w\in H^1(x_{k-1},x_k)\\ \int_{x_{k-1}}^{x_k}(v_{u_k}'\delta_v'+v_{u_k}\delta_v)&=-\int_{x_{k-1}}^{x_k}u_k\delta_v',\quad\forall\delta_v\in H^1(x_{k-1},x_k)\end{aligned}\tag{148}$$

Find $(0,v_{\tilde{\sigma}(x_k)})$ such that

$$\begin{aligned}\int_{x_{k-1}}^{x_k}(v_{\tilde{\sigma}(x_k)}'\delta_v'+v_{\tilde{\sigma}(x_k)}\delta_v)&=-\tilde{\sigma}(x_k)\delta_v(x_k),\quad\forall\delta_v\in H^1(x_{k-1},x_k)\\ \int_{x_k}^{x_{k+1}}(v_{\tilde{\sigma}(x_k)}'\delta_v'+v_{\tilde{\sigma}(x_k)}\delta_v)&=\tilde{\sigma}(x_k)\delta_v(x_k),\quad\forall\delta_v\in H^1(x_k,x_{k+1})\end{aligned}\tag{149}$$

Find $(w_{\tilde{u}(x_k)},v_{\tilde{u}(x_k)})$ such that

$$\begin{aligned}\int_{x_{k-1}}^{x_k}(w_{\tilde{u}(x_k)}'\delta_w'+w_{\tilde{u}(x_k)}\delta_w)&=-\tilde{u}(x_k)\delta_w(x_k),\quad\forall\delta_w\in H^1(x_{k-1},x_k)\\ \int_{x_k}^{x_{k+1}}(w_{\tilde{u}(x_k)}'\delta_w'+w_{\tilde{u}(x_k)}\delta_w)&=\tilde{u}(x_k)\delta_w(x_k),\quad\forall\delta_w\in H^1(x_k,x_{k+1})\\ \int_{x_{k-1}}^{x_k}(v_{\tilde{u}(x_k)}'\delta_v'+v_{\tilde{u}(x_k)}\delta_v)&=\tilde{u}(x_k)\delta_v(x_k),\quad\forall\delta_v\in H^1(x_{k-1},x_k)\\ \int_{x_k}^{x_{k+1}}(v_{\tilde{u}(x_k)}'\delta_v'+v_{\tilde{u}(x_k)}\delta_v)&=-\tilde{u}(x_k)\delta_v(x_k),\quad\forall\delta_v\in H^1(x_k,x_{k+1})\end{aligned}\tag{150}$$

A finite element approximation of the optimal test functions corresponding to the fluxes in (149) and (150) is shown in Figure 33, while Figure 34 and Figure 35 show similar optimal test function approximations corresponding to (147), and Figure 36 and Figure 37 show the optimal test

function approximations corresponding to (148). The optimal test functions shown in these figures correspond to a diffusion coefficient of $\epsilon = 0.01$ and a discretisation of the domain $\Omega = [0, 1]$ into 80 elements. To approximate the optimal test functions using FEM, a discretisation of 40 elements and fourth order polynomials are used. In each of these Figures, σ_i denotes a trial function, and $\hat{w}_{\sigma_i}$, $\hat{v}_{\sigma_i}$, etc, denote the corresponding optimal test function approximation. Unlike the optimal test functions in the 1D non-mixed case, these optimal test functions do no place more weight to the upwind part of their support.

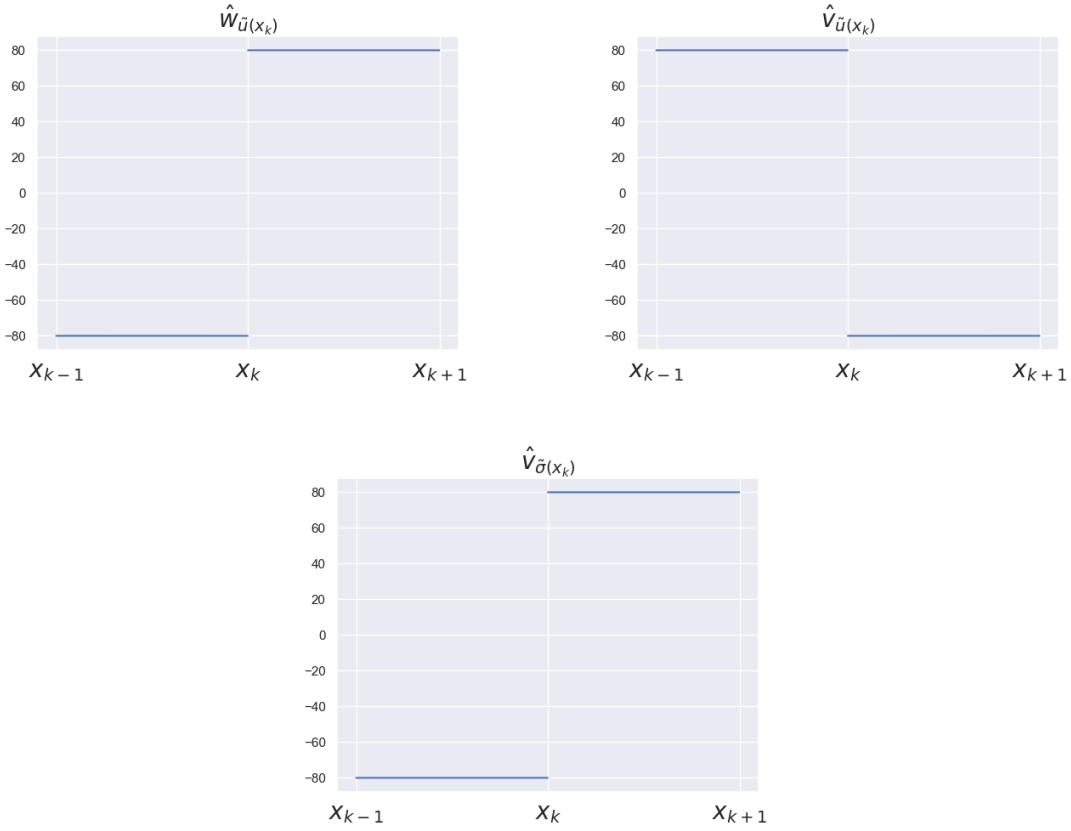


Figure 33: FEM approximation of optimal test functions $(w_{\tilde{u}(x_k)}, v_{\tilde{u}(x_k)})$ and $(0, v_{\tilde{\sigma}(x_k)})$ as defined in (150) and (149) corresponding to fluxes $\tilde{u}(x_k)$ and $\tilde{\sigma}(x_k)$, corresponding to a discretisation using 80 elements.

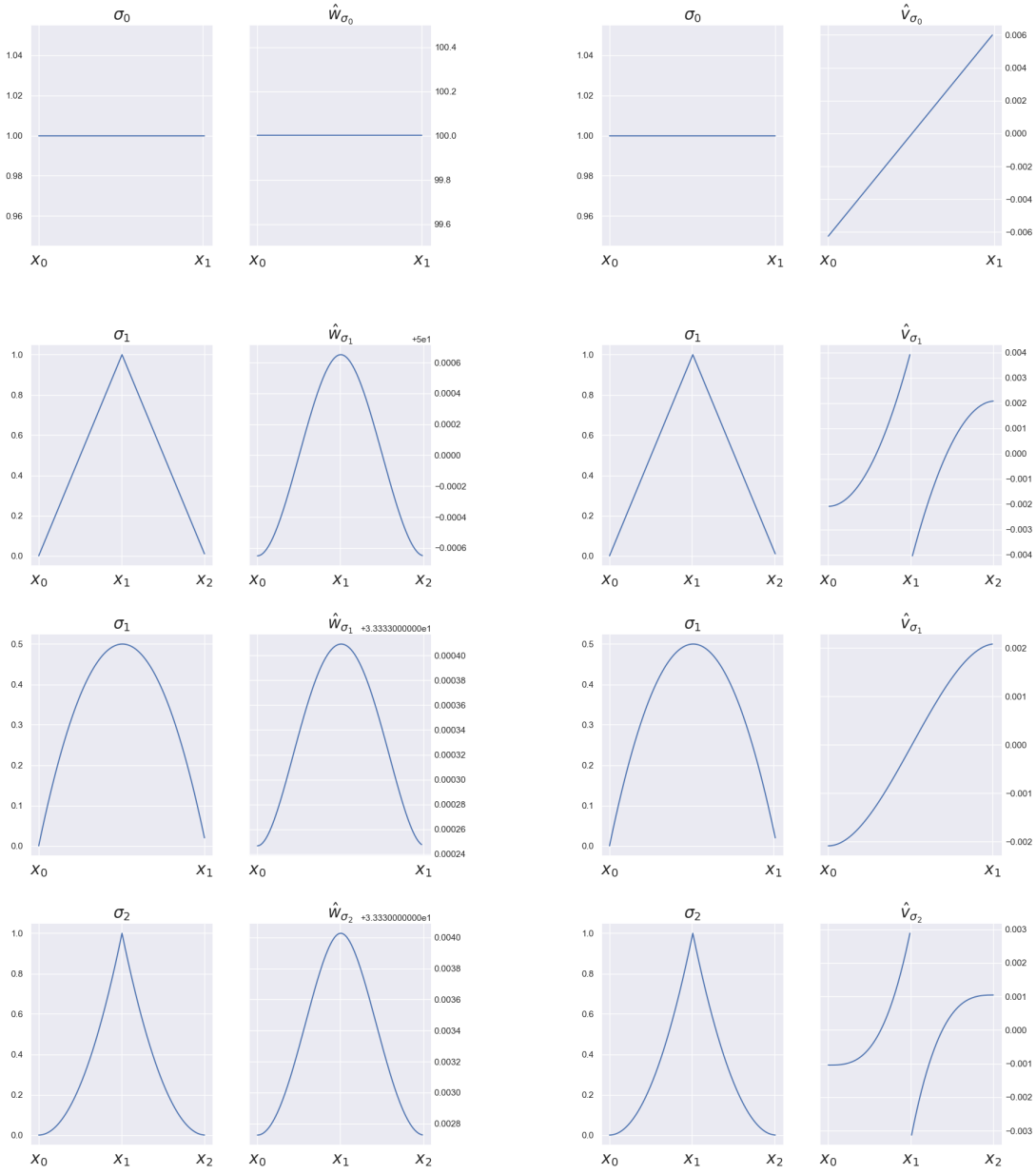


Figure 34: FEM approximation of optimal test functions (w_σ, v_σ) as defined in (147), corresponding to a discretisation using 80 elements and $\epsilon = 0.01$, for constant, piecewise linear, and $C^0(\Omega)$ quadratic trial functions.

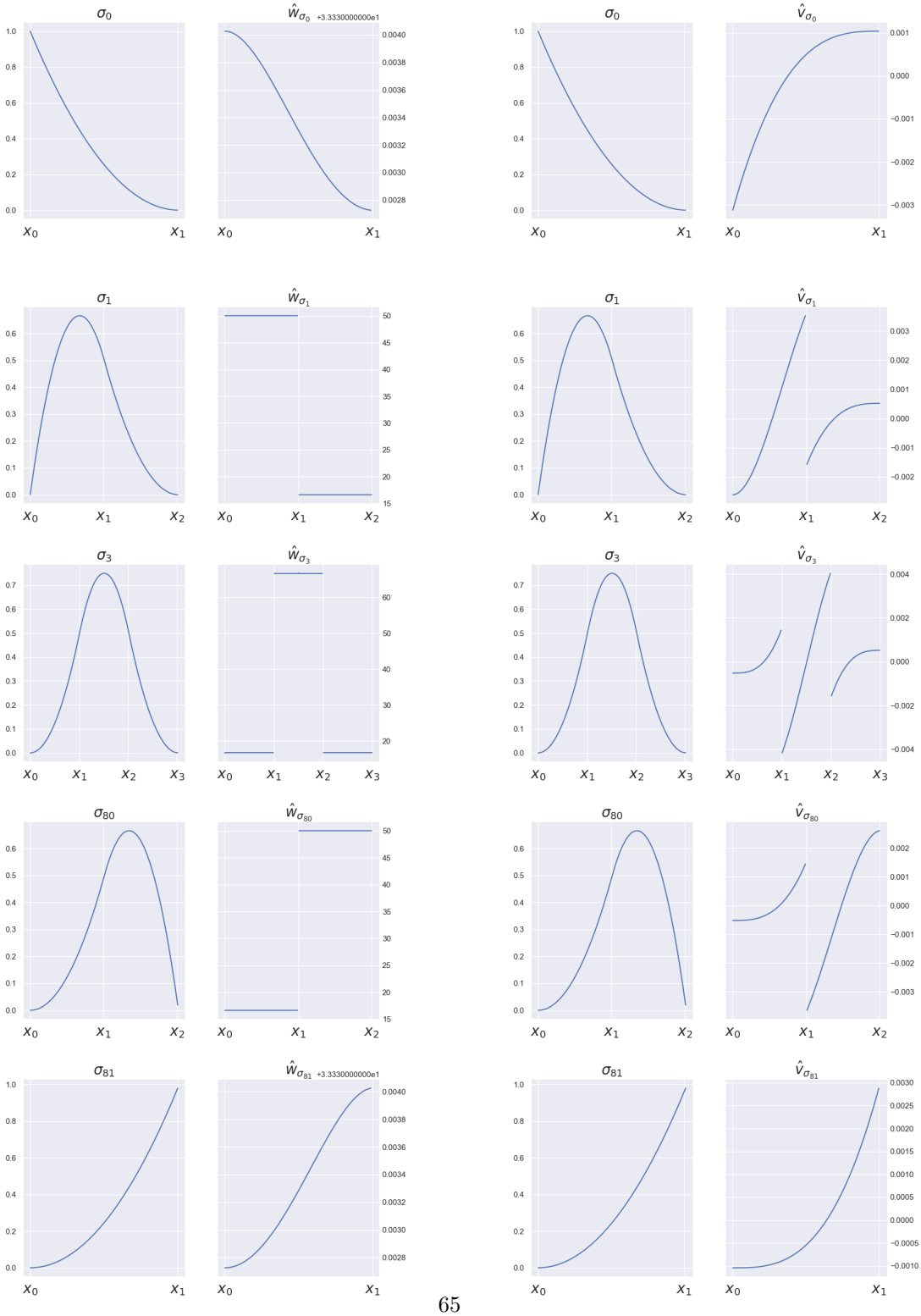


Figure 35: FEM approximation of optimal test functions (w_σ, v_σ) as defined in (147), corresponding to a discretisation using 80 elements and using $\epsilon = 0.01$, for $C^1(\Omega)$ quadratic trial functions.

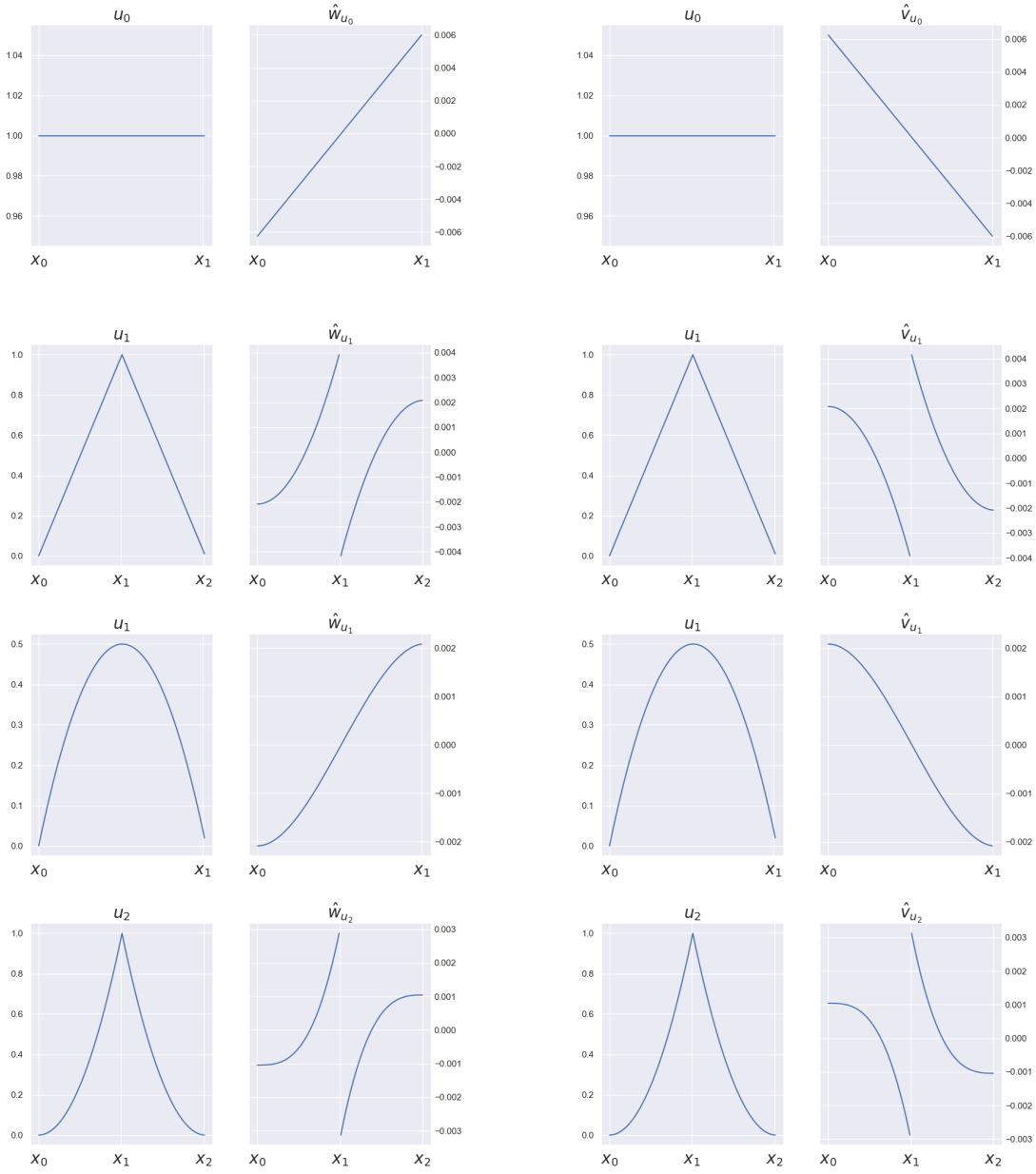


Figure 36: FEM approximation of optimal test functions (w_u, v_u) as defined in (148), corresponding to a discretisation using 80 elements, for constant, piecewise linear, and $C^0(\Omega)$ quadratic trial functions.

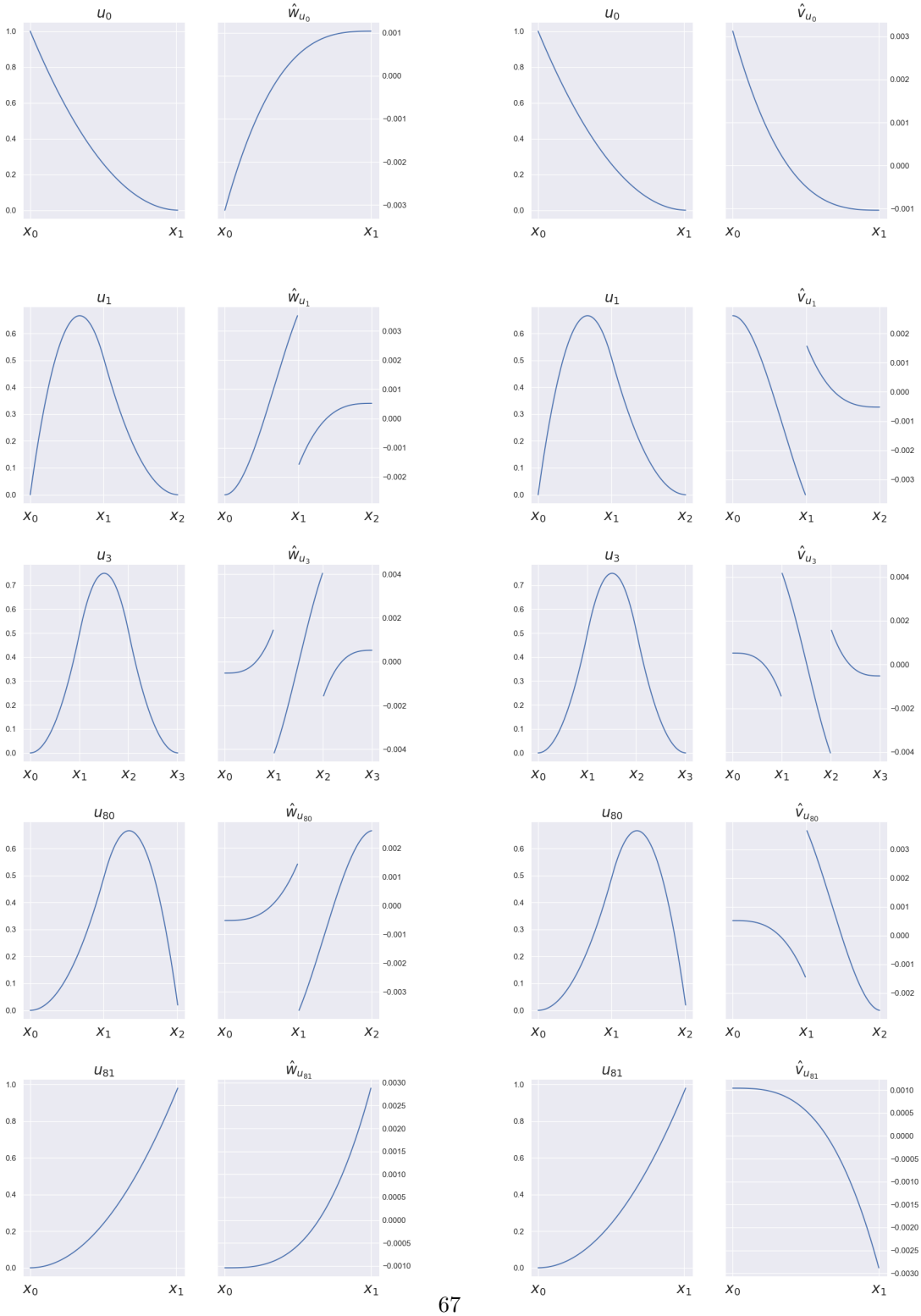


Figure 37: FEM approximation of optimal test functions (w_u, v_u) as defined in (148), corresponding to a discretisation using 80 elements, for $C^1(\Omega)$ quadratic trial functions.

