Stabilizing convection-dominated flow problems using neural networks based on flux-limiting techniques

by

Rana Sannia Ul Haq

in partial fulfillment of the requirements for the degree of

Master of science
in Applied Mathematics

at the Delft University of Technology,
to be defended publicly on Wednesday August 24, 2022 at 13:00.

Thesis commitee:	Dr. M. Möller,	TU Delft, supervisor
	Assistant Prof. dr. D.Toshniwal,	TU Delft, supervisor
	Dr. H. M. Schuttelaars,	TU Delft

An electronic version of this thesis is available at `https://repository.tudelft.nl`.

TU Delft Delft University of Technology

Abstract

Convection-dominated flow problems are well-known to have non-physical oscillations near steep gradients or discontinuities in the solution when solved with standard numerical methods, such as finite elements or finite difference methods. To overcome this limitation, algebraic flux correction (AFC) can be used, which is a stabilization method. However, AFC contains time-consuming computations, therefore, alternative approaches are explored. The rapidly rising field of machine learning in the mathematical world, so called scientific machine learning, has successful applications in solving partial differential equations. In this work, the focus is on convection-dominated flow problems, in particular the steady state convection-diffusion equation in one-dimension. To solve this, two alternative approaches based on neural network-learning have been developed that are able to mimic the AFC limiter with a certain accuracy and performance. In some cases, the neural network-based limiter is outperforming the AFC limiter.

Contents

1 Introduction

There is always a desire for numerical methods to solve partial differential equations (PDEs) to be accurate, fast and generalizable, but this proves to be a challenging task and therefore is not always achieved. This thesis will particularly focus on the convection-dominated flow problems, whereby it is well-known that non-physical oscillations near steep gradients or discontinuities occur in the solutions obtained with standard numerical methods, such as finite difference or finite element methods (FEM). A stabilization method that has been developed to overcome this limitation is the algebraic flux correction (AFC). The idea of AFC is to combine a high-order method, which obtains accurate solutions in smooth regions, but are oscillatory in steep regions, with a low-order method that guarantees no oscillations in the solutions, but are overly diffusive. The difference between these low-and high-order methods is the anti-diffusion, which is added back in a limited amount to the low-order method in a non-linear fashion by the AFC limiter. These extra computational steps are taken to stabilize the numerical solution, but are time-consuming, therefore, alternative methods are explored with machine learning to replace AFC and take it a step further by enhancing it with prior physical knowledge of AFC.

Machine learning (ML) techniques are recently widely being used in classical applied mathematics problems, such as solving PDE problems, but according to the author's best knowledge not much research on applying ML to numerical schemes is concluded, in particular stabilizing methods using neural networks. This new field of research, also known as scientific machine learning, is in rapid development and due to neural networks' universal approximation theorem, it can approximate the underlying nonlinear input-output relationship in complex problems. Moreover, training neural networks takes time, but once trained they can perform computationally cheap evaluations.

The motivation for this research is to be more efficient and potentially more accurate due to the huge potential of machine learning techniques to accelerate and automate the tasks and computations, in this thesis the focus is on convection-dominated flow problems, in particular the stationary convection-diffusion equation in one-dimension.

1.1. Research objectives

The main goal of this research is to investigate if machine learning can mimic what the AFC limiter does without computing it. Specifically, if a neural network can learn the behaviour of the AFC limiter to mimic it. The main research question reads:

Can machine learning be used as a replacement for the AFC limiter?

Several sub-questions to guide the research are as followed:

1. *What are possible network architectures and data sets for that?*
2. *What is the performance impact of the neural network on different problems? In essence, is it generalizable (to other problems)?*
3. *Does the trained neural network outperforms the AFC limiter?*
4. *Does it generalize/applicable to higher dimensions and/or complex geometry problems?*

A python code has been written to implement the problem and numerical methods. For the neural network implementation, Pytorch is used.

1.2. Outline

In this thesis, the discretization regarding the one-dimensional stationary convection-diffusion equation is described in chapter 2, followed by the fundamental mathematical concepts and application of the stabilization method, the algebraic flux correction, in chapter 3. In chapter 4 the concept of a feedforward neural network and its parameter are described, after which the first approach, the learning-based flux limiting, is explained and applied to the problem with the corresponding model's results and analysis in chapter 5. The second approach, the learning-based surrogate model of AFC limiting, is described in chapter 6 along with the developed neural network architectures and data sets and their results and analysis. Finally, the conclusions from both approaches are described and future recommendations are given in chapter 7.

2 Numerical solution of the stationary convection-diffusion equation

In this chapter, the theory regarding the numerical solution of a one-dimensional boundary value problem of the stationary convection-diffusion equation is described. The numerical solution is determined by applying the finite element method (FEM) whereby the variational form, also known as weak form (WF), like in the Galerkin method is computed and is outlined in section 2.1. With the discretization, the computations of the integrals involved in the discrete WF by the Gaussian quadrature rule and the applications of boundary conditions are described in section 2.2. Finally, numerical examples are shown in section 2.3 for the Galerkin method of two boundary value problems, namely the boundary layer problem and the peak problem.

2.1. Weak form and numerical discretization

The convection-diffusion equation describes the motion of particles, like energy or other physical quantities, through a fluid by a combination of convection and diffusion processes. Convection moves the particles on a macroscopic level independently of their cause, while diffusion is a process of moving particles from a region of high concentration to a region of low concentration of that particle [6].

The strong form of the boundary value problem of the 1D stationary convection-diffusion on a physical space, which is an unit interval, is defined as:

$$-d\,(u(x))_{xx} + (vu(x))_x = R \quad \text{in } \Omega = [0,1] \tag{2.1}$$

$$u(0) = \alpha, \quad u(1) = \beta, \tag{2.2}$$

where $d > 0$ is the diffusion coefficient, v is a constant velocity, R is the source and $u(x)$ the solution. When $|v| >> d$, then the problem is *convection-dominated* and the higher-order term with diffusion influences the boundary in the form of boundary layers, consequently the solution is determined by the low-order term corresponding to the convective term [8]. When $|v| << d$, then the problem is *diffusion-dominated*. A non-dimensional ratio can be defined to measure the convection and diffusion processes as the **Péclet number**, $Pe = vL/d$, where L is the length of the domain. Since the domain is on an unit interval, $L = 1$, the formulation is described as $Pe = v/d$, whereby for a positive velocity the flow goes from left to right, while for negative velocity the other way.

The Galerkin method is applied to (2.1) to obtain the weak form of the problem. First, take $u \in S$ as trial space, which defines the space where the solutions are square-integrable, have square-integrable derivatives and obey the Dirichlet boundary conditions (also known as the Sobolev space $H^1(\Omega)$). Secondly, take $w \in V$ as test space, which is similar to space S, but the solutions obey the homogeneous Dirichlet boundary conditions. Thirdly, multiply equation (2.1) by a test function w and integrate over the physical, $\Omega = [0,1]$, followed by using integration by parts on the diffusive term only to get the following:

$$[-du_x w]_0^1 + \int_\Omega du_x w_x + (vu)_x w \, dx = \int_\Omega Rw \, dx. \tag{2.3}$$

To impose the Dirichlet boundary conditions, equation (2.2), w has to satisfy the homogeneous boundary conditions, since these are essential boundary conditions, therefore the term over the entire boundary vanishes. Finally, the WF is defined as following:

$$\text{Find } u \in S \text{ such that } \forall\ w \in V \int_\Omega du_x w_x + (vu)_x w \, dx = \int_\Omega Rw \, dx, \tag{2.4}$$

which can be abbreviated as : $a(u,w) = f(w)$, where

$$a(u,w) = \int_\Omega du_x w_x + (vu)_x w \, dx$$

$$f(w) = \int_\Omega Rw \, dx$$

The discrete WF of the problem using the same trial and test space, gives:

$$\text{Find } u^h \in V^h \text{ such that } \forall\ w^h \in V^h,\ a(u^h, w^h) = f(w^h) \text{ , where} \tag{2.5}$$

$$a(u^h, w^h) = \int_\Omega d(u^h)_x (w^h)_x + ((vu)^h)_x w^h\ dx \tag{2.6}$$

$$f(w^h) = \int_\Omega R w^h\ dx \tag{2.7}$$

Using an equidistant mesh consequently makes the finite difference, finite volume and finite element methods all equivalent. However, staying in the FEM framework for 1D the discrete spaces S^h and V^h are spanned by standard linear basis functions (also known as 'tent' functions) $\{\varphi_j(x)\}$ approximating the solutions in the unit interval domain. To approximate the solution u^h and the convective flux $(vu)^h$, a finite linear combination of basis functions is used as:

$$u^h(x) = \sum_{j=0}^{n-1} u_j \varphi_j(x), \quad (vu)^h = \sum_{j=0}^{n-1} (v_j u_j) \varphi_j(x). \tag{2.8}$$

Since the velocity v is constant it can be taken out of the summation. Substituting this into equation (2.5) and replacing w^h by all basis functions $\varphi_i(x)$ results in the matrix form:

$$(S - K)\mathbf{u} = \mathbf{r}, \tag{2.9}$$

where **u** is the vector of coefficients u_i used in the expansion of the solution provided by equation (2.8), the entries of the discrete diffusion, $S = \{s_{ij}\}$, and convection, $K = \{k_{ij}\}$, operators and the discretized right-hand side vector, $\mathbf{r} = \{r_i\}$. These matrix entries are given as following [7]:

$$k_{ij} = -\ v\ c_{ij}, \qquad c_{ij} = \int_\Omega \varphi_j{}' \varphi_i dx, \tag{2.10}$$

$$s_{ij} = d \int_\Omega \varphi_j{}' \varphi_i{}' dx, \qquad r_i = \int_\Omega R \varphi_i dx \tag{2.11}$$

where $i, j = \{0, 1, 2, \ldots, n-1\}$, n is the number of basis functions and the prime ($'$) gives the first derivative.

2.2. Numerical integration and impose boundary conditions

In FEM, the parametric space, $\hat{\Omega}$, is mapped onto the physical space, Ω, so the mapping is used to rewrite the integrals in the physical to solve them in the parametric space. Furthermore, the linear basis functions defined with order 1, $\hat{\varphi}_j = N_{j,1}(\xi)$, as defined by the Cox-de-Boor recursion formula:
for piecewise constants (p=0) [2]:

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{2.12}$$

and for p = 1, 2, 3, ... :

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi), \tag{2.13}$$

where ξ_i denotes the i^{th} node in the mesh vector $\Xi = \{\xi_0, \xi_1, \ldots, \xi_{n+p}\}$, n is the number of basis functions, p is the order of the polynomial of the basis function and $i \in \{0, \ldots, n-1\}$.

The mesh vector for this problem is $\Xi = \{\xi_0, \xi_1, \ldots, \xi_{n+1}\}$, where $\xi_i \in \mathbb{R}$ is the i^{th} node, i is the node index $i = 0, 1, \ldots, n+1$, and n the number of basis functions. The 1D geometric mapping is defined as:

$$x(\xi) = \sum_j \mathbf{c}_j \hat{\varphi}_j(\xi), \quad \xi \in \hat{\Omega} = [0, 1] \tag{2.14}$$

where $\mathbf{c}_j \in \mathbb{R}$ are defined as control points. To solve the problem in the parametric domain, the coordinates of the physical domain are transformed by the mapping $\phi : \hat{\Omega} \to \Omega$. Here the parametric values ξ are converted to physical coordinates x. In order to get the solution in the physical domain, the mapping should be bijective, such that $\phi^{-1} : \Omega \to \hat{\Omega}$ exists [7].
A well known integration rule [6]:

$$\int_{\Omega} u(x)dx = \int_{\hat{\Omega}} u(x(\xi))|\text{det}Dx(\xi)|d\xi, \tag{2.15}$$

where $Dx(\xi)$ is the Jacobian matrix of the geometrical mapping, but in 1D, this is $\frac{dx}{d\xi}$. However, the parametric space is also a unit interval, $\hat{\Omega} = [0, 1]$, where $x = [1]$, which means that the integrals only have a change of variables and thus is a linear transformation.

The integrals are assembled using the numerical quadrature rule [17] given as:

$$\int_{x_L}^{x_R} f(y)dy \approx \frac{x_R - x_L}{2} \sum_{l=0}^{N} w_l f\Big(\frac{x_R - x_L}{2} x_l + \frac{x_L + x_R}{2}\Big), \tag{2.16}$$

where w_l and x_l are computed with the Gaussian quadrature rule on domain $[-1, 1]$ hence the change of interval is applied for domain $[x_L, x_R]$ and $N = 2$ due to linear basis functions.

For example, the discrete convection operator in the parametric domain is computed as following:

$$k_{ij} = -\,v \int_{\hat{\Omega}} \hat{\varphi_j}'\hat{\varphi_i}\, d\xi \approx -\,v \sum_{k=0}^{n-p-1} \underbrace{\int_{e_k} \hat{\varphi_j}'\hat{\varphi_i} d\xi}_{k_{ij}^{e_k}} = -\,v \sum_{k=0}^{n-p-1} k_{ij}^{e_k}, \tag{2.17}$$

here e_k with $k = 0, 1, \ldots, n-p-1$ defines the elements and since a basis function $N_{i,p}(\xi)$ on mesh vector $\Xi = \{\xi_0, \xi_1, \ldots, \xi_{n+p}\}$ is nonzero on $[\xi_i, \xi_{i+p+1}]$, the e_k are intervals in 1D, where the basis functions are nonzero. Now applying the basis functions and the quadrature rule to $k_{ij}^{e_k}$ gives the following computations:

$$k_{ij}^{e_k} = \int_{e_k} \hat{\varphi_j}'(\xi)\hat{\varphi_i}(\xi)d\xi = \int_{e_k} N'_{j,p}(\xi)N_{i,p}(\xi)d\xi \tag{2.18}$$

$$\approx \frac{\xi_{k+1} - \xi_k}{2} \sum_{l=0}^{N=p+p} w_l N'_{j,p}\Big(\frac{\xi_{k+1} - \xi_k}{2} x_l + \frac{\xi_k + \xi_{k+1}}{2}\Big) N_{i,p}\Big(\frac{\xi_{k+1} - \xi_k}{2} x_l + \frac{\xi_k + \xi_{k+1}}{2}\Big). \tag{2.19}$$

In the assembled matrix $A = S - K$ the boundary conditions of u has to be incorporated as the final step. Due to the support feature of the B-spline basis functions, only the diagonal and off-diagonal axis are nonzero values, resulting in a a sparse matrix:

$$A = \begin{pmatrix} A_{00} & 0 & \cdots & 0 \\ A_{10} & A_{11} & \cdots & A_{1n-1} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & A_{n-1n-1} \end{pmatrix}, \ \mathbf{u} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix}, \ \mathbf{r} = \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-2} \\ r_{n-1} \end{pmatrix}. \tag{2.20}$$

Since the left boundary is $u_0 = u(0) = \alpha$ it will become that $A_{00}u_0 = r_0$, $A_{00} = 1$ and $r_0 = \alpha$. For the right boundary $u_{n-1} = u(1) = \beta$ it becomes $A_{n-1n-1}u_{n-1} = r_{n-1}$, $A_{n-1n-1} = 1$ and $r_{n-1} = \beta$.

2.3. Numerical results

This section presents the results obtained for the Galerkin method applied to the boundary value problem (BVP), the 1D stationary convection-diffusion equation, defined in (2.1). Two BVPs used throughout the thesis are the **boundary layer problem** defined as:

$$-d\,(u(x))_{xx} + (vu(x))_x = 0 \quad \text{in } \Omega = [0, 1] \tag{2.21}$$

$$u(0) = 1, \quad u(1) = 0, \tag{2.22}$$

and the **peak problem** defined as:

$$-d\,(u(x))_{xx} + (vu(x))_x = 1 \quad \text{in } \Omega = [0,1] \tag{2.23}$$

$$u(0) = 0, \quad u(1) = 0, \tag{2.24}$$

The analytical solution to the boundary layer problem is:

$$u(x) = \frac{e^{v/d} - e^{v/d \cdot x}}{e^{v/d} - 1} \tag{2.25}$$

This analytical solution shows a steep gradient for high Péclet numbers as shown in Figure 2.1. The computations were made for on an equidistant mesh with standard linear basis functions. The results obtained from the boundary layer problem for positive velocity shows that as the Péclet number increases the Galerkin solution becomes more unstable and produces non-physical oscillations in the vicinty of steep gradients that are not present in the exact solutions.

As the number of basis functions, n, increases the oscillations are still present (see Figure 2.2), but the amplitude is smaller due to the increase in evaluation points of the integrals to compute the solution. Moreover, the numerical solution for lower Péclet number have less oscillations as the method has less trouble with the convection part.

For the boundary layer problem with negative velocity, the flow direction is from right to left, and flipping the boundary conditions, $u(0) = 0, \quad u(1) = 1$, the numerical solution still produces spurious oscillations as shown in Figure 2.4. Similar oscillations are noticed in the numerical solutions of the peak problem for positive and negative velocities, see Figures 2.3 and 2.4. Therefore, it is necessary to apply a stabilization method for convection-dominated flow problems.

A common stabilization method for the Galerkin method is the Streamline Upwind/Petrov Galerkin method (SUPG), but it has several disadvantages, one being that it does not guarantee to prevent non-physical under-and over-shoots [6]. Therefore, the algebraic flux correction method is chosen as the stabilization method, which has proven to prevent non-physical values for both frameworks FEA and IGA [13, 11, 9, 9, 7].

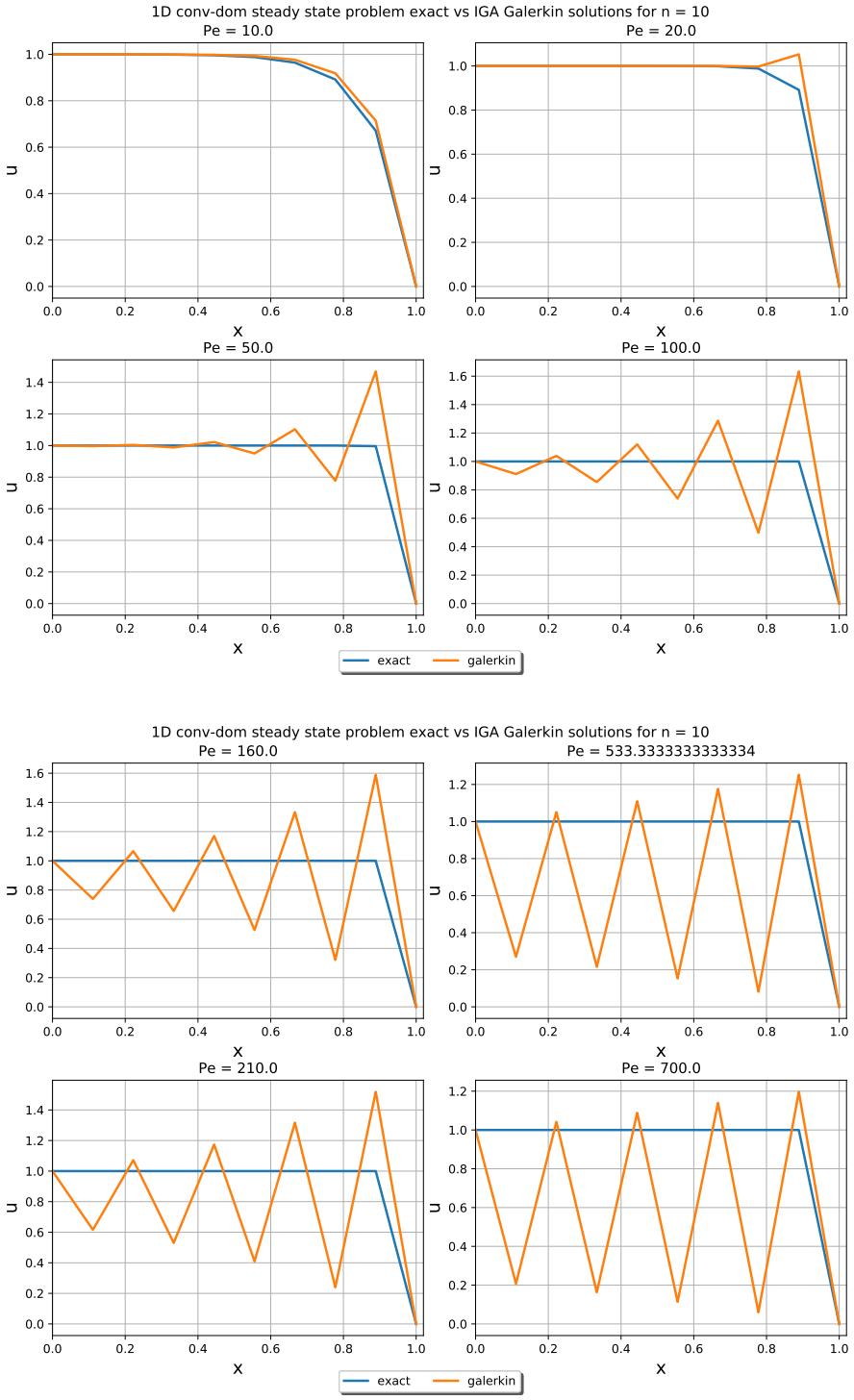


Figure 2.1: Galerkin solution for boundary layer problem with positive velocity and $n = 10$ for different Péclet numbers.

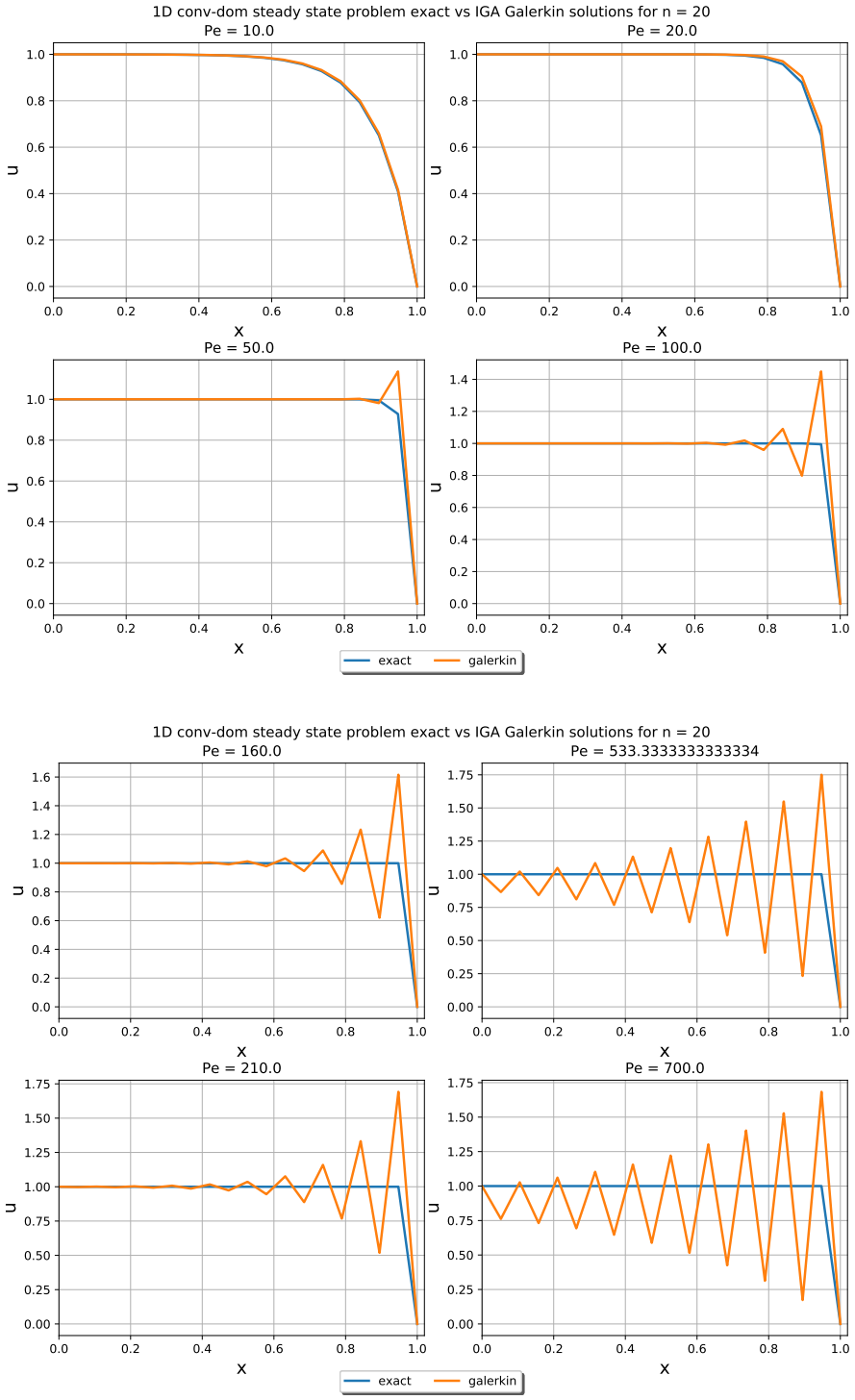


Figure 2.2: Galerkin solution for boundary layer problem with positive velocity and $n = 20$ for different Péclet numbers.

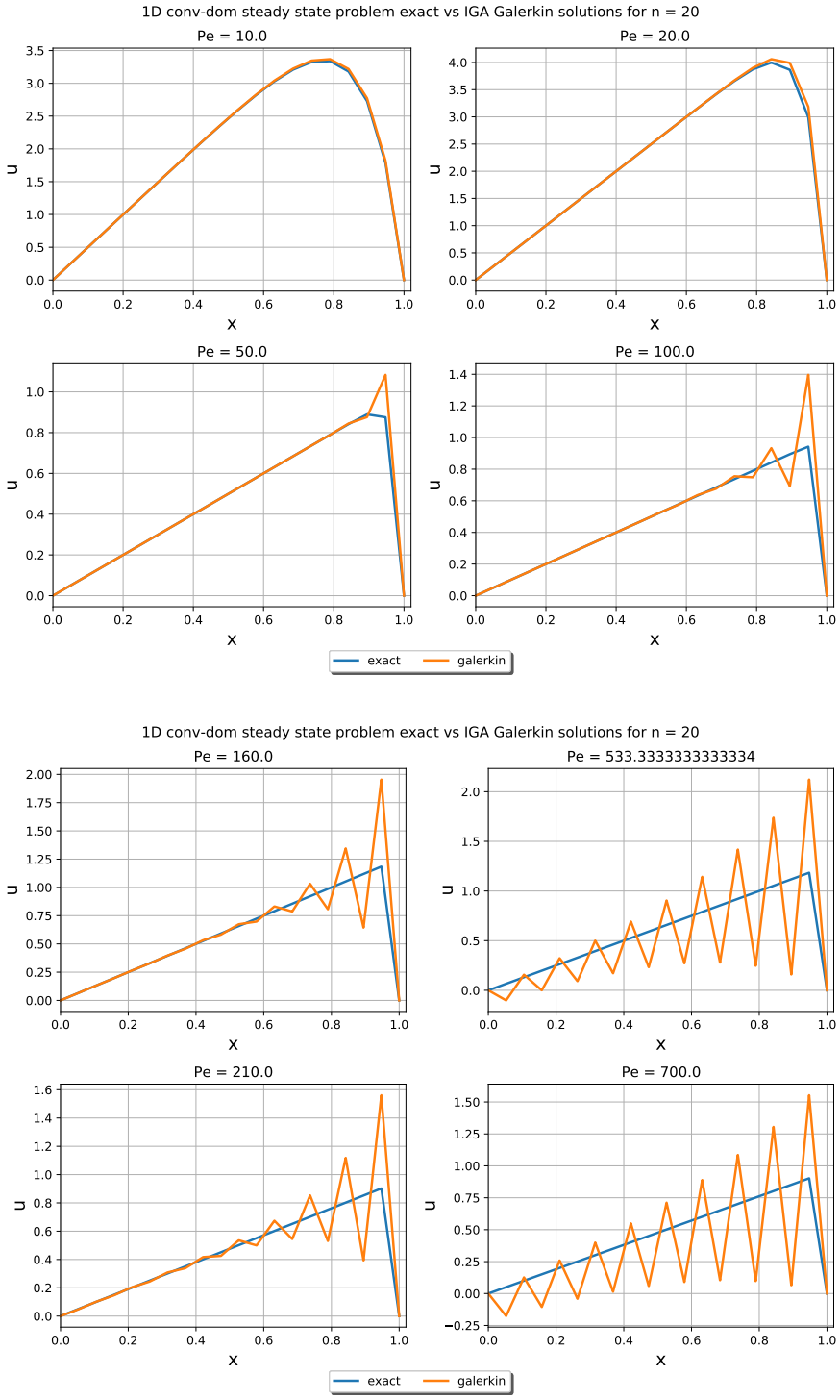


Figure 2.3: Galerkin solution for peak problem with positive velocity and $n = 20$ for different Péclet numbers.

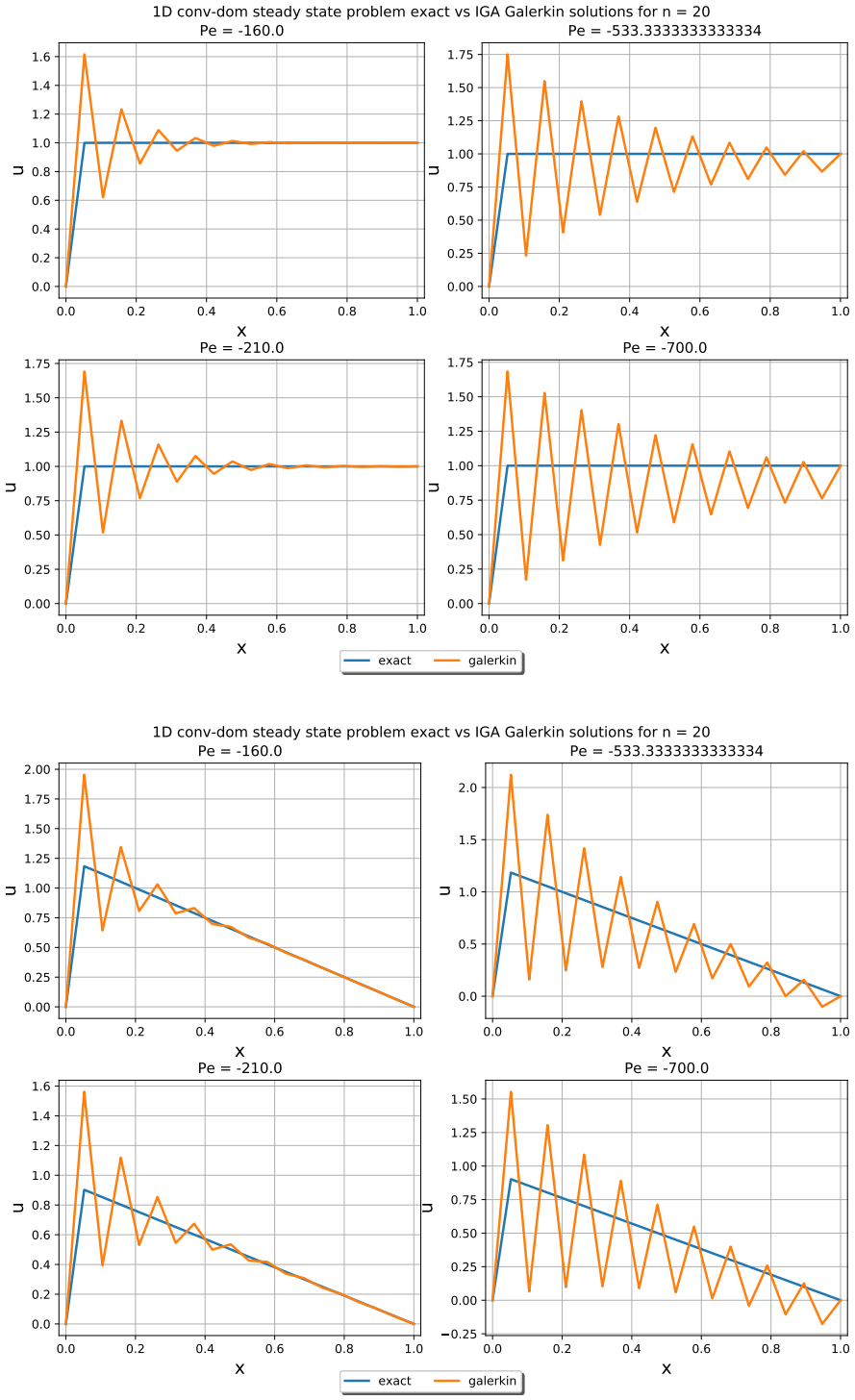


Figure 2.4: Galerkin solution for boundary layer (above) and peak (below) problems with negative velocity and $n = 20$ for different Péclet numbers.

3 Algebraic flux correction

In this chapter, the stationary convection-diffusion problem is stabilized by applying the principles of the algebraic flux correction (AFC) with **TVD-type limiting** and described in section 3.1. This algorithm is introduced by D. Kuzmin in [9] and D. Kuzmin and S. Turek in [14]. These papers as well as [11] contain the complete description of the method and a summary of this family of methods can be found in [11, 12], such as the AFC with flux limiting of FCT-type for time-dependent problems. In section 3.2 the defect correction scheme is described in regards to solving the non-linear system, followed by the application of AFC to the numerical examples in section 3.3 as shown in the previous chapter.

3.1. Algebraic flux correction with TVD-type limiting

Before applying AFC to the stationary convection-diffusion equation in one dimension, it is solved with the standard Galerkin method, which yields the following form:

$$(S - K)\mathbf{u} = \mathbf{r}. \tag{3.1}$$

All algebraic operations performed below are necessary to get the Galerkin solution free of nonphysical oscillations. In particular in the stationary case, operator $K - S$ must not contain any negative off-diagonal entries in order to be a local extremum diminishing scheme [14, 11]. The discrete diffusion operator S [10], does not cause any problem because **all** off-diagonal entries are negative unless the mesh or the diffusion tensor is highly anisotropic, which means that diffusion is direction-dependent, but in this thesis, we limit to isotropic diffusion tensors, which are direction-independent.

Discrete upwind technique is used to compute the symmetric artificial diffusion operator $D = \{d_{ij}\}$ that is added to K, but in a conservative way. D is symmetric and has zero row and column sums (to preserve conservativity of the scheme) (3.3). Square matrices that have all the properties as D are called discrete diffusion operators [13].
For the stationary convection-diffusion ODE system (2.9), the discrete convection operator, K, has to be altered to eliminate all negative off-diagonal. This can be achieved by adding an artificial discrete diffusion operator, D, which is defined as symmetric matrix [11]:

$$D = d_{ij} \quad \text{such that} \quad d_{ij} = d_{ji} \tag{3.2}$$

which have zero row and column sums

$$\sum_i d_{ij} = \sum_j d_{ij} = 0 \tag{3.3}$$

Matrix D is sparse and its nonzero off-diagonal entries d_{ij} may be positive (diffusion) or negative (antidiffusion). Define following operation of so-called conservative flux decomposition for an arbitrary discrete diffusion operator as followed applied to u [11]:

$$(Du)_i = \sum_j d_{ij}u_j = \sum_{j\neq i} d_{ij}u_j + d_{ii}u_i \tag{3.4}$$

$$\overset{(3.3)}{=} \sum_{j\neq i} d_{ij}u_j - \sum_{j\neq i} d_{ij}u_i \tag{3.5}$$

$$= \sum_{j\neq i} d_{ij}(u_j - u_i) \tag{3.6}$$

due to zero row sum property. Then the diffusive terms to node i can be decomposed into a sum of numerical fluxes as follows:

$$(Du)_i = \sum_{j\neq i} f_{ij} \quad \text{where} \quad f_{ij} = d_{ij}(u_j - u_i) \tag{3.7}$$

where f_{ij} are diffusive fluxes.

The optimal entries of the artificial diffusion operator D, a discrete diffusion operator, are given by [13, 14]:

$$d_{ij} = d_{ji} = \max\{0, -k_{ij}, -k_{ji}\}, \quad d_{ii} = -\sum_{j \neq i} d_{ij} \tag{3.8}$$

The modified discrete convection operator $L = \{l_{ij}\}$ is defined as $L = K + D$ resulting in a stabilized linear scheme where the entries of L are non-negative. In practice, D is not assembled explicitly, but entries of L are modified within an edge-by-edge approach. First, we initialize $L := K$, then each pair of nonzero off-diagonal coefficients $l_{ij}$and l_{ji} are analysed such that the smallest entries of K are set to zero and the rest three entries of K are modified to restore row/column sums. So computing d_{ij} with (3.8) and the modifications of entries in L are given by [14]:

$$l_{ii} := l_{ii} - d_{ij}, \qquad l_{ij} := l_{ij} + d_{ij}, \tag{3.9}$$

$$l_{ji} := l_{ji} + d_{ij}, \qquad l_{jj} := l_{jj} - d_{ij} \tag{3.10}$$

The resulting scheme after applying these modifications and the original scheme (2.9) differentiate in a way that can be represented as a sum of skew-symmetric diffusive fluxes $f_{ij}^d = d_{ij}(u_j - u_i)$ between neighboring nodes that have overlapping support of the basis functions. The above modifications lead to a linear scheme given as:

$$(S - L)\mathbf{u} = \mathbf{r} \tag{3.11}$$

As Godunov's theorem [4] stated, linear schemes are only first-order accurate, so to overcome this limitation a **non-linear scheme** is constructed by combing the linear high-order scheme (2.9) and low-order schemes (3.11) in a non-linear fashion given as followed [14]:

$$(S - K^*(\mathbf{u}))\mathbf{u} = \mathbf{r} \tag{3.12}$$

Here, the nonlinear discrete convection operator is defined as [7]:

$$K^*(\mathbf{u}) = L + \bar{F}(\mathbf{u}) = K + D + \bar{F}(\mathbf{u}) \tag{3.13}$$

To remove certain fractions of artificial diffusion which was added to the high-order discretization to suppress spurious oscillations and avoiding loss of accuracy in smooth regions due to excessive diffusion we will modulate the low-order scheme by some non-linear anti-diffusion operator: $\bar{F}(\mathbf{u}) = \{\bar{f}_{ij}(\mathbf{u})\}$, which has the same properties as a discrete diffusion operator, but opposite sign of D, so $-D$. Therefore, decomposition of anti-diffusion operator as a sum of anti-diffusive fluxes such as vector of all $\bar{f}_i(\mathbf{u})$ as $\bar{\mathbf{f}}(\mathbf{u})$:

$$f_i(\mathbf{u}) := (\bar{F}(\mathbf{u})\mathbf{u})_i = \sum_{j \neq i} \bar{f}_{ij}(\mathbf{u}) \tag{3.14}$$

with

$$\bar{f}_{ij}(\mathbf{u}) = \beta_{ij}(\mathbf{u})(u_i - u_j) \tag{3.15}$$

Define the coefficients β_{ij} such that anti-diffusive fluxes are relevant because the anti-diffusive fluxes are defined as the **limited** difference of high-and low-order fluxes. Since the difference between high- and low-order schemes is the added artificial diffusion applied to the solution, this is easily decomposed into fluxes so that we define β_{ij} as [6]:

$$\beta_{ij}(\mathbf{u}) = \alpha_{ij}(\mathbf{u})d_{ij} \tag{3.16}$$

where $\alpha_{ij}(\mathbf{u})$ is an **adaptive flux limiter**. When $\alpha_{ij} = 1$ no limiting is applied since the anti-diffusive fluxes will be equal to the difference of high-and low-order fluxes, which is the diffusive fluxes and returns to the original Galerkin scheme. This limiter is computed by the TVD-type limiting algorithm proposed by Kuzmin in [9]. For the description, the dependency on $(\mathbf{u})$ is omitted to simplify the equations. Let us define the raw anti-diffusive fluxes to be limited as [14]:

$$f_{ij} = d_{ij}(u_i - u_j) \quad \text{and} \quad f_{ij} = -f_{ji}, \tag{3.17}$$

Then the TVD-type limiting algorithm is given as followed[9] : for each pair of overlapping DOFs i and j such that $l_{ji} \geq l_{ij} \geq 0$:

1. Compute the sums of positive and negative anti-diffusive fluxes into i-th DOF to be limited:

$$P_i^+ \coloneqq P_i^+ + \mathsf{max}\{0, f_{ij}\}, \qquad P_i^- \coloneqq P_i^- + \mathsf{min}\{0, f_{ij}\} \tag{3.18}$$

2. Compute upper and lower bounds $Q_i^\pm$ to be imposed on the sums $P_i^\pm$:

$$Q_i^+ \coloneqq Q_i^+ + \mathsf{max}\{0, -f_{ij}\}, \qquad Q_j^+ \coloneqq Q_j^+ + \mathsf{max}\{0, f_{ij}\} \tag{3.19}$$
$$Q_i^- \coloneqq Q_i^- + \mathsf{min}\{0, -f_{ij}\}, \qquad Q_j^- \coloneqq Q_j^- + \mathsf{min}\{0, f_{ij}\} \tag{3.20}$$

3. Determine the nodal correction factors $R_i^\pm$ evaluated at the upwind DOF i:

$$R_i^+ = \mathsf{min}\{1, \frac{Q_i^+}{P_i^+}\}, \qquad R_i^- = \mathsf{min}\{1, \frac{Q_i^-}{P_i^-}\} \tag{3.21}$$

4. Compute the flux limiters:

$$\alpha_{ij} = \begin{cases} R_i^+, & \text{if } f_{ij} > 0 \\ R_i^-, & \text{otherwise} \end{cases} \tag{3.22}$$

Resulting scheme:

$$(S - L)\mathbf{u} = \mathbf{r} + \bar{F}(\mathbf{u})\mathbf{u} \tag{3.23}$$

rewritten as:

$$((S - L)u)_i = r_i + \bar{f}_i(\mathbf{u}) \tag{3.24}$$

for i being the number of the considered degree of freedom, with $\bar{f}_i(\mathbf{u})$ as [14]:

$$\bar{f}_i(\mathbf{u}) = \sum_{j \neq i} \alpha_{ij} f_{ij} \tag{3.25}$$

With this method explicit construction of the anti-diffusion $\bar{F}(\mathbf{u})$ is avoided and the final problem that is a high-resolution scheme has to be solved is:

$$(S - L)\mathbf{u} = \mathbf{r} + \bar{\mathbf{f}}(\mathbf{u}). \tag{3.26}$$

This non-linear algebraic system is solved by defect correction scheme in this thesis.

3.2. Defect correction scheme

The defect correction scheme (DCS) is an iterative method to solve non-linear algebraic systems. The general idea is to start with an initial guess and in each iteration, an evaluation is made of non-linear terms using the solution of the previous iteration. The stopping criteria are set on the desired tolerance or until the limit of iterations is achieved. The algorithm reads as followed [6]:

1. Select an initial guess $\mathbf{u}^{(0)}$, e.g.:

$$\mathbf{u}^{(0)} = 0 \text{ or low-order problem solution: } (S - L)\mathbf{u}^{(0)} = \mathbf{r} \tag{3.27}$$

2. Solve the linear system:

$$(S - L)\mathbf{u}^m = \mathbf{r} + \bar{\mathbf{f}}(\mathbf{u}^{m-1}) \tag{3.28}$$

3. Repeat step 2 until the following condition is fulfilled:

$$||(S - L)\mathbf{u}^{(m)} - \mathbf{r} - \bar{\mathbf{f}}(\mathbf{u}^{(m)})|| < \text{tol or } m = m_{\mathsf{max}}, \tag{3.29}$$
$$\tag{3.30}$$

where tol is prescribed tolerance, m_{max} is prescribed maximal number of iterations and $||\cdot||$ is a norm of interest.

Respectively, the presented algorithm clearly illustrates the idea behind the defect correction approach, but in practice an equivalent algorithm [14] is used more often:

1. Select an initial guess $\mathbf{u}^{(0)}$, e.g.:

$$\mathbf{u}^{(0)} = 0 \text{ or low-order problem solution: } (S-L)\mathbf{u}^{(0)} = \mathbf{r} \tag{3.31}$$

2. Solve the linear system for correction:

$$(S-L)\Delta\mathbf{u}^{(m)} = \mathbf{r} + \bar{\mathbf{f}}(\mathbf{u}^{(m-1)}) - (S-L)\mathbf{u}^{(m-1)} \tag{3.32}$$

   where the right hand side is the residual from the previous approximation.
3. Apply the correction to the solution:

$$\mathbf{u}^{(m)} = \mathbf{u}^{(m-1)} + \Delta\mathbf{u}^{(m)} \tag{3.33}$$

4. Repeat step 2 until the following condition is fulfilled:

$$\frac{||\Delta\mathbf{u}^{(m)}||}{||\mathbf{u}^{(m)}||} < \text{tol or } m = m_{\text{max}} \tag{3.34}$$

The necessary ingredients to implement the TVD-type limiting approach is presented and applicable to the convection-diffusion problem.

3.3. Numerical results

The results obtained after stabilizing the Galerkin method with AFC are computed for the boundary layer and peak problem defined in section 2.3. For different Péclet numbers the low-order upwind solutions and the solutions obtained by the high-resolution method: AFC of TVD-type limiting, are shown in Figure 3.1 for the boundary layer problem and in Figure 3.3 for the peak problem. Moreover, corresponding $P^{\pm}, Q^{\pm}, R^{\pm}$ values of the AFC algorithm after one iteration of the DCS results are included to show understand how the TVD-type limiting works (see Figures 3.2 and 3.4). At the first iteration for both problems, the P^{+} values show how much flux wants to go into the node (knots), while the Q^{+} values show which bounded flux is allowed to go into the node (knots). The ratio of them is defined by the R^{+} values, which evaluates how strong the limitation can be for the incoming fluxes.

It can be seen as the Péclet number increases, the flux values increase and therefore the R^{+} values increases to prevent having oscillations in the solution. At regions where the solution is smooth, the flux values are zero and therefore how much these fluxes are limited is not of importance. These AFC results give a better understanding of how the limiter behaves and will assist in the development of potential neural network architecture that can mimic the behaviour of the limiter.

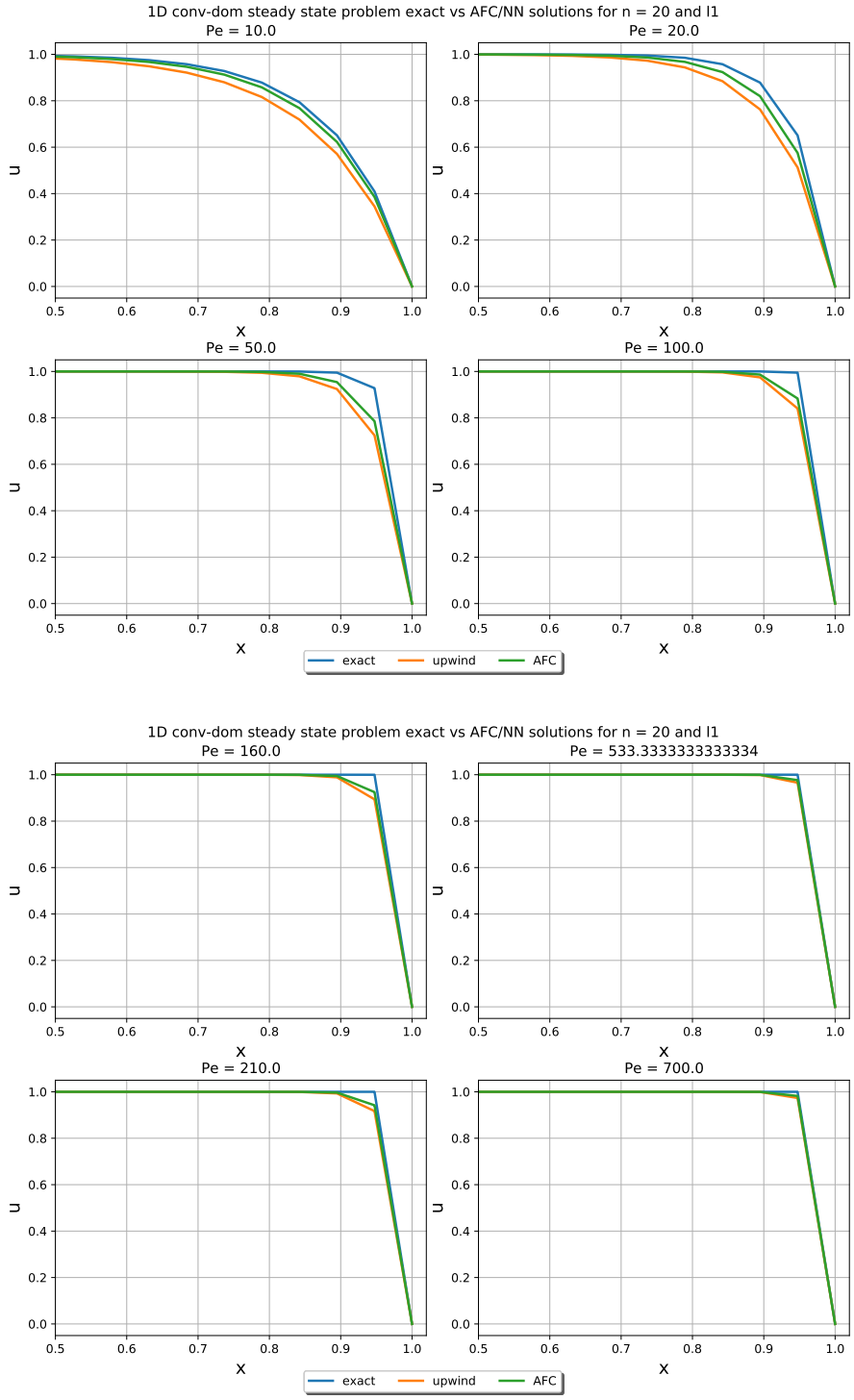


Figure 3.1: Upwind (low-order) and AFC (stabilized high-order) solution for boundary layer problem with positive velocity and $n = 20$ for different Péclet numbers.

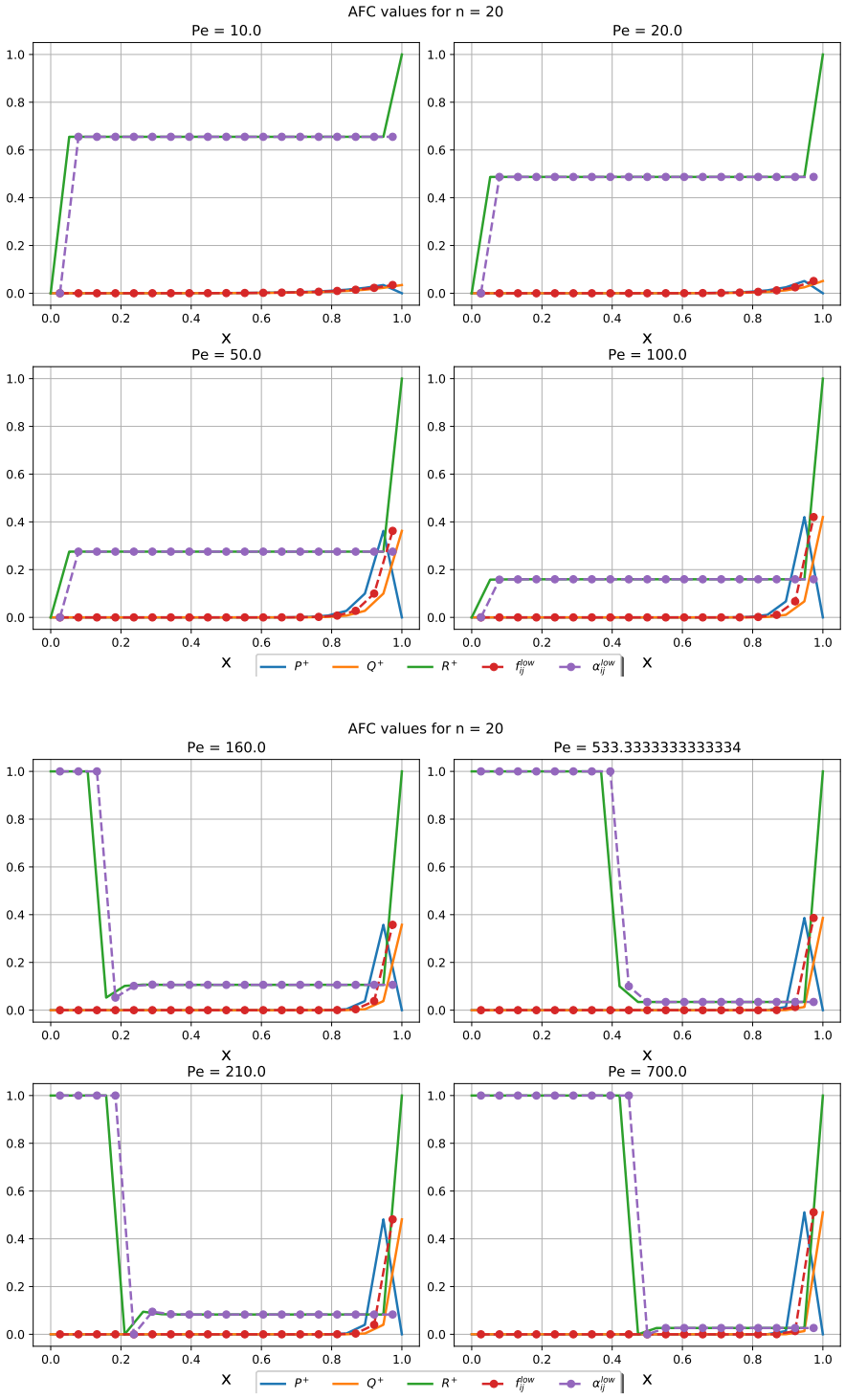


Figure 3.2: AFC values after one iteration for the boundary layer problem with positive velocity and $n = 20$ for different Péclet numbers.