5.2.2 Implementing the Optimal Test Functions

Having found the optimal test functions $(w_{\sigma_k}, v_{\sigma_k})$, (w_{u_k}, v_{u_k}), $(w_{\tilde{u}(x_k)}, v_{\tilde{u}(x_k)})$, and $(0, v_{\tilde{\sigma}(x_k)})$, the weak form becomes

Find $\sigma_k, u_k \in L^2(x_{k-1}, x_k)$ and fluxes $\tilde{u}(x_k), k = 1, \ldots, N-1$, and $\tilde{\sigma}(x_k), k = 0, \ldots, N$ such that

$$
\begin{cases}
\dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_{\sigma_k} + \int_{x_{k-1}}^{x_k} u_k w'_{\sigma_k} + \tilde{u}(x_{k-1}) w_{\sigma_k}(x_{k-1}) - \tilde{u}(x_k) w_{\sigma_k}(x_k) + \int_{x_{k-1}}^{x_k} \sigma_k v'_{\sigma_k} - \int_{x_{k-1}}^{x_k} u_k v'_{\sigma_k} \\
\qquad - \tilde{u}(x_{k-1}) v_{\sigma_k}(x_{k-1}) + \tilde{u}(x_k) v_{\sigma_k}(x_k) + \tilde{\sigma}(x_{k-1}) v_{\sigma_k}(x_{k-1}) - \tilde{\sigma}(x_k) v_{\sigma_k}(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_{\sigma_k} \\
\dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_{u_k} + \int_{x_{k-1}}^{x_k} u_k w'_{u_k} + \tilde{u}(x_{k-1}) w_{u_k}(x_{k-1}) - \tilde{u}(x_k) w_{u_k}(x_k) + \int_{x_{k-1}}^{x_k} \sigma_k v'_{u_k} - \int_{x_{k-1}}^{x_k} u_k v'_{u_k} \\
\qquad - \tilde{u}(x_{k-1}) v_{u_k}(x_{k-1}) + \tilde{u}(x_k) v_{u_k}(x_k) + \tilde{\sigma}(x_{k-1}) v_{u_k}(x_{k-1}) - \tilde{\sigma}(x_k) v_{u_k}(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_{u_k} \\
\dfrac{1}{\epsilon}\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k w_{\tilde{u}(x_k)} + \int_{x_{k-1}}^{x_k} u_k w'_{\tilde{u}(x_k)} + \tilde{u}(x_{k-1}) w_{\tilde{u}(x_k)}(x_{k-1}) - \tilde{u}(x_k) w_{\tilde{u}(x_k)}(x_k) + \int_{x_{k-1}}^{x_k} \sigma_k v'_{\tilde{u}(x_k)} \\
\qquad - \displaystyle\int_{x_{k-1}}^{x_k} u_k v'_{\tilde{u}(x_k)} - \tilde{u}(x_{k-1}) v_{\tilde{u}(x_k)}(x_{k-1}) + \tilde{u}(x_k) v_{\tilde{u}(x_k)}(x_k) + \tilde{\sigma}(x_{k-1}) v_{\tilde{u}(x_k)}(x_{k-1}) \\
\qquad - \tilde{\sigma}(x_k) v_{\tilde{u}(x_k)}(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_{\tilde{u}(x_k)} \\
\displaystyle\int_{x_{k-1}}^{x_k} \sigma_k v'_{\tilde{\sigma}(x_k)} - \int_{x_{k-1}}^{x_k} u_k v'_{\tilde{\sigma}(x_k)} - \tilde{u}(x_{k-1}) v_{\tilde{\sigma}(x_k)}(x_{k-1}) + \tilde{u}(x_k) v_{\tilde{\sigma}(x_k)}(x_k) + \tilde{\sigma}(x_{k-1}) v_{\tilde{\sigma}(x_k)}(x_{k-1}) \\
\qquad - \tilde{\sigma}(x_k) v_{\tilde{\sigma}(x_k)}(x_k) = \displaystyle\int_{x_{k-1}}^{x_k} f v_{\tilde{\sigma}(x_k)}
\end{cases}
\tag{151}
$$

The results of implementing this weak form for $\epsilon = 0.01$ are shown in Figure 38. Both the errors $\|u - u^h\|_{L^2(\Omega)}$ and $\|\sigma - \sigma^h\|_{L^2(\Omega)}$ converge nicely. While the convergence of the error in σ^h is not shown in [2], the convergence of u^h in [2] is the same to the one shown in 38. Although the order of convergence is not reported for these graphs, the difference between the error at 10 and 70 degrees of freedom in this thesis is very similar to the difference shown in [2].

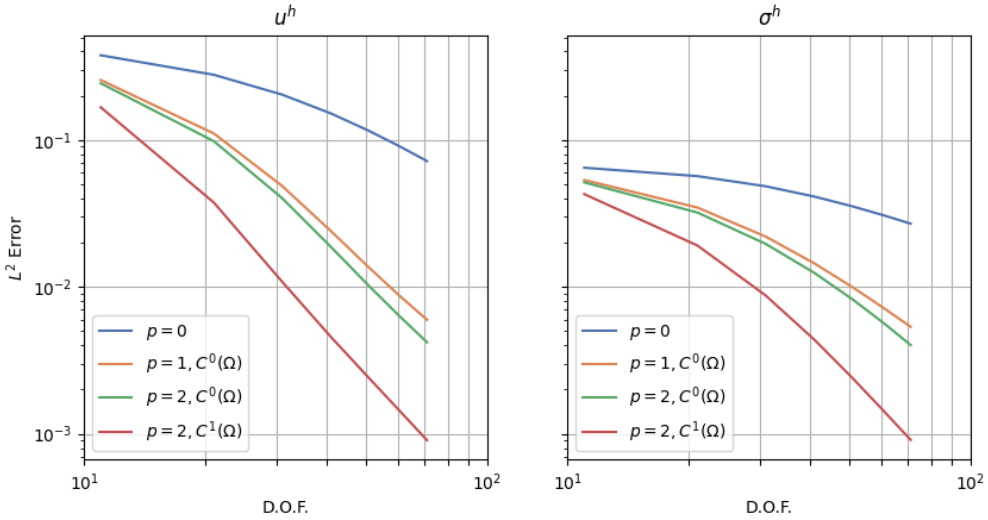


Figure 38: Implementation of the optimal test functions shown in Figure 33, Figure 34, Figure 35, Figure 36, and Figure 37, for piecewise constant, piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions, and for $\epsilon = 0.01$.

5.2.3 Training the DeepONets

Similar to the 1D non-mixed case, DeepONets are used to approximate the optimal test functions. To approximate the optimal test functions (w_σ, v_σ) and (w_u, v_u) in (147) and (148) respectively, four different DeepONets are trained, called *DeepONet-1D-*w_σ, *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u, corresponding to a discretisation of the domain Ω into 80 elements and $\epsilon = 0.01$. For every DeepONet a training dataset is built by approximating the optimal test functions corresponding to the piecewise constant, piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions, at random points on the domain. As was done in the non-mixed case, the x values that represent points on the domain are scaled to the interval $[0, 1]$. Using this approach, a datapoint in one of the four resulting datasets is a triplet that looks like this:

$$(\boldsymbol{u}_i, x, \hat{v}_{u_i}(x)) \tag{152}$$

Here $\boldsymbol{u}_i$ is vector that contains the values of trial function u_i evaluated at a set of fixed sensors, x a random point from the interval $[0, 1]$ that corresponds to a point on the original domain on which the trial function is non-zero, and $\hat{v}_{u_i}(x)$ is an approximation of the optimal test function. Similar to before, 100 sensors will be used to evaluate $\boldsymbol{u}_i$ on. The same DeepONet network configuration that was used in the previous 1D section will be used, using three layers in the trunk network, each using 100 neurons, and two layers in the branch network, each using 100 neurons. Lastly, the ReLU function will be used as an activation function again.
Since the approximation error of the DeepONets proved to be a problem in the previous section, the approach in this section will start by using FEM to approximate the optimal test functions corresponding to the fluxes as defined in (149) and (150), and the derivatives of the optimal test functions defined in (147) and (148). The DeepONets will initially just be used to implement the

optimal test functions corresponding to (147) and (148).
Figure 39 shows the convergence of the MSE loss during the training of the four networks. While the errors of *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u converge nicely, the error of *DeepONet-1D-*w_σ barely improves after the first epoch. Looking at the DeepONets approximations of the optimal test functions in Figure 40, Figure 41, Figure 42, and Figure 43, it becomes clear that the networks *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u indeed approximate the optimal test functions very accurately, while the *DeepONet-1D-*w_σ does not do as good of a job. In each of these Figures, the trial function is again denoted by σ_i or u_i and the corresponding optimal test functions by $\hat{w}_{\sigma_i}$, $\hat{w}_{u_i}$, etc.

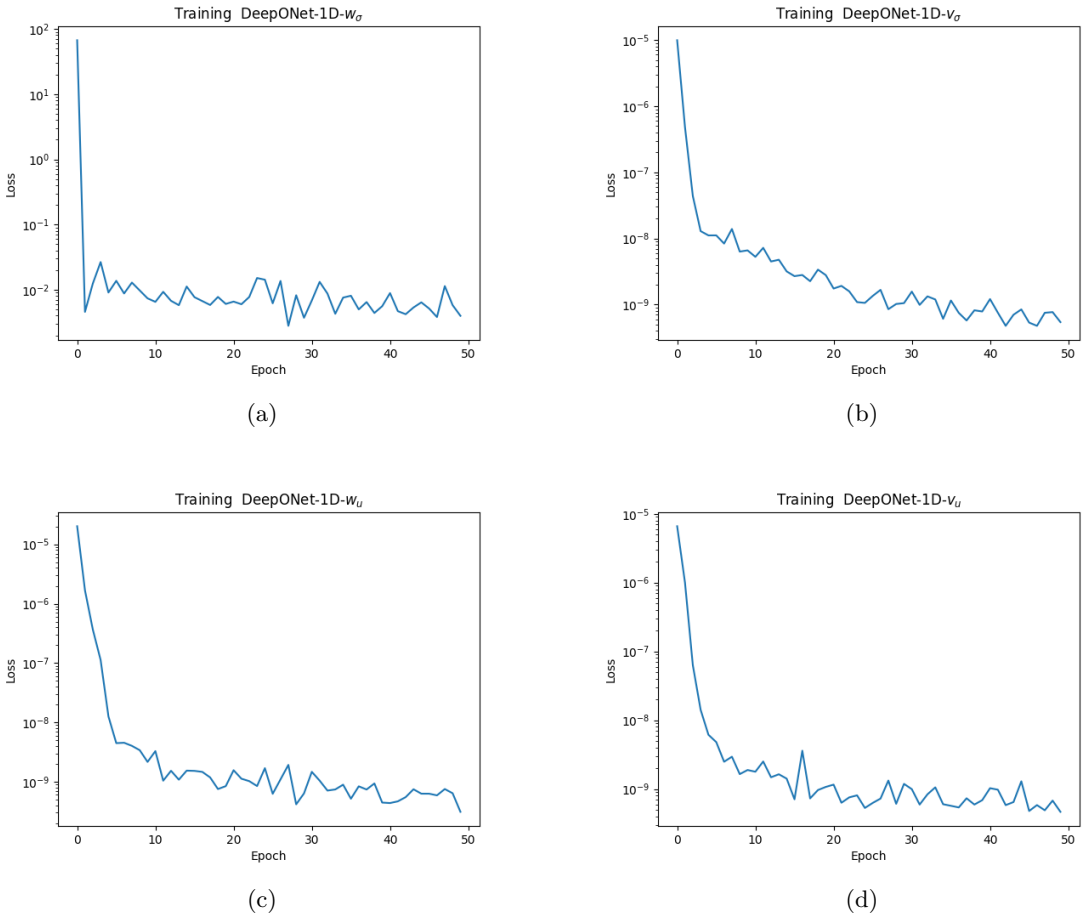


Figure 39: Convergence of MSE loss during training of DeepONets *DeepONet-1D-*w_σ, *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u for 50 epochs each.

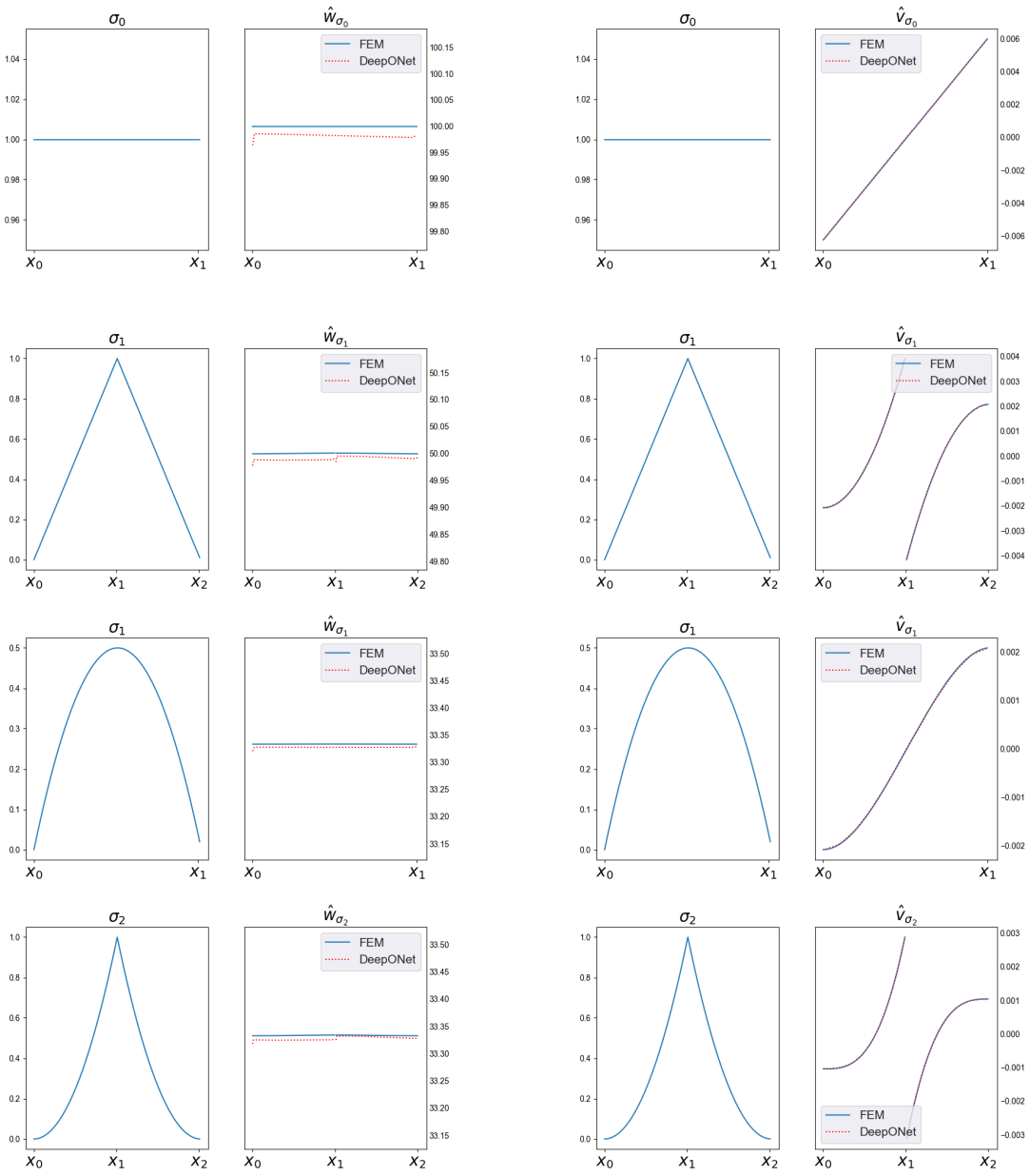


Figure 40: FEM vs *DeepONet-1D-w_σ* and *DeepONet-1D-v_σ* approximation of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for piecewise constant, piecewise linear, and $C^0(\Omega)$ trial functions.

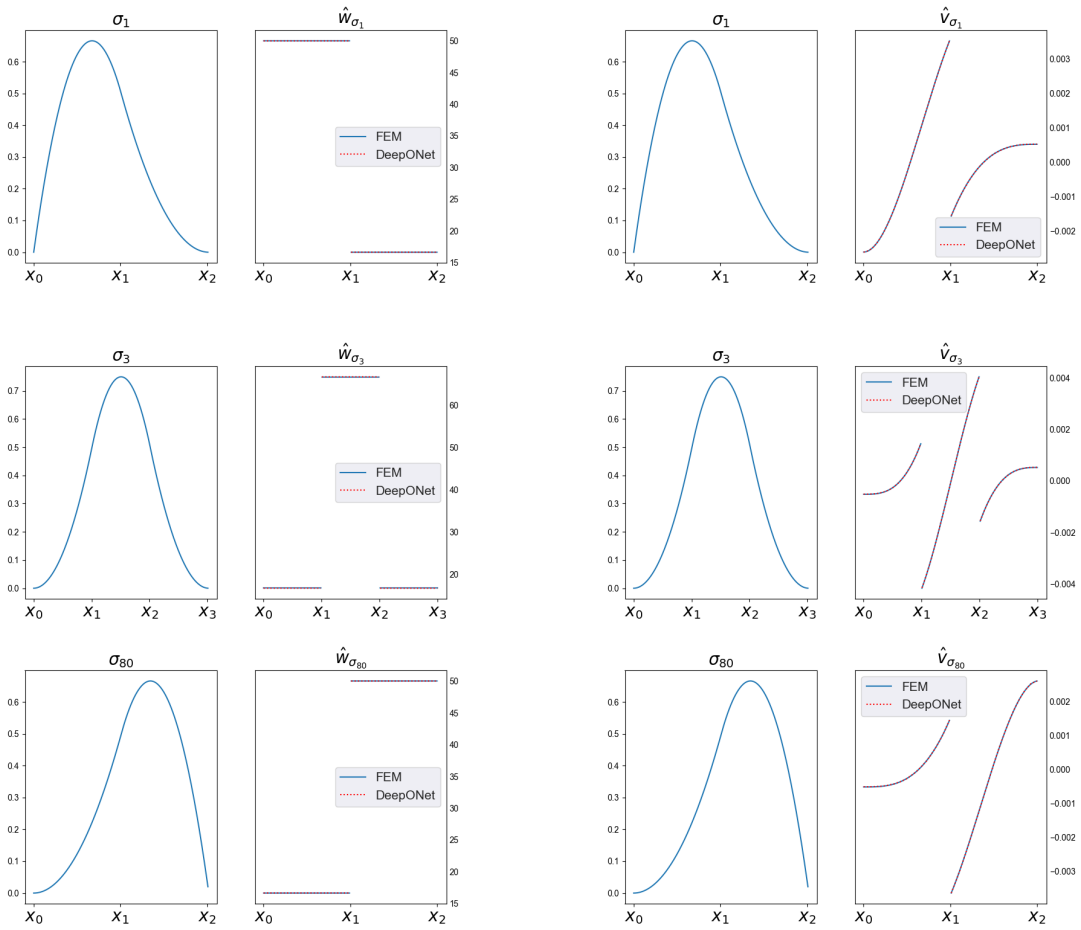


Figure 41: FEM vs *DeepONet-1D-w_σ* and *DeepONet-1D-v_σ* approximation of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for $C^1(\Omega)$ quadratic trial functions.

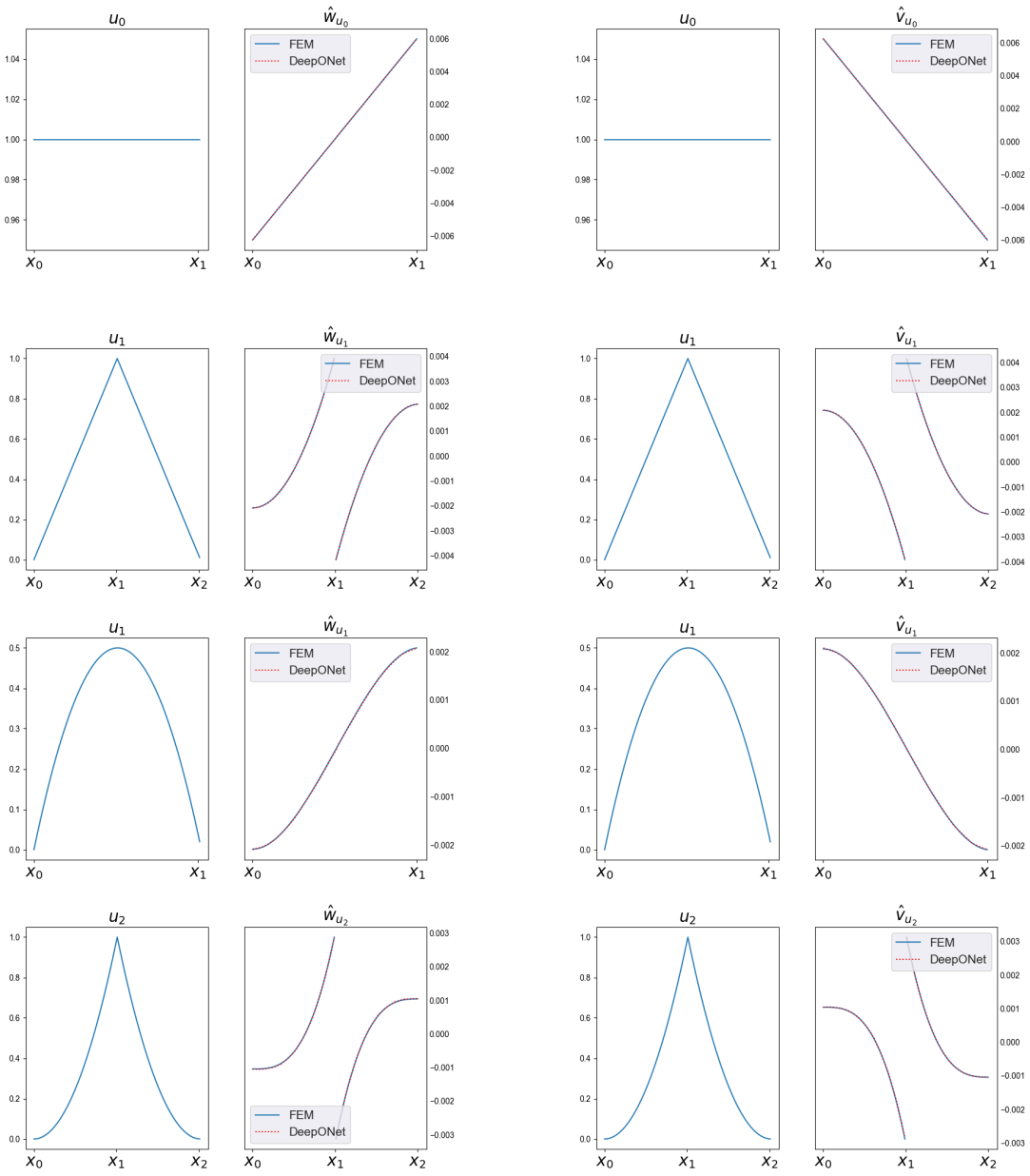


Figure 42: FEM vs *DeepONet-1D-w_u* and *DeepONet-1D-v_u* approximation of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for piecewise constant, piecewise linear, and $C^0\Omega$) quadratic trial functions.

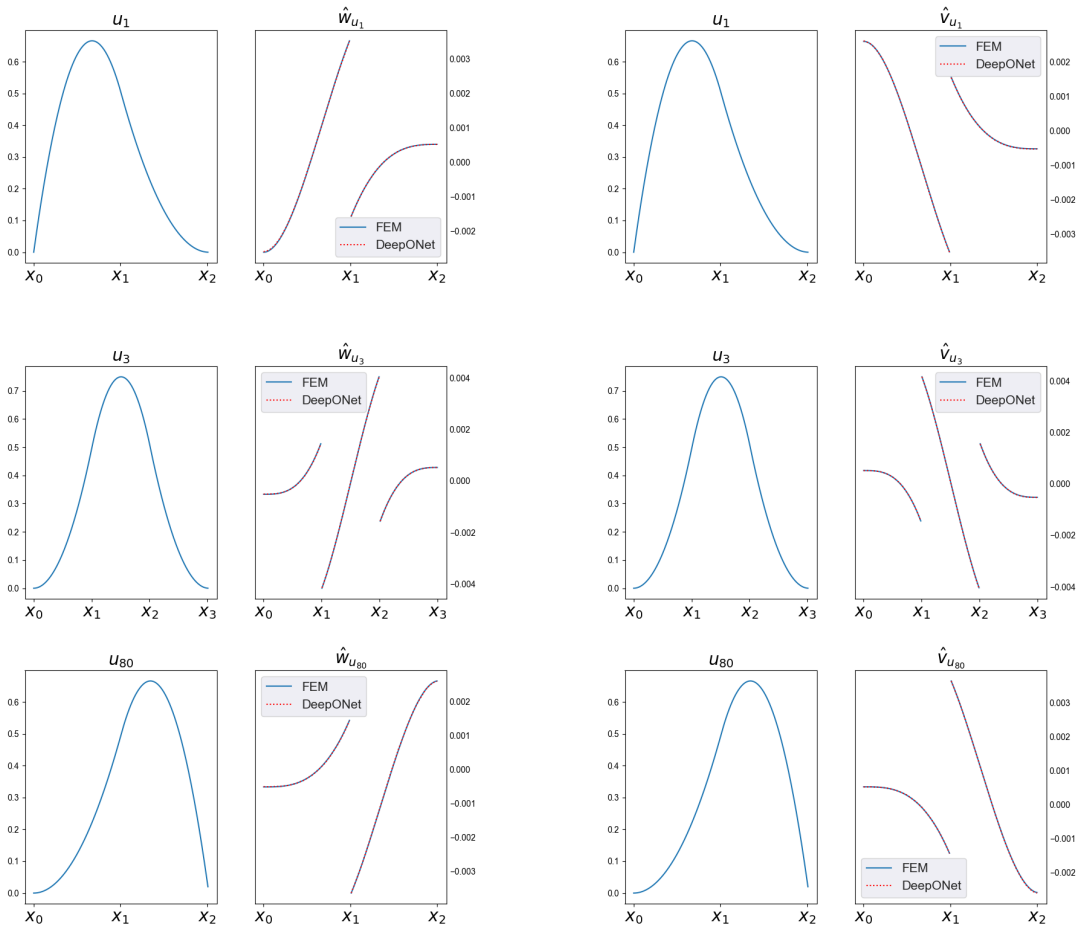


Figure 43: FEM vs *DeepONet-1D-w_u* and *DeepONet-1D-v_u* approximation of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for $C^1(\Omega)$ quadratic trial functions.

5.2.4 Implementing the DeepONets

As was mentioned in the previous section, the initial testing for this mixed weak form is done by using DeepONets to approximate the optimal test functions corresponding to (147) and (148) only. The optimal test functions corresponding to the fluxes in (149) and (150) are generated using FEM, as will the derivatives of the optimal test functions shown in (147) and (148). The results of implementing *DeepONet-1D-σ_w*, *DeepONet-1D-σ_v*, *DeepONet-1D-u_w*, and *DeepONet-1D-u_v* into the finite element method, using piecewise constant, piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions are shown in Figure 48. The corresponding errors in L^2 are shown in Figure 44 and 45. Clearly the method using the four DeepONets does considerable worse than the method using FEM generated optimal test functions. Especially in the $C^0(\Omega)$ case, where the solution starts to exhibit extreme oscillations for both u^h and

Trial functions	FEM w/ four DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.114	0.0534
Piecewise linear	0.210	0.00428
$C^0(\Omega)$ quadratic	0.214	0.000618
$C^1(\Omega)$ quadratic	0.215	0.000652

Figure 44: $\|u - u^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, with a discretisation of 80 elements, using all four DeepONets.

Trial functions	FEM w/ four DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.0461	0.024
Piecewise linear	0.0752	0.00414
$C^0(\Omega)$ quadratic	0.2621	0.000592
$C^1(\Omega)$ quadratic	0.0737	0.000619

Figure 45: $\|\sigma - \sigma^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, with a discretisation of 80 elements, using all four DeepONets.

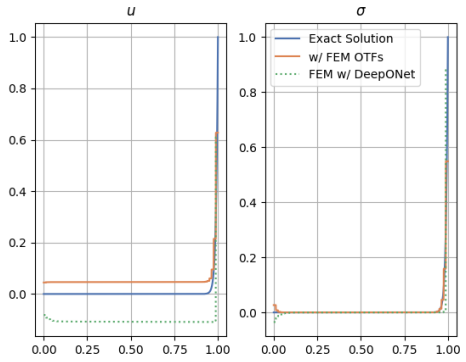
σ^h. Looking at the errors in Figures 44 and 45 it becomes clear how much worse the method performs when the optimal test functions are being generated by the DeepONets. In not one of these cases is the method using DeepONets remotely close to the original method. Based on the convergence of the MSE error of the four networks and the comparison between FEM and DeepONet generated optimal test functions shown in Figure 40, it makes sense to think that the problems shown in Figure 48 are caused by the inaccurate approximation of optimal test function w_σ by *DeepONet-1D-*w_σ. To test this hypothesis, the optimal test functions generated by the networks *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u are implemented into (142), while approximating the optimal test function w_σ using FEM. The results of this approach are shown in Figure 46, Figure 47, and Figure 49 and prove that it is indeed the approximation error of *DeepONet-1D-*w_σ that is the problem. Using FEM to approximate w_σ instead of using *DeepONet-1D-*w_σ leads results comparable to the original FEM implementation. Therefore it makes sense to try and improve the approximation results of this network, which is the goal of the next section.

Trial functions	FEM w/ three DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.053	0.0534
Piecewise linear	0.00428	0.00428
$C^0(\Omega)$ quadratic	0.000618	0.000618
$C^1(\Omega)$ quadratic	0.000654	0.000652

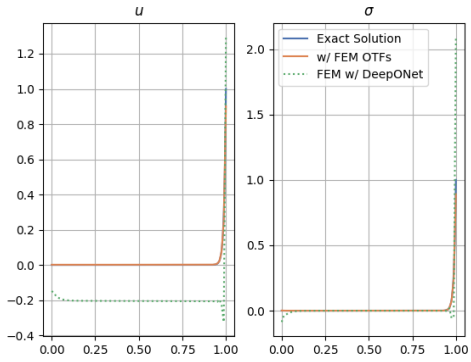
Figure 46: $\|u - u^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, with a discretisation of 80 elements, not using *DeepONet-1D-*w_σ.

Trial functions	FEM w/ three DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.024	0.024
Piecewise linear	0.00414	0.00414
$C^0(\Omega)$ quadratic	0.000593	0.000592
$C^1(\Omega)$ quadratic	0.000620	0.000619

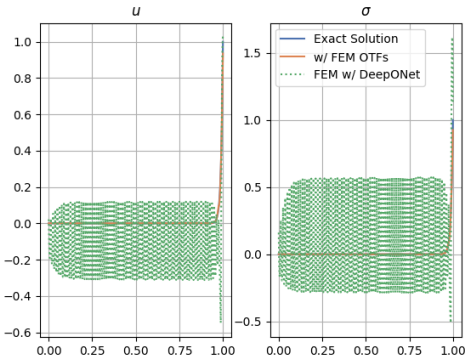
Figure 47: $\|\sigma - \sigma^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, with a discretisation of 80 elements, not using *DeepONet-1D-w_σ*.



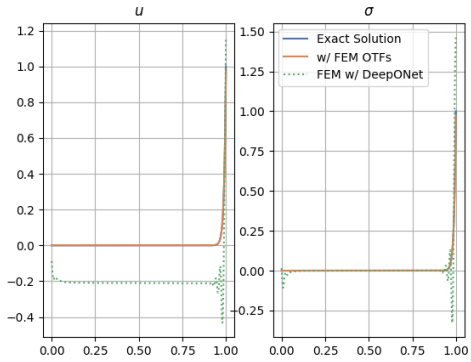
(a) Using piecewise constant trial functions.



(b) Using piecewise linear trial functions.

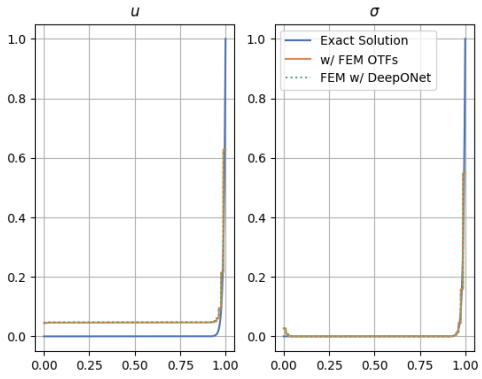


(c) Using $C^0(\Omega)$ quadratic trial functions.

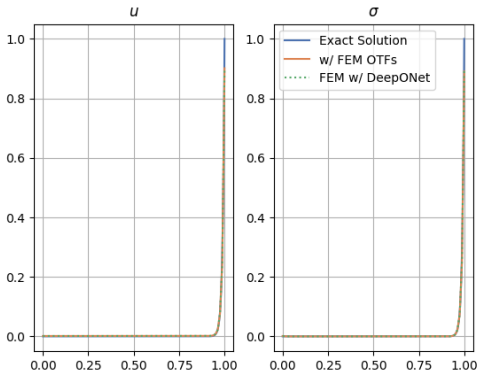


(d) Using $C^1(\Omega)$ quadratic trial functions.

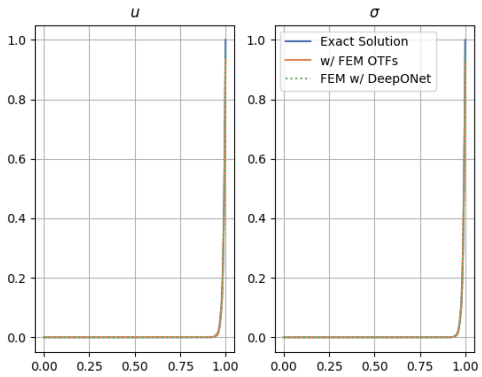
Figure 48: Comparing the exact solution of (135) with the implementation of (142) using FEM generated optimal test functions and using all four DeepONets to generate the optimal test functions as shown in Figure 40, Figure 41, Figure 42, and Figure 42, with $f = 0$ and $\epsilon = 0.01$, using a discretisation of 80 elements.



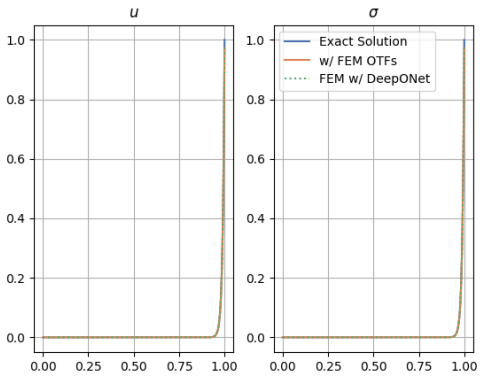
(a) Using piecewise constant trial functions.



(b) Using piecewise linear trial functions.



(c) Using $C^0(\Omega)$ quadratic trial functions.



(d) Using $C^1(\Omega)$ quadratic trial functions.

Figure 49: Comparing the exact solution of (135) with the implementation of (142) using FEM generated optimal test functions and using *DeepONet-1D-*v_σ, *DeepONet-1D-*w_u, and *DeepONet-1D-*v_u to generate the optimal test functions as shown in Figure 40, Figure 41, Figure 42, and Figure 42, with $f = 0$ and $\epsilon = 0.01$, using a discretisation of 80 elements. If you compare these plots to the ones shown in Figure 49, it becomes clear that the approximation error of *DeepONet-1D-*w_σ is the root cause of the deterioration of the finite element solution associated with using the four DeepONets.

5.2.5 Training a New DeepONet

In an attempt to improve the approximation error of w_σ, a new DeepONet is trained. The network is trained in the same way as the *DeepONet-1D-*w_σ network, but uses a sine activation function in all layers where the ReLU activation function was used, which is why it is called *DeepONet-1D-*w_σ*-sin*. Besides using a different activation function, the new network is trained for 150 epochs (instead of 50). Figure 50 shows the MSE loss during training of *DeepONet-1D-*w_σ*-sin*. Comparing

this plot to the convergence of the loss of the *DeepONet-1D-w_σ* network from the previous section (see Figure 39 a)), it might seem that the new network does not do a much better job. However, when looking at its approximations of the optimal test functions shown in Figure 51, it becomes clear that the new network is a big improvement.

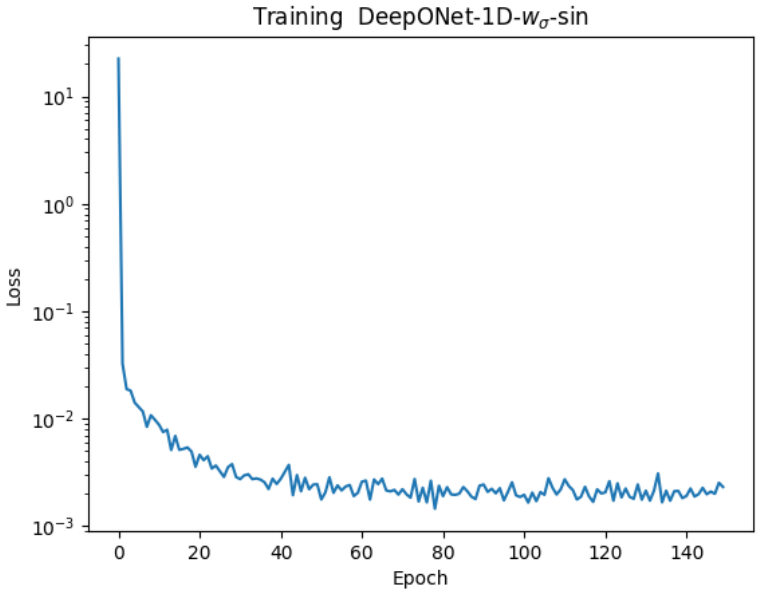


Figure 50: Convergence of MSE loss during training of DeepONet *DeepONet-1D-w_σ-sin*.

The results of implementing the new *DeepONet-1D-w_σ-sin* network together with *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u*, are shown in Figure 52, 53, and 54. While the newly trained DeepONet does a much better job at approximating the optimal test functions, the improvement is not good enough. Using the newly trained network is an improvement over the original *DeepONet-1D-w_σ* network, but the results are still nowhere near the original implementation using FEM generated optimal test functions. Using the DeepONets to approximate the optimal test functions leads to much larger errors for both u and σ. The only case where the error is somewhat close to the original implementation is for the piecewise trial functions. For those trial functions the approximation of σ is reasonably accurate, as shown in 53 and which is again confirmed in Figure 54.

In the previous section it was shown that by using FEM to approximate w_σ and using the three networks *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u* to approximate the rest of the trial functions, results similar to the fully FEM generated optimal test function approach could be achieved. To isolate the impact of the approximation error of *DeepONet-1D-w_σ-sin* on the finite element implementation, a new test is run. Now every optimal test function is generated using FEM except for the optimal test function w_σ, which is approximated by the neural network *DeepONet-1D-w_σ-sin*. The results are shown in Figure 55.

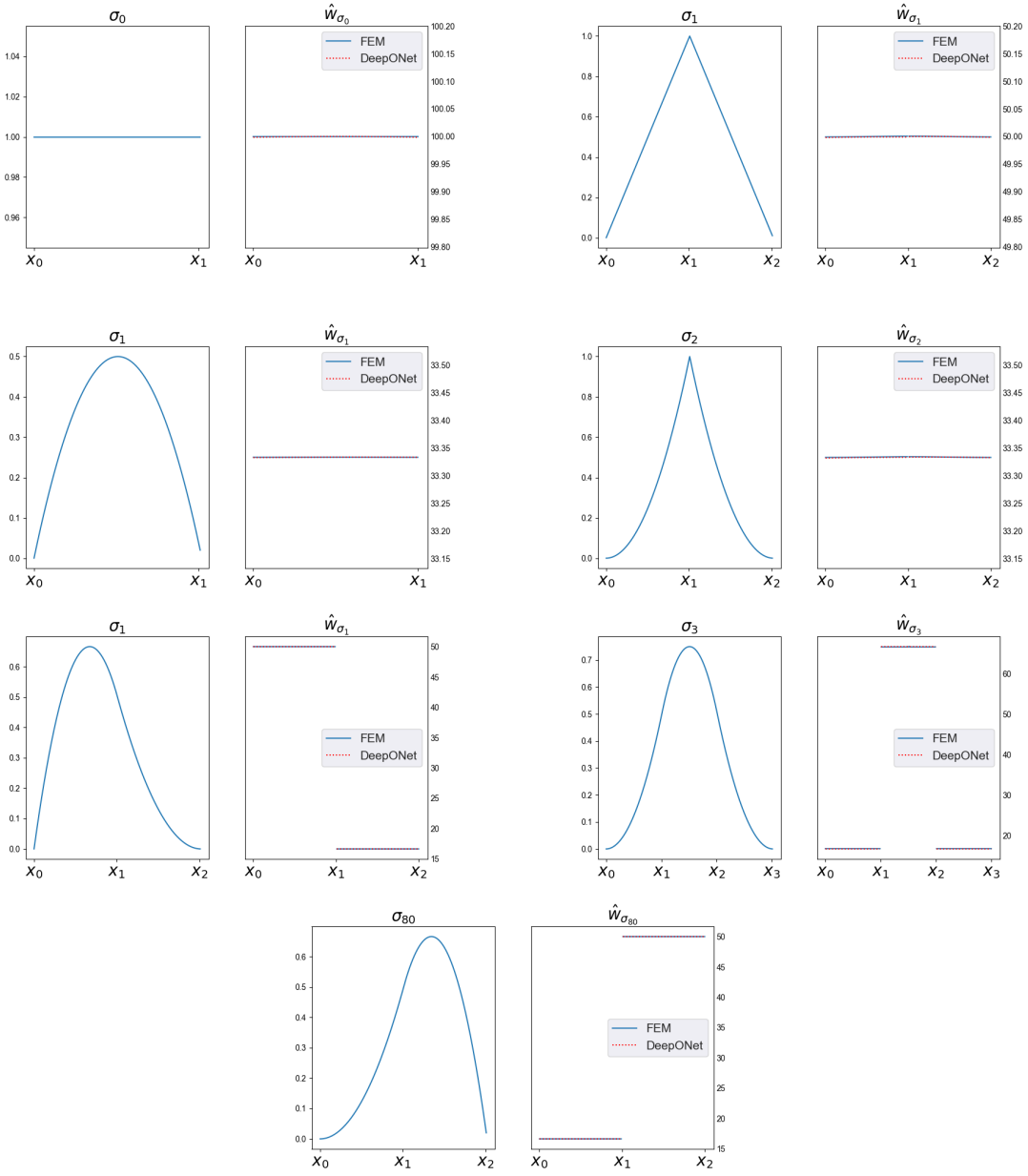


Figure 51: FEM vs newly trained *DeepONet-1D-w_σ-sin* approximation of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for piecewise constant, piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions.

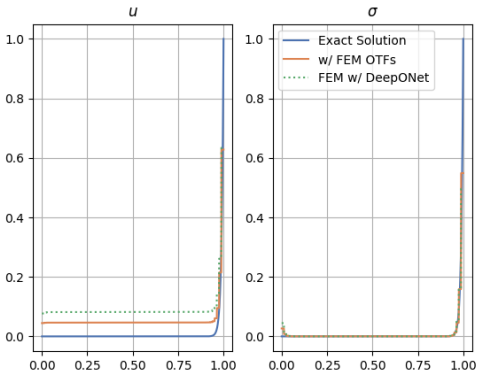
Trial functions	FEM w/ four DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.086	0.0534
Piecewise linear	0.0583	0.00428
$C^0(\Omega)$ quadratic	0.0255	0.000618
$C^1(\Omega)$ quadratic	0.0356	0.000652

Figure 52: $\|u-u^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, for a discretisation using 80 elements, using the newly trained *DeepONet-1D-w_σ-sin* and the three networks that were trained in the previous section, *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u*.

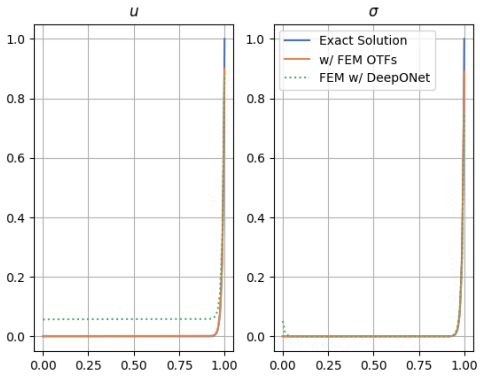
Trial functions	FEM w/ four DeepONets	FEM w/ FEM generated OTFs
Piecewise constant	0.0257	0.0241
Piecewise linear	0.0115	0.00414
$C^0(\Omega)$ quadratic	0.02751	0.000592
$C^1(\Omega)$ quadratic	0.00692	0.000619

Figure 53: $\|\sigma-\sigma^h\|_{L^2(\Omega)}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, for a discretisation using 80 elements, using the newly trained *DeepONet-1D-w_σ-sin* and the three networks that were trained in the previous section, *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u*.

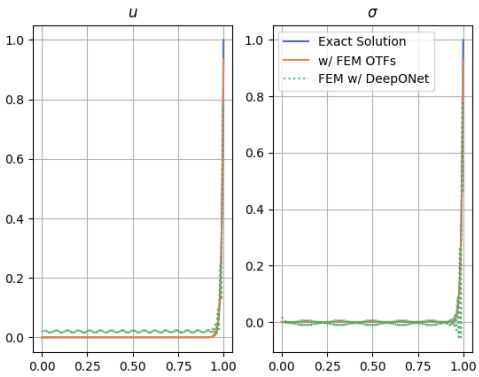
Although replacing the approximations of *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u* with finite element generated optimal test functions does lead to better results (compare results of the $C^0(\Omega)$ quadratic trial functions in Figure 54 with Figure 55), the results are not nearly as good as the original implementation that uses only finite element generated optimal test functions. As was mentioned *DeepONet-1D-w_σ* is now the only neural network that is used in the implementation. The fact that a small error in a single optimal test function can cause such a big difference in the finite element method resembles the results that were presented in the non-mixed setting of the 1D section. This time however, the diffence between the FEM approximation of w_σ and the *DeepONet-w_σ-sin* approximation appears to be even smaller than the approximation errors that were seen in the non-mixed case.
It makes sense to further investigate the *DeepONet-1D-w_σ-sin* approximation that was shown in Figure 51. Figure 56 and 57 show an examination of the *DeepONet-1D-w_σ* approximation of the discontinuous pieces, while zooming in considerably. The network is very accurate (notice the very small rate of change on the y-axis), but is unable to correctly approximate the shape of the optimal test functions. It appears this small difference between the DeepONet output and the finite element approximations of the optimal test functions are causing big differences in the corresponding finite element implementations.
In the next section an experiment will be run where a small perturbation that is of similar magnitude as the approximation errors of *DeepONet-1D-w_σ* is added to the FEM approximtated optimal test function w_σ. It will be shown that adding such a small perturbation has similar effects on the finite element solution as using *DeepONet-1D-w_σ* had.



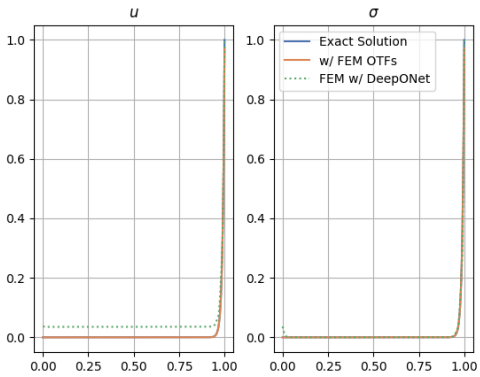
(a) Using piecewise constant trial functions.



(b) Using piecewise linear trial functions.

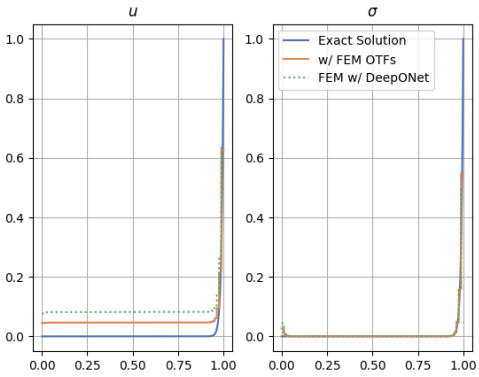


(c) Using $C^0(\Omega)$ quadratic trial functions.