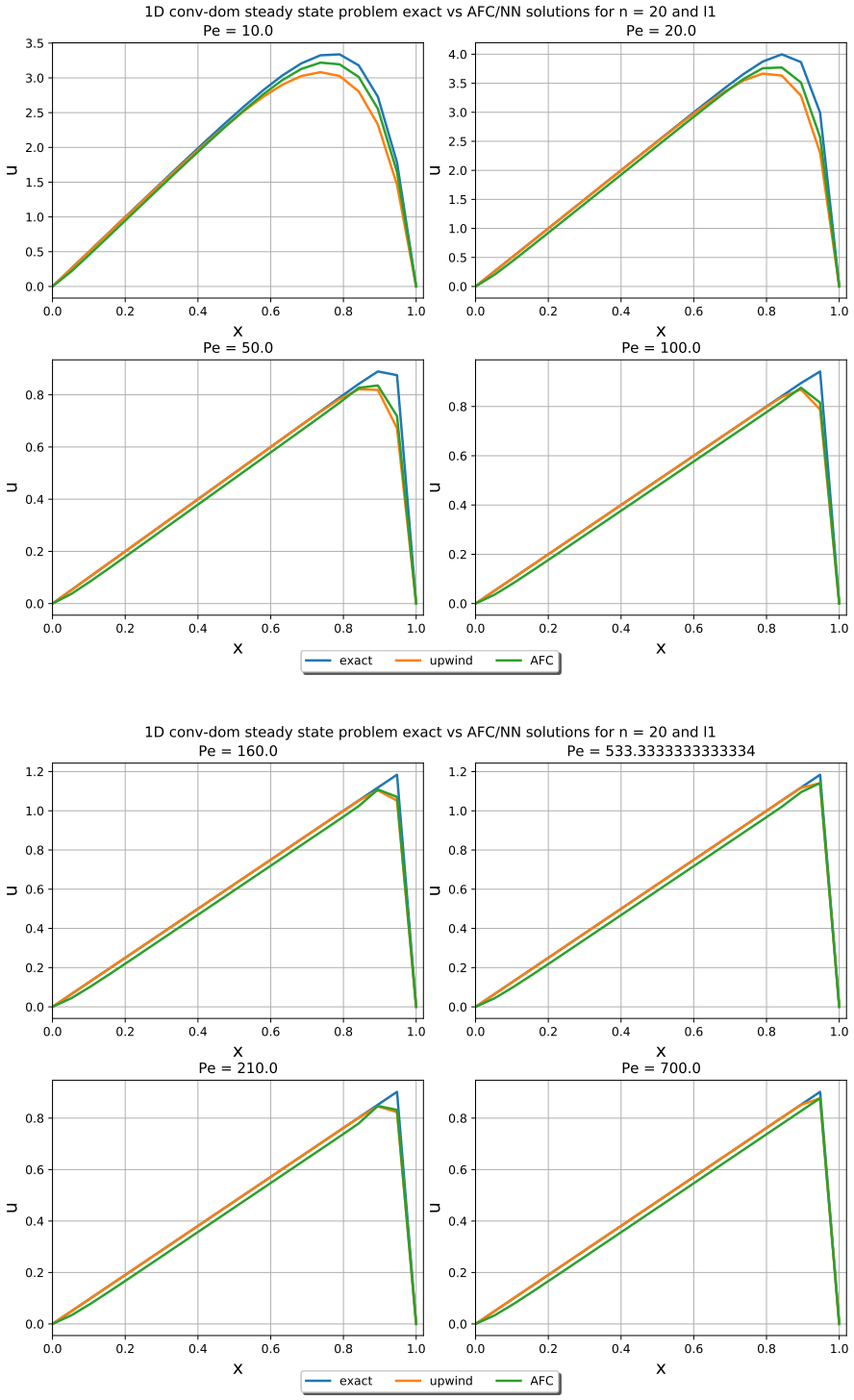


Figure 3.3: Upwind (low-order) and AFC (stabilized high-order) solution for peak problem with positive velocity and $n = 20$ for different Péclet numbers.

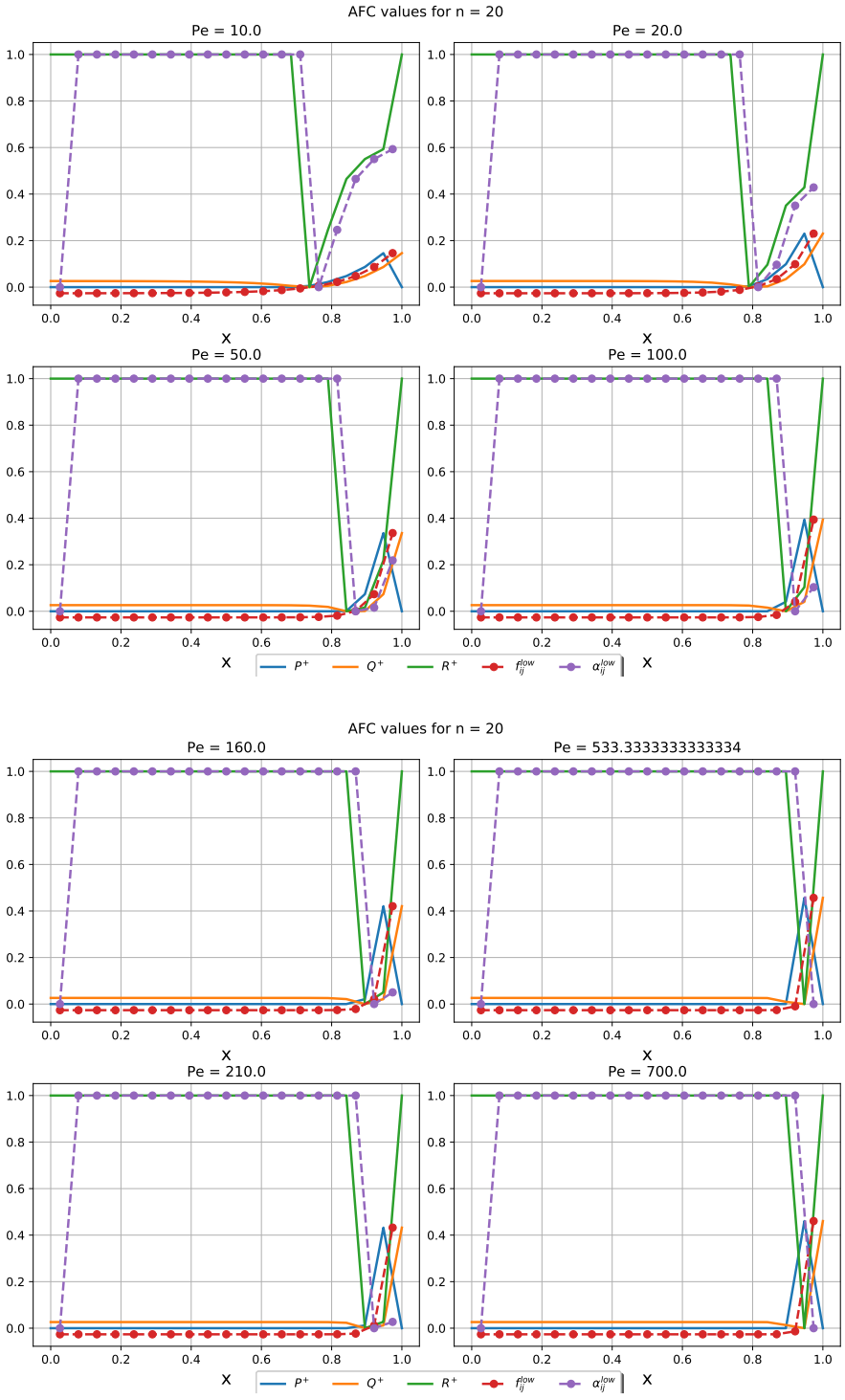


Figure 3.4: AFC values after one iteration for the peak problem with positive velocity and $n = 20$ for different Péclet numbers.

4 Scientific machine learning

This chapter gives a short overview of a neural network and its components, such as the architecture of a feedforward neural network is explained in section 4.1, and the important aspects of training them in section 4.2.

4.1. Network architecture

The most classical type of deep learning models are **feedforward neural networks** or **multilayer perceptrons** (MLPs), which are also called **Deep feedforward networks**. The goal of a feedforward network is to approximate some function f. For example for a classifier, $\mathbf{y} = f(\mathbf{x}; \theta)$ that maps an input **x** to an output **y** and learns the value of the parameters θ, such that f is the best function approximation. The reason to call these models feedforward is that the information propagates only in one direction, so no feedback connections are included where the output is fed back to itself. They are called networks because they are composed of many different functions.

For example, taking three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$, connected in a chain, in the form of $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. These are common structures of neural networks. Here $f^{(1)}$ is called the first layer or **input layer** of the network, $f^{(2)}$ the second layer and so on. The length of the chain gives the **dept** of the model from which the term 'deep learning' came. The final layer is the **output layer**. Hidden layers are the layers in between the input and output layer and here the input **x** is evaluated without knowing the desired output **y**. The neural in these networks appear since they are inspired by neurons in biology, like the human brain. Each hidden layer consists of vectors and the elements in that vector resembles neurons. The number of neurons of these hidden layers determines the **width** of the model [5].

An example of a neuron that is connected by directed links in a neural network is shown in Figure 4.1 with the important descriptions of the parameters given in Table 4.1.

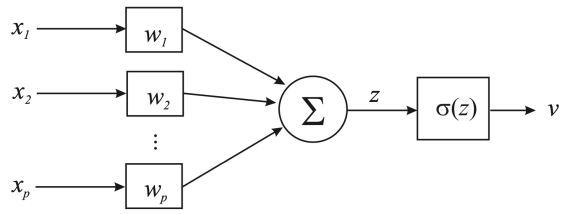


Figure 4.1: Artificial neuron [1]

Parameter	Description
x_i	i^{th} input of the neuron
w_i	Adaptive weight for x_i
z	Weighted sum of inputs: $z = \sum_{i=1}^{p} W_i x_i = \mathbf{W^T x}$
$\sigma(z)$	Activation function
v	Output of neuron

Table 4.1: Parameters of artificial neuron

Each artificial neuron has an activation function, which takes in the summed weighted inputs and maps these onto the functions and returns an 'activation' value as output. This correlates to its own mapping of the input with a certain output. During training, the weights are adjusted and the neuron learns to make better approximation of the mapping from the input to the output. The activation functions play a crucial role for neural networks to describe nonlinear functions, otherwise it would only describe linear functions despite the setting. A bias can be added to shift the activation of a neuron in order to find optimal weights. The most common used activation functions are shown in Figure 4.2.

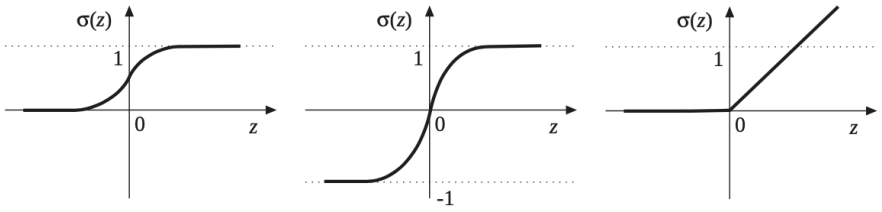


Figure 4.2: From left to right; Sigmoid, tangent hyperbolic (tanh), Rectified Linear Unit (ReLU) [1]

The equations of the activation functions are:

Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$
Tangent hyperbolic (tanh)	$\sigma(z) = \frac{2}{1+e^{-2z}-1}$
Rectified Linear Unit (ReLU)	$\sigma(z) = \begin{cases} 0 & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases}$

The network's output are values, which makes it regression-based problem instead of the (traditional) classification problem.

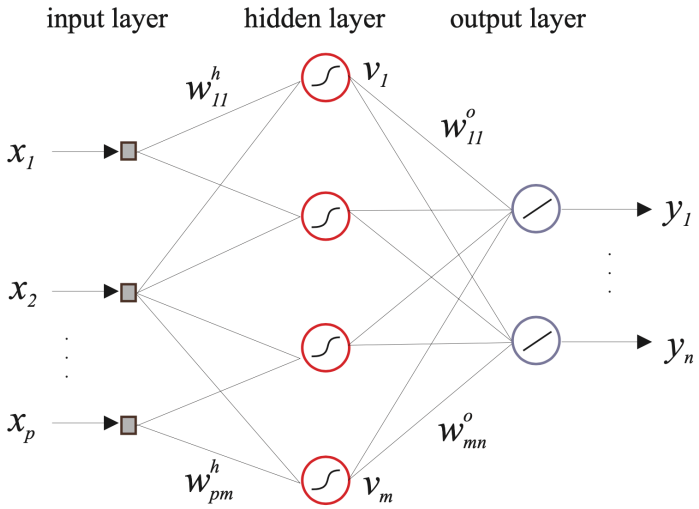


Figure 4.3: Feedforward neural network [1].

- Activation of hidden-layer neuron j:

$$z_j = \sum_{i=1}^{p} W_{ij}^h x_i + b_j^h$$

- Output of hidden-layer neuron j:

$$v_j = \sigma(z_j)$$

- Output of output-layer neuron l:

$$y_l = \sum_{j=1}^{m} W_{jl}^o v_j + b_l^o$$

The input-output mapping can be given in matrix notation, such as:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X}_b \mathbf{W}^h, \quad \mathbf{X}_b = [\mathbf{X}\ \mathbf{1}] \\ \mathbf{V} &= \sigma(\mathbf{Z}) \\ \mathbf{Y} &= \mathbf{V}_b \mathbf{W}^o, \quad \mathbf{V}_b = [\mathbf{V}\ \mathbf{1}] \end{aligned}$$

An example of a feedforward neural network is shown in Figure 4.3. The general goal of training a neural network is to find the optimal weight matrix **W** and sometimes bias vector **B**, given as $\theta = [\mathbf{W}\ \mathbf{B}]$, by minimizing a loss function $L(f(\mathbf{x};\theta), \mathbf{y})$ for all inputs **x**. The following steps occur during this training, called supervised learning, after initialization of the weights and (optional) the bias to small random values:

1. **Forward computation**: feed the input data into the network and proceeds through the hidden layers to compute the output and the loss of the prediction $f(\mathbf{x};\theta)$ and target **y**

2. **Backward computation**: calculate the gradients of the loss function with respect to the weights, known as the process of back-propagation: $\nabla L(\mathbf{W}_n) = [\frac{\partial L}{\partial w_i}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_M}]^T$
3. **Optimization**: adjust the weights based on the gradients to minimize the loss, known as the process of gradient descent: $\mathbf{W}_{n+1} = \mathbf{W}_n - \alpha_n \nabla L(\mathbf{W}_n)$

4.2. Training and regularization

Minimizing the loss function by a neural network is called training. It approximates the unknown nonlinear function that is underlying in the given input and output data mapping. The loss function is usually obtained by computing, for example, the mean square error as:

$$L(f(\mathbf{x};\theta),\mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i,\theta) - y_i)^2.$$

An optimization algorithm is used to handle this training process. The most common one is the Stochastic Gradient Descent (SGD), which is based on firs-order gradients and updates the weights and biases in the direction of the steepest descent of the loss function. This algorithm computes the gradients with respect to a subset of the data and the size of it is called the batch size, which is one of the hyperparameters that are optimized empirically. Another hyperparameter in the SGD is the learning rate, which determines what the step size for the weight/bias update at every iteration.

Another optimizer is Adam, which is a modification of the SGD algorithm and computationally efficient. It requires less tuning and increases the step size of the update if the direction of the gradient keeps descending, but decreases the step size over time, specifically if the gradients are large. This makes it a powerful optimizer and it tends to outperform SGD in practice.

To regulate the performance of the neural network on unobserved data, also called generalization, three sets are created from the data, namely training, validation and test set. The training set is to train the neural network and the validation set is used to select the right hyperparameters, such as the number hidden layers or the number of predetermined training iterations, also called epochs. The test set is an effort to improve the performance and verify that the hyperparameters have not been over-fitted to the validation set. This is often observable when the validation loss start to increase after some epochs while the training loss decreases.

5 Learning-based flux limiting

In this chapter, the learning-based flux limiting approach is presented in section 5.1 to attempt to mimic AFC with neural networks models as described in section 5.2, followed by the descriptions of the parameters and loss functions used to train the neural networks in section 5.3. The results are given in section 5.4 for each model. Finally, an analysis and discussion are given based on the results in section 5.5.

5.1. Reverse Engineering α_{ij}

The neural network (NN) learning-based flux limiting approach is developed using AFC with TVD-type limiting, henceforth called the AFC limiter, to solve the stationary convection-diffusion problem in one dimension. To solve this non-linear problem the defect correction scheme (DCS) is used, which takes up extra computational steps involving the AFC limiter and therefore is time-consuming. These calculations are circumvented by reverse engineering the final limiting coeffcients α_{ij} as produced by the AFC limiter. The converged AFC solution, $\mathbf{u}^{\text{AFC}}$, obtained from the DCS at iteration $m = n$, and the low-order upwind solution, $\mathbf{u}^{\text{low}}$, at $m = 1$ are used in the system,

$$(S - L)\mathbf{u}^m = \mathbf{r} + \bar{\mathbf{f}}(\mathbf{u}^{m-1}),$$

that is solved by the DCS with m as iteration. After filling these solutions and rewriting it the following is solved:

$$\bar{\mathbf{f}}(\mathbf{u}^{\text{low}}) = (S - L)\mathbf{u}^{AFC} - \mathbf{r}, \tag{5.1}$$

where the anti-diffusive correction for the i^{th} DOF $\bar{\mathbf{f}}_i(\mathbf{u}^{\text{low}})$ is computed as followed:

$$\bar{\mathbf{f}}_i(\mathbf{u}^{\text{low}}) = \sum_{j \neq i} \bar{\alpha}_{ij} f_{ij}(\mathbf{u}^{\text{low}}),$$

where $f_{ij}(\mathbf{u}^{\text{low}}) = d_{ij}(u_i^{\text{low}} - u_j^{\text{low}})$ is the anti-diffusive flux with d_{ij} as the artificial diffusion coefficient and $\bar{\alpha}_{ij}$ is the aggregated adaptive flux limiter, which now represents the one step limiter and is called **the reverse engineered** α_{ij}, or in short α_{ij}^{REV}. To compute this, an example with five basis functions ($n = 5$) is as followed: Let $v > 0$ such that

$$\begin{aligned}
\bar{\mathbf{f}}_0(\mathbf{u}^{\text{low}}) &= +\alpha_{01} f_{01} && \Rightarrow \quad \alpha_{01} = \bar{\mathbf{f}}_0(\mathbf{u}^{\text{low}})/f_{01} \\
\bar{\mathbf{f}}_1(\mathbf{u}^{\text{low}}) &= -\alpha_{01} f_{01} + \alpha_{12} f_{12} && \Rightarrow \quad \alpha_{12} = (\bar{\mathbf{f}}_1(\mathbf{u}^{\text{low}}) + \alpha_{01} f_{01})/f_{12} \\
\bar{\mathbf{f}}_2(\mathbf{u}^{\text{low}}) &= -\alpha_{12} f_{12} + \alpha_{23} f_{23} && \Rightarrow \quad \alpha_{23} = (\bar{\mathbf{f}}_2(\mathbf{u}^{\text{low}}) + \alpha_{12} f_{12})/f_{23} \\
\bar{\mathbf{f}}_3(\mathbf{u}^{\text{low}}) &= -\alpha_{23} f_{23} + \alpha_{34} f_{34} && \Rightarrow \quad \alpha_{34} = (\bar{\mathbf{f}}_3(\mathbf{u}^{\text{low}}) + \alpha_{23} f_{23})/f_{34} \\
\bar{\mathbf{f}}_4(\mathbf{u}^{\text{low}}) &= -\alpha_{34} f_{34}
\end{aligned}$$

Note that for $f_{ij} = 0$ the α_{ij} is set to 1 similarly to the AFC limiter and the value of the last node is not used, but serves as a verification. For $v < 0$, the anti-diffusion correction $\bar{\mathbf{f}}(\mathbf{u}^{\text{low}})$ is computed in a similar way, but starts with computing first the α_{34} term as given by:

$$\begin{aligned}
\alpha_{34} &= \bar{\mathbf{f}}_4(\mathbf{u}^{\text{low}})/f_{34} \\
\alpha_{23} &= (\bar{\mathbf{f}}_3(\mathbf{u}^{\text{low}} + \alpha_{34} f_{34})/f_{23} \\
\alpha_{12} &= (\bar{\mathbf{f}}_2(\mathbf{u}^{\text{low}}) + \alpha_{23} f_{23})/f_{12} \\
\alpha_{01} &= (\bar{\mathbf{f}}_1(\mathbf{u}^{\text{low}}) + \alpha_{12} f_{12})/f_{01}.
\end{aligned}$$

AFC algorithm vs NN learning-based flux limiting

1. High-order method:
   (e.g. Galerkin FEM)

$$(S-K)\mathbf{u}=\mathbf{r}$$

2. Low-order method:

$$(S-L)\mathbf{u}=\mathbf{r}$$

$L=K+D$ results by adding the artificial diffusion D

AFC algorithm (TVD-type)

3. Apply the correction to the solution:

$$(S-L)\mathbf{u}=\mathbf{r}+\bar{\mathbf{f}}(\mathbf{u})$$

where the anti-diffusive correction at i is:

$$\bar{f}_i(\mathbf{u})=\sum_{j\neq i}\alpha_{ij}(\mathbf{u})f_{ij}(\mathbf{u}),$$
$$f_{ij}(\mathbf{u})=d_{ij}(u_i-u_j)$$
$\alpha_{ij}(\mathbf{u}^{(m-1)})$ is determined by the algorithm: (3.17)-(3.22)

NN algorithm

3. Apply the correction to the solution:

$$(S-L)\mathbf{u}^{\text{AFC}}=\mathbf{r}+\bar{\mathbf{f}}(\mathbf{u}^{\text{low}})$$

where the anti-diffusive correction at i is:

$$\bar{f}_i(\mathbf{u}^{\text{low}})=\sum_{j\neq i}\alpha_{ij}^{\text{REV}}(\mathbf{u}^{\text{low}})f_{ij}(\mathbf{u}^{\text{low}}),$$
$$f_{ij}(\mathbf{u}^{\text{low}})=d_{ij}(u_i^{\text{low}}-u_j^{\text{low}})$$
$$\alpha_{ij}^{\text{REV}}(\mathbf{u}^{\text{low}})=NN\ (\mathbf{u}^{\text{low}})$$

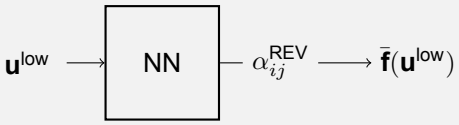


Figure 5.1: A summary of the AFC algorithm and NN learning-based flux limiting algorithm for solving the nonlinear system.

5.2. Development of neural network models

The learning-based flux limiting approach is based on supervised learning where the neural network approximates the underlying function that is known to us through the relationship between the input variables and output variables. In this case the input data consists of some form of the upwind solutions and the output data of the reverse engineered α_{ij} as shown in figure 5.1. This relationship or mapping provides the neural network the ability to learn the behaviour of the AFC limiter, which is the goal. For this purpose the feedforward neural network is used and via observing the training and validation set losses or errors the architecture and hyperparameters were optimized. However, these parameters might not be the most optimal ones and could be automatically tune by programs, like Ray Tune [16], but these are usually computationally costly. The architecture is optimized by initially using one hidden layer with few neurons and then increasing the layers and number of neurons per layer when the training and validation losses were decreasing and the performance was improving.

An initial model is developed by fixing the input values of the neural network as given in Figure 5.2. These input values takes into account the neighboring low-order solutions corresponding to the output value α_{ij}^{REV}, which provides more information about the solution around element $[u_i, u_{i+1}]$ that can guide the neural network in understanding the relationships/mapping better. This model is defined as the **solution model**, as shown in Figure 5.3 (top left).

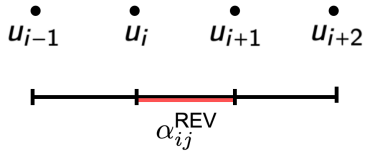


Figure 5.2: The low-order upwind solution as input and reverse engineered α_{ij} as output corresponding to element $[u_i, u_{i+1}]$ for the neural network.

As an alternative, another model approach is considered that has besides the solution input values also the anti-diffusive flux values $f_{ij} = d_{ij}(u_i - u_j)$, where u_i values are the low-order upwind solutions and contains the physics aspect namely the artificial diffusion coefficient d_{ij}. Moreover, these fluxes are also used to compute the AFC limiter in algorithm: (3.17)-(3.22), which also provides more information to the model to predict the α_{ij}^{REV} values. This approach defines the **solution + fluxes model** as shown in Figure 5.3 (top right).

Yet another alternative is to compensate for the mesh size by taking the gradients of the low-order solution as the input values because the mesh size determines the numerical accuracy of the solutions and might influence the accuracy of the neural network as well if is not filtered out during training. For example, fine meshes (equivalent to small mesh sizes) are numerically more accurate than coarse meshes (equivalent to large mesh sizes). Therefore, taking the gradient of the low-order solution incorporates the mesh size dependency as smaller mesh sizes can determine a more accurate solution. This defines the **gradient model** as shown in Figure 5.3 (bottom left).

The final model approach developed is inspired from the last two previous models, namely by combining the gradient values of the upwind solution with the anti-diffusive fluxes, to get the best of both models in one. This is defined the **gradient + fluxes model**, as shown in Figure 5.3(bottom right).

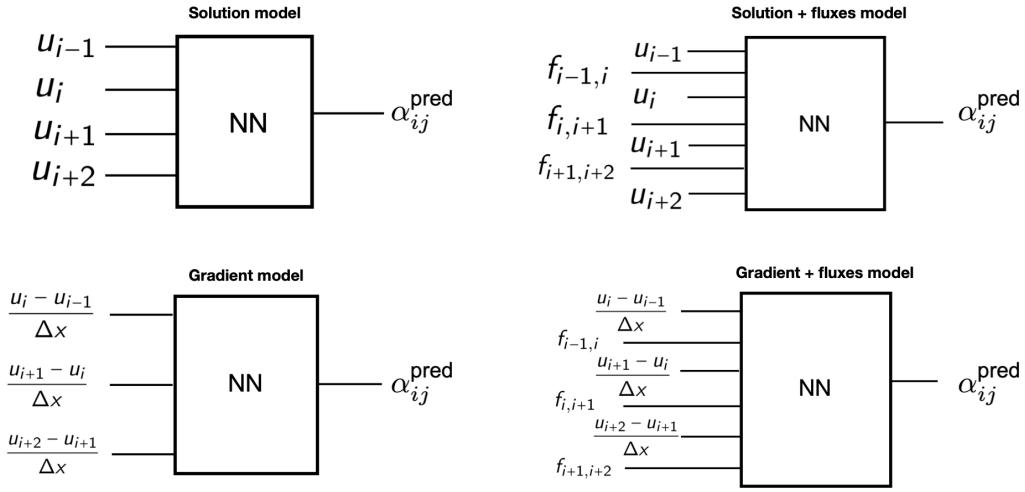


Figure 5.3: The four model approaches.

To sum up, the four potential model approaches are given in Figure 5.3 with the corresponding input and output variables. These models are trained with the solutions of the **boundary layer problem** defined in (2.21) and to find a suitable data that represents the task of limiting similar to AFC the best, two data sets, **single-mesh data** and **multi-mesh data**, were tested. To generate enough data to train a neural network 144 various Péclet numbers ranging from 8 to 125 were used and other parameters are summarized in Table 5.1.

Parameters	**single-mesh data**	**multi-mesh data**
number of basis functions (n)	10	10, 25, 50
artificial diffusion coefficient (d_{ij})	0.36-0.5	0.36-0.5
velocity constant(s) (v)	0.72-1	0.72-1
diffusion coefficient (d)	0.008 - 0.09	0.008 - 0.09
solution profiles	upwind BL problem	upwind BL problem
Péclet number (Pe)	8 to 125	8 to 125

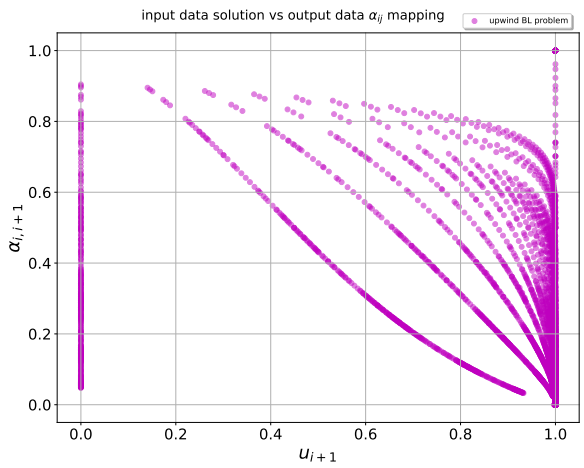
Table 5.1: Single-mesh and multi-mesh data generated by the solutions of the boundary layer (BL) problem to train the neural network models and consists of 1026 data points and 9348 points respectively. For each data, the training data is 70% of the data, validation data is 23% and test data is 7%.

To understand the relationship between the input variables and output variable three examples of the input and output mapping with multi-mesh data for the models is given in Figure 5.4. Firstly, the solution input value u_{i+1} versus the output value $\alpha_{i,i+1}^{\text{pred}}$ is mapped in Figure 5.4a. Note that for zero or one solution values, the alpha values become irrelevant because the boundary conditions have to be satisfied here. The boundary layer solution is steepest around $x = 1$, therefore more limiting is required here and consequently the alpha values become mostly between zero and one. This is in general for all mappings more important at high Péclet values, therefore, the alpha values are closer to zero, while at low Péclet values the solution is not that steep so alpha values can be larger. For multi-mesh data, on fine meshes more points are evaluated and therefore more accurate limiting is possible.

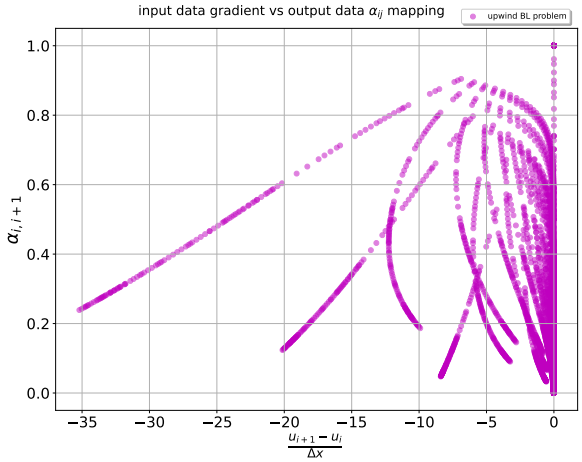
Secondly, the gradient input value $\frac{u_{i+1}-u_i}{\Delta x}$ versus the output value $\alpha_{i,i+1}^{\text{pred}}$ is mapped in Figure 5.4b. For the boundary layer problem, the gradient becomes more negative as the solution decreases, so more limiting is needed and the alpha values become smaller. At straight solution regions the differences between the points are zero and thus the alpha values can be any value between zero or one.

Lastly, the anti-diffusive flux input value $f_{i,i+1}$ versus the output value $\alpha_{i,i+1}^{\text{pred}}$ is mapped in Figure 5.4c. Note that the previous mapping can be interpreted as the opposite of this mapping because both mappings take the difference of the solution values, but in the opposite way. Besides, this mapping does not divide by the mesh size, but is multiplied with the artificial diffusion coefficient $d_{i,i+1}$, so it contains the physics of the problem. Moreover, in the mapping a large slope is visible, which corresponds to the last solution points close to the steep region, whereas the other mapping between flux values zero and 0.1 corresponds to the steep region of the solution.

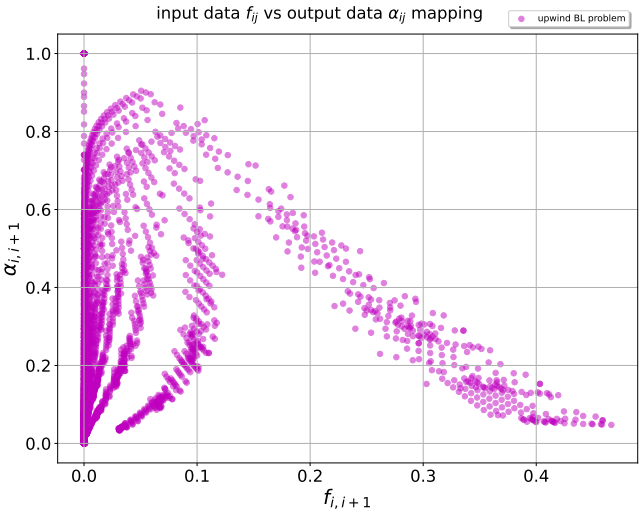
Furthermore, high Péclet solutions have steeper regions and therefore have larger fluxes, but these have to be limited and therefore correspond to smaller alpha values. Fluxes around zero do not need to be limited and therefore can have alpha value zero, one or in between because that does not affect the solution results. This also means that a small error in the α_{ij} predictions for a large flux can create overshoots in the solution, so it is important that the predictions for large fluxes are as small as possible equivalent to being sufficiently accurate.



(a) Input-output mapping of solution value u_{i+1}



(b) Input-output mapping of gradient value $\frac{u_{i+1}-u_i}{\Delta x}$



(c) Input-output mapping of anti-diffusive flux value $f_{i,i+1}$

Figure 5.4: One input value to output value mapping for the different models with the solution value u_{i+1}, gradient value $\frac{u_{i+1}-u_i}{\Delta x}$ and anti-diffusive flux $f_{i,i+1}$ value obtained by the upwind solutions of the boundary layer (BL) problem and mulit-mesh data.

5.3. Training: loss functions and hyperparameters

The goal of training a neural network is to minimize the loss functions, so a suitable measure of the error between the output of the network and the given target has to be considered. A straightforward choice is taking the Mean Squared Error (MSE) of the target, α_{ij}^{REV}, and the output of the neural network, $\alpha_{ij}^{\text{pred}}$. However, the limiter α_{ij}^{REV} is a dimensionless number between 0 and 1 and contains no other information about the anti-diffusive fluxes that have to be limited. Therefore, other loss functions are considered where suitable terms are included.

In AFC a limited amount of the anti-diffusive fluxes is added in a non-linear fashion to the system to suppress the non-physical oscillations from reoccurring. This is defined by the anti-diffusive correction $\bar{\mathbf{f}}(\mathbf{u})$, which sums up the limited anti-diffusive fluxes over the domain and is given as $\alpha_{ij} f_{ij}(\mathbf{u})$. This is represents a more realistic limiting behaviour and therefore is used as alternative second loss function. Note that the output of the network is the aggregated α_{ij} value and to compute the anti-diffusive fluxes f_{ij} the upwind solution $\mathbf{u}^{\text{low}}$ is used, which also forms the basis as the input of the neural network models.

A third loss function is considered as well, where the explicit computation of the anti-diffusive correc-

tion as given by (3.25) is taken into account, but it also inherits the non-linear behaviour of adding the fluxes similar to the AFC algorithm. Below the explicit loss function formulas are given, where N is the batch size and n are the numbers between 1 and N. The loss functions are applied to each model and compared with each other for each model.

LF1. MSE of predicted and target α_{ij} - **alpha loss function**:

$$\text{LF1} = \frac{1}{N}\sum_{n=1}^{N}(\alpha_{ij}^{\text{pred}(n)} - \alpha_{ij}^{\text{REV}(n)})^2 \tag{5.2}$$

LF2. MSE of predicted and target $\alpha_{ij} f_{ij}(\mathbf{u}^{\text{low}})$ - **limited (anti-diffusive) fluxes loss function:**

$$\text{LF2} = \frac{1}{N}\sum_{n=1}^{N}\left((\alpha_{ij}^{\text{pred}} f_{ij})^{(n)} - (\alpha_{ij}^{\text{REV}} f_{ij})^{(n)}\right)^2 \tag{5.3}$$

LF3. MSE of predicted and target $\bar{\mathbf{f}}(u^{\text{low}})$ - **limited (anti-diffusive) correction loss function:**

$$\text{LF3} = \frac{1}{N}\sum_{n=1}^{N}\left((\bar{f}^{\text{pred}}(\mathbf{u}^{low}))^{(n)} - (\bar{f}^{\text{REV}}(\mathbf{u}^{low}))^{(n)}\right)^2, \tag{5.4}$$

where $\bar{\mathbf{f}}_i^{\text{REV}}(\mathbf{u}^{\text{low}}) = \sum_{j\neq i}\alpha_{ij}^{\text{REV}} f_{ij}(\mathbf{u}^{\text{low}})$.

Here $\alpha_{ij}^{\text{pred}}$ is the output of the NN and α_{ij} values are the reversed engineered α_{ij} , abbreviated as α_{ij}^{REV} and described in section 5.1. For the third loss function, LF3, the error is computed for the $\bar{\mathbf{f}}^{\text{pred}}(\mathbf{u}^{\text{low}})$ and requires to have neighboring α_{ij} and f_{ij} values as described in section 5.1, therefore, it requires a complete different split of the data to achieve the training, validation and test sets than for LF1 and LF3 where the data is randomly shuffled. This causes the training to be more computationally inefficient, which can play a major role in the results. The architecture and hyperparameters used in each model approach are shown in Table 5.2.

Parameters	**All models with LF1 and LF2**	**All models with LF3**
Hidden layer (HL)	3	3
Neurons per HL	12, 15, 15	12, 20, 20
HL activation function	tanh	tanh
OL activation function	Sigmoid	Sigmoid
Optimizer	Adam	Adam
Learning rate	0.001	0.001
Epochs	1000	30
Batch size	64	n (number of basis functions)
Initial weights	Xavier normalized distribution	Xavier normalized distribution
Initial biases	$\mathcal{N}(0, 0.25)$	$\mathcal{N}(0, 0.25)$

Table 5.2: Hyperparameters for the different neural network models with loss function 1, 2 and 3.

The tanh activation function is considered in each hidden layer based on the mapping of the input and output data of the models. The Sigmoid activation function is considered in the output layer to keep the output value of the model between 0 and 1 since the α_{ij} is between these values. Here the Xavier normalized distribution, also known as Glorot initialization, is the normal distribution, $\mathcal{N}(0, \sigma^2)$, where

$$\sigma = gain \times \sqrt{\frac{2}{fan_{\text{in}} + fan_{\text{out}}}},$$

and fan_{in} is for example the number neurons in the input layer connected with weights to a hidden layer and fan_{out} the number of neurons in the hidden layer. The idea of Xavier normalized distribution is so

that the neural network converges faster [3] for tanh activation function and the recommended gain according to the PyTorch function documentation is 5/3 [15]. It was also observed that stable training is achieved with this initialization of the weights in combination with the bias initialization with the normal distribution.

5.4. Results

After training each model, the results for the two data sets, single-and multi-mesh data, are compared with each other on the trained boundary layer problem. To evaluate the robustness of the models, they are also tested on an untrained **peak problem** as defined in (2.23). These problems are tested on different meshes ($n = 5, 10, 20, 50$) and inside and outside trained range of Péclet values ($Pe = 4-700$).

To test and compare the performance of each of the models the L^2-norm of the relative error of the AFC solution, u^{AFC}, the neural network solution, u^{NN}, which is computed with the predicted α_{ij} versus the exact solution, u^{exact}, as followed (the computation is found in the appendix):

$$\frac{||u^{\text{AFC}} - u^{\text{NN}}||_2}{||u^{\text{exact}}||_2}$$

Here the relative error defines the accuracy of the solution obtained by the neural network, in other words, if the absolute error of the AFC and NN solution compared to the exact solution is significant (small or large). For example, the absolute error of the AFC and NN solution is small in the case when the exact solution L^2 norm is large, consequently leading to higher accuracy of the NN solution. This error is calculated for each loss function for all models for comparison.

Moreover, the L^2-norm of the α_{ij} error is computed as vectors, so $\forall i, j$:

$$||\alpha_{ij}^{\text{pred}} - \alpha_{ij}^{\text{AFC}}||_2 = \left(\sum_{m=0}^{n-2} |\alpha^{\text{pred}}(m)_{ij} - \alpha^{\text{AFC}}(m)_{ij}|^2 \right)^{1/2}$$

The model that mimics the AFC limiter the best is based on the lowest errors for the solutions for the two data sets and two test problems, but also on whether the model solutions are similar or close to the AFC solutions for most of the meshes and Péclet values. Furthermore, when the model solutions exceed the AFC and or the analytical solutions then it represents an *overshoot*. However, when the model solution are exceeding the AFC solution, but not the analytical solution then it *outperforms* AFC. Finally, interesting solution profiles results are shown in the results below and the L^2-norm relative solution error and α_{ij} error for all tested meshes and problems for each model can be found in the appendix A.

5.4.1. Solution model approach

For the trained boundary layer problem and both data sets, the solution model with limited fluxes loss function (LF2) is most accurate on all meshes and for most Péclet values, but multi-mesh data has smaller solution errors compared to the single-mesh data, see for example Figure 5.5. However, the alpha L2 errors are smaller for alpha loss function (LF1) on coarse meshes but not for LF2. This might be because the neural network optimizes its network-parameters better based on LF1 instead of LF2 as the α_{ij}^{REV} value is the output value and to optimize for the limited anti-diffusive fluxes $\alpha_{ij}^{\text{REV}} f_{ij}$ is more complex. Moreover, on fine meshes, the solution errors for all loss functions are in the same order, so solution profiles results are needed to observe which loss functions performs the best and has the most accurate results, for example see Figure 5.6. Furthermore, for single-mesh data, the model based on the limited correction loss function (LF3) is less accurate than other loss functions considering the under-and/or overshoots present in the solutions.

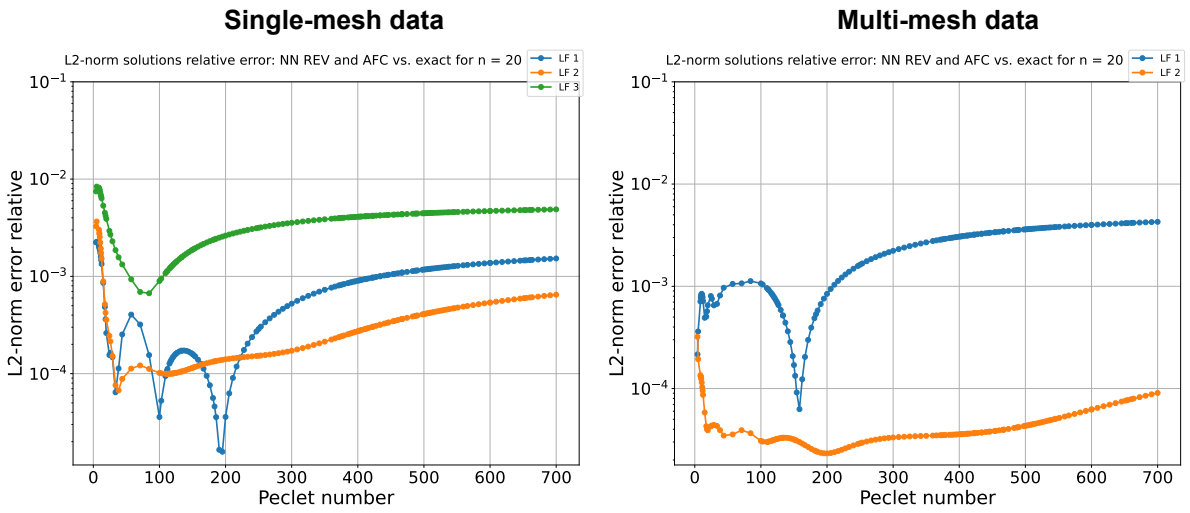


Figure 5.5: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for fine mesh ($n = 20$) for solution model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

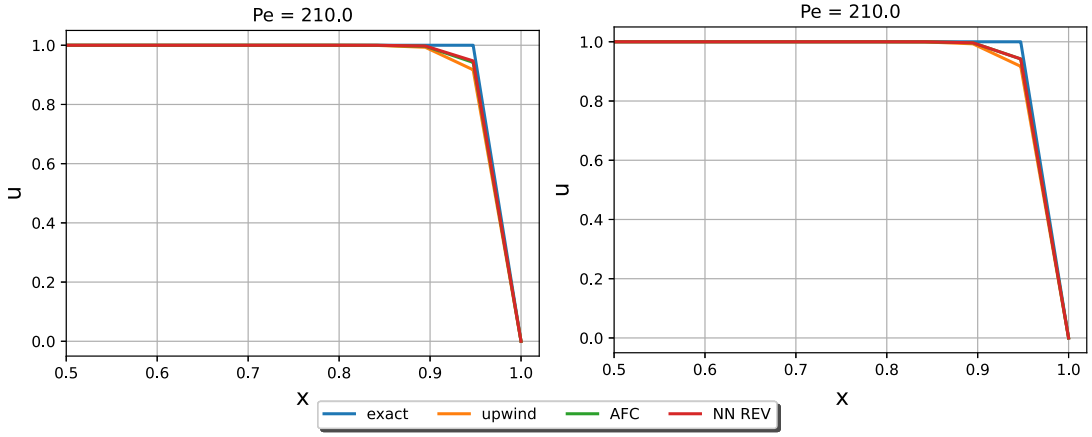


Figure 5.6: The boundary layer solutions of the solution model on a coarse mesh ($n = 20$) for alpha loss function (LF1, left) versus limited fluxes loss function (LF2, right) for multi-mesh data.

For the untrained peak problem, the solution error results on fine meshes, $n = 20, 50$, are more accurate than on coarse meshes, $n = 5, 10$, for both data sets (see one of the examples in Figure 5.7). On coarse meshes, significant overshoots in the model solution are present for all loss functions and Péclet numbers, so the errors are too high. On fine meshes, the solutions are close to the AFC solutions for $n = 50$, but for $n = 20$ still overshoots are present in the solution for most Péclet values, for example see Figure 5.8. For single-mesh data, limited correction loss function (LF3) results have most similar solutions as AFC at the steep region of solution, other loss functions models are similar to each other and have solutions close to AFC. Moreover, the solution errors for LF3 are less oscillatory than for alpha (LF1) and limited fluxes loss functions (LF2). This could mean that this loss function model has more stable limiting behaviour. For multi-mesh, LF2 is most accurate for most Péclet values at the steep region of the solution, although, the differences in the errors with LF1 are very small.

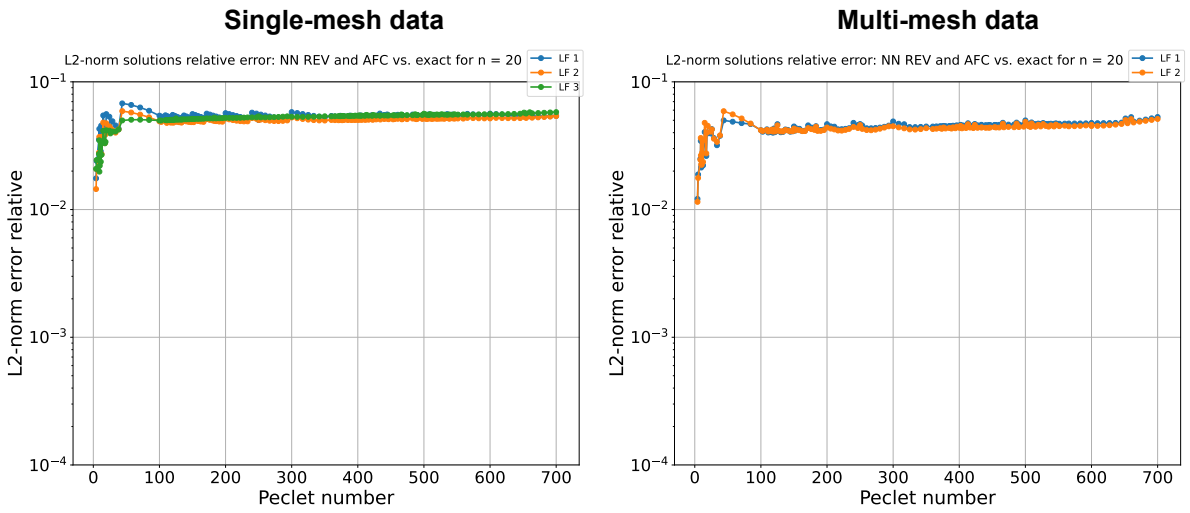


Figure 5.7: The L^2-norm relative error of the **(untrained) peak problem** solutions for fine mesh ($n = 20$) for solution model trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

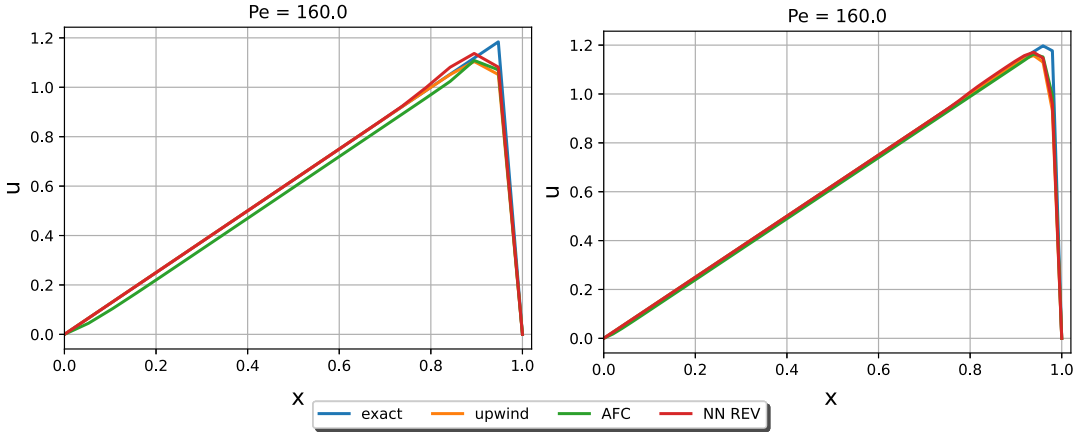


Figure 5.8: The peak problem solutions of the solution model on a coarse mesh ($n = 20$, left) versus fine mesh ($n = 50$, right) for limited fluxes loss function (LF2) with multi-mesh data.