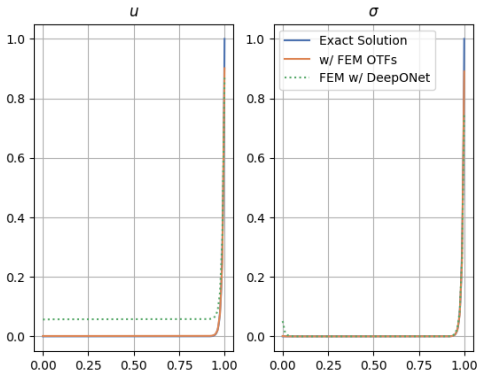


(d) Using $C^1(\Omega)$ quadratic trial functions.

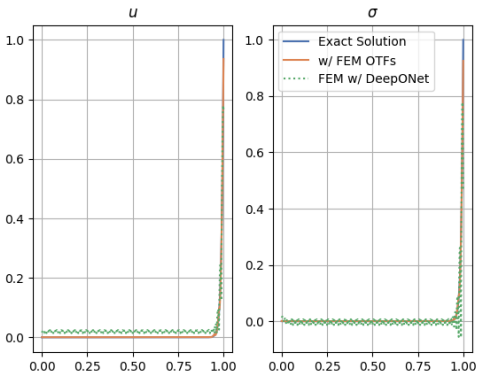
Figure 54: Comparing the exact solution of (135) with the implementation of (142) using FEM generated optimal test functions and using the newly trained *DeepONet-1D-w_σ-sin*, and the three previously trained networks *DeepONet-1D-v_σ*, *DeepONet-1D-w_u*, and *DeepONet-1D-v_u* to generate the optimal test functions as shown in Figure 40, Figure 41, Figure 42, and Figure 42, for $f = 0$ and $\epsilon = 0.01$, using a discretisation of 80 elements.



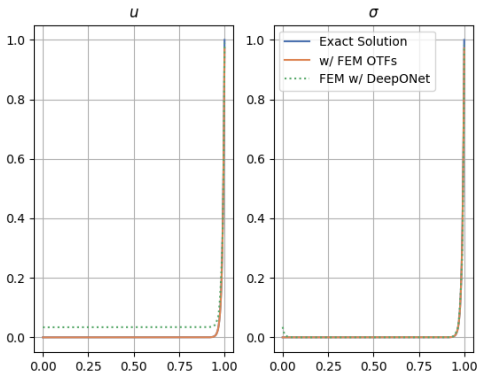
(a) Using piecewise constant trial functions.



(b) Using piecewise linear trial functions.



(c) Using $C^0(\Omega)$ quadratic trial functions.



(d) Using $C^1(\Omega)$ quadratic trial functions.

Figure 55: Comparing the exact solution of (135) with the implementation of (142) using FEM generated optimal test functions and using the newly trained *DeepONet-1D-w_σ-sin* to approximate w_σ as shown on the left side of Figure 40 and Figure 41, while generating all the other optimal test functions using FEM, for $f = 0$ and $\epsilon = 0.01$, using a discretisation of 80 elements. The approximation error of *DeepONet-w_σ* may be small, but its impact on the finite element solution is very big.

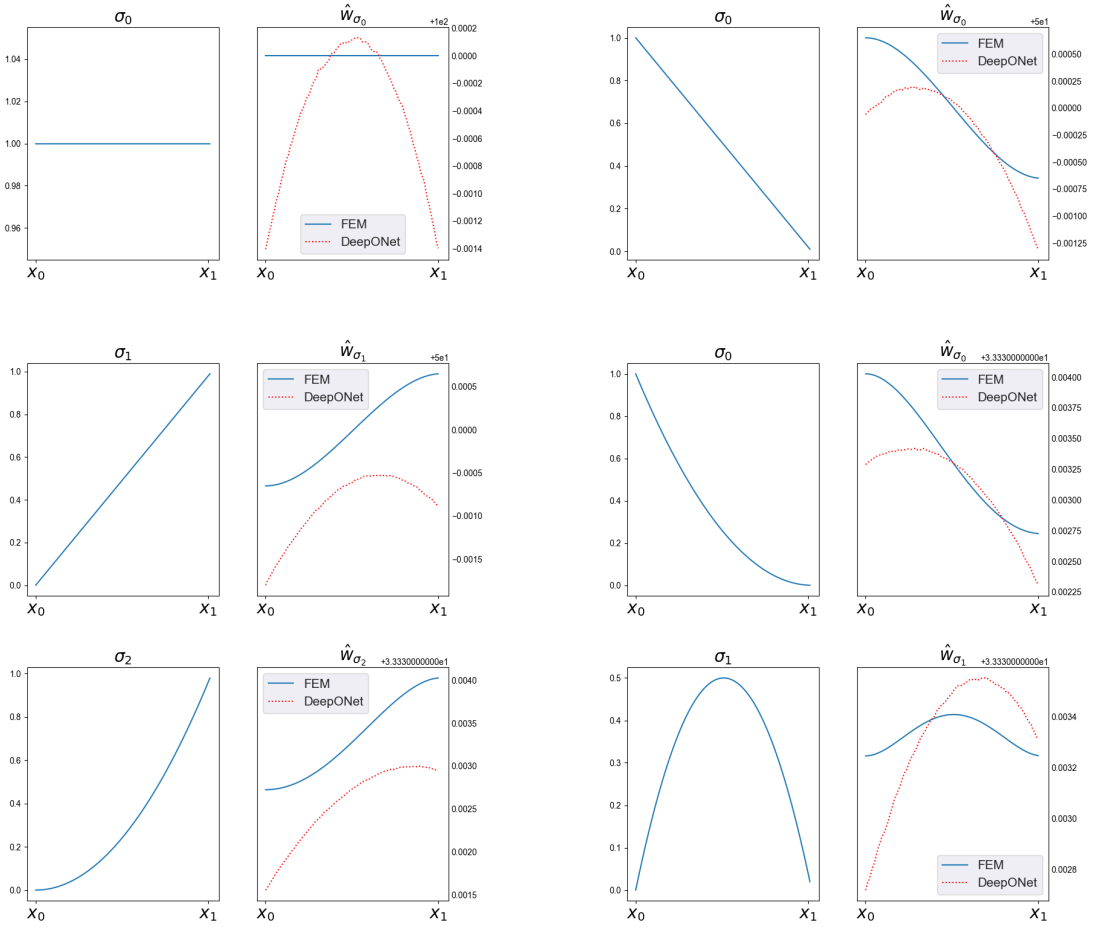


Figure 56: FEM vs newly trained *DeepONet-1D-w_σ-sin* approximation of discontinuous pieces of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for piecewise constant, piecewise linear, and $C^0(\Omega)$ quadratic trial functions.

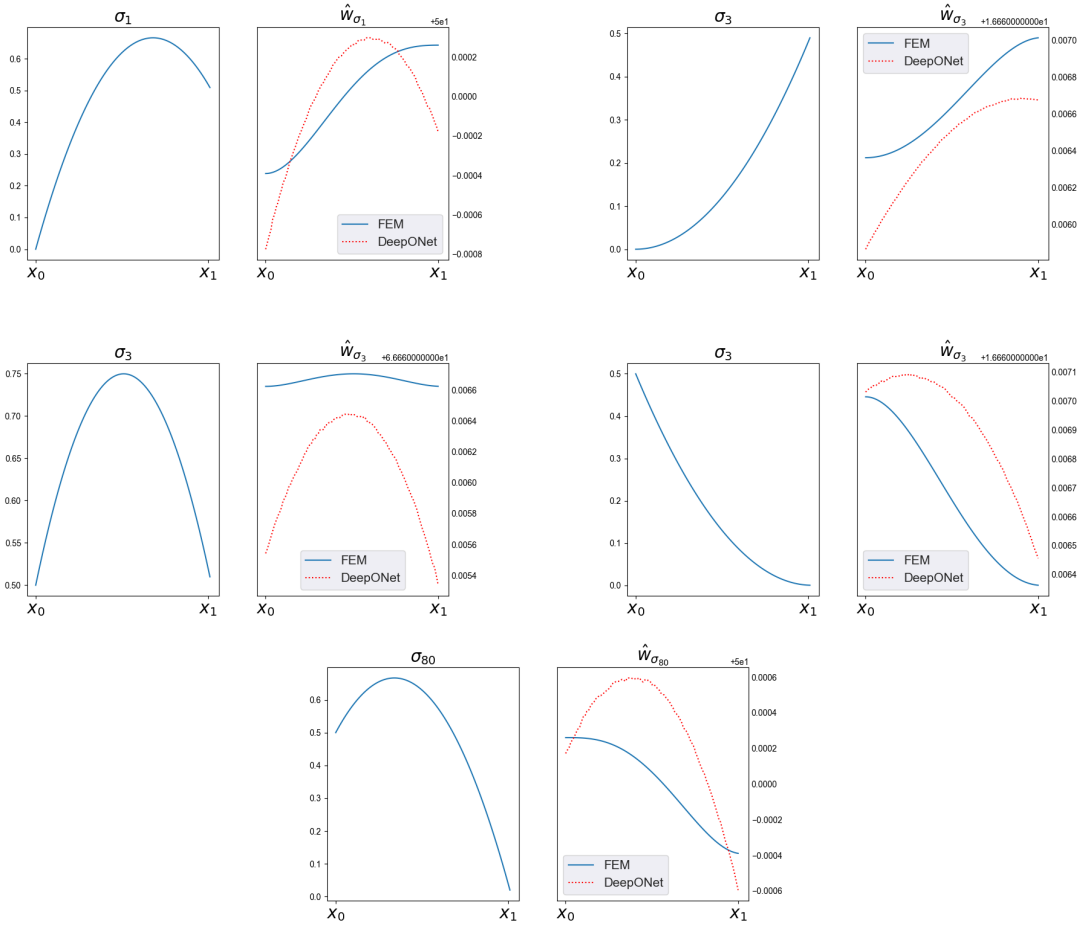


Figure 57: FEM vs newly trained *DeepONet-1D-w_σ-sin* approximation of discontinuous pieces of the optimal test functions in (147) corresponding to a discretisation of 80 elements and $\epsilon = 0.01$, for $C^1(\Omega)$ quadratic trial functions.

5.2.6 Adding Small Perturbations to Optimal Test Function w_σ

As a final test the original DPG [2] is implemented while adding a small perturbation to the optimal test function w_σ. The perturbations are sampled randomly from the interval $[-2 * E, 2 * E]$, where E is the mean of the absolute value of the approximation errors of *DeepONet-1D-w_σ-sin* at the quadrature points. The approximation errors for the different types of trial functions are shown in Figure 58. The results of implementing the DPG method using this perturbation to w_σ are shown in Figure 59, Figure 60, and Figure 61.

Trial functions	E
Piecewise constant	0.0006458
Piecewise linear	0.0006473
$C^0(\Omega)$ quadratic	0.0003618
$C^1(\Omega)$ quadratic	0.0002595

Figure 58: MSE at the quadrature points of *DeepONet-1D-w_σ-sin* when compared to a FEM approximation, corresponding to different trial functions.

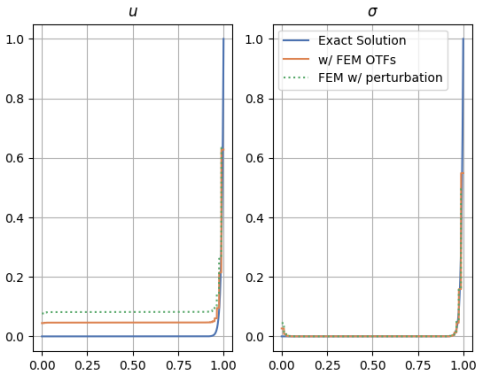
Trial functions	FEM w/ perturbation	FEM w/ FEM generated OTFs
Piecewise constant	0.0856	0.0534
Piecewise linear	0.13176	0.00428
$C^0(\Omega)$ quadratic	0.0254	0.000618
$C^1(\Omega)$ quadratic	0.0343	0.000652

Figure 59: $\|u - u^h\|_{L^2}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, for a discretisation using 80 elements, using finite element generated optimal test functions while adding a small perturbation to w_σ.

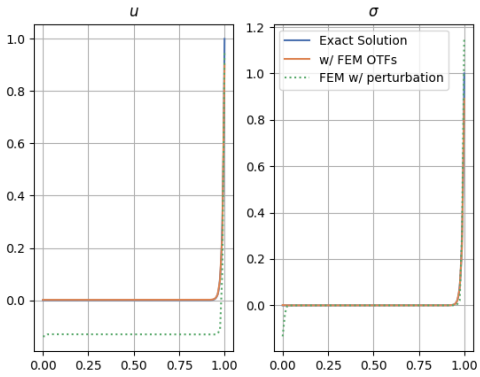
Trial functions	FEM w/ perturbation	FEM w/ FEM generated OTFs
Piecewise constant	0.0257	0.0241
Piecewise linear	0.0219	0.00414
$C^0(\Omega)$ quadratic	0.02782	0.000592
$C^1(\Omega)$ quadratic	0.006184	0.000619

Figure 60: $\|\sigma - \sigma^h\|_{L^2}$ error comparison, corresponding to (142) with $f = 0$ and $\epsilon = 0.01$, for a discretisation using 80 elements, using finite element generated optimal test function while adding a small perturbation to w_σ.

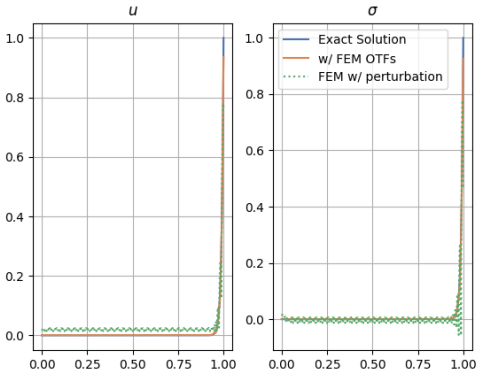
The perturbation causes the approximate solution to become much worse, similarly to the *DeepONet-w_σ* approximation error. Given the fact that some of the optimal test functions (we have found at least one) are so sensitive to small perturbations probably means that the DPG method in this form doesn't lend itself well to the use of machine learning. Neural networks like the DeepONet that use supervised learning will never be, at least intentionally, as accurate as the finite element method. Other networks like the VPINN of WAN are not suitable (especially in this DPG case where the optimal test functions are allowed to be discontinuous), because those architectures are able to approximate the optimal test function corresponding to a single trial function at a time. An argument can be made that those networks would have been applicable in the previous section as the number of unique optimal test function was relatively low. In the DPG case however, many different VPINNs or WANs would have to be trained to approximate all the discontinuous optimal test function pieces.



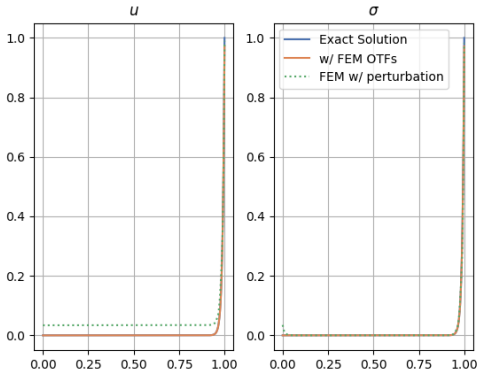
(a) Using piecewise constant trial functions.



(b) Using piecewise linear trial functions.



(c) Using $C^0(\Omega)$ quadratic trial functions.



(d) Using $C^1(\Omega)$ quadratic trial functions.

Figure 61: Comparing the exact solution of (135) with the implementation of (142) using the correct finite generated optimal test functions and using finite element generated optimal test functions while adding a small perturbation to w_σ, for $f = 0$ and $\epsilon = 0.01$, using a discretisation of 80 elements.

5.3 Non-mixed Weak Formulation in 2D

5.3.1 Problem Definition

Let $\Omega = [0,1] \times [0,1]$ and let $\Gamma = \partial\Omega$. To test the 2D implementation of optimal test function the following two dimensional advection-diffusion problem will be considered

$$\begin{aligned} &\text{Find } u \text{ such that:} \\ &\begin{cases} -\nabla \cdot (\epsilon \nabla u) + \mathbf{b} \cdot \nabla u = f, & \text{in } \Omega \\ u = 0, & \text{on } \Gamma \end{cases} \end{aligned} \tag{153}$$

where ϵ denotes the diffusion coefficient and $\mathbf{b}$ is the advection coefficient. As was the case in the 1D setting, only ϵ will be used to change the Peclet number, and the advection will be set to $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

5.3.2 Weak Formulation

To approximate the solution to this problem, the following weak form will be used

$$\begin{aligned} &\text{Find } u \in U \text{ such that} \\ &\sum_{K_m \in \mathcal{P}_h} \epsilon \int_{K_m} \nabla u \cdot \nabla v + \mathbf{b} \cdot \nabla u \, v = \sum_{K_m \in \mathcal{P}_h} \int_{K_m} f v \quad \forall v \in V \end{aligned} \tag{W1}$$

where $\mathcal{P}_h$ is the partition of Ω into square subdomains $K_1, K_2, ..., K_N$ and where the trial and test space are defined as $U = H^1(\Omega)$ and $V = H^1(\Omega)$. By introducing the following notation

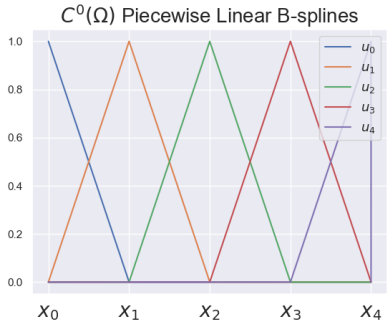
$$\begin{aligned} b(u,v) &= \sum_{K_m \in \mathcal{P}_h} \epsilon \int_{K_m} \nabla u \cdot \nabla v \; + \; \mathbf{b} \cdot \nabla u \, v \\ l(v) &= \sum_{K_m \in \mathcal{P}_h} \int_{K_m} f v \end{aligned} \tag{154}$$

The weak formulation can be written as follows:

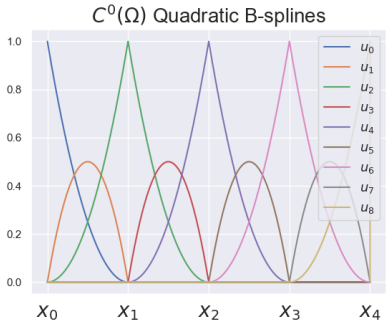
$$\begin{aligned} &\text{Find } u \in U \text{ such that} \\ &\qquad b(u,v) = l(v), \quad \forall v \in V \end{aligned} \tag{155}$$

5.3.3 Finite Element Implementation

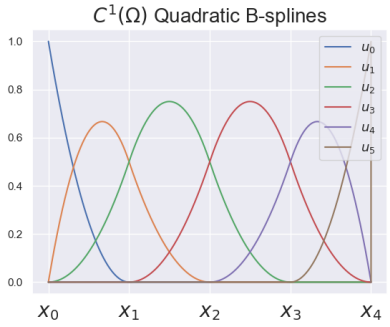
For the 2D problem two dimensional shape functions are needed. Here the shape functions that will be used as trial and test functions are the tensor product of the B-splines that were used in the 1D sections, shown in Figure 62 for a four element discretisation.



(a)



(b)



(c)

Figure 62: The one-dimensional B-splines that were used as trial and test functions in the 1D finite element implementation.

The two dimensional shape functions will be denoted as $u_{i,j}(x, y)$, where the i and j subscript indicate which 1D B-splines were used to build the 2D version. For example, the two dimensional shape function that is the product of the piecewise linear B-splines $u_1(x)$ and $u_1(y)$ is shown in Figure 63 and will be denoted as $u_{1,1}(x, y)$. Each piecewise linear B-spline is non-zero on two elements, and the new two-dimensional shape function will therefore be non-zero on four elements. The 2D shape functions that are the tensor product of the $C^0(\Omega)$ and $C^1(\Omega)$ quadratic B-splines are shown in Figure 64 and Figure 65 respectively. Here the two-dimensional shape functions that use either the first or last B-spline corresponding to each class of functions are not shown. As the boundary conditions in this weak form will be strongly enforced these shape functions are not used in the finite element scheme. One thing to note is that in all the plots in this

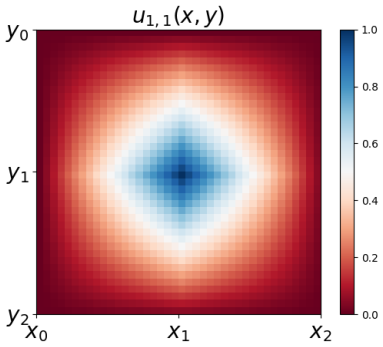
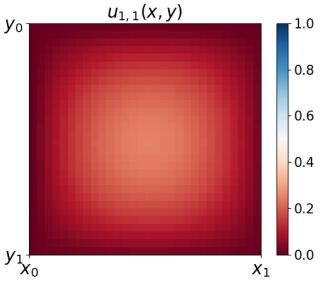
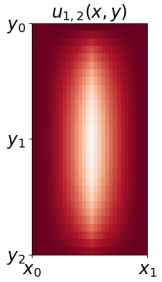


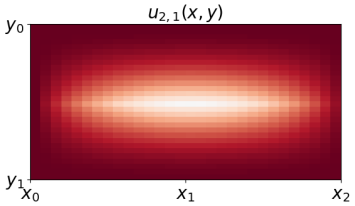
Figure 63: 2D shape function $u_{1,1}$ that is product of piecewise linear B-splines $u_1(x)$ and $u_1(y)$. Note that the y-axis is inverted.



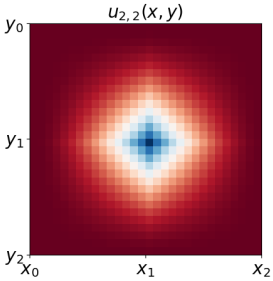
(a) 2D shape function $u_{1,1}$ that is the product of two 1D shape functions $u_1(x)$ and $u_1(y)$.



(b) 2D shape function $u_{1,2}$ that is the product of two 1D shape functions $u_1(x)$ and $u_2(y)$.



(c) 2D shape function $u_{2,1}$ that is the product of two 1D shape functions $u_2(x)$ and $u_1(y)$.



(d) 2D shape function $u_{2,2}$ that is the product of two 1D shape functions $u_2(x)$ and $u_2(y)$.

Figure 64: Shape functions that are the product of the quadratic $C^0(\Omega)$ B-splines show in Figure 62. Note that the first and last B-splines are not used.

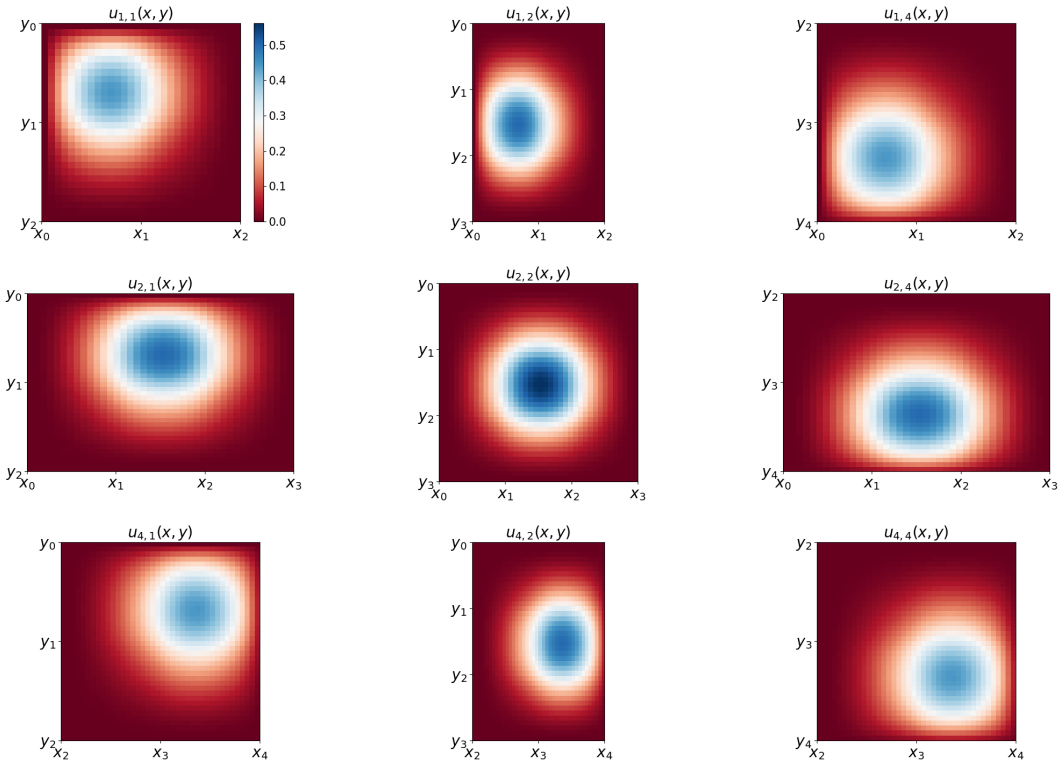
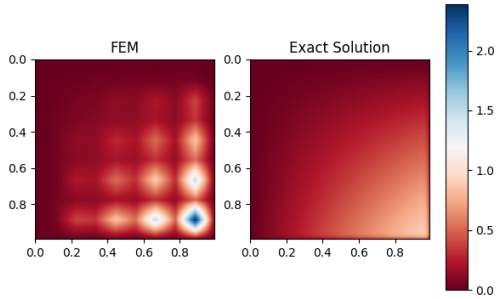


Figure 65: Shape functions that are the product of the quadratic $C^1(\Omega)$ B-splines show in Figure 62. The first and last B-splines are not used in the finite element implementations in this section.

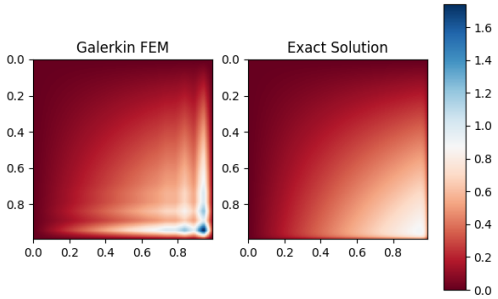
two dimensional setting the y-axis is reversed, meaning that the origin sits in the top left corner of every plot. Since the advection coefficient is set to $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ in this section, that means that the flow points from the upper left corner to the lower right corner.
Figure 66 shows an implementation of the Galerkin method resulting from (155) for $\epsilon = 0.01$ using test and trial functions that are the tensor products of the piecewise linear B-splines shown in Figure 63, using a discretisation of 100 elements, and by choosing the source term f such that the solution is given by

$$u(x,y) = \left[x + \frac{e^{Pe \cdot x} - 1}{1 - e^{Pe}} \right] \left[y + \frac{e^{Pe \cdot y} - 1}{1 - e^{Pe}} \right] \tag{156}$$

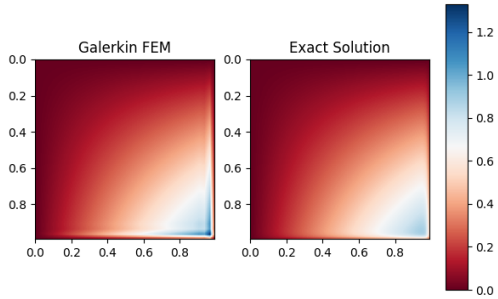
Setting $\epsilon = 0.01$ means that the problem is advection dominated. Similarly to the 1D setting, the finite element solution in the 2D problem starts oscillating as it approaches the steep gradient on the lower right corner of the plot. It's clear that the method resulting from (155) is well defined and leads to a stable scheme. As the number of elements used in the discretisation is increased the oscillations in the approximate solution become less and less, as the method converges to the exact solution.



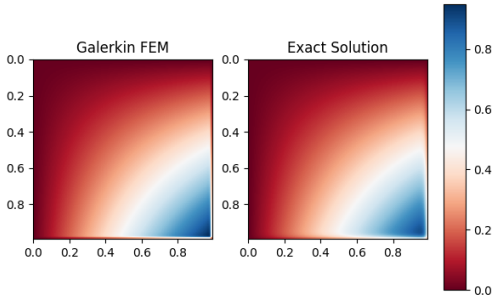
(a) 10×10 element discretisation.



(b) 20×20 element discretisation.



(c) 35×35 element discretisation.



(d) 50×50 element discretisation.

Figure 66: Implementation of the Galerkin finite element method resulting from (155) using test and trial function that are shown in 63, for $\epsilon = 0.01$, where the source term f is chosen such that the solution is given by (156).

5.3.4 Globally Continuous Optimal Test Functions

To derive the optimal test functions corresponding to weak formulation (155) the following problem needs to be solved

$$b(u, v) = (Tu, v)_V, \quad \forall v \in V \tag{157}$$

where Tu denotes the optimal test function corresponding to u, $b(u, v)$ denotes the bilinear form in (154), and $(\cdot, \cdot)_V$ denotes the inner product of the test space V. In this example, the mesh dependent H^1 inner product will be used, e.g.,

$$(v, w)_V = \int_\Omega (h^2 \nabla v \cdot \nabla w + v\, w) \tag{158}$$

Here, h is equal to the diagonal of the elements used in the discretisation. To find the optimal test function $v_{u_{i,j}}$ corresponding to trial function $u_{i,j}$, the following problem needs to be solved

$$\begin{cases} \text{Find } v_{u_{i,j}}, \text{ such that} \\ \epsilon \int_{S_{u_{i,j}}} (\nabla u_{i,j} \cdot \nabla \delta_v + \mathbf{b} \cdot \nabla u_{i,j}\, \delta_v) = \int_{S_{u_{i,j}}} (h^2\, \nabla v_{u_{i,j}} \cdot \nabla \delta_v + v_{u_{i,j}}\, \delta_v), \quad \text{in } S_{u_{i,j}} \\ v_{u_{i,j}} = 0, \quad \text{on } \partial S_{u_{i,j}} \end{cases} \tag{159}$$

where $S_{u_{i,j}}$ denotes the union of elements on which $u_{i,j}$ has non-zero support. The homogeneous boundary condition on $\partial S_{u_{i,j}}$ is required to ensure that the optimal test functions are in $H^1(\Omega)$. Figure 67 shows the optimal test function corresponding to the trial function that is the tensor product of piecewise linear B-splines (shown in Figure 63).

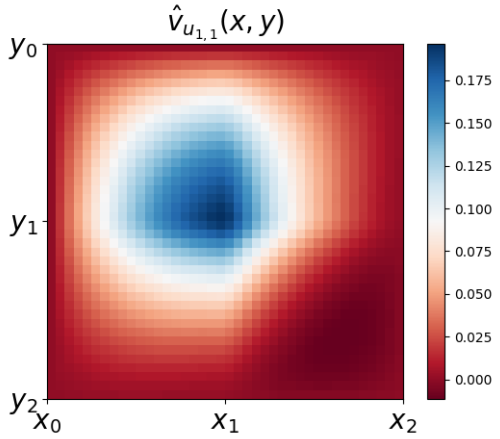


Figure 67: Globally continuous optimal test function defined by (159), with $\epsilon = 0.1$ and a discretisation of 4×4 elements, corresponding to the shape functions that are the tensor product of $C^0(\Omega)$ piecewise linear B-splines.

The optimal test function shows a clear upwinding effect, placing more weight on the upwind part of the element (again, $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ and the origin sits on the top left of the plot). The optimal test functions corresponding to the shape functions that are generated by the $C^0(\Omega)$ and $C^1(\Omega)$ quadratic B-splines are shown in Figure 68 and Figure 69 respectively. The upwind effect is not as easy to spot for every one of these trial functions, due to the fact that some of the trial functions themselves place more weight on the downwind part of the element. Consider for example the optimal test functions that are shown in either the last row or column of Figure 68. These optimal test functions correspond to trial functions that use the second-last 1D B-splines on either the x or y axis (or both). Since these B-splines place a lot more of their weight on the downwind part of the elements, there appears to be no upwind effect.
For the optimal test functions that do not correspond to trial functions with a lot of weight downwind, the upwind effect is more easily recognisable. Take the optimal test function shown in the top right corner of Figure 68, e.g., $\hat{v}_{u_{1,2}}(x, y)$, corresponding to the trial function that is the product of the $C^0(\Omega)$ quadratic trial functions $u_1(x)$ and $u_2(y)$. There is slightly more weight on the upper

and left side of the plot. What will be interesting to see in later sections is that as the Péclet number gets larger, the upwind effect will become stronger, as the need for stabilization increases.

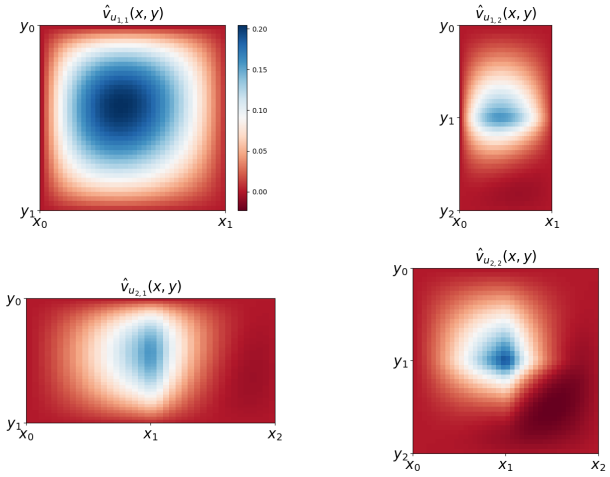


Figure 68: Optimal test functions defined by (159) for $\epsilon = 0.1$ with a discretisation of 4×4 elements, corresponding to shape functions that are the tensor product of the quadratic $C^0(\Omega)$ B-splines, shown in Figure 64.

Figure 70 shows a comparison between finite element methods that use optimal test functions and finite element methods that do not. It is clear that implementing the optimal test functions into the finite element method leads to a much smaller error in the H^1 norm. The difference between the methods that use optimal test functions is particularly visible for problems with a large peclet number. While the difference between the Galerkin implementation and the optimal test function implementation is not as big in the left plot of 70, the plot on the right which corresponding to $\epsilon = 0.01$ shows that using optimal test functions leads to a big improvement in the approximation.

In the previous 1D sections it was found that there exist weak formulations for which the optimal test function approximations need to be very accurate to improve the error in the finite element implementation. Small perturbations to the optimal test functions could had extreme ramifications for the approximate finite element solutions. In the next few subsections it will be shown that neural networks can be successfully implemented into the finite element method, can greatly improve the approximation error of the finite element method, and can stabilize solutions. In fact, it will be shown that by passing ϵ as a variable to a trained DeepONet, a single neural network can improve the finite element implementation corresponding to many differrent values of ϵ.

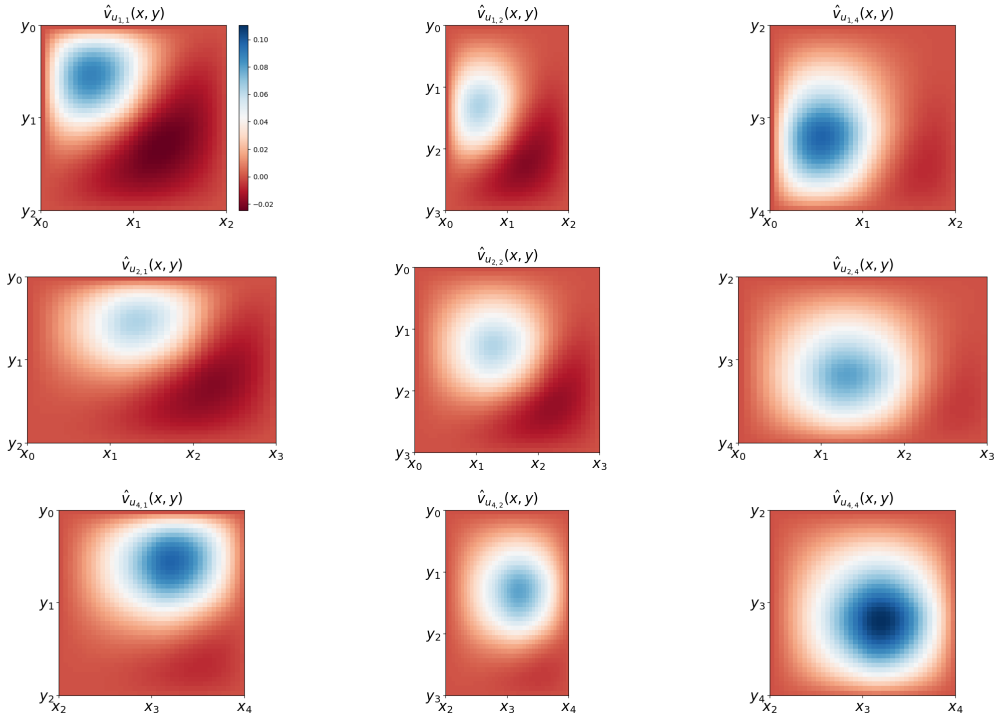
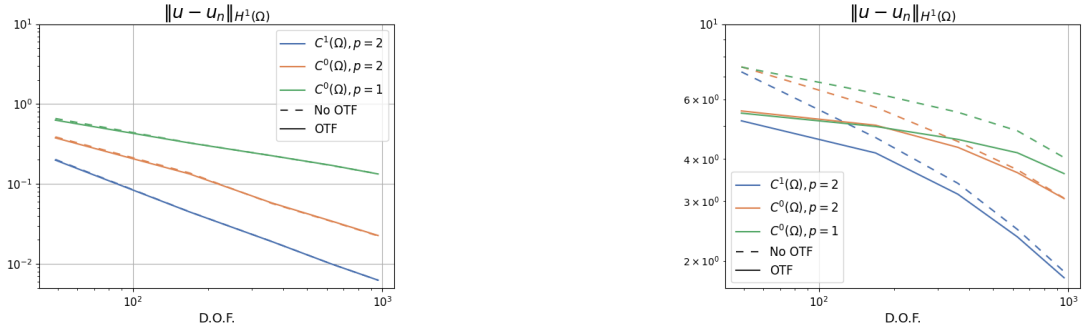


Figure 69: Optimal test functions defined by (159) for $\epsilon = 0.1$ with a discretisation of 4×4 elements, corresponding to shape functions that are the tensor product of the quadratic $C^1(\Omega)$ B-splines shown in Figure 65.



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.01$

Figure 70: Convergence of the error $\|u - u_n\|_{H^1(\Omega)}$ of the approximate solution to (155), with and without using optimal test functions, for different values of ϵ.

5.3.5 Training the DeepONet

In the rest of this chapter, several DeepONets will be trained. The DeepONet that will be trained in this subsection is called *DeepONet-2D* and can be used to approximate the optimal test functions defined in (159) for a discretisation of 4×4 elements and a diffusion coefficient of $\epsilon = 0.1$. Since the DeepONets will have to deal with two dimensional input functions the DeepONets will be made wider. Specifically, the branch network will use two layers that have a width of 300 and 200 neurons, and the trunk network will use three layers that have width 100, 200, and 300. These configurations are based on the DeepONets that are used in [**?**], that also deal with two dimensional input functions. Similarly to the DeepONets used in the 1D section, the ReLU activation function will be used. To train the *DeepONet-2D* a dataset is generated by repeatedly approximating the solution $v_{u_{i,j}}(x, y)$ to (159) for trial function $u_{i,j}$ at random point (x, y) for each of the 14 trial functions shown in Figure 63, Figure 64, and Figure 65. The solutions are approximated using the finite element method. Using this approach each point in the training dataset is a triplet that looks like this

$$(\boldsymbol{u}_{i,j}, (x, y), \hat{v}_{u_{i,j}}(x, y)) \tag{160}$$

Here $\boldsymbol{u}_{i,j}$ denotes a vector that holds the values of trial function $u_{i,j}(x, y)$ evaluated at grid points (called sensors) on the element on which it is non-zero.
For *DeepONet-2D*, a grid of 20×20 sensors is used for each trial function. The optimal test functions are approximated using a discretisation of 26×26 elements, using $C^1(\Omega)$ quadratic B-splines for both the trial and test functions. For each of the 14 trial functions the corresponding optimal test functions are approximated at 10.000 randomly sampled points each, meaning that the training dataset consists of 140.000 points in total. The DeepONet is trained by iterating over this dataset with SGD, doing 200 epochs in total, using the mean squared error loss function.
The result of the training in shown in Figure 71. The model appears to converge well and exhibits a very small loss after 200 epochs.

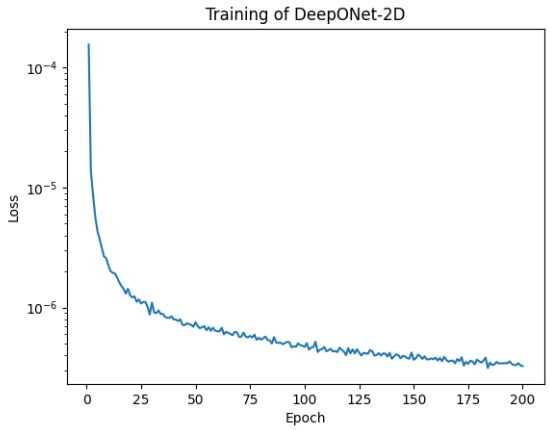


Figure 71: Average MSE loss during training of *DeepONet-2D*.

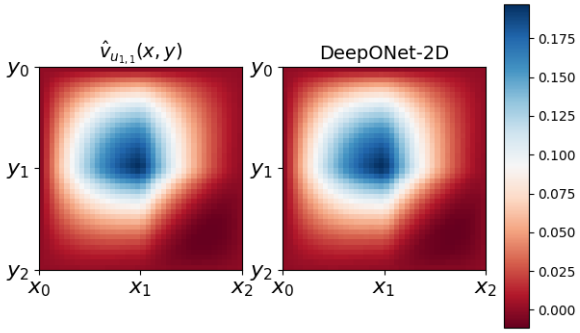


Figure 72: Comparison between FEM approximation and DeepONet approximation of optimal test function $v_{u_{1,1}}(x, y)$, corresponding to trial function $u_{1,1}$ shown in Figure 63.

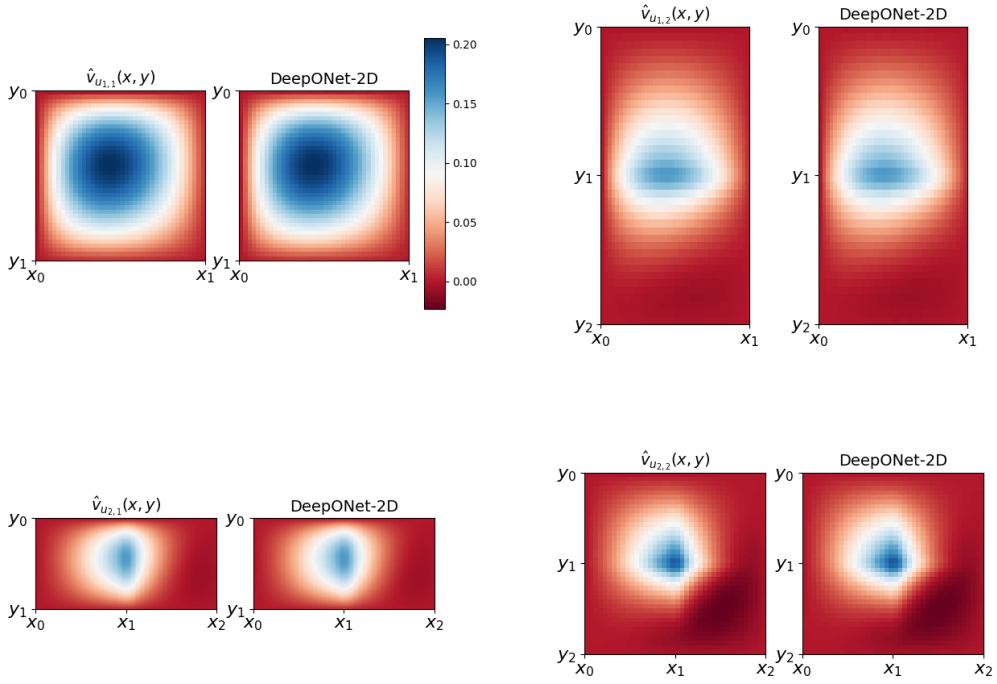


Figure 73: Comparison of FEM approximation and DeepONet approximation of optimal test function corresponding to the trial functions shown in Figure 64.

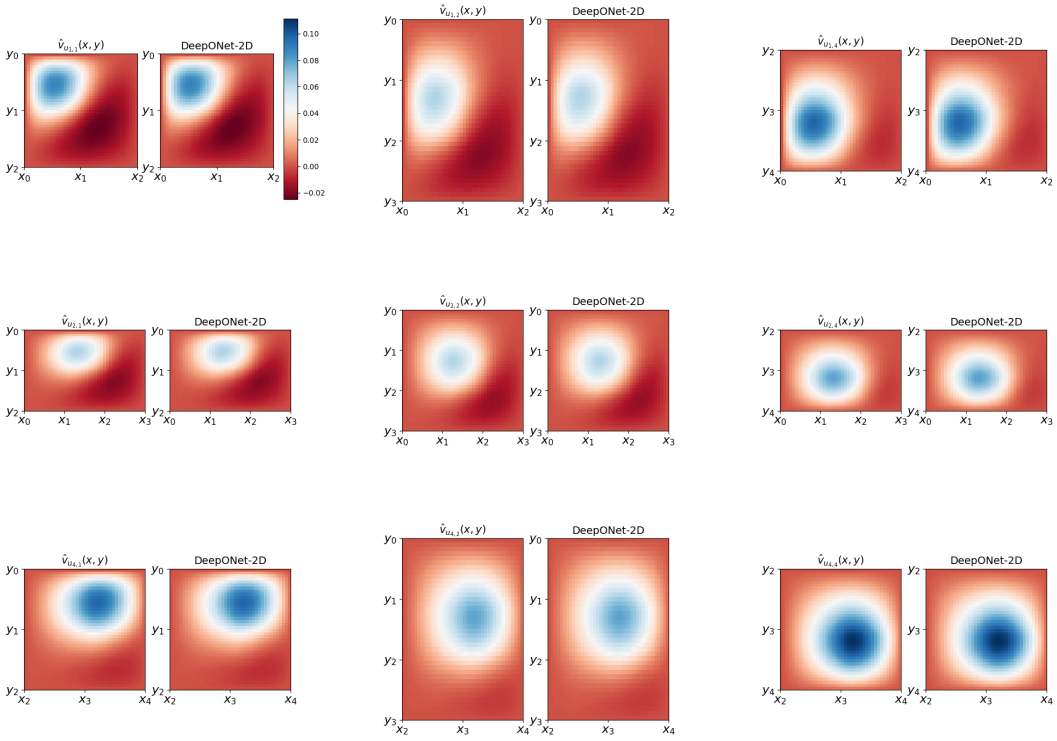


Figure 74: Comparison of FEM approximation and DeepONet approximation of optimal test function corresponding to the trial functions shown in Figure 65.