To conclude, for the boundary layer problem the model trained on multi-mesh data with limited fluxes loss function (LF2) is based on observations of the errors and solution profiles the best on all meshes and Péclet values. For the peak problem, the model gives for both data sets similar results and are only most accurate on the finest mesh ($n = 50$). However, the multi-mesh data results are considered more accurate for $Pe > 100$ than the single-mesh data, therefore, this data is most suitable to generalize this model to untrained or other problems only on fine meshes.

5.4.2. Solution + fluxes model approach

For solution + fluxes model, the boundary layer problem results with limited fluxes loss function (LF2) and multi-mesh data are most accurate for all meshes and Péclet values than single-mesh data with LF2 (see one example in Figure 5.9). For single-mesh data, the solutions for alpha loss function (LF1) and LF2 are similar on most meshes, but for LF2 they are still better. Moreover, the solutions for LF3 are overshooting AFC for high Péclet values and over-and undershooting AFC for low Péclet values, but in some cases it outperforms AFC, see for example Figure 5.10 $n = 20$ for high Péclet values. For multi-mesh data, the coarse mesh solutions have significant overshoots for LF1, but are similar to AFC for fine meshes. Finally, the addition of the fluxes to the input data leads to no oscillations in the errors, which could mean that this model has a more consistent limiting behaviour.

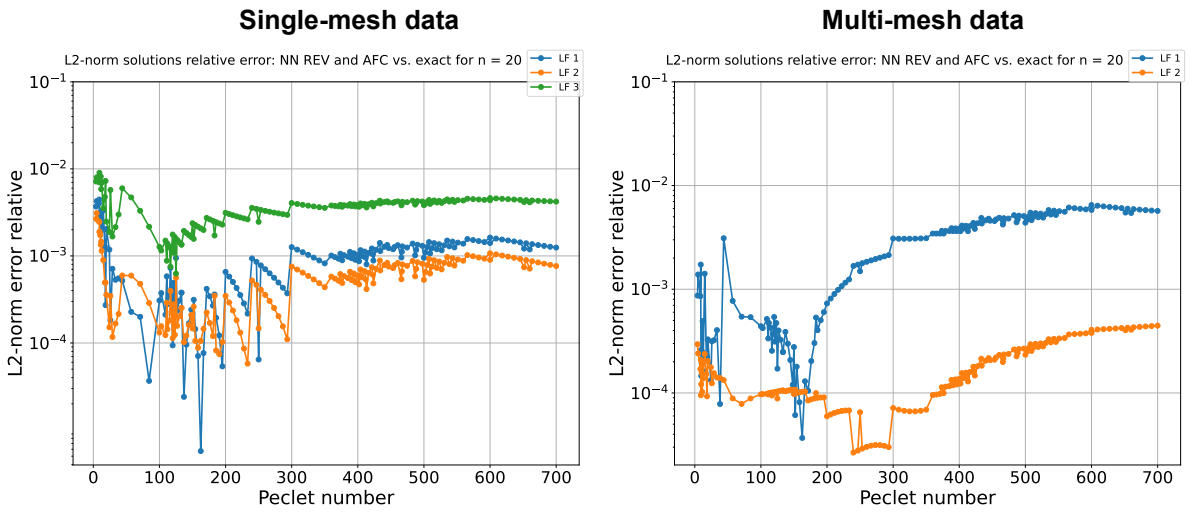


Figure 5.9: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for fine mesh ($n = 20$) for solution + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

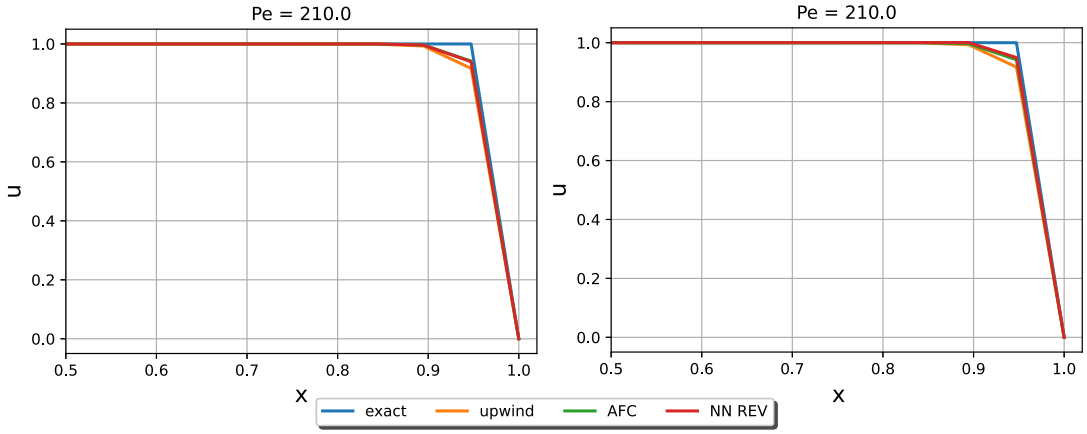


Figure 5.10: The boundary layer solutions of the solution + fluxes model on a coarse mesh ($n = 20$) for limited fluxes function (LF2, left) versus limited correction loss function (LF3, right) for single-mesh data.

For the peak problem, the results for both data sets on fine meshes are more accurate than on coarse meshes, which resulted in solutions containing overshoots for most Pèclet values. However, the solutions on fine meshes such as $n = 20$ still contains small overshoots, but are on $n = 50$ most similar to the AFC solutions, which was also observed for solution model. Based on the solution errors, multi-mesh data is most accurate for limited fluxes loss function (LF2) for all meshes and $Pe > 100$ compared to alpha loss functions (LF1) and to single-mesh data, see one example in Figure 5.11. On the other hand, the results for $Pe < 100$ are better for LF1 on all meshes.

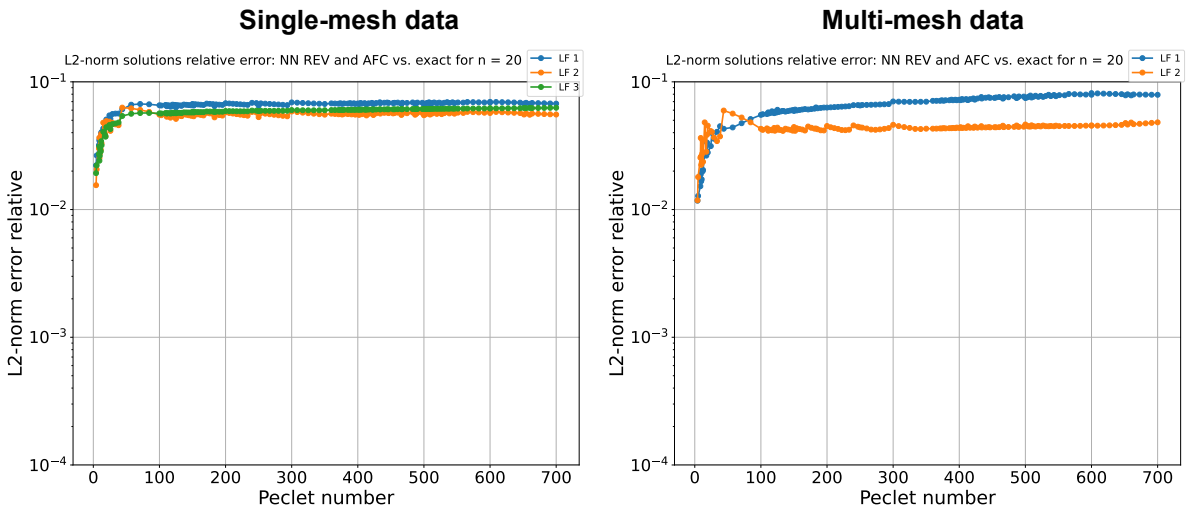


Figure 5.11: The L^2-norm relative error of the **(untrained) peak problem** solutions for fine mesh ($n = 20$) for solution + fluxes model trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

To sum up, for both problems the most accurate results for solution + fluxes model were obtained with the multi-mesh data and limited fluxes loss function (LF2) on all meshes and most of the Péclet values. So this model would be most suitable to be generalized for other problems only on fine meshes.

5.4.3. Gradient model approach

Gradient model learned a completely different input and output relation than the previous models, so different limiting behaviour is expected. However, on most meshes ($n \geq 10$), the model is most accurate with limited fluxes loss function (LF2) for all Péclet values, as shown in for example Figure 5.12, and has boundary layer problem solutions most similar to the AFC solutions, but for single-mesh data it is also accurate with alpha loss function (LF1) on fine meshes, see for example Figure 5.13. Other loss functions have solutions that include significant overshoot on coarse meshes and small over-and/or undershoots on fine meshes. For this model, the α_{ij} errors are on a coarse mesh smallest for LF1 with single-mesh data, which has been observed for the solution model as well and confirms that LF1 is easier to optimize for than others on coarse meshes.

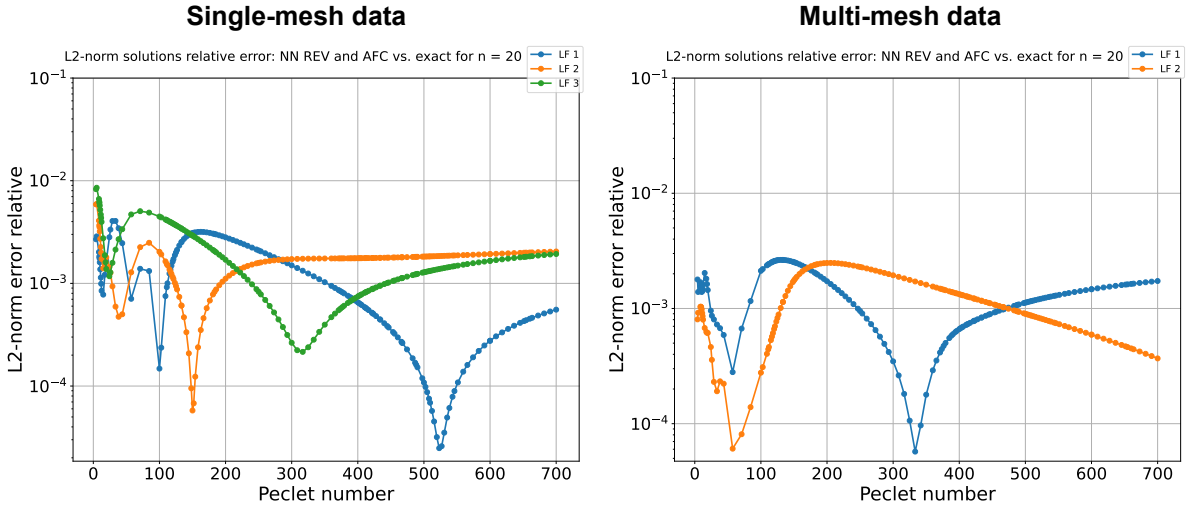


Figure 5.12: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for fine mesh ($n = 20$) for gradient model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

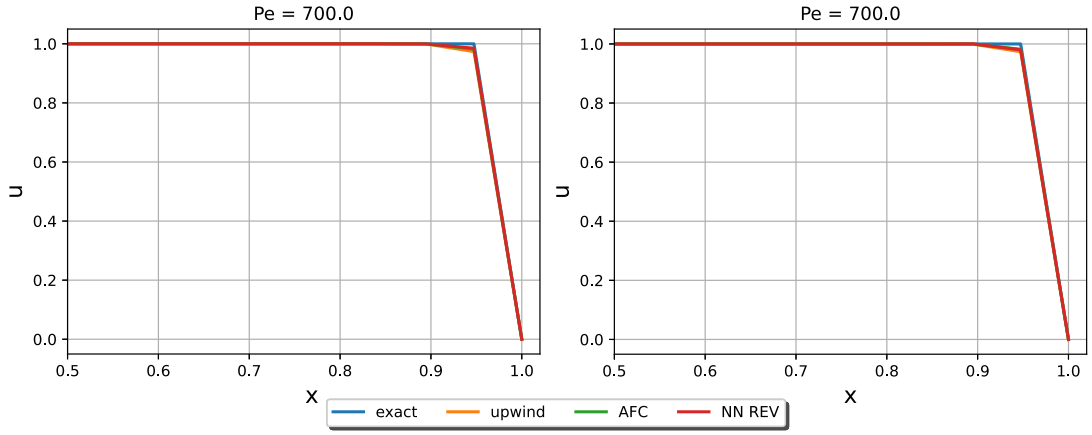


Figure 5.13: The boundary layer solutions of the gradient model on a coarse mesh ($n = 20$) for alpha loss function (LF1, left) with single-mesh data versus limited fluxes function (LF2, right) with multi-mesh data.

The impact of the mapping is more noticeable in the results for the untrained peak problem because this model has a higher capacity to be generalized. Since on most meshes and for most Pèclet values the solutions are similar to the AFC solutions and even in some cases outperforms AFC (see one example in Figure 5.15). For multi-mesh data this is obtained with limited fluxes loss function (LF2) and for single-mesh data with the alpha loss function (LF1) even though LF3 has smallest errors on most meshes, see one example in Figure 5.14, this is because the solution is closer to AFC, but has significant overshoot at the steep region compared to LF1, which has no overshoot.

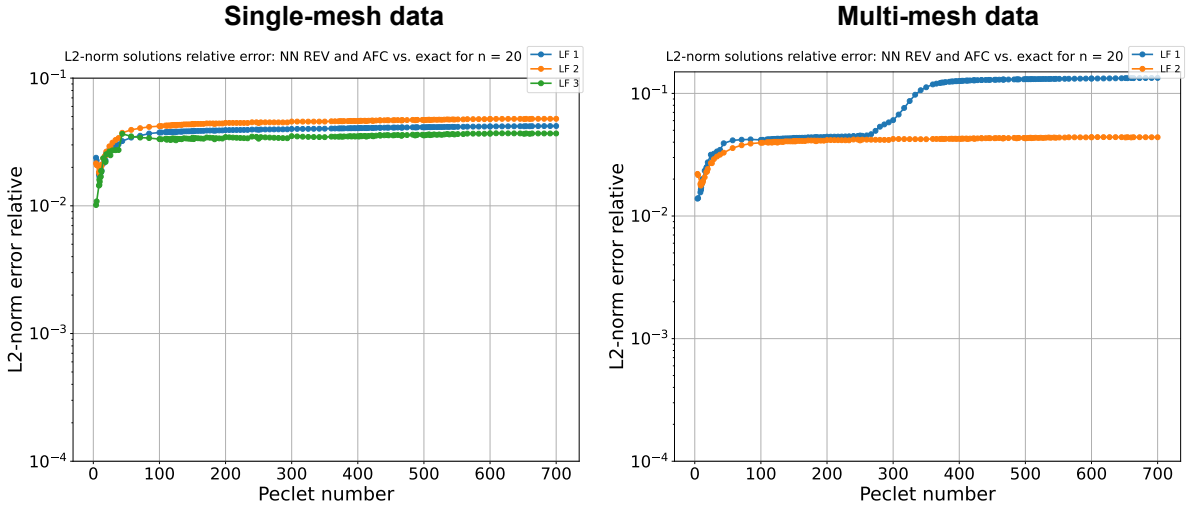


Figure 5.14: The L^2-norm relative error of the **(untrained) peak problem** solutions for fine mesh ($n = 20$) for gradient model trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

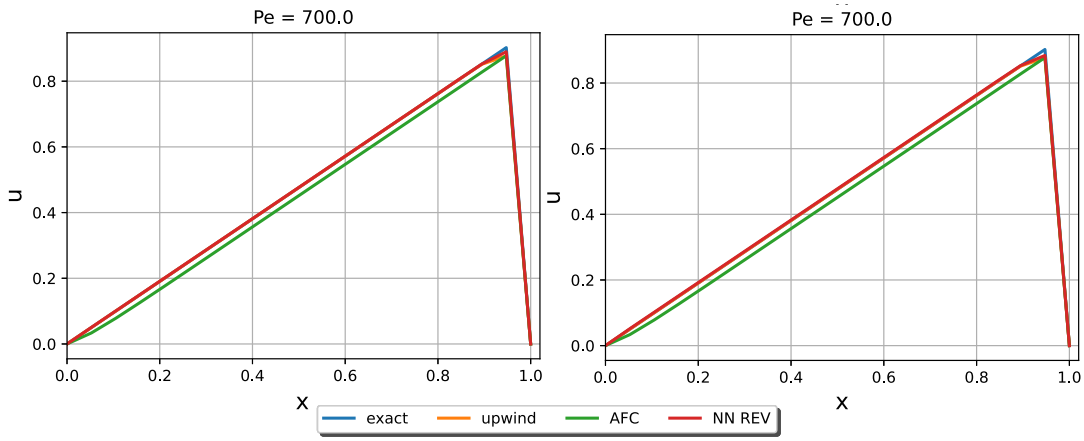


Figure 5.15: The peak problem solutions of the gradient model on a coarse mesh ($n = 20$) for alpha loss function (LF1, left) with single-mesh data versus limited fluxes function (LF2, right) with multi-mesh data.

To conclude, for both problems and data sets the gradient model performs well with the limited fluxes loss function (LF2) on most meshes and for most Péclet values, but for single-mesh data the alpha loss function (LF1) is well generalizable to untrained problems and for multi-mesh this is achieved with LF2.

5.4.4. Gradient + fluxes model approach

For the boundary layer problem, the results of the gradient + fluxes model are similar to the gradient model. On most of the meshes the results for single-mesh data are most accurate for alpha loss function (LF1), while, for multi-mesh data it is limited fluxes loss funtion (LF2), see for example Figure 5.16, that obtains accurate results. The corresponding solutions are on most of the meshes and Péclet numbers similar to AFC, but the solutions for single-mesh data outperforms the AFC solutions in some cases.

Moreover, on single-mesh data the solutions on coarse meshes are best with LF1 for most Péclet numbers, for $n = 5$ both LF2 and limited correction loss function (LF3) have overshoots, but for $n = 10$ the solutions are similar to AFC for all loss functions; on fine meshes it is observed that the solutions of LF1 are overshooting slightly AFC (outperforms as well), while of LF2 are undershooting AFC slightly (see for example Figure 5.17), and LF3 are similar or close to AFC, for all Péclet values. So, overall LF1 or LF3 could be chosen based on desired mesh for this data. For multi-mesh general observation would be that for LF2 the solutions are most similar to the AFC solutions, while the solutions of LF1 are only more accurate for some high Péclet value cases on fine meshes, like reflected by the solution errors.

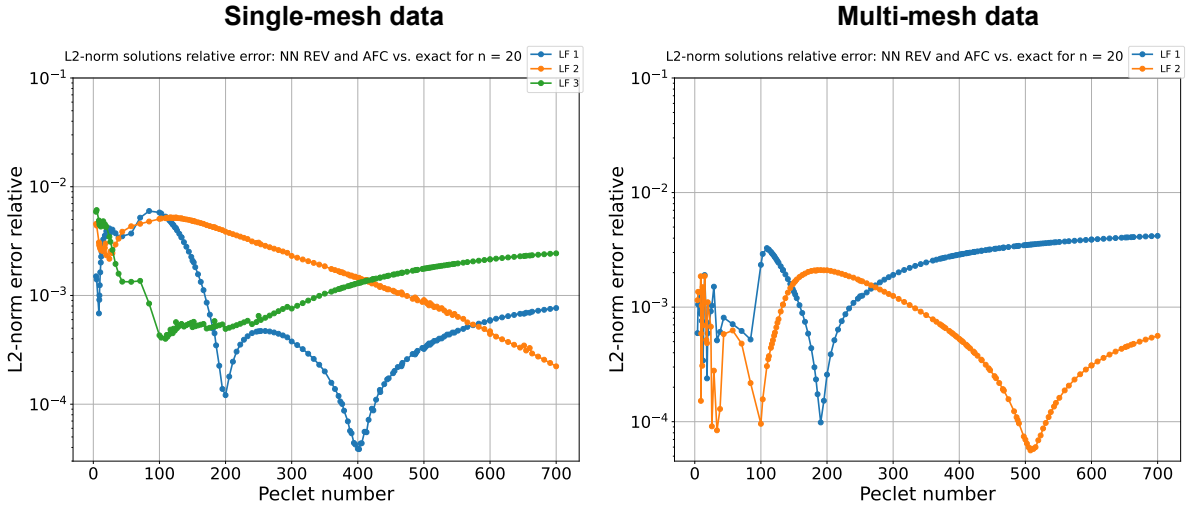


Figure 5.16: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for fine mesh ($n = 20$) for gradient + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

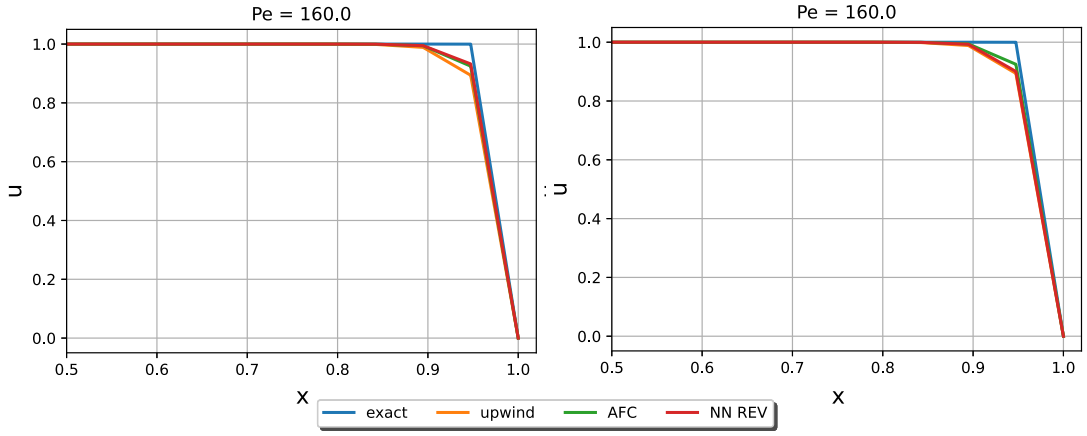


Figure 5.17: The boundary layer solutions of the gradient + fluxes model on a coarse mesh ($n = 20$) for alpha loss function (LF1, left) versus limited fluxes function (LF2, right) with single-mesh data.

The solutions of the peak problem are most accurate for single-mesh data with alpha loss function (LF1) and for multi-mesh data with limited fluxes loss function (LF2) on most of the meshes and Péclet numbers, see one example Figure 5.18 and the solution for LF2 in Figure 5.19. These results lead to solutions that are most similar to AFC and even outperforms AFC in some cases for both data sets likewise to the gradient model. However, for single-mesh data LF2 solutions are also close to AFC only on fine meshes and for the limited correction loss function (LF3) the solution have significant overshoots, for example see solution for LF3 in Figure 5.19. For multi-mesh data and single-mesh data the LF2 differences of the solutions are very small on fine meshes.

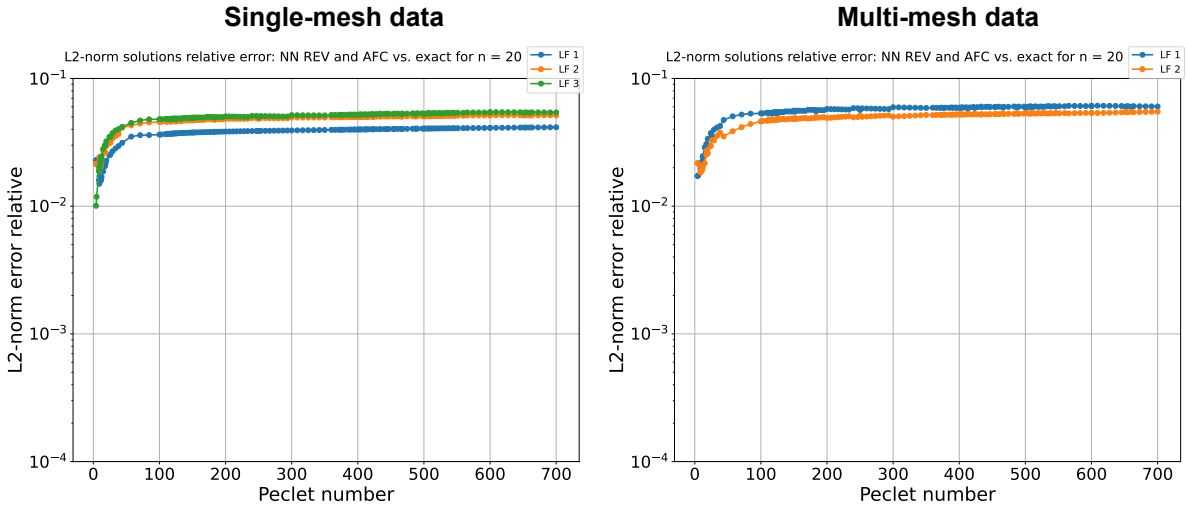


Figure 5.18: The L^2-norm relative error of the **(untrained) peak problem** solutions for finest mesh ($n = 20$) for gradient + fluxes model trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited fluxes LF2 (orange) and limited correction LF3 (green).

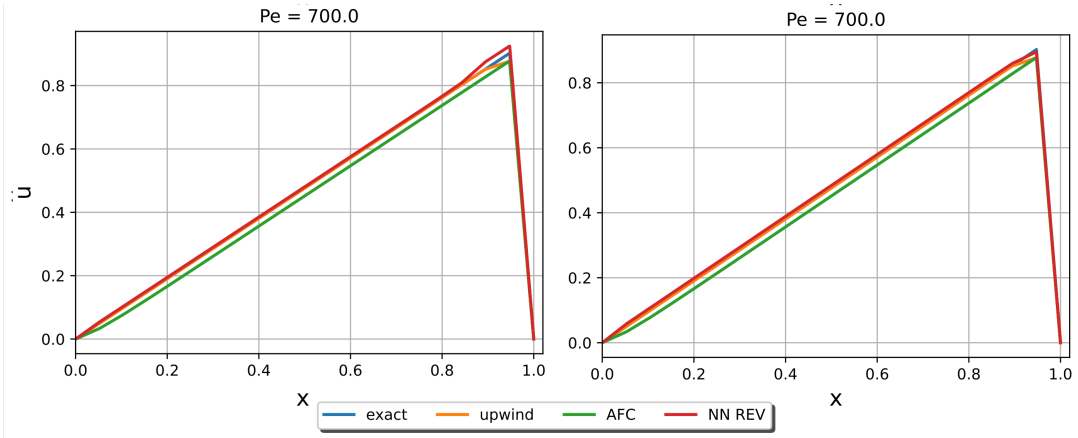


Figure 5.19: The peak problem solutions of the gradient + fluxes model on a coarse mesh ($n = 20$) for limited correction loss function (LF3, left) with single-mesh data versus limited fluxes function (LF2, right) with multi-mesh data.

To sum up, the gradient + fluxes model is most accurate with single-mesh data for alpha loss function (LF1) and with multi-mesh data for limited fluxes loss function (LF2) on most meshes and Pèclet values for both problems. Thus, both data sets have the potential to be generalized to untrained or other problems.

5.5. Analysis

In this section, the results are discussed and analysed. Four models have been proposed and each model has been trained on two different data sets, namely single and multi-mesh data. These data sets are generated by the low-order solutions of the boundary layer problem and also trained on three different loss functions: alpha α_{ij} loss function (LF1), limited anti-diffusive fluxes $\alpha_{ij} f_{ij}$ loss function (LF2) and limited anti-diffusive correction $\bar{\mathbf{f}}(\mathbf{u}^{\text{low}})$ loss function (LF3). All the models have been tested on the trained boundary layer problem and untrained peak problem. A summary of the most accurate models with the corresponding parameters is given in Table 5.3.

Models	**Data**	**LF**	**Meshes** (n) + Pe	**Generalizability**
solution model solution+fluxes model	multi-mesh	2	for all	on a fine mesh ($n = 50$) $\forall Pe \geq 50^*$
gradient model	single-mesh multi-mesh	1 2	on most ($n \geq 10$)	on most meshes and Pe *
gradient+fluxes model	single-mesh multi-mesh	1 2	on most* on most	on most meshes and Pe * on most meshes ($n \geq 10$) and Pe *

$* =$ outperforms AFC observably

Table 5.3: Summarized the best achievable models with the corresponding parameters for the different data sets (single-mesh ($n = 10$) and multi-mesh($n = 10, 25, 50$)), loss functions (α_{ij} loss function (LF1), limited anti-diffusive fluxes $\alpha_{ij} f_{ij}$ loss function (LF2) and limited anti-diffusive correction $\bar{\mathbf{f}}(\mathbf{u}^{\text{low}})$ loss function (LF3)), and for tested meshes ($n = 5, 10, 20, 50$) and Péclet (Pe) values based on the trained boundary layer (BL) problem results and the generalizability based on the untrained peak (PK) problem results.

Mesh data analysis

The multi-mesh data achieves for each model accurate results, which could be expected since it contains coarse and fine meshes and so it works on most if not all meshes for the boundary layer problem and even for the peak problem it acquires results that outperform AFC. Single-mesh data is only compatible with gradient and gradient+fluxes models because it takes into account the mesh size and therefore the neural network can learn better the AFC limiting behaviour as the solution values are scaled and do not depend on the mesh size anymore.

Loss function analysis

As shown in Table 5.3 each model achieves more accurate results with LF2 compared to other loss functions because LF2 takes into account the flux values, which gives more information than only the alpha values, which is only a number between one and zero. Since the fluxes determine how much limiting is required, therefore, optimizing the limited anti-diffusive fluxes $\alpha_{ij} f_{ij}$ gives more accurate limiting behaviour and results. The Table also shows that LF3 does not provide accurate results with any model and data sets, because it circumvents the summation of the limited anti-diffusive fluxes and therefore the neural network is trained differently, which affects the performance and training took longer, so it is more difficult to optimize for.

In general, LF2 results are less accurate with single-mesh data compared to multi-mesh data because single-mesh data does not provide sufficient information about the fluxes for the neural network to optimize well with LF2. This could also be the case for the models where the flux information is not sufficiently extracted from the solution input data because gradient-based models trained with single-mesh data achieved accurate results with LF1. This could be because the gradient input data does provide more information about limiting like AFC and therefore a simple loss function would work and be optimized easier for this data.

To conclude, the alpha values of AFC and the models do not need to be similar to have accurate performance and results, see for example LF2 which has the most accurate results for each model, this means that when optimizing for LF2, the errors of the limited anti-diffusive fluxes of AFC and the model are minimized, which is more important to be similar than the alpha values to be similar.

Comparison models

Based on the generalizability, gradient and gradient+fluxes models would be most suitable to mimic the AFC limiting behaviour. The difference between these models is that the flux values are added, which gives the neural network more information to learn and optimize for the loss functions. For example, Table 5.3 shows that the model trained with single-mesh is the only one that outperforms AFC for both boundary layer and peak problem on most meshes and Péclet values. Furthermore, for the solution and solution+fluxes models adding the fluxes to the input data did not have the same impact on the ability of the neural network to learn or optimize better since the same results are obtained for both models.

Outperformance: AFC vs models

To outperform AFC, the models' solutions are better than the AFC solutions and therefore are closer to the analytical solution (and not exceeding it). From Table 5.3, it can be concluded that all models are outperforming AFC for the untrained peak problem, which entails that less limiting is required than AFC for this problem and that this less limiting behaviour is learned by the trained boundary layer problem. This behaviour is learned by the solution-based models accurately only for fine meshes and large Péclet values, but by gradient-based models for most meshes and Péclet values. This shows that the relationship between the input and output data plays a significant role in training a neural network to learn the AFC limiter behaviour or more generally a numerical stabilization method.

Generalizability analysis

From Table 5.3 it follows that the solution model and solution+fluxes model are both more accurate than the gradient model and gradient+fluxes model for the boundary layer problem on all meshes. However, the latter models are more generalizable to untrained or other problems because of the different input and output mapping. Finally, solution values as input data or solution-based models are only able to learn the trained problem better and are less generalizable to other problems. Although, gradient values as input data or gradient-based models are generalizable because they are able to extract a general limiting pattern like AFC and are not specifically learning the mapping for the trained problem. So gradient values provide more insight or information into the AFC limiting behaviour than the solution values. In short, gradient-based models perform sufficiently accurate on most of the meshes for both data sets, thus the affect of the numerical accuracy is less on these models.

Broad recommendations

It is recommended to train with data obtained with a fine mesh ($n > 10$), which occurs when multi-mesh data is used, this increases the numerical accuracy and consequently the model's accuracy. However, single-mesh data ($n = 10$) can be used when the appropriate input values are used, as is the case for

the gradient-based models. Moreover, to train a neural network the input and output relationship has to be well compatible for the type of application since this will decide what the neural network has to approximate with a suitable function. Likewise, this also applies to the type of loss function that is used. For AFC, the gradient and/or fluxes as input values were the most optimal choices with relatively simple loss functions, such as the optimization of the alpha errors (LF1) or the limited anti-diffusive flux errors (LF2). To further evaluate the generalizability, the models can be tested on variations of the boundary layer and peak problems, for instance by changing the boundary conditions.

6 Learning-based surrogate model of AFC limiting

In this chapter, the neural network-based surrogate model of AFC limiting approach is developed and described at first. The main focus of this approach is to find a suitable data set and neural network architecture that can mimic the behaviour of the AFC limiter given by the algorithm: (3.17)-(3.22). Therefore, two data sets: constant diffusion data and variable diffusion data, explained in section 6.1 and three different architectures: feedforward neural network, splitted neural network and physics-informed neural network, described in section 6.2, were considered. These networks were trained on the two data sets and optimized with Ray Tune as explained in section 6.3, which tunes the hyperparameters and lowers the manual efforts to obtain an optimal model. Finally, the neural networks are tested on different problems to achieve the accuracy and performance and these results are described in section 6.4, after which the analysis and discussion of the results follow in section 6.5.

An alternative technique to using a neural network in the AFC algorithm to solve the stationary convection-diffusion equation is to use it as a surrogate model in the defect correction scheme [14] as described in section 3.2. Here the neural network computes the α_{ij} values in a nonlinear fashion similar to the AFC limiter of TVD-type. The idea is that the neural network learns the behaviour of the AFC limiter, where the limiter is the benchmark.

The non-linear algebraic system of the stationary convection-diffusion equation solved by the defect correction scheme (DCS), from equation (3.26), is given as:

$$(S - L)\mathbf{u} = \mathbf{r} + \bar{\mathbf{f}}(\mathbf{u}) \tag{6.1}$$

where the anti-diffusive correction $\bar{\mathbf{f}}(\mathbf{u})$ put into the system, from equation (3.25), is written as:

$$\bar{f}_i(\mathbf{u}) = \sum_{j \neq i} \alpha_{ij} f_{ij}, \tag{6.2}$$

where $f_{ij} = d_{ij}(u_i - u_j)$ is the anti-diffusive flux with d_{ij} as the artificial diffusion coefficient and α_{ij} is the adaptive flux limiter. The limiter is computed using the f_{ij} values by **the P, Q, and R equations of the AFC with TVD-type limiting** defined in Kuzmin's algorithm: (3.17)-(3.22), which is also used by the DCS and henceforth will be called the AFC limiter. The concept is to replace these equations by the neural network, so it computes the anti-diffusive correction $\bar{\mathbf{f}}(\mathbf{u})$ instead of the AFC limiter by using as input the anti-diffusive fluxes, f_{ij} $\forall i, j$, and giving as an output the α_{ij} $\forall i, j$ values, as shown in Figure 6.1. Consequently, the neural network relates the input data to the output data to mimic the P, Q, and R equations, which defines the behaviour of the AFC limiter and therefore it functions as a surrogate model in the DCS.

Moreover, the behaviour of the AFC limiter is not entirely based on the α_{ij} values, but the contribution of the anti-diffusive fluxes f_{ij} have to be taken into consideration as well. Therefore, the behaviour should be based more on the limited anti-diffusive flux as given by $\alpha_{ij} f_{ij}$ and consequently also on the (converged) net limited anti-diffusive flux $\bar{f}_i(\mathbf{u})$ because this is the part of the anti-diffusive flux that is actually put into the system to remove excessive artificial diffusion in regions where possible without generating spurious oscillations. For example, when the solution has regions where the slope is zero the anti-diffusive flux f_{ij} is zero and therefore α_{ij} can be zero, one or any value in between without generating oscillations in the solution.

So, the behaviour of the AFC limiter can be interpreted as followed: when AFC is limiting more than required the solution is closer to the upwind (low-order) solution as less net limited anti-diffusive flux is put into the system as a consequence less artificial diffusion is removed. However, limiting less than required the solution is closer to the Galerkin (high-order) solution as more net limited anti-diffusive flux is put into the system as a consequence more artificial diffusion is removed.

AFC algorithm vs NN learning-based surrogate model of AFC limiting

1. Select the initial guess $\mathbf{u}^{(0)}$, e.g.:

$$\mathbf{u}^{(0)} = \text{ upwind solution (low-order)},$$
$$(S-L)\mathbf{u}^{(0)} = \mathbf{r}$$

2. Solve the linear system for correction:

$$(S-L)\Delta\mathbf{u}^{(m)} = \mathbf{r} + \boxed{\bar{\mathbf{f}}(\mathbf{u}^{(m-1)})} - (S-L)\mathbf{u}^{(m-1)},$$

where the anti-diffusive correction at i is:

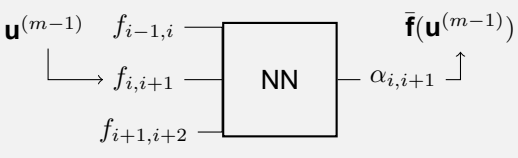
$$\bar{f}_i(\mathbf{u}^{(m-1)}) = \sum_{j\neq i} \alpha_{ij}(\mathbf{u}^{(m-1)}) f_{ij}(\mathbf{u}^{(m-1)}),$$
$$f_{ij}(\mathbf{u}^{(m-1)}) = d_{ij}(u_i^{m-1} - u_j^{m-1})$$

AFC algorithm

$\alpha_{ij}(\mathbf{u}^{(m-1)})$ is determined by the algorithm: (3.17)-(3.22)

NN algorithm

$$\alpha_{ij}(\mathbf{u}^{(m-1)}) = NN\ (f_{ij}(\mathbf{u}^{(m-1)}))$$



3. Apply the correction to the solution:

$$\mathbf{u}^{(m)} = \mathbf{u}^{(m-1)} + \Delta\mathbf{u}^{(m)}$$

4. Repeat step 2 unless following condition is fulfilled:

$$\frac{||\Delta\mathbf{u}^{(m)}||}{||\mathbf{u}^{(m)}||} < tol \text{ or } m = m_{max}$$

Figure 6.1: Summary of DCS with AFC and NN algorithms for solving the nonlinear system.

6.1. Type of input and output data

To find a suitable mapping between the input and output data to train the neural network in mimicing the behaviour of the AFC limiter different approaches were considered. Firstly, the anti-diffusive fluxes f_{ij} as input data and the corresponding α_{ij} values as output (see local window approach Figure 6.2) of each iteration of the DCS for a boundary layer problem with single-mesh size ($n = 50$) was considered. However, the mapping of the data appeared to be ill-conditioned because the fluxes around zero corresponded to a large variation in the α_{ij} values as shown in the left plot in Figure 6.3 for $Pe = \pm 100, \pm 700$, therefore, restricting the learning space of the neural network.

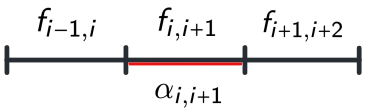


Figure 6.2: Local window approach for input flux values, f_{ij}, and output value, α_{ij}.

Secondly, more data is generated to broaden the space of the mapping, random solution profiles, like cosine and sine functions, and the Galerkin and upwind solutions of the boundary layer and peak problems both with negative and positive velocities were considered. For these profiles, the anti-diffusive flux f_{ij} values were computed with their corresponding α_{ij} values for one AFC-TVD-loop iteration (step 2 in DCS), which is equivalent to using the P, Q and R equations once. For example, the mapping of the upwind solution of the boundary layer and peak problems are shown in the right plot of Figure 6.3 for $Pe = \pm 100, \pm 700$. Note that as the Péclet number increases the solutions get steeper therefore the f_{ij} increases consequently decreases the α_{ij} values to limit oscillations from occurring in the solution.

The three neural networks were trained with this data with the local window approach. Although, the predictions for the α_{ij} values did not achieve symmetric solution profiles of the boundary layer and peak problem in both velocity directions for all trained neural networks. This was especially noticeable for the $n = 20$ results, thus, eliminating the need for the cosine and sine functions with this approach.

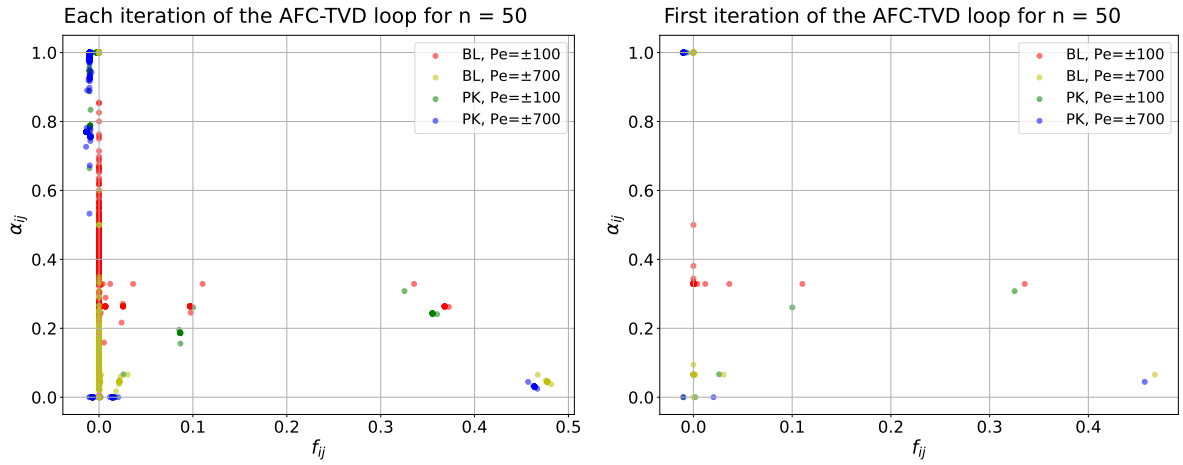


Figure 6.3: For $n = 50$ the mapping between f_{ij} and α_{ij} of each iteration of the TVD loop (left) and of the first iteration of the TVD loop (right) with the upwind solutions of the boundary layer (BL) and peak (PK) problems, both for Péclet number ± 100 $(v = \pm 1, d = \pm 1/100)$ and ± 700 $(v = \pm 1, d = \pm 1/700)$.

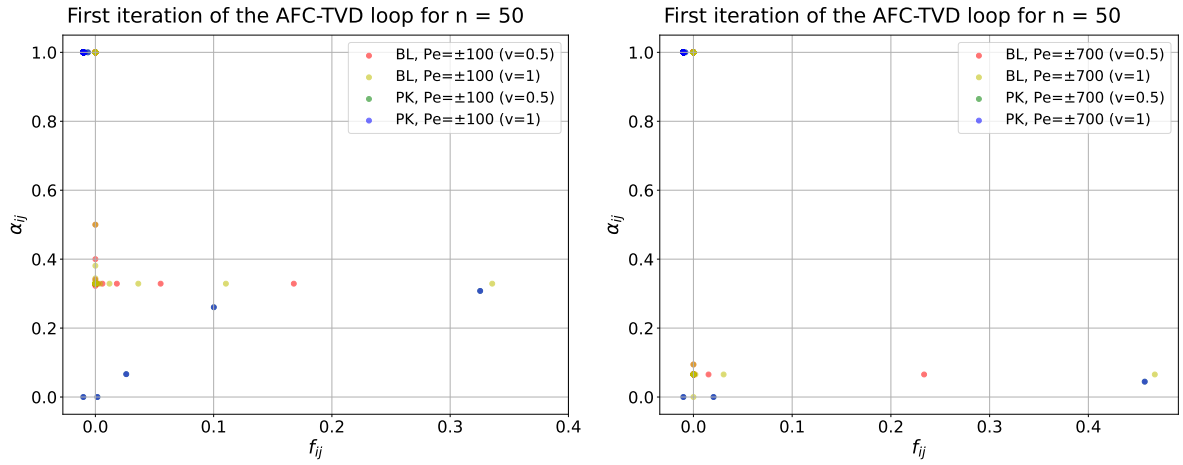
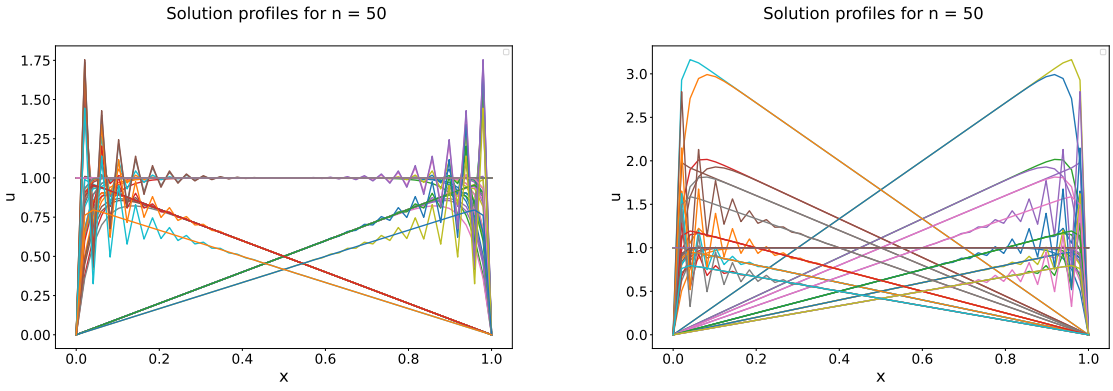


Figure 6.4: For $n = 50$ the mapping between f_{ij} and α_{ij} of the first iteration of the TVD loop for the boundary layer (BL) and peak (PK) problem, both for Péclet number ± 100 (left) and ± 700 (right) with variable velocities.

As a result, the neural networks were trained with the local window approach, where the anti-diffusive fluxes $f_{i-1,i}, f_{i,i+1}, f_{i+1,i+2}$, correspond to the output value, $\alpha_{i,i+1}$ (as shown in DCS Figure 6.1). The data is generated for one iteration of the DCS, so the fluxes are computed only once and used to compute the α_{ij} value by using the P, Q, and R equations once. The fluxes were computed

by $f_{ij} = d_{ij}(u_i - u_j)$ where the artificial diffusion coefficient d_{ij} is related to the velocity constant used in the problem. Therefore, two different data sets were investigated to find the optimal data set to train the network that learns the AFC limiter equations. One data set is created with a **constant diffusion coefficient**, $d_{ij} = 0.5$, and one data set with a **variable diffusion coefficient**, $d_{ij} = v/2$ (the $1/2$ comes from the Gauss quadrature rule see section 2.2) with v as the velocity constant. The former data set has $v = 1$ and the latter set has different velocity constants.

Moreover, the u_i values were sampled from the Galerkin and upwind solution profiles of the boundary layer and peak problems in both velocity directions for fixed number of basis functions $n = 50$. These solution profiles have steep, non-physical oscillatory and smooth regions (see Figure 6.5) where AFC limits accordingly. For example, in oscillatory regions the α_{ij} values are zero, while in smooth or continuous regions the α_{ij} values are close to one. In steep regions of the solutions, α_{ij} values are between zero and one, which are the most challenging regions as oscillations can reappear in the solutions when the Galerkin solutions or oscillations are not limited enough. The goal for the neural network is to learn to recognize these types of regions and limit accordingly.



(a) constant diffusion coefficient $d_{ij} = 0.5$

(b) variable diffusion coefficient d_{ij}

Figure 6.5: Solution profiles for computing the fluxes to train the neural networks. (a) The Galerkin and upwind solutions, both for peak and boundary layer problems and Péclet number from ± 30 to ± 700. (b) The Galerkin and upwind solutions for peak problem and upwind solutions for boundary layer problem and number from ± 50 to ± 700.

Furthermore, a fine mesh ($n = 50$) is chosen to represent the steep regions in the solutions accurately with sufficient nodes, where AFC has to limit. Both problems are used with positive velocity and negative velocity to create more data, but also to train the neural network to behave symmetrically and consistently for these problems as the AFC limiter is capable of doing this. The problems are also considered for a wide range of Péclet numbers, where higher Péclet numbers are included. A summary of the training data sets, the constant and variable d_{ij}, are given in Table 6.1 with the specific parameters used. In Figures 6.5 the corresponding solution profiles used are presented. Important to consider here is that for the boundary layer problem as the Péclet number increases, the solutions become steeper at the right boundary ($x = 1$) for positive velocity or at the left boundary ($x = 0$) for negative velocity. If different velocity constants are used for similar Péclet numbers the solutions do not change, but the diffusion coefficients d_{ij} do change consequently changing the fluxes f_{ij} and therefore the α_{ij} values, so the mapping changes as shown in Figure 6.4. Moreover, as the Péclet number increases for the peak problem the solutions also get steeper at the peak region, which is near the right boundary ($x = 1$) for positive velocity or at the left boundary ($x = 0$) for negative velocity, considering constant velocity is used, for example $v = 1$. However, as the velocity constant increases for similar Péclet number the slope of the line before the peak occurs decreases making it a less steep solution while maintaining the steepness of the peak region for the corresponding Péclet number. Furthermore, the different velocities changes the d_{ij} and the $(u_i - u_j)$ values of the solution profiles and therefore the change of velocity is cancelled out in the computations of the fluxes f_{ij}. Thus, the mapping for similar Péclet values with different velocities are identical as shown in Figure 6.4.

Parameters	**constant d_{ij} data**	**variable d_{ij} data**
artificial diffusion coefficient (d_{ij})	0.5	0.15-0.6
velocity constant(s) (v)	1	0.3-1.2
diffusion coefficient (d)	0.0015.. - 0.03	0.00109.. - 0.009
number of basis functions (n)	50	50
solution profiles	Galerkin, upwind $\pm$ PK/BL	Galerkin$\pm$PK, upwind$\pm$ PK/BL
Péclet number (Pe)	$\pm$ 30 to $\pm$ 700	$\pm$ 50 to $\pm$ 700

Table 6.1: Constant and variable artificial diffusion coefficient data sets to train the neural networks on the Galerkin and/or upwind solutions of the boundary layer (BL) and peak (PK) problems. The constant d_{ij} data has $v = 1.2$ for $Pe = \pm 700$ instead of one.

As a result of the chosen solution profiles the flux input data and alpha output data form the mapping as shown in Figure 6.6 and Figure 6.7 for constant and variable diffusion coefficient data, respectively. For both data sets, the alpha values are zero for a wide range of flux values, which corresponds to the oscillations in the Galerkin solutions, but also alpha values are close to one for a small range of flux values close to zero, which relates to smooth regions in the solutions that are not limited by AFC. Moreover, it can be observed that for a certain range of positive fluxes, the mapping has a slope. As the flux values increases the alpha values decreases indicating that the regions in the solutions get steeper, which correspond to increasing Péclet numbers. However, a main difference between the two data sets can be observed from the relations between the alpha values and the input values $f_{i-1,i}$ and $f_{i+1,i+2}$. For the constant diffusion data, the slope relation is present in both cases, while for the variable diffusion data a more spread out relation is visible. The latter is caused by the different velocities used in the data, which will influence the neural network to learn differently than for the former data set.