Figure 72 shows a comparison between the FEM approximation of optimal test function $v_{u_{1,1}}(x, y)$ and the *DeepONet-2D* approximation. The network has converged to the optimal test function and it shows that DeepONets can approximate the optimal test functions very accurately. Figure 73 and Figure 74 show a comparison between the FEM approximation of the optimal test functions corresponding to the trial functions shown in 64 and 65, and the *DeepONet-2D* approximations. Again, the *DeepONet-2D* approximate every optimal test function very accurately. Now that the DeepONet is trained and tested, it can be implemented into the finite element method.

5.3.6 Implementing *DeepONet-2D*

DeepONet-2D is implemented into the finite element method in a similar fashion as the networks from the 1D section. To approximate the gradients of the optimal test functions, the gradient of *DeepONet-2D* is used. Since *DeepONet-2D* was trained in Tensorflow, automatic differentiation is used to calculate the gradients. Figure 75 shows a comparison between the FEM approximated partial derivative $v_{u_{1,1}}(x, y)$ and the derivative of the DeepONet with respect to x.

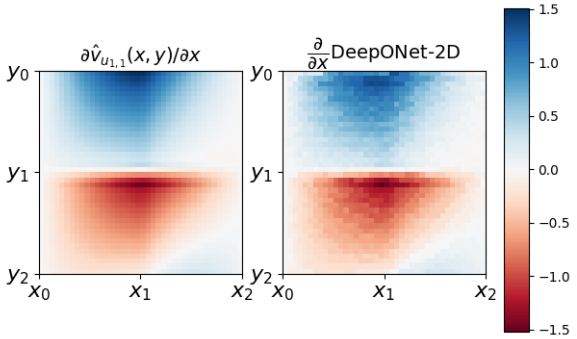
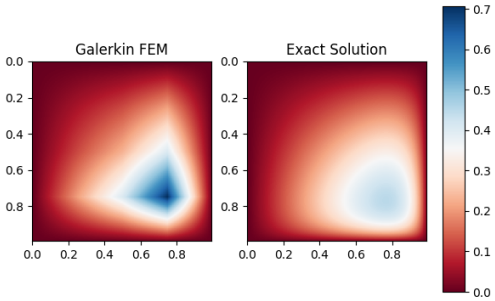


Figure 75: Comparison of FEM approximation of partial derivative of optimal test function $v_{u_{1,1}}$ and the approximation by using the *DeepONet-2D* gradient.

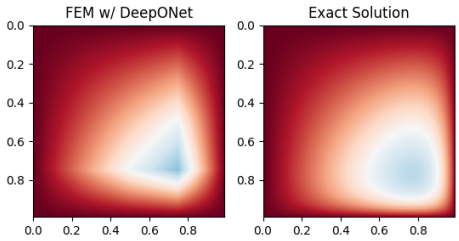
While the DeepONet gradient provides a reasonably accurate approximation, there does exist a considerable difference between the finite element approximation of the gradient and the DeepONet approximation (which is the gradient of *DeepONet-2D*). As was observed in the 1D section, the blocky approximation on the right hand side of 75 is a result of the fact that the ReLU activation function is used in the DeepONet. Recall from the 1D section that the ReLU function and its derivative look like this

$$\begin{aligned} f(x) &= \max(0, x) \\ f'(x) &= \begin{cases} 0, & \text{if} \quad x < 0 \\ 1, & \text{if} \quad x > 0 \end{cases} \end{aligned} \tag{161}$$

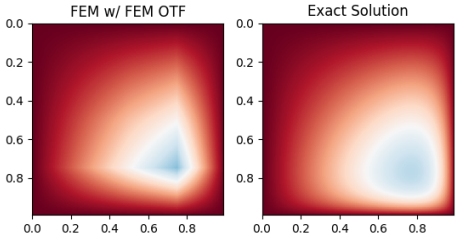
Since the ReLU activation function has a discontinuous derivative, the gradient of *DeepONet-2D* is the sum of discontinuous functions. In the 1D section this was solved for by using separate DeepONets to approximate the derivatives of the optimal test functions.
While approximating the continuous gradient of the optimal test function using the neural network gradient might not ideal, it leads to nice results in this 2D setting. Using *DeepONet-2D* together with its gradient in the finite element implementation of (155) with trial functions that are the products of C^0 piecewise linear trial functions as shown in 63, leads to an improvement in the finite element approximation. The results are shown in Figure 76. The methods using optimal test functions (either FEM or *DeepONet-2D* generated) lead to a significant improvement in the error measured in the $\|\cdot\|_{H^1(\Omega)}$ norm. These results are somewhat surprising considering the fact that the implementations used in 1D did not respond well to perturbations in either the optimal test functions or the optimal test functions' derivatives. It is true that the *DeepONet-2D* approximates the optimal test functions very accurately, but based on the previous experiments it would have seemed likely that the gradient approximation shown in Figure 75 would not be good enough. It seems that the importance of accuracy when implementing the optimal test functions depends on the corresponding weak formulation.



(a) Using test functions that are tensor product of $C^0(\Omega)$ B-splines as shown in 63, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.891$.



(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.827$.

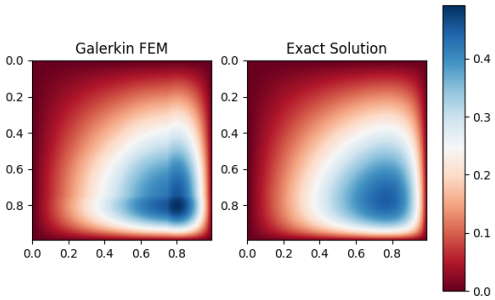


(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.826$.

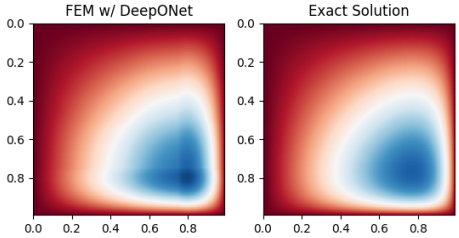
Figure 76: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 4×4 elements for $\epsilon = 0.1$, with trial functions that are the tensor product of the $C^0(\Omega)$ piecewise-linear B-splines as shown in 63.

Implementing the *DeepONet-2D* generated optimal test functions proves to be a success for schemes that use the higher order trial functions as well. Figure 77 shows a comparison between the regular Galerkin method based on trial and test functions as shown in Figure 64 (that are the product of the $C^0(\Omega)$ quadratic B-spline products), and two methods that use the *DeepONet-2D* generated and FEM generated optimal test functions. Figure 78 shows a similar comparison but for schemes that use the trial functions that are shown in Figure 65 (that are the product of the $C^1(\Omega)$ quadratic

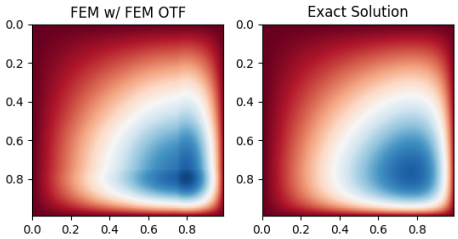
B-spline products). In both cases, the version that uses *DeepONet-2D* generated optimal test functions performs better than the original Galerkin method.



(a) Using test functions that are tensor product of $C^0(\Omega)$ B-splines as shown in Figure 64, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.248$.

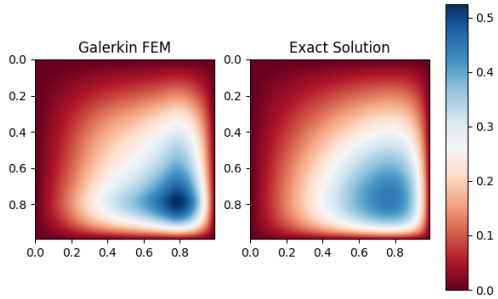


(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.245$.

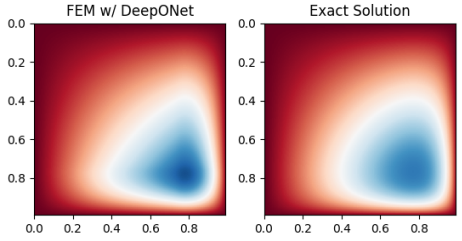


(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.245$.

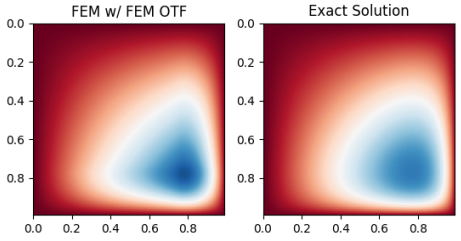
Figure 77: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 4×4 elements for $\epsilon = 0.1$, with trial functions that are the tensor product of the $C^0(\Omega)$ quadratic B-splines as shown in 64.



(a) Using test functions that are tensor product of $C^1(\Omega)$ B-splines as shown in Figure 65, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.295$.



(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.287$.



(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 0.287$.

Figure 78: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 4×4 elements for $\epsilon = 0.1$, with trial functions that are the tensor product of the $C^1(\Omega)$ quadratic B-splines as shown in Figure 65.

5.3.7 Training *DeepONet-2D-2*

Another DeepONet is trained to generate optimal test functions corresponding to (159), but this time for a larger Peclet number, by setting $\epsilon = 0.01$. This new network will be called *DeepONet-2D-2*. Network *DeepONet-2D-2* is trained in the exact same fashion as the network from the previous section and uses the same configuration. The losses during training are plotted in Figure 79.

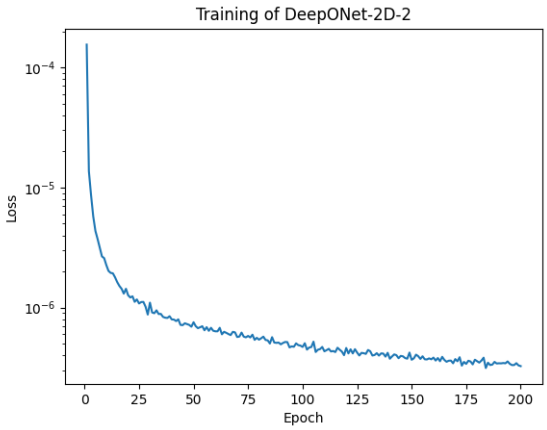


Figure 79: Average MSE loss during training of *DeepONet-2D-2*.

Like the DeepONet that was trained in the previous section, the model converges well and after 200 epochs the training MSE loss is very small. Figure 80, Figure 81 and Figure 82 confirm that *DeepONet-2D-2* is able to approximate the optimal test functions with a very high accuracy. Note that for the $C^1(\Omega)$ shape functions the subscripts of the optimal test functions are different. This has to do with the fact that a discretisation of 10×10 elements is used instead of the discretisation that was used previously with 4×4 elements.

Comparing the optimal test functions shown in Figures 80-82 that correspond to a diffusion coefficient of $\epsilon = 0.1$, to the optimal test functions shown in Figures 72-74 that correspond to a diffusion coefficient, it can be observerd that the upwind effect gets stronger as the Péclet number becomes larger. As the diffusion coefficient increases, the need for stabilisation is greater, and more weight is being placed on the upwind part of the element. To see this, consider Figure 74 and Figure 82. First focus on Figure 74 and look at the optimal test functions in the last row or the last column. These optimal test functions correspond to shape functions where at least one of the 1D B-splines used to build them is the second-last B-spline shown in Figure 62 c). The second-last $C^1(\Omega)$ B-spline puts a lot more weight on the downwind part of the domain which makes the upwind effect less easy to spot. Now consider Figure 82 and focus on the same row/column. The weight of the optimal test function has moved upwind considerably.
Figure 83 shows a comparison between the regular Galerkin method based on trial and test functions as shown in Figure 63 (that is the $C^0(\Omega)$ one dimensional piecewise-linear B-spline product), and two methods that use the *DeepONet-2D-2* generated and FEM generated optimal test functions. The *DeepONet-2D* generated optimal test functions appear to get rid of most of the oscillations in the approximate solution. Figure 85 and 85 show similar results for the higher order trial functions. In both cases using the *DeepONet-2D-2* generated optimal test functions leads to a much smaller approximation error, and visibly milder oscillations.

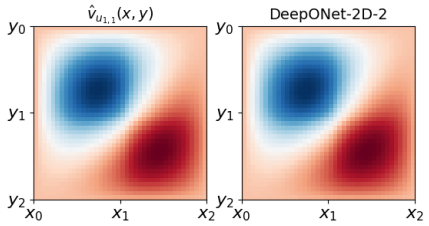


Figure 80: Comparison of FEM approximation and *DeepONet-2D-2* approximation of optimal test function $v_{u_{1,1}}(x, y)$, corresponding to trial function $u_{1,1}$ shown in Figure 63 corresponding to (159) with $\epsilon = 0.01$, for a discretisation using 10×10 elements.

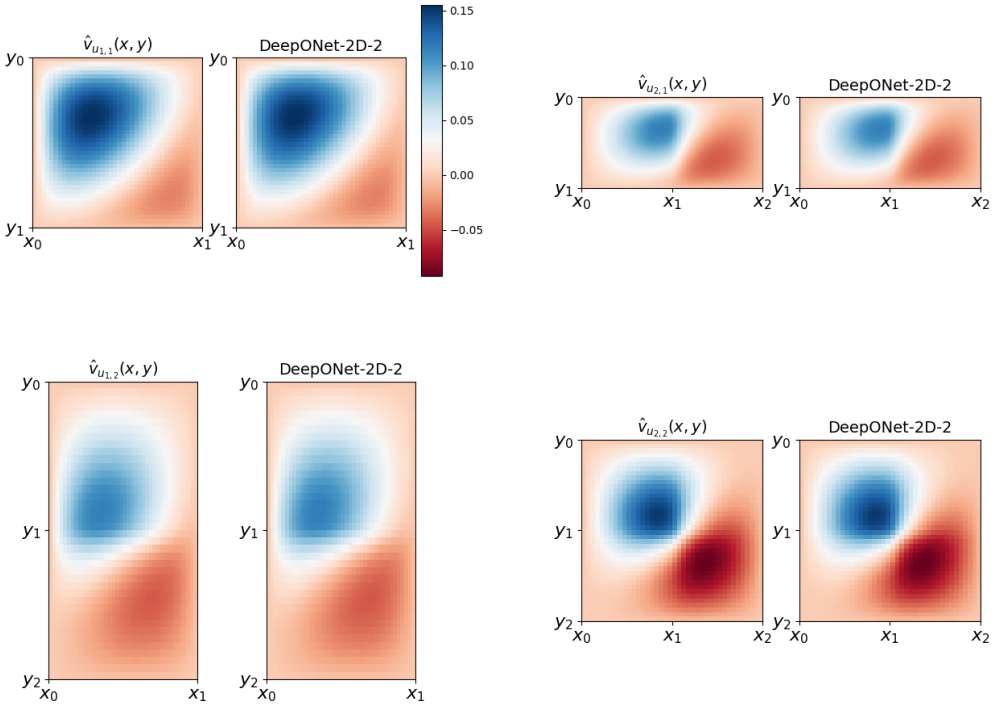


Figure 81: Comparison of FEM approximation and *DeepONet-2D-2* approximation of optimal test functions corresponding to the $C^0(\Omega)$ B-spline products as shown in 64, for problem (159) with $\epsilon = 0.01$.

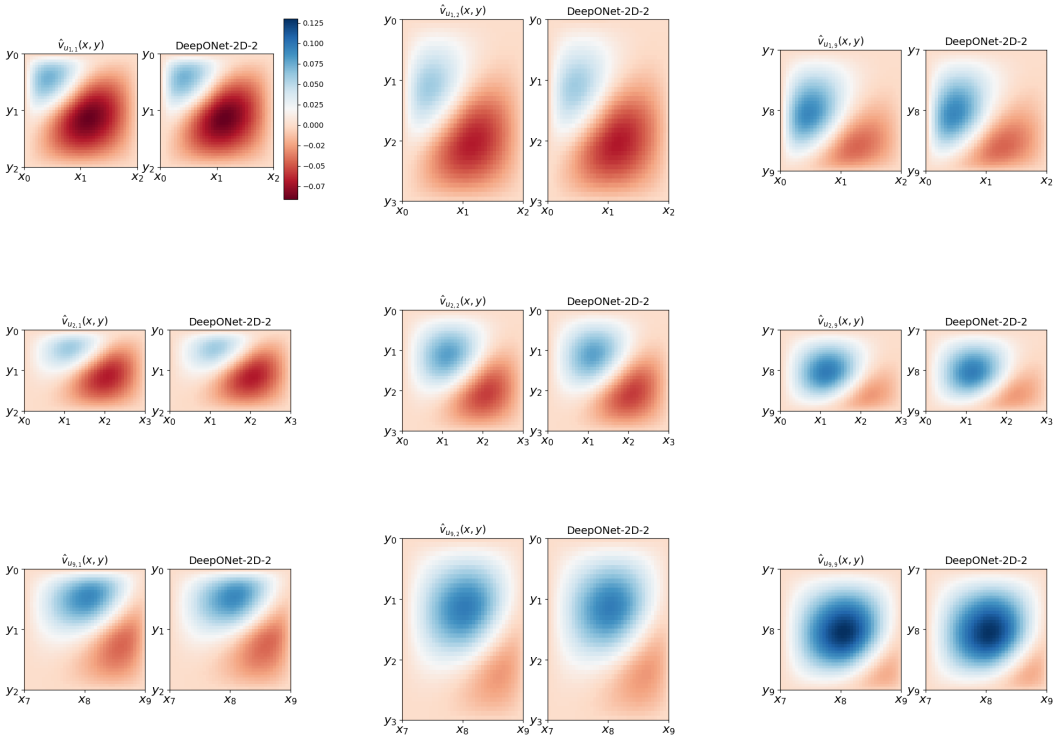
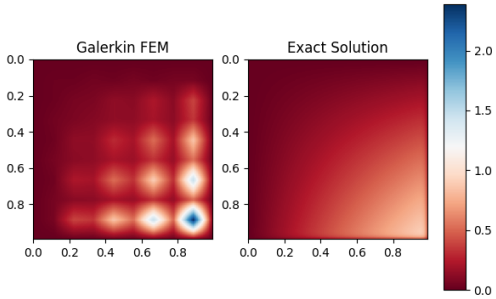
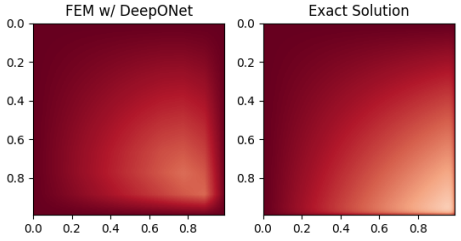


Figure 82: Comparison of FEM approximation and DeepONet approximation of optimal test function corresponding to the trial functions shown in Figure 65, for problem (159) with $\epsilon = 0.01$.

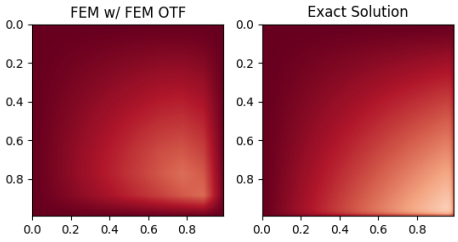
Compared to the case where $\epsilon = 0.1$, the difference in the error measured in the H^1 norm between the Galerkin method and the methods using optimal test functions (either FEM or *DeepONet-2D-2* generated) is much bigger. One thing to note here is that the results using $C^0(\Omega)$ quadratic trial functions result in a lower error and a less oscillatory approximate solution than the method using $C^1(\Omega)$ trial functions. This shouldn't happen generally and is a result of the fact that *DeepONet-2D-2* was trained for a fixed discretisation. Since the $C^0(\Omega)$ and $C^1(\Omega)$ trial functions use the same discretisation, the implementation using $C^0(\Omega)$ quadratic trial functions has a lot more degrees of freedom, as can be seen in Figure 62.



(a) Using test functions shown in Figure 63 that are tensor product of $C^0(\Omega)$ piecewise-linear B-splines, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 6.616$.

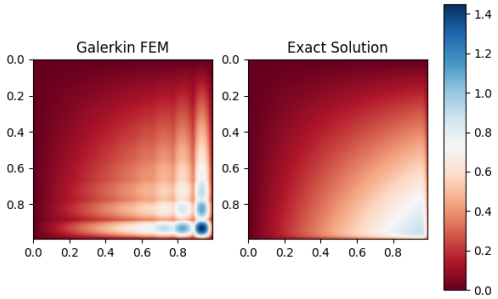


(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 5.224$.

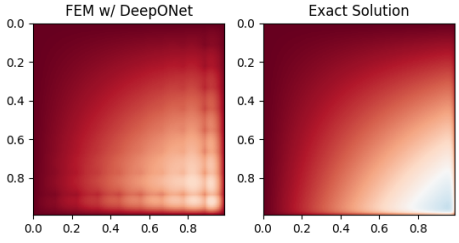


(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 5.221$.

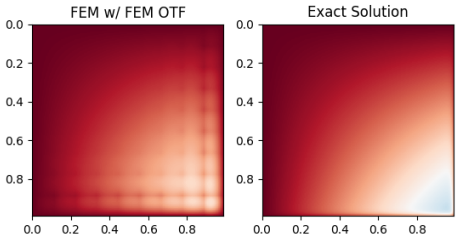
Figure 83: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D-2* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 10×10 elements for $\epsilon = 0.01$, with trial functions that are the tensor product of the $C^0(\Omega)$ piecewise-linear B-splines as shown in Figure 63.



(a) Using test functions that are tensor product of $C^0(\Omega)$ quadratic B-splines as shown in Figure 64, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 4.499$.

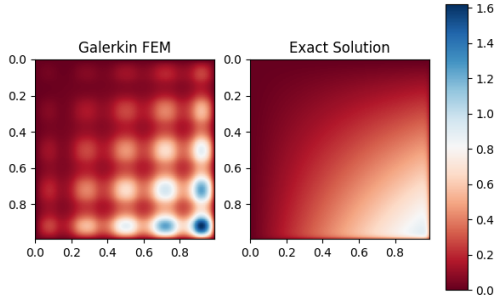


(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 4.317$.

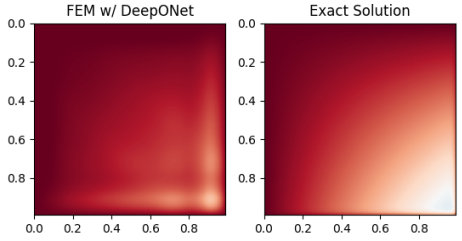


(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 4.309$.

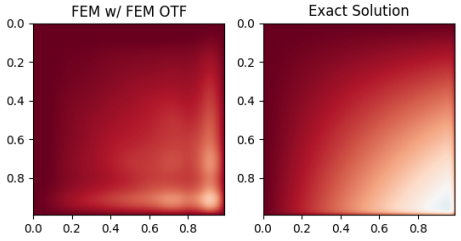
Figure 84: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 10×10 elements for $\epsilon = 0.01$, with trial functions that are the tensor product of the $C^0(\Omega)$ quadratic B-splines as shown in Figure 64.



(a) Using test functions that are tensor product of $C^1(\Omega)$ B-splines as shown in Figure 65, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 5.209$.



(b) Using *DeepONet-2D* generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 4.538$.



(c) Using FEM generated optimal test functions, resulting in an approximation error of $\|u - u_n\|_{H^1(\Omega)} = 4.513$.

Figure 85: Comparison of regular FEM implementation of (155) to a FEM implementation that uses *DeepONet-2D* b) and FEM c) generated optimal test functions. All implementations use a discretisation of 10×10 elements for $\epsilon = 0.01$, with trial functions that are the tensor product of the $C^1(\Omega)$ quadratic B-splines as shown in Figure 65.