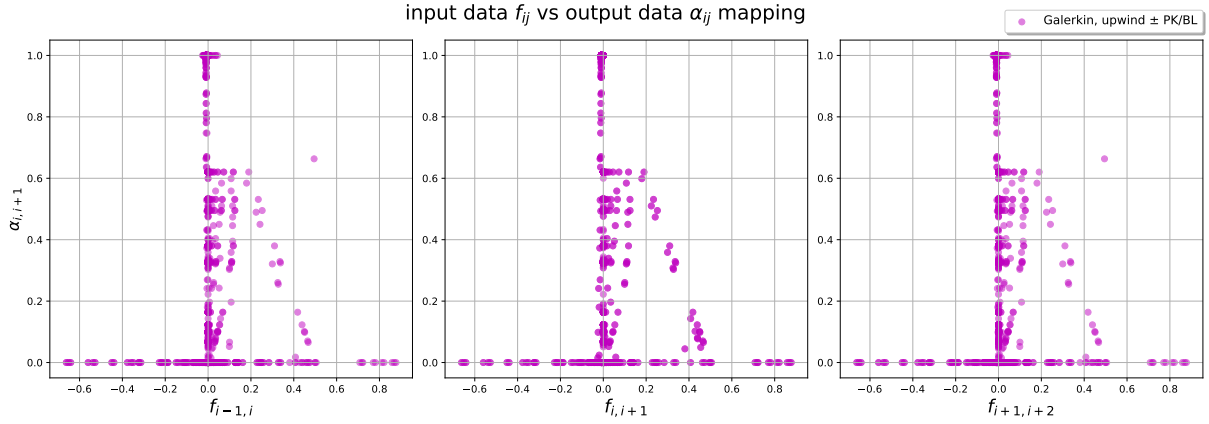


Figure 6.6: Input-output mapping to train the neural networks for constant diffusion coefficient $d_{ij} = 0.5$ constructed from Galerkin and upwind solutions for both the peak and boundary layer problems for fixed number of basis functions, $n = 50$, and Péclet numbers ranging from $+30$ to $+700$.

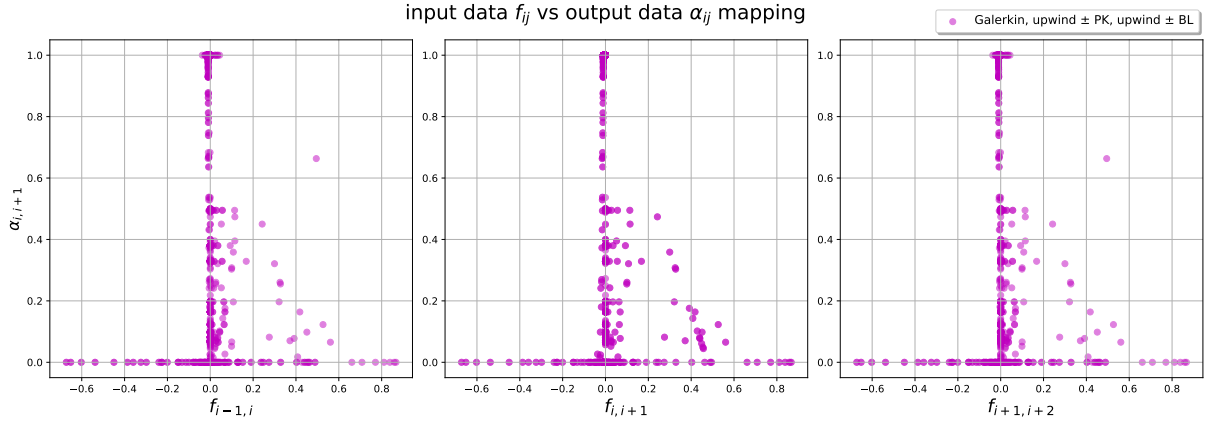


Figure 6.7: Input-output mapping to train the neural networks for variable diffusion coefficient d_{ij} constructed from Galerkin and upwind solutions for the peak problem and upwind solutions for the boundary layer problem for fixed number of basis functions, $n = 50$, and Péclet numbers ranging from ± 50 to ± 700.

6.2. Neural network architects: FFNN, Splitted NN and PINN

Feedforward Neural Network

As has been used in previous chapters, the feedforward neural network (FFNN) is used here since it will give a reference for the ability to train and learn the behaviour of AFC limiter compared to other neural network architectures. A FFNN of five hidden layers is used after optimizing the architecture manually for two to eight hidden layers by observing the training and validation losses. Circumstantially, five main computation steps are involved in the AFC limiter, namely taking the maximum and minimum of the fluxes, adding the values together, taking a division between the Q and P values, taking the minimum between one and fraction P and Q and finally the *if* statement to compute alpha. However, this does not necessarily imply that each hidden layer will learn one step. All hidden layers have tanh as activation function because the fluxes are between negative and positive values and the value of the neurons will be kept between minus one and one. The output layer has a Sigmoid activation function to keep the alpha value between zero and one. As the loss function the MSE is taken between the α_{ij} from AFC and the predicted α_{ij} of the network as defined in previous chapter with equation (5.2). The next architecture is a variation of the FFNN, namely the hidden layers are partially connected.

Splitted Neural Network

The Splitted neural network (SpNN) is developed inspired from the AFC limiter, so the P, Q and R equations. As shown in Figure 6.9 the first hidden layer is a split layer that creates two separate networks. The idea behind splitting the network into two is to mimic the P^+, Q^+ and P^-, Q^- equations separately in each sub-network because these equations compute R^+ and R^- , respectively, and these two contribute to computing the α_{ij}. The activation functions chosen are also based on the AFC limiter as shown in Figure 6.8. The $ReLU(z) = \max(0, z)$ and $-ReLU(-z) = \min(0, z)$ represent the positive flux contributions (P^+, Q^+) and the negative flux contributions (P^-, Q^-) respectively. The $\tanh(z)$ and $-\tanh(z)$ account for the contribution of the fluxes, but also to mimic the division between the $Q^\pm$ and $P^\pm$ values while keeping the output values between 1 and -1. The $\min(1, z)$ has to function like the *if* statement for choosing the R^+ or R^- and is the last step to compute the α_{ij} value. Moreover, in hidden layer 1 two different activation functions are applied in each network, which serves as taking the positive (f_{ij}) and negative ($-f_{ij}$) fluxes like in the $Q^\pm$ equations. These activation functions have been applied in various configurations in the hidden layers while manually tuning the hyperparameters (explained in the section 6.3) and tested on the boundary layer and peak problems. Based on this the best configuration of the model as shown in Figure 6.9 got the solutions closest to AFC, which might represent the separation of the $P^\pm, Q^\pm$ equations in another way, for example separating the $P^\pm$ and $Q^\pm$ in each sub-network. Lastly, similar to the FFNN the loss function is the MSE between the α_{ij} from AFC and the predicted α_{ij} of the network as defined equation (5.2).

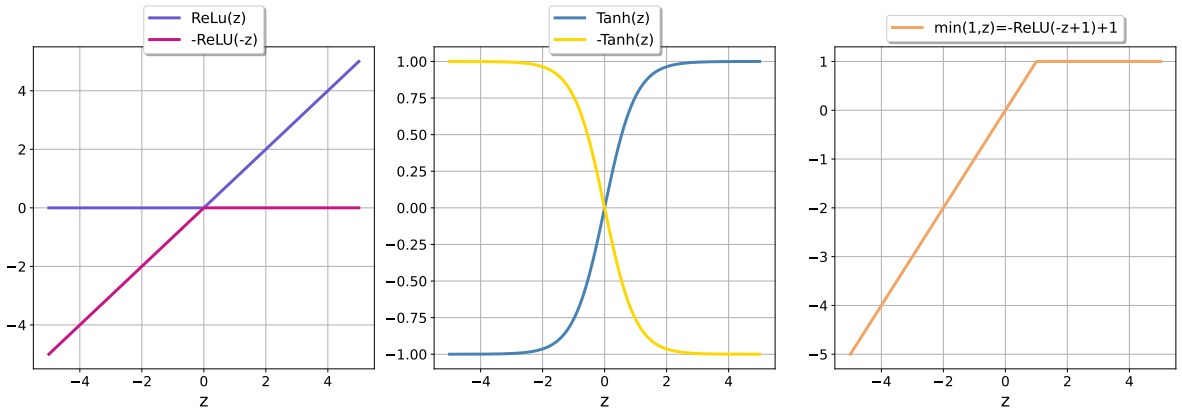


Figure 6.8: Activation functions used in SpNN.

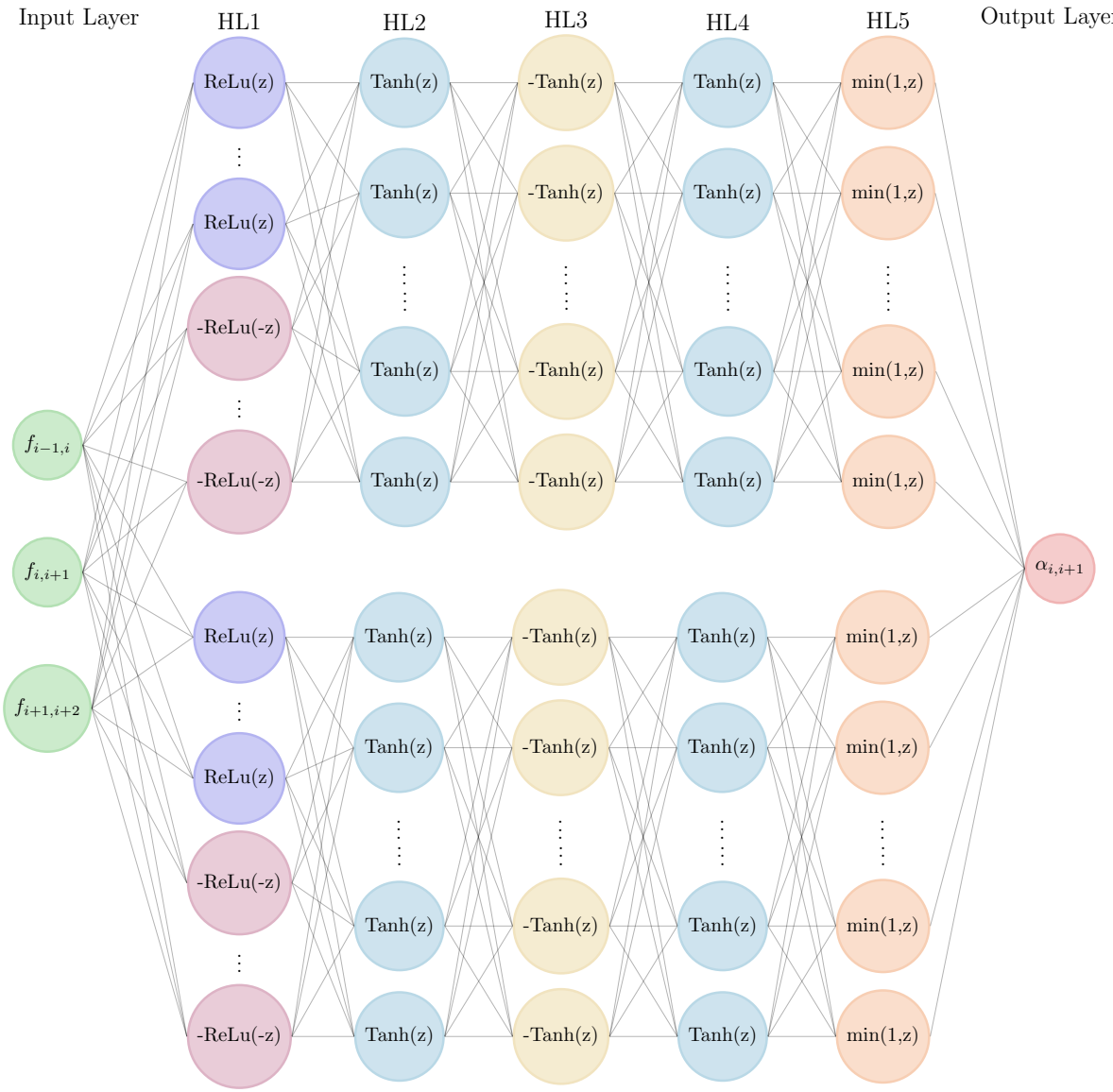


Figure 6.9: Splitted Neural Network with input, output layer and five hidden layers (HL1-5) with different activation functions in each layer based on the AFC limiter.

Physics-informed Neural Network

In recent years a rapid rising scientific machine learning technique has been developed in diverse applications that integrates knowledge of physical phenomenon with deep learning to solve problems involving Partial Differential Equations (PDEs) Physics-Informed Neural Networks (PINNs). The network is trained in order to approximate the solutions of the PDEs by minimizing the loss function, which obeys the physical constraints. The loss function includes a supervised loss that comes from the initial and boundary conditions and an unsupervised loss of the PDE, which is the residual of the PDE. However, instead of approximating the solution of the PDE we have to approximate the α_{ij} with the AFC limiter by the network. Therefore, the P, Q and R equations are included into the loss function that computes the $\alpha_{ij}^{\text{PINN}}$ as followed:

$$L \overset{\text{MSE}}{=} \frac{1}{N}\sum_{n=1}^{N}(\alpha_{i,i+1}^{\text{pred}(n)} - \alpha_{i,i+1}^{\text{PINN}\,(n)})^2, \quad \text{where} \quad \alpha_{i,i+1}^{\text{PINN}} := \begin{cases} \min\{1, \dfrac{Q_i^+}{P_i^+}\}, & \text{if } f_{i,i+1} > 0 \\ \min\{1, \dfrac{Q_i^-}{P_i^-}\}, & \text{otherwise} \end{cases} \tag{6.3}$$

with redefined $P^{\pm}$ and $Q^{\pm}$ equations:

$$P_i^+ \coloneqq \mathsf{max}\{0, f_{i,i-1}\}+\mathsf{max}\{0, f_{i,i+1}\}, \qquad Q_i^+ \coloneqq \mathsf{max}\{0, -f_{i,i-1}\} + \mathsf{max}\{0, -f_{i,i+1}\} \tag{6.4}$$

$$P_i^- \coloneqq \mathsf{min}\{0, f_{i,i-1}\} \ +\mathsf{min}\{0, f_{i,i+1}\}, \qquad Q_i^- \coloneqq \mathsf{min}\{0, -f_{i,i-1}\} + \mathsf{min}\{0, -f_{i,i+1}\}. \tag{6.5}$$

The $\alpha_{i,i+1}^{\mathsf{PINN}}$ equations hold only for the upwind node and its flux contributions from the left and right node for the positive velocity problems ($v > 0$). To be able to work for the negative velocity problems ($v < 0$), the i and j are switched and therefore we have to consider $f_{i+1,i+2}$ instead of $f_{i-1,i}$. Thus, the above equations will be as followed:

$$\alpha_{i+1,i}^{\mathsf{PINN}} \coloneqq \begin{cases} \mathsf{min}\{1, \dfrac{Q_{i+1}^+}{P_{i+1}^+}\}, & \text{if } f_{i+1,i} > 0 \\ \mathsf{min}\{1, \dfrac{Q_{i+1}^-}{P_{i+1}^-}\}, & \text{otherwise} \end{cases} \tag{6.6}$$

where

$$P_{i+1}^+ \coloneqq \mathsf{max}\{0, f_{i+1,i}\}+\mathsf{max}\{0, f_{i+1,i+2}\}, \qquad Q_{i+1}^+ \coloneqq \mathsf{max}\{0, -f_{i+1,i}\} + \mathsf{max}\{0, -f_{i+1,i+2}\} \tag{6.7}$$

$$P_{i+1}^- \coloneqq \mathsf{min}\{0, f_{i+1,i}\} \ +\mathsf{min}\{0, f_{i+1,i+2}\}, \qquad Q_{i+1}^- \coloneqq \mathsf{min}\{0, -f_{i+1,i}\} + \mathsf{min}\{0, -f_{i+1,i+2}\}. \tag{6.8}$$

The α_{ij} is computed at the time of computation and back-propagation is performed on these equations to update the weights and biases of neural network via automatic differentiation. Thus, the PINN is trained to find the best weights and biases by minimizing the loss and therefore representing the underlying non-linear input and output relationship of the AFC limiter.

6.3. Hyperparameter tuning

There are two main types of hyperparameters: model and algorithm hyperparameters. Model hyperparameters determine the model architecture like the number of fully connected layers in a network, while the algorithm hyperparameters are involved in the learning process of the network, like the batch size and learning rate. Optimizing these parameters can make the difference between an average model and a highly accurate model. Moreover, manually finding the optimal hyperparameters is a time-consuming process and a waste of resource power, so it is necessary to automatize this process. Therefore, hyperparameter optimization (HPO) or tuning is used as it shows to be critically beneficial for quickly maximizing model performance and minimizing the training cost.

Ray Tune [16], a Python library, is a HPO tool that can be used at any tuning scale from laptops to multiple machines without changing your code completely, so only a few lines of extra code are needed. This is the reason to choose Ray Tune and that it can be applied to various machine learning frameworks, like PyTorch, TensorFlow and Keras. Moreover, it includes a variety of state of the art and popular optimization algorithms, such as Population Based Training (PBT) and Bayesian optimization, which enables the user to explore them that matches their model. These algorithms reduce the cost of training by terminating bad trials early, choosing better parameters to evaluate or even change hyperparameters during training to optimize schedules.

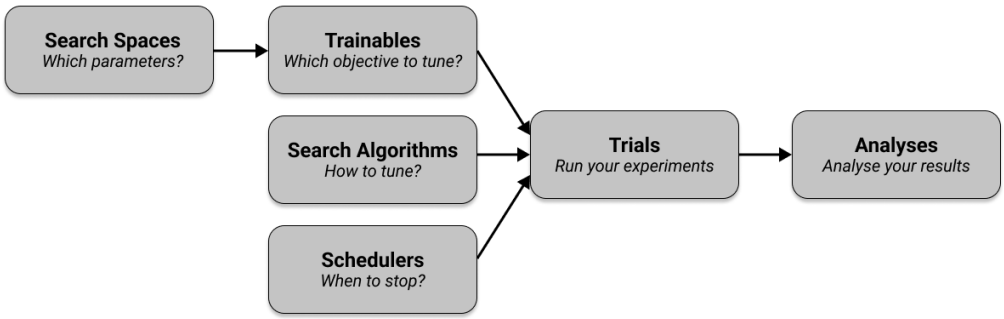


Figure 6.10: Key concepts of hyperparameter tuning with Ray Tune. Diagram from [16].

In Figure 6.10 the key concepts of Ray Tune are given. Firstly, the search space defines the values for the hyperparameters to optimize and how they are sampled, for example random normal or integers. This information is passed to the trainable, which has the objective functions that has to be tuned, which in our case is the neural network. During the training loop the trainable reports back scores for which the objective function is optimized and here the validation losses are chosen as the scores. To effectively optimize the hyperparameters a search algorithm is used that selects the hyperparameter configurations to evaluate. The default and most basic algorithms are random and grid search, where random search samples the parameter randomly from a distribution and grid search explores every configuration of the parameter. The algorithm Bayesian optimization/Hyperband (BOHB) is used for our approach since it terminates bad trials and also uses Bayesian Optimization to improve the hyperparameter search. To speed up the hyperparameter tuning process schedulers can be used that stop, pause or tweak the hyperparameters of running trials, but do not select which parameter configurations to evaluate. The first-in-first-out (FIFO) schedular is used as default, which as the name suggests goes through the trials in the same order they were created by the search algorithm and does not stop searches early. BOHB is intended to be paired with a specific schedular class, namely HyperBandForBOHB, which is a variant of the HyperBand schedular that is an early stopping algorithm [16]. Finally, various methods can be used to analyse the training process by Ray Tune, for example how the best trial or the best hyperparameter configuration for that trial can be accessed.

Best hyperparameters	**FFNN**	**SpNN**	**PINN**
Neurons per HL	42, 25, 24, 19, 20	16, 40, 56, 40, 14	27, 45, 32, 19, 28
Learning rate	0.001231..	0.003534..	0.001997..
Epochs	647	893	1221
Batch size	30	56	20

Table 6.2: Constant d_{ij} data: Best hyperparameter with Ray Tune of the three neural networks: FFNN, SpNN, PINN with five hidden layers.

Best hyperparameters	**FFNN**	**SpNN**	**PINN**
Neurons per HL	24, 37, 29, 30, 34	40, 10, 10, 14, 14	17, 23, 44, 20, 45
Learning rate	0.001804..	0.004606..	0.002445..
Epochs	1005	1168	1312
Batch size	60	24	20

Table 6.3: Variable d_{ij} data: Best hyperparameter with Ray Tune of the three neural networks: FFNN, SpNN, PINN with five hidden layers.

In Tables 6.2 and 6.3 the best hyperparameter configurations of the three neural networks for constant diffusion data and variable diffusion data are given respectively. Based on manual optimization of the hyperparameters, the search spaces were defined in a small or large range for each neural network. Therefore, the SpNN has a specific range for the model parameters and also due to its architecture,

while for the FFNN and PINN the neurons per hidden layer range is from 10 to 50. Moreover, the variable diffusion data is larger than the constant diffusion data, but a larger batch size range, namely 16 to 64, is used as the search space for both data sets. For the learning rate in the Adam optimizer a tuning range for the constant diffusion of 1e-4 to 1e-2 is considered, while for variable diffusion data a range of 1e-3 to 5e-3 is chosen because for this data set the losses were more stable.

Furthermore, initial weights and biases were first manually optimized before Ray Tune is used. Constant initial weights and biases did not yield better results than random values therefore normal or uniform distributions have been considered. By looking at the input and output values and the types of activation functions used in the neural networks, the weights were initialized with the Xavier Normalized distribution and the weights with the normal distribution.

6.4. Results

For the constant and variable diffusion data, the three trained and optimized neural networks are tested on the trained **boundary layer and peak problems** to benchmark their performance and accuracy. It must be pointed out that the peak problem solutions have a slope while the boundary layer problem solutions have a constant value before the steep regions occur, therefore, this problem has a more complex limiting behaviour. Moreover, the networks are tested on an untrained problem, namely where the boundary conditions of the boundary layer problem are switched, such as: $u(0) = 0$ and $u(1) = 1$, which is called the **variation of the boundary layer problem**. This is to obtain the limitations of the models (since the left side of the mapping plays a crucial part for that problem) and how well they are generalized. These problems are tested on a coarse mesh ($n = 20$) and fine meshes ($n = 50, 80$), since the mesh size ($1/n - 1$) influences the numerical accuracy and as consequence will influence the accuracy of the models, but how severe that is has to be investigated. Furthermore, these problems are tested for a Péclet range from ± 10.30 to ± 707.96, which is inside and outside the trained range, to observe if the models are limiting like AFC because as the Péclet numbers increase the solutions get steeper and therefore require more limiting. Moreover, AFC limits symmetrically for the negative and positive velocities, therefore the models are tested for both velocities for all problems.

To measure and compare the performance of the models the L^2-norm of the relative error of the solutions as described in chapter 5 is computed. Additionally, the solution profiles of the neural networks and numerical methods versus the exact solutions are computed for Péclet numbers $\pm 10, 20, 50, 100$ in the low Péclet range and $\pm 160, 210, 533.33, 700$ in the high Péclet range. For these values the pointwise difference between the exact solution versus the AFC and neural network solution are also computed in order to observe the differences between the solutions and if the neural network solution exceeds the analytic solution and/or the solution of AFC creating overshoots mainly at the steep region. Also, for converged α_{ij} the limited anti-diffusive flux, $\alpha_{ij} f_{ij}$, of the neural network and AFC is computed to see how much flux is actually limited and added back to the system. These results assist with the understanding how the models are limiting for which some interesting examples are shown and other cases can be found in appendix B.1. Finally, to gain more insight into the neural networks for both data sets the input and output mapping of predictions against AFC as target values are generated and can be found in the appendix B.2.

6.4.1. Feedforward neural network

The predictions of FFNN trained on the constant diffusion data are more accurate than those of the variable diffusion data based on the observations of the input and output mapping with the true values.

Figure 6.11 shows qualitatively that the models perform well both on the trained constant and variable diffusion data since the boundary layer problem solutions are similar to the AFC solutions, so the errors are in an acceptable range. However, they limit noticeably more than AFC for the low Péclet range ($Pe \leq (\pm)50$) on a coarse mesh resulting in the solutions being lower than the AFC solutions thus being less accurate, an example for $n = 20$ can be seen in Figure 6.12. The decreasing error trend in both velocity directions for all meshes shows that the models are limiting more as the Péclet number increases, which is expected and essential to limit the growing oscillations and therefore mimics AFC accurately and symmetrically.