5.3.8 Training *DeepONet-2D-VP*

Now that a problem is found for which the DeepONets can be implemented to improve the finite element approximation, it makes sense to try and answer the third research question in this thesis. Namely whether it is possible to train a neural network to generate optimal test functions, using problem specific parameters like the diffusion coefficient as variables, to improve the stability/accuracy of finite element methods. The last deeponet that is trained in this thesis will try to do just that and will therefore be called *DeepONet-2D-VP*, as it is going to deal with variable Péclet

(VP) numbers. This network will use the exact same configuration as the other two networks in this section, but instead of generating the solutions to (159) for a single value of ϵ this time solutions are generated for different ϵ ranging between 2.5×10^{-3} and 1. Using ϵ as an additional variable to the *DeepONet-2D-VP* network, each point in the the training dataset now looks like this:

$$(\boldsymbol{u}_{i,j}, (x, y, \epsilon), \hat{v}_{i,j}(x, y) \tag{162}$$

Here $\boldsymbol{u}_{i,j}$ again denotes the vector with the values of the input function $u_{i,j}$ evaluated at the fixed sensors, and $\hat{v}_{i,j}(x, y)$ denotes the finite element approximation of the corresponding optimal test function. Like the points on the domain (x, y), the values for ϵ that are fed to the network are scaled to the interval $[0, 1]$. This step is important for the training process, as the values of (x, y) and ϵ have very different scales.
Figure 86 shows the MSE loss of the *DeepONet-2D-VP* during training. The error after training is much higher when compared to those of the *DeepONet-2D* and *DeepONet-2D-2* as shown in Figure 71 and Figure 79 respectively. This makes sense since *DeepONet-2D-VP* now needs to take into account an additional variable, using the same configuration.

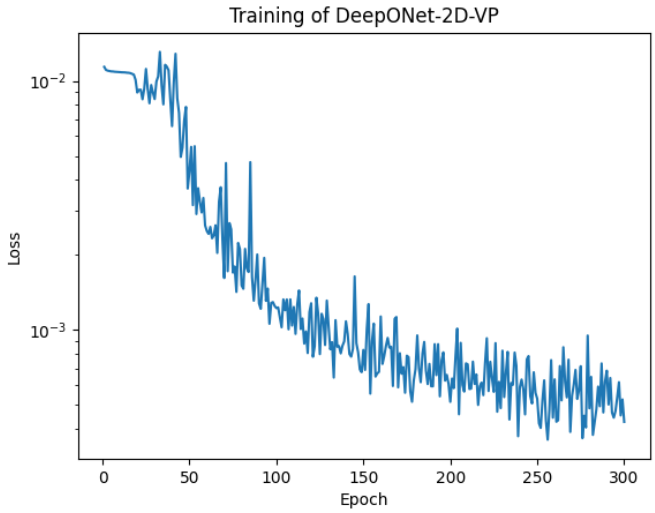


Figure 86: MSE loss during training of *DeepONet-2D-VP*.

Figure 87 shows a comparison between the *DeepONet-2D-VP* and the FEM generated optimal test functions corresponding to the trial functions shown in Figure 64, for a diffusion coefficient $\epsilon = 0.0025$. While the *DeepONet-2D-VP* does not approximate the optimal test functions as well as the two previous models did, it is still very precise.
To test the *DeepONet-2D-VP* network, it is used in the finite element implementation of problem (155) for multiple values of ϵ and is compared to implementations that use finite element generated optimal test functions and Galerkin finite element implementations. The results are shown in Figure 88 for the trial functions that are the tensor product of the $C^0(\Omega)$ piecewise linear, $C^0(\Omega)$ and $C^1(\Omega)$ quadratic trial functions.

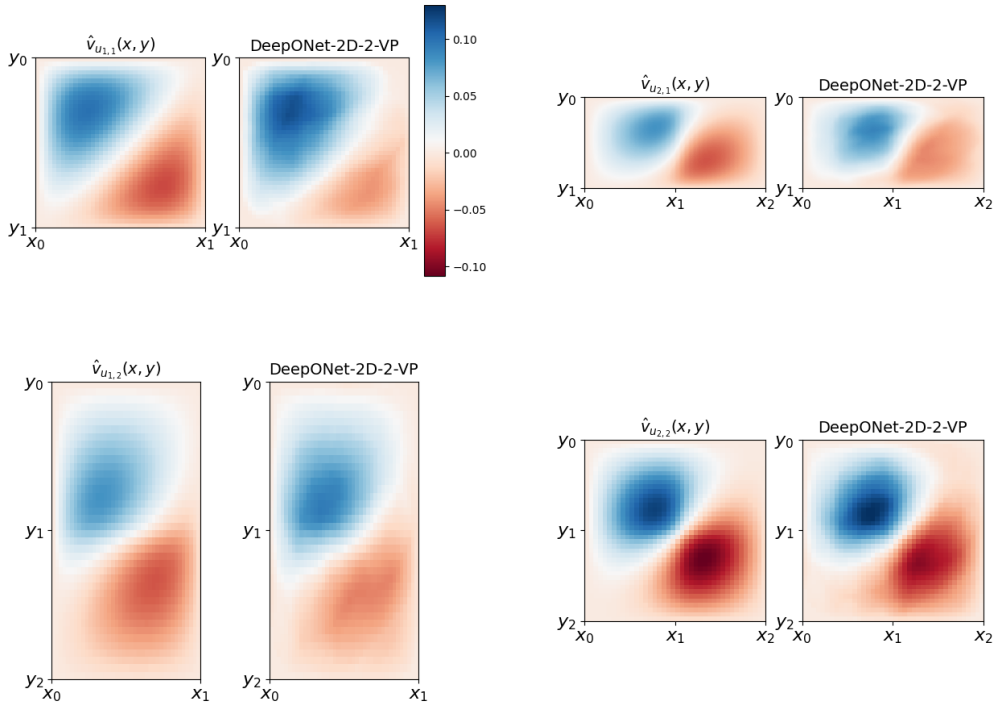
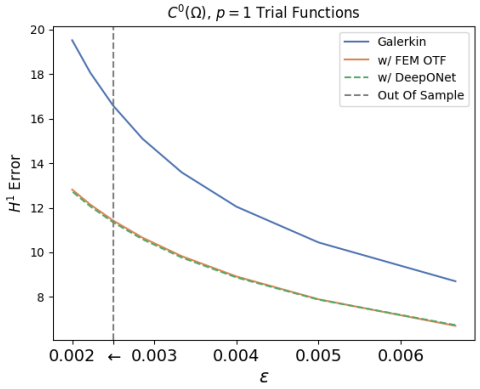
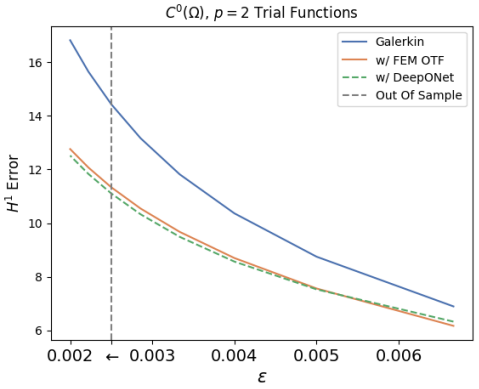


Figure 87: Comparison of FEM approximation and *DeepONet-2D-VP* approximation of optimal test functions corresponding to the $C^0(\Omega)$ B-spline products as shown in Figure 64, for problem (159) with $\epsilon = 0.0025$.

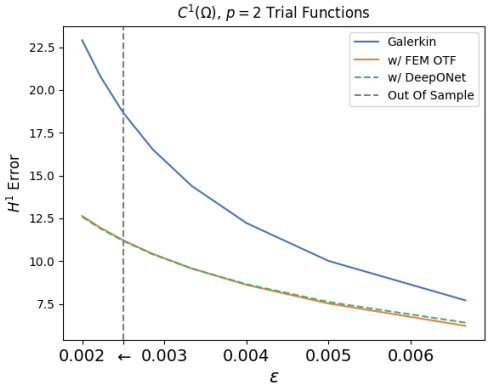
It is quite remarkable how well the DeepONet implementation works in this case, especially after the dissapointing results in 1D. For all the types of different trial functions, implementing *DeepONet-2D-VP* leads to a decrease in the approximation error that is on par with the results of the implementations that use FEM generated optimal test functions. For some values of ϵ the implementations with *DeepONet-2D-VP* generated optimal test functions actually do better than the implementations with fem generated optimal test functions. This is effect is particularly visible for the implementations that use $C^0(\Omega)$ quadratic B-spline tensor products, i.e., Figure 88 b). It appears that in these cases the network accidentally got to a better approximation of the optimal test functions (the FEM generated optimal test functions are just approximations after all).
It is worth mentioning that *DeepONet-2D-VP* was trained by randomly sampling Péclet values Pe_i from the interval $[1, 400]$ and by setting the diffusion coefficient to $\epsilon_i = 1/Pe_i$. It follows that *DeepONet-2D-VP* has only seen diffusion coefficients from the interval $[0.0025, 1]$ during its training, meaning that the predictions shown in Figure 88 for $\epsilon < 0.0025$ are out of sample predictions. The grey vertical line in the plots indicates the out of sample threshold, where the arrow below the plot points of the direction of the region that is out of sample.



(a) Using test functions that are tensor product of the piecewise-linear $C^0(\Omega)$ B-splines as shown in Figure 63.



(b) Using test functions that are tensor product of $C^0(\Omega)$ B-splines as shown in Figure 64



(c) Using test functions that are tensor product of $C^1(\Omega)$ B-splines as shown in Figure 65.

Figure 88: Comparison of Galerkin FEM implementation of (155) to FEM implementations that use *DeepONet-2D-VP* and FEM generated optimal test functions for varying values of ϵ. All implementations use a discretisation of 10×10 elements. Here, the grey vertical lines and the arrows below them indicate the point from which the optimal test functions are predicted for ϵ values that are out of sample (the arrows point in the direction of values that are out of sample).

The fact that is possible to train a network that can generate optimal test functions for a range of values for ϵ and improve the stability/accuracy of the finite element method can be very valuable. A single neural network could potentially be integrated in a software package that can is used to solve PDEs using the finite element method. A user would have access to optimal test function approximations that could used almost immediately. Another very useful application of the *DeepONet-2D-VP* network would be the case where ϵ is not a constant but instead varies over the domain. After training a network, instead of computing the optimal test functions for each element separately they could be generated instantaneously.

6 Conclusion

The goal of this master thesis was to find out whether machine learning approaches can be used to generate optimal test functions that improve the stability/accuracy of the finite element method. This idea was tested on three different weak forms corresponding to the steady state advection-diffusion equation in 1D and in 2D. In every case the Péclet number was chosen such that the problem was advection dominated and the Galerkin finite element solution exhibited oscillations near a boundary layer. For one of the problems in 1D a mixed weak formulation was implemented by introducing auxiliary variables for the fluxes and derivative of the solution. For the other two problems in 1D and 2D a regular weak formulation was used (no auxiliary variables). As there can exist quite a large number of unique optimal test functions per problem, Deep Operator Networks (DeepONets) were used to generate the optimal test functions. The DeepONet can learn to approximate operators, which is very convenient in the case of generating optimal test functions. The architecture is able to take in a trial function as a variable and predict the values of the corresponding trial functions at points in a domain. This means that instead of training a single neural network per optimal test function, a single network can be trained to approximate multiple optimal test functions. Let's consider the results by revisiting the research questions that formed the underpinning of the experiments in this thesis.

6.1 Revisiting the Research Questions

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?*

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?*

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?*

It was found that the applicability of deep learning generated optimal test functions varies based on the particular weak form that is used. In the 1D non mixed weak formulation case the accuracy of the finite element methods turned out to be very sensitive to small perturbations in the optimal test functions when a non-zero source term was present. In the 1D mixed formulation case similar results were found, as small perturbations to the optimal test functions were enough to cause a substantial decrease in the accuracy of the finite element solution. Although the DeepONets learned to generate the optimal test functions very accurately, in these cases their small approximation errors led to much greater errors in the finite element implementations for which they were used.
For the non-mixed 1D problem the impact of the perturbation was so significant that the methods using DeepONet generated optimal test functions produced approximate solutions that were worse that the original Galerkin implementations when using the $C^0(\Omega)$ and $C^1(\Omega)$ quadratic trial functions. To improve the solution in this non-mixed case, a new network was trained to approximate the derivatives of the optimal test functions. While the approximate solution of the finite element methods using this network did improve, they did not reach the accuracy of the Galerkin imple-

mentations.
For the mixed form in 1D, the DeepONets were used just to approximate only a handful of the optimal test functions. Moreover, the derivatives of these optimal test functions were generated by the finite element method from the start. Even under those conditions the implementation using DeepONets showed a considerable decrease in accuracy when compared to the implementations that used FEM generated optimal test functions. To conclude the 1D mixed form experiment a small perturbation was added to one of the optimal test functions that seemed to cause the worsening of the finite element solution. The method that used the perturbed optimal test function showed a significant decrease in accuracy.

One of the interesting findings of this thesis was that there do exist weak formulations for which using DeepONets generated optimal test functions can improve the stability/accuracy of the finite element method. In the 2D case it was shown that DeepONets can even use problem specific parameters like the diffusion coefficient ϵ as variables, and produce optimal test function approximations for many variations of a single weak form that are accurate enough to improve the approximate finite element solutions. Here, it turned out that the goals set out in the three research questions could be achieved. The DeepONets can approximate the optimal test function corresponding to a trial function (research question 1), corresponding to variable trial functions (research question 2), and using problem specific parameters like the diffusion coefficient as variables (research question 3), to improve the stability/accuracy in the finite element method. By training a DeepONet to take in problem specific parameters like the diffusion coefficient, it becomes possible to use a single network to improve the stability/accuracy of the finite element method for a large variety of problems without any further computations. One of the cases where this could be very useful is problems where the diffusion coefficient is not constant but changes over the problem domain.

6.2 Discussion and Future Research

The results in this master thesis have shown that there are cases where optimal test functions generated by machine learning can be used to improve the stability/accuracy of the finite element method. That being said, a lot of questions remain. This subsection will briefly address a number of points that have not be covered successfully in this report and could be the focus of future research.

1. In the 1D non-mixed case it has been shown that the finite element approximations are very sensitive to perturbations to the optimal test functions, but it is not necessarily clear how and when the DeepONet approximation errors impact the stability/accuracy of the solution. As the optimal test functions are approximated extremely accurately the point could be made that the lack of accuracy in the derivatives' approximation is causing the problems. That suspicion seemed to be confirmed when the DeepONet generated test functions were used together with fem generated derivatives. That implementation yielded results that were identical to those that used only fem generated derivatives and optimal test functions. However, this does not seem consistent with the findings in the $f = 0$ case. There the DeepONet implementation is so successful that it is barely distinguishable from the implementations that use FEM generated optimal test functions. If the approximation error in the derivative of the optimal test function is what's causing the deterioration of the finite element solution, then why didn't it cause those problems for the $f = 0$ case? To gain a better understanding of the problem

it would be nice to add different types of perturbations to the optimal test function and its derivative (and to both of them separately), and see how the finite element solution would respond.

2. In the 1D mixed formulation, the DeepONet implementation seemed to work for all but one of the optimal test functions. This optimal test functions was scaled by $1/\epsilon$ and appeared to be approximated not accurately enough. It was shown that adding a tiny perturbation to the FEM generated approximation of this optimal test function was enough to make the finite element solution deteriorate dramatically. It would have been nice to see whether changing the weak formulation by moving ϵ to different parts of the problem, other test functions would have become the bottleneck.

3. In the 2D non-mixed case the implementation was successful; using DeepONet generated optimal test functions improved the stability and accuracy of the finite element method. *Why* the implementation was successful is unclear though. The 1D problems showed that the finite element method can be very sensitive to perturbations in the optimal test functions. It is unclear whether the success in 2D was the cause of using a weak form for which the approximation errors did not pose an issue, or whether the optimal test functions that were used were more accurate (compared to the 1D case). Although the results seem to suggest that the first of those statements is true (that the weak form is not that sensitive to perturbations in the optimal test functions), it would be worthwhile to test this hypothesis. This could be done by adding perturbations to the fem generated optimal test functions and comparing the corresponding solutions.

4. While the DeepONet that was used in the 2D section was trained to use the problem specific parameter ϵ as a variable, it did not use the direction of the flow. It would be nice to use both the Péclet number and the advection direction as variables to the network as doing so would make it applicable to all variations of the weak form that was used. A good next step would be to also pass the element length used in finite element discretisation as a variable to the network. A drawback of the current setup is that one network is needed per discretisation, which puts a big restriction on its usability. Based on the results of this thesis using the flow direction and the element length as additional variables is possible, and doing so successfully appears to be just a matter of finding the right network configuration.

Appendices

A 2D Mixed Weak Formulation

Here, an unsuccessful attempt will be made to implement 2D mixed weak formulation that is the same as the one used in [35]. Consider the following 2D advection-diffusion equation with Dirichlet boundary conditions

$$\begin{cases} -\nabla \cdot (\epsilon \nabla u) + \mathbf{b} \cdot \nabla u = f, & \text{in } \Omega \\ u = 0, & \text{on } \partial\Omega \end{cases} \tag{163}$$

where ϵ denotes the diffusion coefficient, $\mathbf{b}$ the advection velocity, and where $\Omega = [0,1] \times [0,1]$. A mixed FE methodology is applied by introducing the variable $\mathbf{q} = \epsilon \nabla u$ as an auxiliary variable. Using this new variable the second order PDE in (163) can be reformulated as a first order system

$$\begin{cases} \mathbf{q} - \epsilon \nabla u = 0 & \text{in } \Omega, \\ -\nabla \cdot \mathbf{q} + \mathbf{b} \cdot \nabla u = f & \text{in } \Omega \\ u = 0, & \text{on } \partial\Omega \end{cases} \tag{164}$$

To derive a DPG weak formulation of (163), as in [35], first the original domain Ω is divided into a partition $\mathcal{P}_h$ of square sub-domains K_m such that

$$\Omega = \text{int}\Big(\bigcup_{K_m \in \mathcal{P}_h} \overline{K_m} \Big) \tag{165}$$

Then, by multiplying the first two equations in (164) with a test function and using Green's identity, and by enforcing the Dirichlet boundary condition on u strongly (and thereby enforcing $v_{|\partial\Omega} = 0$), the following weak form can be derived:

$$\begin{aligned} &\text{Find } (u, \mathbf{q}) \in U(\Omega) \text{ such that:} \\ &\sum_{K_m \in \mathcal{P}_h} \Big\{ \int_{K_m} \Big[(\mathbf{q}_m - \epsilon \nabla u_m) \cdot \mathbf{w}_m + \mathbf{q}_m \cdot \nabla v_m + (\mathbf{b} \cdot \nabla u_m)\, v_m \Big] dx \\ &\qquad - \int_{\partial K_m \setminus \partial\Omega} \gamma_{\mathbf{n}}^m(\mathbf{q}_m)\, \gamma_0^m(v_m)\, ds \Big\} = \sum_{K_m \in \mathcal{P}_h} \Big\{ \int_{K_m} f v_m \, dx \Big\}, \\ &\forall (v, \mathbf{w}) \in V(\mathcal{P}_h) \end{aligned} \tag{166}$$

with the trial and test space defined as follows

$$\begin{aligned} U(\Omega) &\overset{\text{def}}{=} \Big\{ (u, \mathbf{q}) \in H^1(\Omega) \times H(\text{div}, \Omega) : \gamma_0^m(u_m)_{|\partial K_m \cap \Gamma_D} = 0, \forall K_M \in \mathcal{P}_h \Big\}, \\ V(\mathcal{P}_h) &\overset{\text{def}}{=} \Big\{ (v, \mathbf{w}) \in H^1(\mathcal{P}_h) \times H(\text{div}, \Omega) : \gamma_0^m(v_m)_{|\partial K_m \cap \Gamma_D} = 0, \forall K_M \in \mathcal{P}_h \Big\} \end{aligned} \tag{167}$$

where γ_0^n and $\gamma_{\mathbf{n}}^m$ denote the trace and normal trace operators, as in [35]. Now, let's introduce the following notation

$$B\big((u,\mathbf{q});(v,\mathbf{w})\big) \stackrel{\text{def}}{=} \sum_{K_m\in\mathcal{P}_h}\Bigg\{\int_{K_m}\Big[(\mathbf{q}_m-\epsilon\nabla u_m)\cdot\mathbf{w}_m+\mathbf{q}_m\cdot\nabla v_m+(\mathbf{b}\cdot\nabla u_m)v_m\Big]dx \\ -\int_{\partial K_m\setminus\partial\Omega}\gamma_{\mathbf{n}}^m(\mathbf{q}_m)\,\gamma_0^m(v_m)\,ds\Bigg\} \tag{168}$$

$$F\big((v,\mathbf{w})\big) \stackrel{\text{def}}{=} \sum_{K_m\in\mathcal{P}_h}\Bigg\{\int_{K_m} f v_m\,dx\Bigg\} \tag{169}$$

Using this new compact notation, (166) can be written as follows:

$$\begin{aligned}&\text{Find } (u,\mathbf{q})\in U(\Omega) \text{ such that:}\\ &\qquad B\big((u,\mathbf{q});(v,\mathbf{w})\big)=F\big((v,\mathbf{w})\big),\quad \forall(v,\mathbf{w})\in V(\mathcal{P}_h)\end{aligned} \tag{170}$$

A.1 Deriving the Optimal Test Functions

To derive the FE discretisation, the authors in [35] first introduce the family of invertible maps, $\{\boldsymbol{F}_m:\hat{K}\subset\mathbb{R}^2\to\Omega\}$, such that every $K_m\in\mathcal{P}_h$ is the image of the element $\hat{K}$ through one of the mapppings $\boldsymbol{F}_m$. Using this mapping the space of trial functions can be defined as follows:

$$U^h(\Omega)\stackrel{\text{def}}{=}\Bigg\{(\phi^h,\boldsymbol{\theta}^h)\in C^0(\Omega)\times[C^0(\Omega)]^2:(\phi^h_{|K_m},\boldsymbol{\theta}^h_{|K_m})=(\hat{\phi},\hat{\boldsymbol{\theta}})\circ\boldsymbol{F}_m,\\ \hat{\phi}\in P^{p_m}(\hat{K})\wedge\hat{\boldsymbol{\theta}}\in[P^{p_m}(\hat{K})]^2,\quad\forall K_m\in\mathcal{P}_h\Bigg\} \tag{171}$$

Here, p_m denotes the polynomial degree of approximation on K_m. The approximate solutions u^h and $\boldsymbol{q}^h=\{q_x^h,q_y^h\}^T$ are linear combinations of trial functions $(e^i(\boldsymbol{x}),(E_x^j(\boldsymbol{x}),E_y^k(\boldsymbol{x})))\in U^h(\Omega)$, i.e.,

$$u^h(\boldsymbol{x})=\sum_{i=1}^N u_i^h e^i(\boldsymbol{x}),\quad q_x^h(\boldsymbol{x})=\sum_{j=1}^N q_x^{h,j}E_x^j(\boldsymbol{x}),\quad q_y^h(\boldsymbol{x})=\sum_{k=1}^N q_y^{h,k}E_y^k(\boldsymbol{x}) \tag{172}$$

Here, u_i^h, $q_x^{h,j}$, and $q_x^{h,k}$ for $i=1,\dots,N$, $j=1,\dots,N$ and $k=1,\dots,N$ are the degrees of freedom that will be found by solving the system resulting from the finite element implementation. To derive the optimal test functions, the following inner product, $(\cdot,\cdot)_{V(\mathcal{P}_h)}:V(\mathcal{P}_h)\times V(\mathcal{P}_h)\to\mathbb{R}$, will be used

$$((r,\mathbf{z}),(v,\mathbf{w}))_{V(P_h)}\stackrel{\text{def}}{=}\sum_{K_m\in P_h}\int_{K_m}\Big[h_m^2\nabla r_m\cdot\nabla v_m+r_m v_m+\mathbf{z}_m\cdot\mathbf{w}_m\Big]dx \tag{173}$$

Now, each of the $3N$ trial functions $e^i(\boldsymbol{x}),E_x^j(\boldsymbol{x})$, and $E_y^k(\boldsymbol{x})$ is paired with a vector-valued test function. Specifically, $e^i(\boldsymbol{x})$ is paired with $(v_u^i,\mathbf{w}_u^i)\in V(\mathcal{P}_h)$, $E_x^j(\boldsymbol{x})$ with $(v_{q_x}^i,\mathbf{w}_{q_x}^i)\in V(\mathcal{P}_h)$, and $E_y^k(\boldsymbol{x})$ with $(v_{q_y}^i,\mathbf{w}_{q_y}^i)\in V(\mathcal{P}_h)$. These optimal test functions can be found by solving the following

variational problems

$$\begin{aligned}
\Big((r,\mathbf{z}),(v_u^i,\mathbf{w}_u^i)\Big)_{V(P_h)} &= B((e^i,\mathbf{0}),(r,\mathbf{z})), && \forall (r,\mathbf{z})\in V(P_h), \quad i=1,\ldots,N,\\
\Big((r,\mathbf{z}),(v_{q_x}^i,\mathbf{w}_{q_x}^i)\Big)_{V(P_h)} &= B((0,(E_x^j,0)),(r,\mathbf{z})), && \forall (r,\mathbf{z})\in V(P_h), \quad j=1,\ldots,N,\\
\Big((r,\mathbf{z}),(v_{q_x}^i,\mathbf{w}_{q_y}^i)\Big)_{V(P_h)} &= B((0,(0,E_y^k)),(r,\mathbf{z})), && \forall (r,\mathbf{z})\in V(P_h), \quad k=1,\ldots,N
\end{aligned} \tag{174}$$

The local formulation of the first equation in (174), the problem that can be solved to find the local restriction of the optimal test function $(v_u^i,\mathbf{w}_u^i)$ corresponding to trial function e^i, is as follows

$$\begin{aligned}
\int_{K_m}\Big(h_m^2\nabla r_m\cdot\nabla v_{u_m}^i + r_m v_{u_m}^i + \mathbf{z}_m\cdot\mathbf{w}_{u_m}^i\Big)dx = \\
\int_{K_m}\Big(-\epsilon\nabla e_m^i\cdot\mathbf{z}_m + \mathbf{b}\cdot\nabla e_m^i\, r_m\Big)dx, \quad \forall (r_m,\mathbf{z}_m)\in V(\mathcal{P}_h)
\end{aligned} \tag{175}$$

This problem can be decoupled into two equations. The problem becomes: find $v_{u_m}^i$ and $\mathbf{w}_{u_m}^i$ such that

$$\begin{aligned}
\int_{K_m}\Big(h_m^2\nabla r_m\cdot\nabla v_{u_m}^i + r_m v_{u_m}^i\Big) &= \int_{K_m}\mathbf{b}\cdot\nabla e_m^i\, r_m, \quad \forall r_m\in H^1(\mathcal{P}_h)\\
\int_{K_m}\mathbf{z}_m\cdot\mathbf{w}_{u_m}^i &= \int_{K_m}-\epsilon\nabla e_m^i\cdot\mathbf{z}_m, \quad \forall\,\mathbf{z}\in[L^2(\Omega)]^2
\end{aligned} \tag{176}$$

Using $\mathbf{z}_m=\begin{bmatrix}z_{1_m} & z_{2_m}\end{bmatrix}^T$ and $\mathbf{w}_{u_m}^i=\begin{bmatrix}w_{u_{1_m}}^i & w_{u_{2_m}}^i\end{bmatrix}^T$, the second equation in (176) can be rewritten and decoupled from

$$\int_{K_m}\big(z_{1_m}\,w_{u_{1_m}}^i + z_{2_m}\,w_{u_{2_m}}^i\big) = \int_{K_m}-\epsilon\left(\frac{\partial e_m^i}{\partial x}\,z_{1_m}+\frac{\partial e_m^i}{\partial y}\,z_{2_m}\right) \tag{177}$$

into

$$\begin{aligned}
\int_{K_m} z_{1_m}\,w_{u_{1_m}}^i &= \int_{K_m}-\epsilon\,\frac{\partial e_m^i}{\partial x}\,z_{1_m}\\
\int_{K_m} z_{2_m}\,w_{u_{2_m}}^i &= \int_{K_m}-\epsilon\,\frac{\partial e_m^i}{\partial y}\,z_{2_m}
\end{aligned} \tag{178}$$

Therefore, combining the first equation of (176) and the two equations in (178), finding the local restriction of optimal test function $(v_u^i,\mathbf{w}_u^i)$ corresponding to trial function e^i, is equivalent to solving

$$\begin{aligned}
\int_{K_m}\Big(h_m^2\nabla r_m\cdot\nabla v_{u_m}^i + r_m v_{u_m}^i\Big) &= \int_{K_m}\mathbf{b}\cdot\nabla e_m^i\, r_m, \quad \forall r_m\in H^1(\mathcal{P}_h)\\
\int_{K_m} z_{1_m}\,w_{u_{1_m}}^i &= \int_{K_m}-\epsilon\,\frac{\partial e_m^i}{\partial x}\,z_{1_m}, \quad \forall z_{1_m}\in L^2(\Omega)\\
\int_{K_m} z_{2_m}\,w_{u_{2_m}}^i &= \int_{K_m}-\epsilon\,\frac{\partial e_m^i}{\partial y}\,z_{2_m}, \quad \forall z_{2_m}\in L^2(\Omega)
\end{aligned} \tag{179}$$

for $v^i_{u_m}, w^i_{u_{1_m}}$, and $w^i_{u_{2_m}}$ separately. Note that it follows here that

$$\begin{aligned} w^i_{u_{1_m}} &= -\epsilon \frac{\partial e^i_m}{\partial x} \\ w^i_{u_{2_m}} &= -\epsilon \frac{\partial e^i_m}{\partial y} \end{aligned} \tag{180}$$

The local formulations of the second and third equations in (174) can be decoupled in a similar fashion. Consider the local formulations of the second and third equation of (174)

$$\begin{aligned} \int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{x_m}} + r_m v^i_{q_{x_m}} + \mathbf{z}_m \cdot \mathbf{w}^i_{q_{x_m}} \right) dx &= \int_{K_m} \left(\begin{bmatrix} E^j_x & 0 \end{bmatrix} \cdot \mathbf{z}_m + \begin{bmatrix} E^j_x & 0 \end{bmatrix} \cdot \nabla r_m \right) \\ &\quad - \int_{\partial K_m \setminus \partial\Omega} \gamma^m_{\mathbf{n}}(\begin{bmatrix} E^j_x & 0 \end{bmatrix}) \, \gamma^m_0(r_m) \\ \int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{y_m}} + r_m v^i_{q_{y_m}} + \mathbf{z}_m \cdot \mathbf{w}^i_{q_{y_m}} \right) dx &= \int_{K_m} \left(\begin{bmatrix} 0 & E^k_y \end{bmatrix} \cdot \mathbf{z}_m + \begin{bmatrix} 0 & E^k_y \end{bmatrix} \cdot \nabla r_m \right) \\ &\quad - \int_{\partial K_m \setminus \partial\Omega} \gamma^m_{\mathbf{n}}(\begin{bmatrix} 0 & E^k_y \end{bmatrix}) \, \gamma^m_0(r_m) \end{aligned} \tag{181}$$

By writing out some of the dot products and decoupling the equations, solving the two problems in (181) can be reduced to solving

$$\begin{aligned} \int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{x_m}} + r_m v^i_{q_{x_m}} \right) &= \int_{K_m} E^j_x \frac{\partial r_m}{\partial x} - \int_{\partial K_m \setminus \partial\Omega} n_1 \, E^j_x \, r_m \\ \int_{K_m} \mathbf{z}_m \cdot \mathbf{w}^i_{q_{x_m}} &= \int_{K_m} \begin{bmatrix} E^j_x, 0 \end{bmatrix} \cdot \mathbf{z}_m \end{aligned} \tag{182}$$

and

$$\begin{aligned} \int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{y_m}} + r_m v^i_{q_{y_m}} \right) &= \int_{K_m} E^k_y \frac{\partial r_m}{\partial y} - \int_{\partial K_m \setminus \partial\Omega} n_2 \, E^k_y \, r_m \\ \int_{K_m} \mathbf{z}_m \cdot \mathbf{w}^i_{q_{y_m}} &= \int_{K_m} \begin{bmatrix} 0, E^k_y \end{bmatrix} \cdot \mathbf{z}_m \end{aligned} \tag{183}$$

where n_1 and n_2 denote the first and second element in the normal vector $\mathbf{n}$. Now, the second equation in (182) and the second equation in (183) can be rewritten using $\mathbf{z}_m = \begin{bmatrix} z_{1_m} & z_{2_m} \end{bmatrix}^T$, $\mathbf{w}^i_{q_{x_m}} = \begin{bmatrix} w^i_{q_{x_{1_m}}} & w^i_{q_{x_{2_m}}} \end{bmatrix}^T$ and $\mathbf{w}^i_{q_{y_m}} = \begin{bmatrix} w^i_{q_{y_{1_m}}} & w^i_{q_{y_{2_m}}} \end{bmatrix}^T$ to arrive at the following

$$\begin{aligned} \int_{K_m} z_{1_m} \, w^i_{q_{x_{1_m}}} + z_{2_m} \, w^i_{q_{x_{2_m}}} &= \int_{K_m} E^j_x \, z_{1_m}, \quad \forall (z_{1_m}, z_{2_m}) \in [L^2(\Omega)]^2 \\ \int_{K_m} z_{1_m} \, w^i_{q_{y_{1_m}}} + z_{2_m} \, w^i_{q_{y_{2_m}}} &= \int_{K_m} E^j_y \, z_{2_m}, \quad \forall (z_{1_m}, z_{2_m}) \in [L^2(\Omega)]^2 \end{aligned} \tag{184}$$

It follows that $w^i_{q_{x_{2_m}}} = w^i_{q_{y_{1_m}}} = 0$, and that $w^i_{q_{x_{1_m}}}$ and $w^i_{q_{y_{2_m}}}$ need to satisfy

$$\begin{aligned} \int_{K_m} z_{1_m} \, w^i_{q_{x_{1_m}}} &= \int_{K_m} E^j_x \, z_{1_m}, \quad \forall z_{1_m} \in L^2(\Omega) \\ \int_{K_m} z_{2_m} \, w^i_{q_{y_{2_m}}} &= \int_{K_m} E^j_y \, z_{2_m}, \quad \forall z_{2_m} \in L^2(\Omega) \end{aligned} \tag{185}$$

Note that this immediately gives the solutions

$$\begin{aligned} w^i_{q_{x_{1_m}}} &= E^j_x \\ w^i_{q_{y_{2_m}}} &= E^j_y \end{aligned} \tag{186}$$

To summarise the results of the previous derivations; the optimal test functions (w_{u_1}, w_{u_2}), w_{q_x}, and w_{q_y} shown in (180), and (186) respectively. The rest of the local restrictions of the optimal test functions corresponding to (174) can be found by solving

$$\int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{u_m} + r_m v^i_{u_m} \right) = \int_{K_m} \mathbf{b} \cdot \nabla e^i_m \, r_m, \quad \forall r_m \in V(\mathcal{P}_h) \tag{187}$$

for $v^i_{u_m}$, by solving

$$\int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{x_m}} + r_m v^i_{q_{x_m}} \right) = \int_{K_m} E^j_x \frac{\partial r_m}{\partial x} - \int_{\partial K_m \setminus \partial\Omega} n_1 \, E^j_x \, r_m, \quad \forall r_m \in V(\mathcal{P}_h) \tag{188}$$

for $v^i_{q_{x_m}}$, and finally by solving

$$\int_{K_m} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{y_m}} + r_m v^i_{q_{y_m}} \right) = \int_{K_m} E^k_y \frac{\partial r_m}{\partial y} - \int_{\partial K_m \setminus \partial\Omega} n_2 \, E^k_y \, r_m, \quad \forall r_m \in V(\mathcal{P}_h) \tag{189}$$

for $v^i_{q_{y_m}}$. One important point is that it appears to be necessary to enforce boundary conditions on the optimal test functions v_u, v_{q_x}, and v_{q_y} on the domain boundary. This requirement follows from the definition of the test space, e.g.,

$$V(\mathcal{P}_h) \overset{\text{def}}{=} \left\{ (v, \mathbf{w}) \in H^1(\mathcal{P}_h) \times H(\text{div}, \Omega) : \gamma_0^m(v_m)_{|\partial K_m \cap \Gamma_D} = 0, \forall K_M \in \mathcal{P}_h \right\} \tag{190}$$

where the trace of the function v is set to zero.

A.2 Finite Element Implementation

Implementing the optimal test functions into the finite element method comes down to testing (170) with the optimal test functions $(v_{q_x}, \begin{pmatrix} w_{q_{x_1}} & 0 \end{pmatrix})$, $(v_{q_y}, \begin{pmatrix} 0 & w_{q_{y_2}} \end{pmatrix})$, and $(v_u, \begin{pmatrix} w_{u_1} & w_{u_2} \end{pmatrix})$ that are the linear span of the optimal test function approximations in (174). The new weak formulation is shown in (194), where V^h_{opt} is the span of the optimal test function approximations found by solving (174). Building a stiffness matrix for this problem can be done by building the matrices associated with the terms in (194) and using them as building blocks. The matrices are

$$Q_x W_{q_{x_1}}, Q_x V_{q_x}, Q_y V_{q_x}, U W_{q_{x_1}}, U V_{q_x} \tag{191}$$

used to build the parts corresponding to the optimal test function $(v_{q_x}, (w_{q_{x_1}}, 0))$ and

$$Q_x V_{q_y}, Q_y W_{q_{y_2}}, Q_y V_{q_y}, U W_{q_{y_2}}, U V_{q_y} \tag{192}$$

used to build the parts corresponding to the optimal test function $(v_{q_y}, (0, w_{q_{y_2}}))$, and

$$Q_x W_{u_1}, Q_x V_u, Q_y W_{u_2}, Q_y V_u, U W_{u_1}, U W_{u_2}, U V_u \tag{193}$$

Find $(u^h, \begin{pmatrix} q_x^h & q_y^h \end{pmatrix}) \in H^1(\Omega) \times H(\mathrm{div}, \Omega)$ such that:

$$
\begin{cases}
\displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} \left[q_{x_m} \, w_{q_{x_{1m}}} - \epsilon \frac{\partial u_m}{\partial x} w_{q_{x_{1m}}} + q_{x_m} \frac{\partial v_{q_{x_m}}}{\partial x} + q_{y_m} \frac{\partial v_{q_{x_m}}}{\partial y} + (\mathbf{b} \cdot \nabla u_m) \, v_{q_{x_m}} \right] \right. \\
\qquad \left. - \int_{\partial K_m \setminus \partial \Omega} \left(n_1 \, \gamma_0^m(q_{x_m}) + n_2 \, \gamma_0^m(q_{y_m}) \right) \gamma_0^m(v_{q_{x_m}}) \right\} = \displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} f \, v_{q_{x_m}} \right\} \\
\displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} \left[q_{y_m} \, w_{q_{y_{2m}}} - \epsilon \frac{\partial u_m}{\partial y} w_{q_{y_{2m}}} + q_{x_m} \frac{\partial v_{q_{y_m}}}{\partial x} + q_{y_m} \frac{\partial v_{q_{y_m}}}{\partial y} + (\mathbf{b} \cdot \nabla u_m) \, v_{q_{y_m}} \right] \right. \\
\qquad \left. - \int_{\partial K_m \setminus \partial \Omega} \left(n_1 \, \gamma_0^m(q_{x_m}) + n_2 \, \gamma_0^m(q_{y_m}) \right) \gamma_0^m(v_{q_{y_m}}) \right\} = \displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} f \, v_{q_{y_m}} \right\} \\
\displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} \left[q_{x_m} w_{u_{1m}} + q_{y_m} w_{u_{2m}} - (\epsilon \frac{\partial u_m}{\partial x} w_{u_{1m}} + \epsilon \frac{\partial u_m}{\partial y} w_{u_{2m}}) \right. \right. \\
\qquad \left. + q_{x_m} \frac{\partial v_{u_m}}{\partial x} + q_{y_m} \frac{\partial v_{u_m}}{\partial y} + (\mathbf{b} \cdot \nabla u_m) \, v_{u_m} \right] \\
\qquad \left. - \int_{\partial K_m \setminus \partial \Omega} \left(n_1 \, \gamma_0^m(q_{x_m}) + n_2 \, \gamma_0^m(q_{y_m}) \right) \gamma_0^m(v_{u_m}) \right\} = \displaystyle\sum_{K_m \in \mathcal{P}_h} \left\{ \int_{K_m} f \, v_{u_m} \right\}
\end{cases}
\tag{194}
$$

$$
\forall (v_{q_x}, \begin{pmatrix} w_{q_{x_1}} & 0 \end{pmatrix}), (v_{q_y}, \begin{pmatrix} 0 & w_{q_{y_2}} \end{pmatrix}), (v_u, \begin{pmatrix} w_{u_1} & w_{u_2} \end{pmatrix}) \in V_{\mathrm{opt}}^h
$$

used to build the parts corresponding to the optimal test function $(v_u, (w_{u_1}, w_{u_2}))$. Here every first and second capital letter are used to indicate which trial and which test function are used respectively. For example, the matrix $Q_y V_{q_y}$ corresponds to the terms in (194) for trial function q_y and test function v_{q_y}. Using these matrices the following system can be built and solved

$$
\begin{bmatrix}
Q_x W_{q_{x_1}} + Q_x V_{q_x} & Q_y V_{q_x} & U W_{q_{x_1}} + U V_{q_x} \\
Q_x V_{q_y} & Q_y W q_{y_2} + Q_y V_{q_y} & U W_{q_{y_2}} + U V_{q_y} \\
Q_x W_{u_1} + Q_x V_u & Q_y W_{u_2} + Q_y V_u & U W_{u_1} + U W_{u_2} + U V_u
\end{bmatrix}
\begin{bmatrix} C_{q_x} \\ C_{q_y} \\ C_u \end{bmatrix}
=
\begin{bmatrix} F V_{q_x} \\ F V_{q_y} \\ F V_u \end{bmatrix}
\tag{195}
$$

where C_{q_x}, C_{q_y}, and C_u denote the vectors with degrees of freedom corresponding to q_x, q_y and u, and FV_{q_x}, FV_{q_y}, and FV_u denote the vectors associated with the terms on the right hand side of the equations in (194). In this system the boundary conditions on u are enforced strongly as was done in [35].

A.3 Incorrect Results

The convergence of the approximation errors measured in several norms corresponding to the method that follows from (194) are shown in Figure 89. The implementation is incorrect as the order of convergence corresponding to the different errors is much lower than reported in [35]. For example, the paper states that the $L^2(\Omega)$ norm of the error $u - u^h$ exhibit convergence rates of order $p+1$ and p, where p refers to the order of the trial functions used in the implementation. A rough estimation of the slopes shown in Figure 89 corresponding to $\|u - u_n\|_{H^1(\Omega)}$ gives $-0.50, -1.03$, and -1.15 for the piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions respectively. Besides the incorrect order of convergence, something definitely seems of when looking at the error measured in the $L^2(\Omega)$ norm corresponding to σ for the $C^1(\Omega)$ trial functions. There, the convergence rate starts to flatten out.

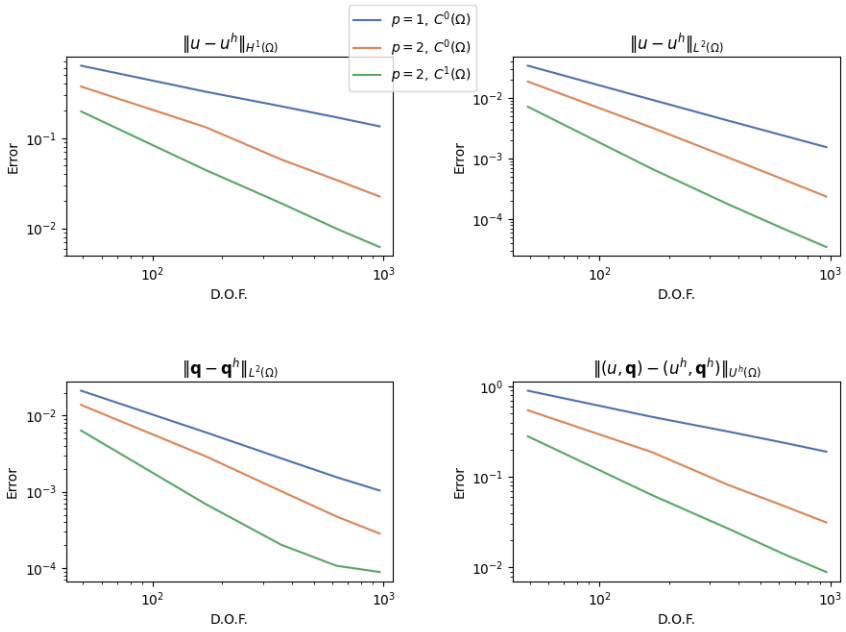


Figure 89: Convergence of method in (194) for $C^0(\Omega)$ piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions, for $\epsilon = 0.1$ and $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

Besides an incorrect implementation, a possible explanation for the results could be the accuracy of the optimal test function approximation.

A.4 Potential DeepONet Implementation

It is questionable whether implementing neural networks into weak formulation (194) makes sense, given the nature of the optimal test function problems in (188) and in (189). To see why, let's focus on (188) and think about the different sides of the element boundary that can be included in the boundary integrals.
For elements that do not intersect the domain boundary, so called "interior" elements, all the sides of the element are included in the boundary. Therefore, for the interior element K_{int_m}, the expression that needs to be solved to find the optimal test functions is simply

$$\int_{K_{\text{int}_m}} \left(h_m^2 \nabla r_m \cdot \nabla v^i_{q_{x_m}} + r_m v^i_{q_{x_m}} \right) = \int_{K_{\text{int}_m}} E_x^j \frac{\partial r_m}{\partial x} - \int_{\partial K_{\text{int}_m}} n_1 E_x^j r_m, \quad \forall r_m \in V(\mathcal{P}_h) \quad (196)$$

Here, a single DeepONet can be trained to approximate the solutions to (194) for all the $C^0(\Omega)$ piecewise linear, $C^0(\Omega)$ quadratic, and $C^1(\Omega)$ quadratic trial functions.
However, for boundary elements there exist eight different versions of (188), depending on which faces lie on the domain boundary. That means that eight different DeepONets would have to be trained to approximate v_{q_x} on the boundary. This in turn would imply that in total 19 DeepONets would have to be trained to approximate all the optimal test functions used in (194), eight

DeepONets for the boundary elements plus one DeepONet for the interior element for v_{q_x}, eight DeepONets for the boundary elements plus one DeepONet for the interior element for v_{q_y}, and one DeepONet for v_u.
It has to be said that it's probably possible to reduce the number of DeepONets that would have to be trained here. By using transformations on the optimal test functions corresponding to boundary elements, it seems possible to reuse some of them (for example by mirroring or scaling them). However, this approach will not be tried in this thesis due to time limits.

7 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.

[39] Elham Wasei, *Investigating Physics-Informed Neural Networks for Solving PDEs.* TU Delft, 2020.