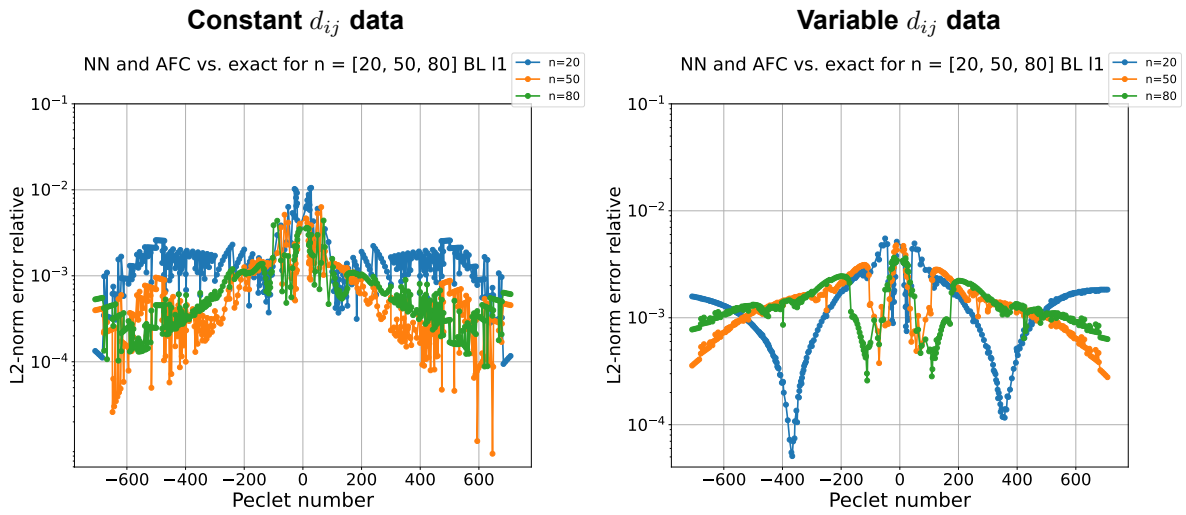


Figure 6.11: FFNN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

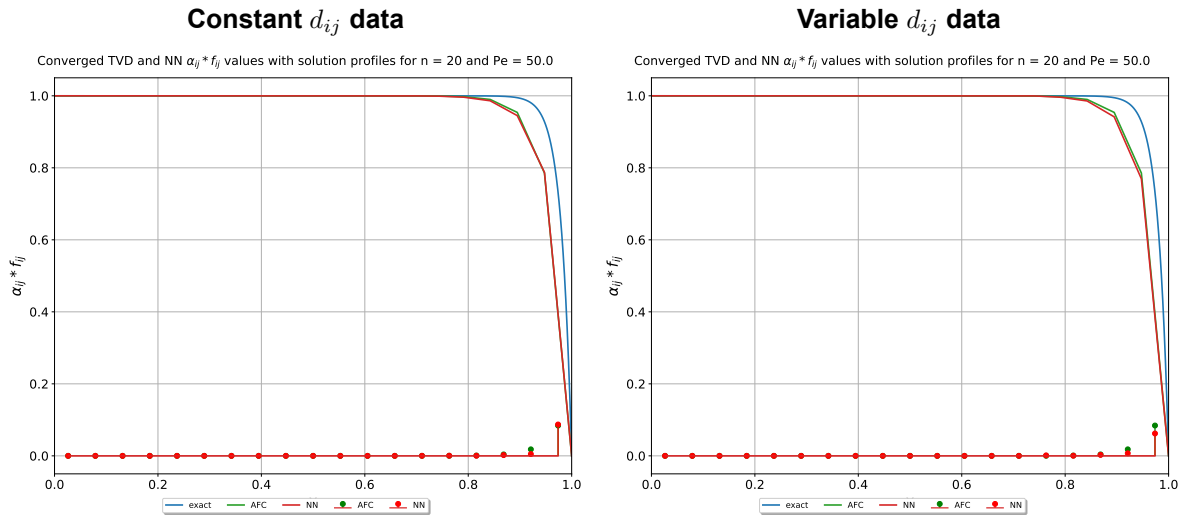


Figure 6.12: FFNN limited anti-diffusive flux results with the NN and AFC solution profiles against the exact solution for constant (left) and variable (right) d_{ij} data for $n = 20$ and $Pe = 50, 210$ for the boundary layer problem.

Figure 6.13 clearly shows that the errors are ten times larger as expected due to the nature of the solutions of the peak problem for both diffusion data sets compared to the previous problem. On the coarse mesh ($n = 20$) the models' solutions have noticeable overshoots for high Péclet value ($Pe \geq (\pm)100$), therefore, not limiting enough compared to the fine meshes, which shows negligible amount of overshoots and in some cases outperforms the AFC solutions (by being more accurate than the AFC solutions and closer to the analytical solutions) as shown for one case in Figure 6.14 with the solution differences. The constant diffusion data model is less accurate on the coarse mesh compared to the variable diffusion data model because the solutions also contain more oscillations as the Péclet values increases. Thus, variable diffusion data model is more accurate based on the $n = 20$ results for this problem and has a symmetric limiting pattern for very fine or coarse mesh.

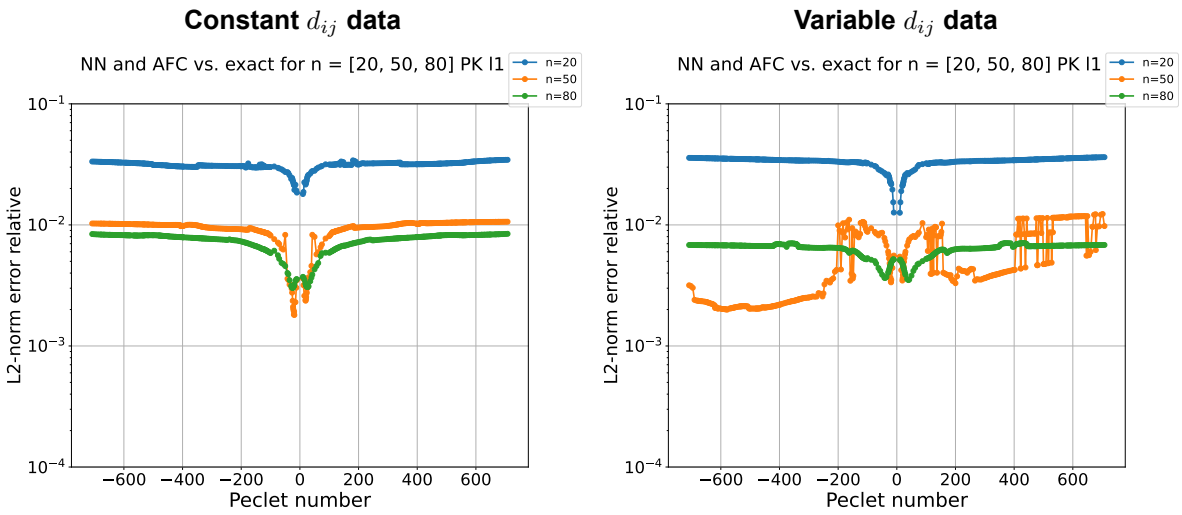


Figure 6.13: FFNN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

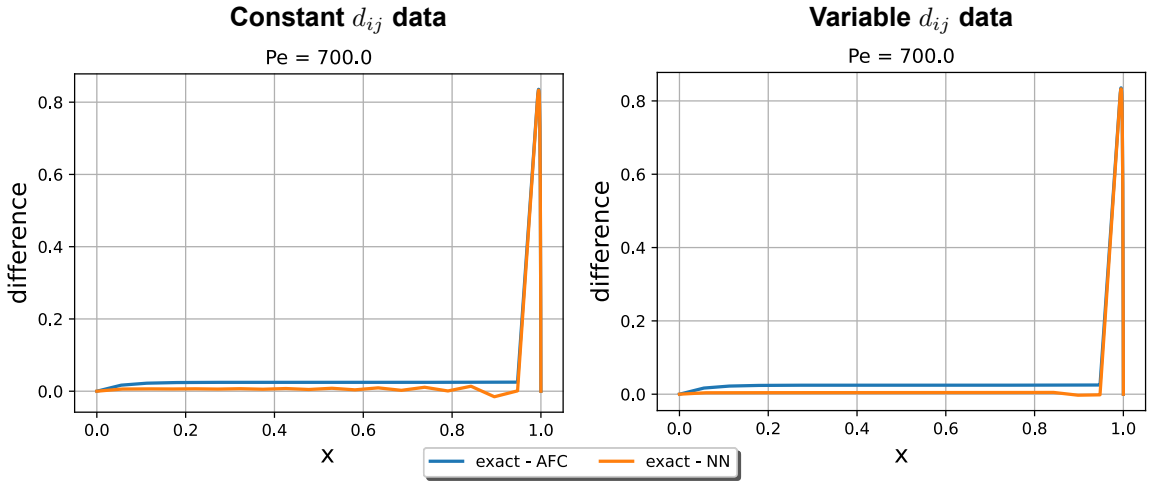


Figure 6.14: FNN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (left) and variable diffusion data (right).

Figure 6.15 shows that the errors for the models on untrained problem, that is the variation of the boundary layer problem, are 10 to 100 times larger compared to the the trained problems. However, the constant diffusion model produces more accurate results compared to the variable diffusion model for all Péclet values on all meshes. From the solution profiles, see examples in Figure 6.16, and differences results it can observed that in general on a coarse mesh the solutions contain a significant amount of overshoots than on fine meshes due to the fact that less points represent the steep region. Furthermore, the variable diffusion data model has a large amount of oscillations on the fine meshes for high Péclet values ($Pe \leq 100$) compared to the constant diffusion model, which has a negligible amount. Overall, the model trained on constant diffusion data is more symmetric and accurate and therefore more generalizable compared to model trained on the variable diffusion data.

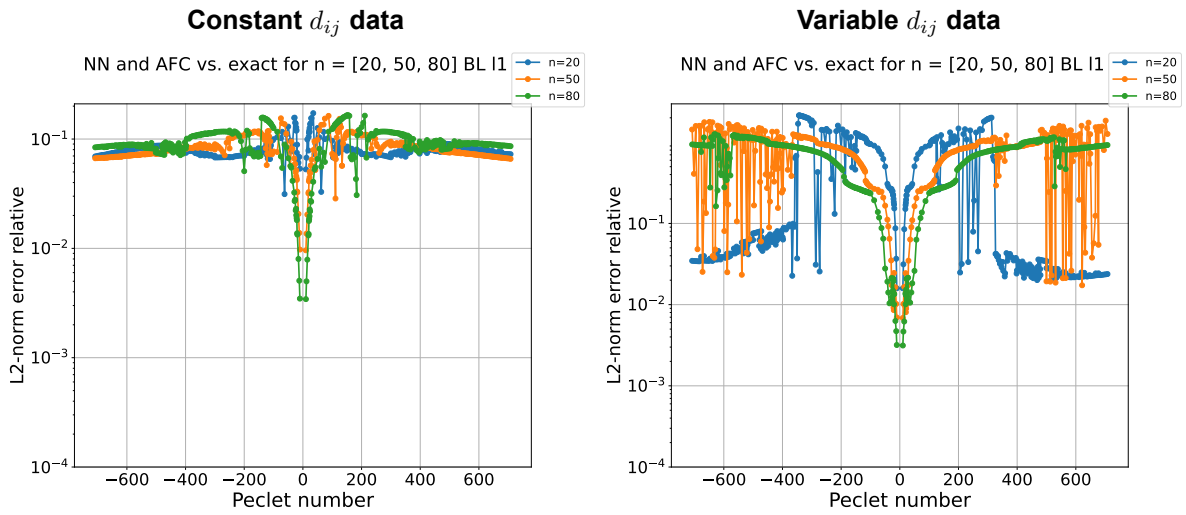


Figure 6.15: FFNN L2 relative error for **variation of boundary layer problem (untrained)** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

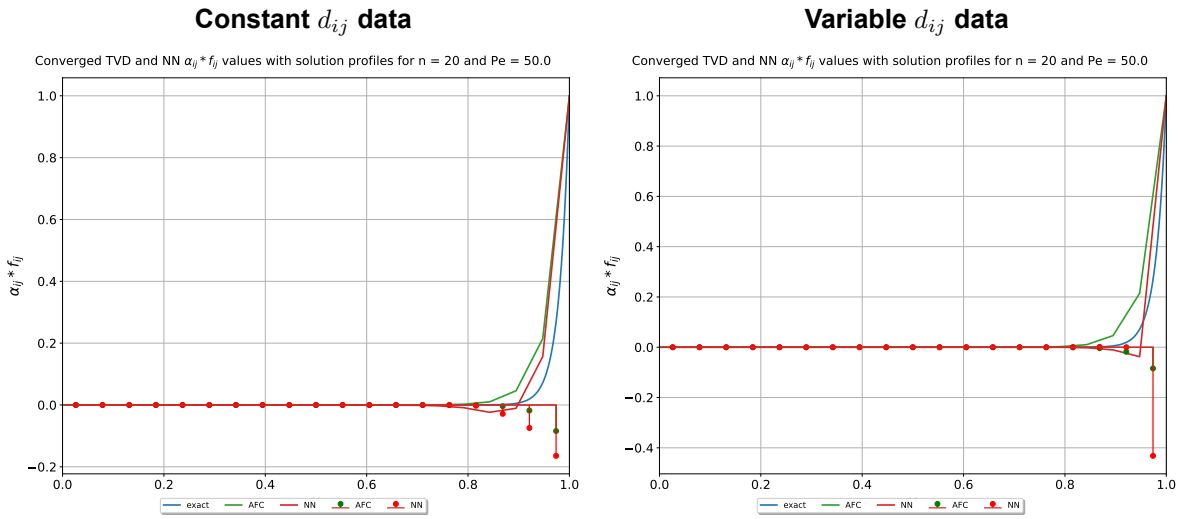


Figure 6.16: FFNN limited anti-diffusive flux results with the NN and AFC solution profiles against the exact solution for constant (left) and variable (right) d_{ij} data for $n = 20$ and $Pe = 50$ for the untrained problem.

A summary of the results for the tested problems is given in Table 6.4 for the FFNN. Based on the L^2 error, the solution profiles, differences and limited anti-diffusive fluxes plots the results of the problems have been categorized in three classes of accuracy. Firstly, the sufficiently accurate class is where the results matches AFC the most or is similar to AFC for all meshes and Péclet numbers. Secondly, the moderately accurate class is where the results are close to AFC for fine meshes ($n = 50, 80$), but solutions on a coarse mesh ($n = 20$) are significantly under-and/or overshooting the AFC solutions and therefore improving the model is recommended. Thirdly, the not accurate class is where the solutions contain oscillations and/or overshoots the AFC and analytical solutions for all n values and therefore improving the model is required.

FeedForward Neural Network	**Constant d_{ij} data**	**Variable d_{ij} data**
$\pm$ BL problem	$\surd$	$\surd$
$\pm$ PK problem	$+/-$	$\surd$
$\pm$ varBL problem	$+$	$-$

$\surd$ = sufficiently accurate; $+$ = moderately accurate; $-$ = not accurate;

Table 6.4: FFNN comparison table between the constant and variable d_{ij} data tested on the trained problems: boundary layer (BL) and peak (PK) problem, and the untrained problem: variation of boundary layer (varBL) problem in positive and negative velocity directions ($\pm$) for all meshes.

By having these two diffusion data sets gives the opportunity to understand how the FFNN architecture learns and what the impact of the architecture is on the performance and accuracy. Thus, a general observation is that the FFNN is better generalized with the constant diffusion data when compared with the variable diffusion data, which also results from the input and output mapping predictions and indicates that the former data set has a simpler mapping which is more suitable for this architecture to learn the AFC limiter. In section 6.4.4 the variation of the boundary layer problem is included in the training data to see if these results can be improved.

6.4.2. Splitted neural network

The predictions of SpNN trained on both diffusion data sets are similarly based on the observations of the input and output mapping with the true values.

Figure 6.17 shows that the SpNN models for both data sets have similar error range and also a decreasing trend as the FFNN results for the boundary layer problem on all meshes. This means it is also limiting more as the Péclet number increases, but both data sets have different limiting behaviour than AFC. The constant diffusion data model limits less than AFC in some Péclet cases on all meshes, while, the variable diffusion model limits more than AFC in those cases. Nevertheless, the solutions of the models are similar to AFC for all meshes and Péclet values, but that of the variable diffusion data are closer to the AFC solutions on a coarse mesh for the low Péclet range ($Pe = \pm 10 - \pm 100$) and are more symmetric in both velocity directions.

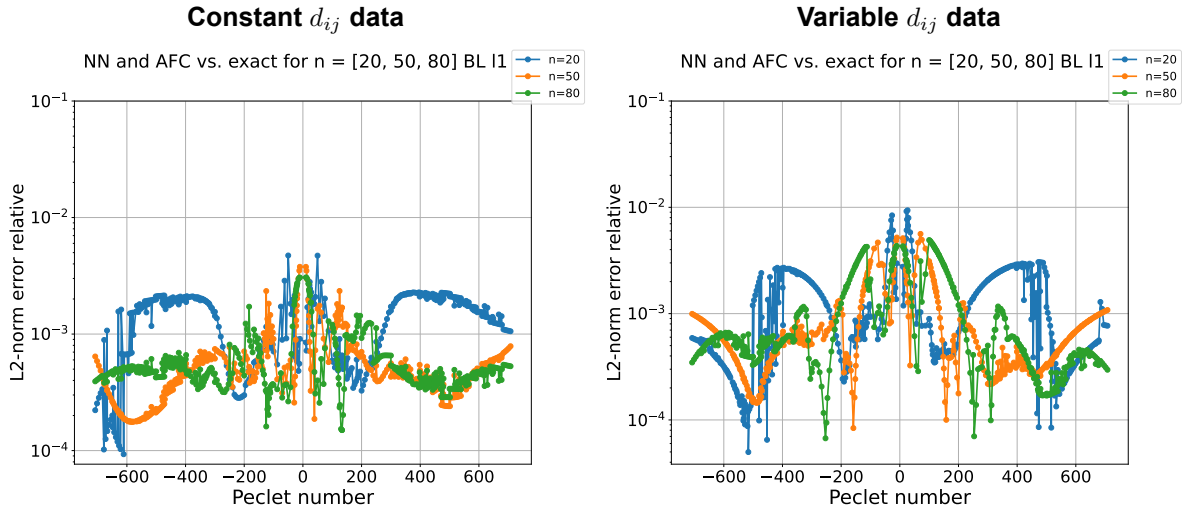


Figure 6.17: SpNN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

From Figure 6.18 it is clear that variable diffusion data model is performing more symmetrically than the constant diffusion data model on all meshes. Moreover, for both models the limiting behaviour on a coarse mesh differs significantly (see one example in Figure 6.19). For example, at low Péclet range ($Pe = \pm 10 - \pm 100$) they both generate similar solutions as AFC, but at high Péclet values ($Pe > (\pm)100$) the solutions of the constant diffusion model have small overshoots in the vicinity of the steep region, therefore, not limiting enough like AFC. Whereas, the solutions of the variable diffusion model is limiting more than AFC at the smooth region and accurately limits the oscillations in the steep region (see one example in Figure 6.20). In general for the peak problem, the SpNN outperforms AFC on accuracy and performance with the variable diffusion data on all meshes, while, with the constant diffusion data it generates unstable and inconsistent limiting behaviour.

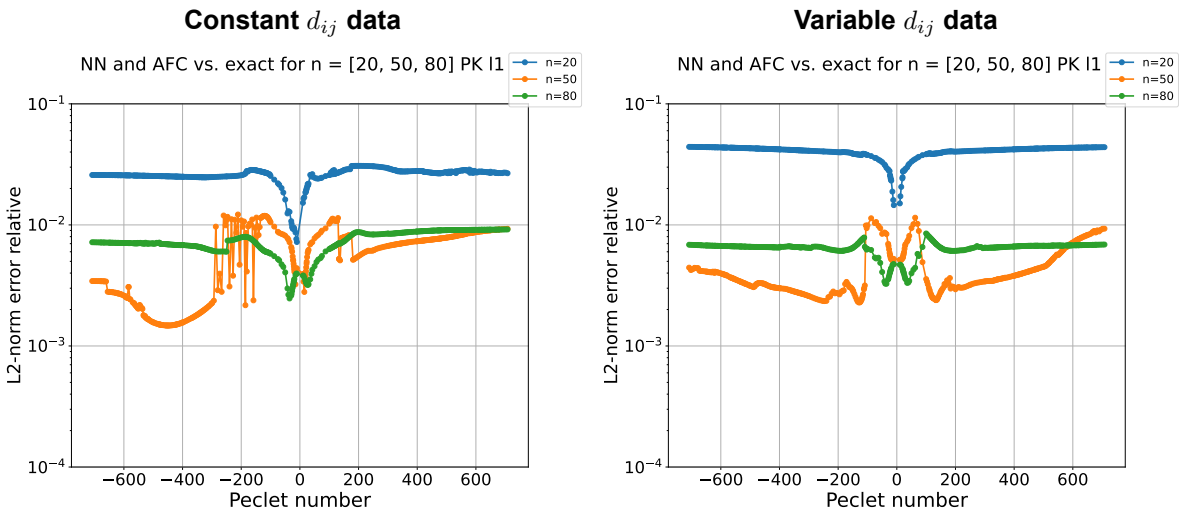


Figure 6.18: SpNN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

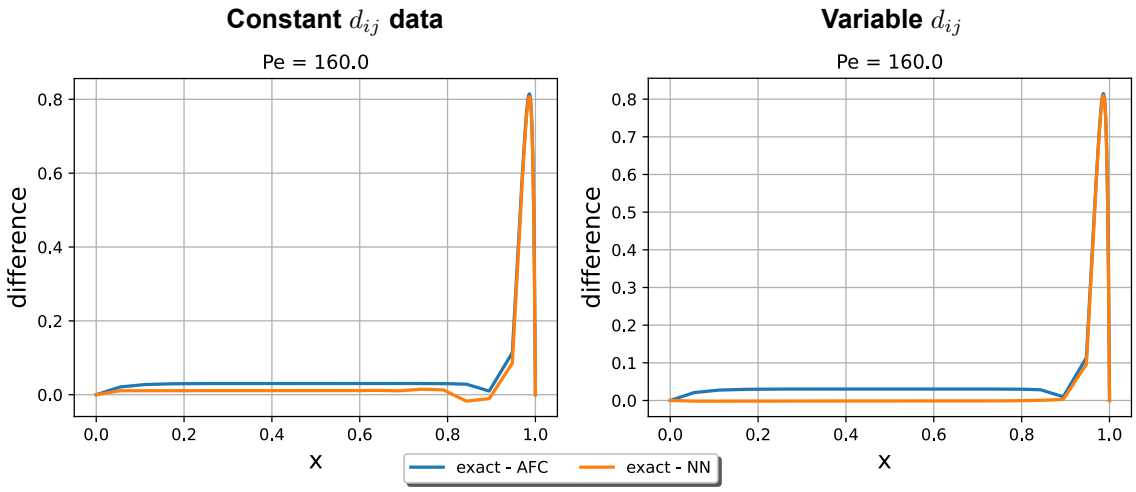


Figure 6.19: SpNN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (left) and variable diffusion data (right).

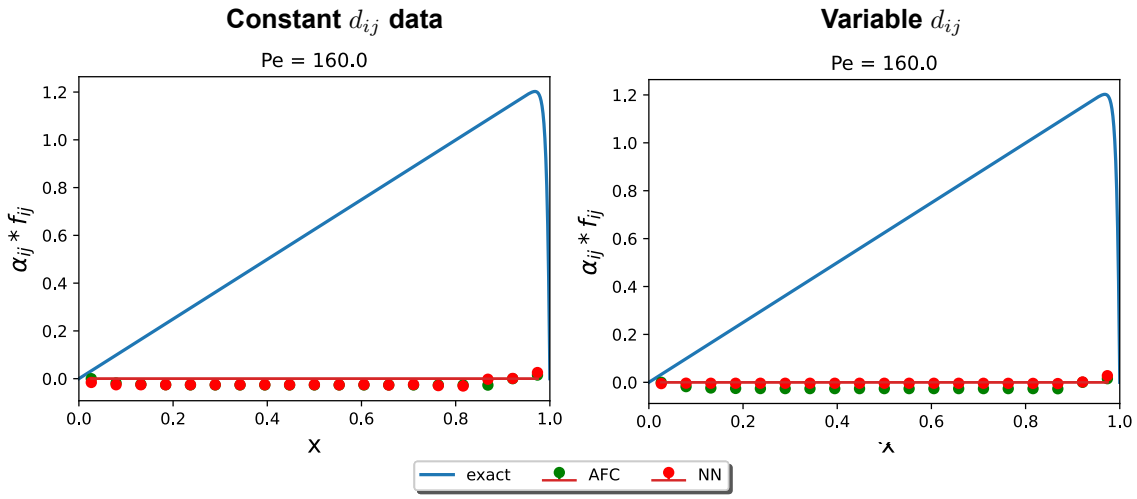


Figure 6.20: SpNN limited anti-diffusive flux results with the exact solution for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (left) and variable diffusion data (right).

Figure 6.21 qualitatively shows that the variable diffusion model is more accurate and symmetric than the constant diffusion model for all meshes and Péclet values. The constant diffusion data model has solutions with increasing overshoot amplitude as the Péclet number increases, which appear on the coarse mesh at lower Péclet values compared to the fine meshes, therefore, not limiting sufficiently. The variable diffusion model prevents oscillations and overshoots from occurring in the solutions, but at the cost of limiting more than required, which is expected due to the mapping. An example for both data sets is shown in Figure 6.22. Moreover, it has been observed that for all meshes the solutions are closer to the upwind solutions for low Péclet values, therefore, limiting more than AFC, but as the Péclet number increases the solutions get closer to AFC. Moreover, the model was also tested on more coarser meshes ($n = 5, 10$) for all problems and all Péclet values and the solutions were free of overshoots and oscillations and were more accurate on the smooth regions than AFC, which makes the model mesh independent.

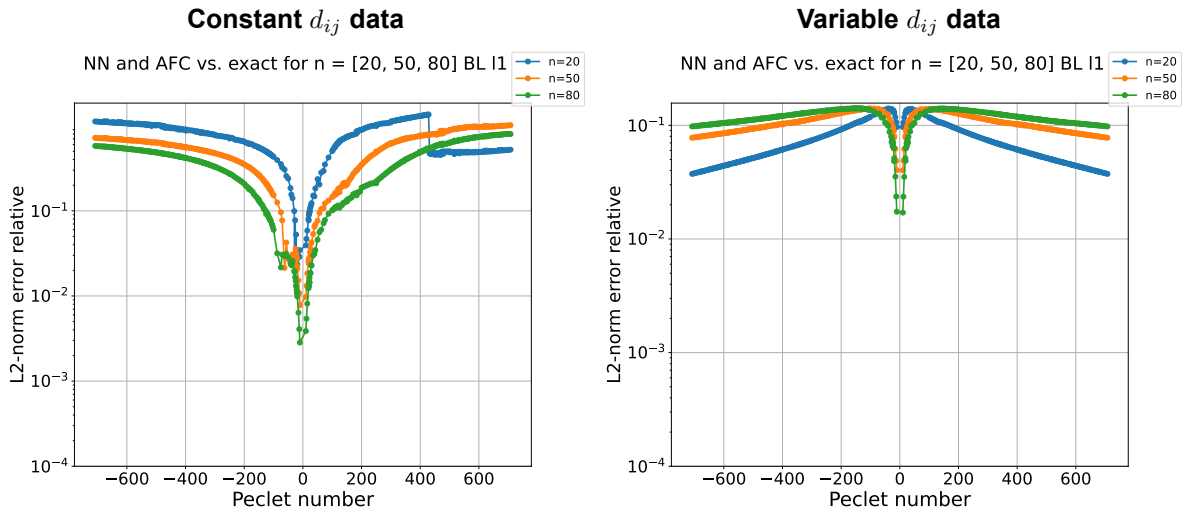


Figure 6.21: SpNN L2 relative error for **variation of boundary layer problem (untrained)** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

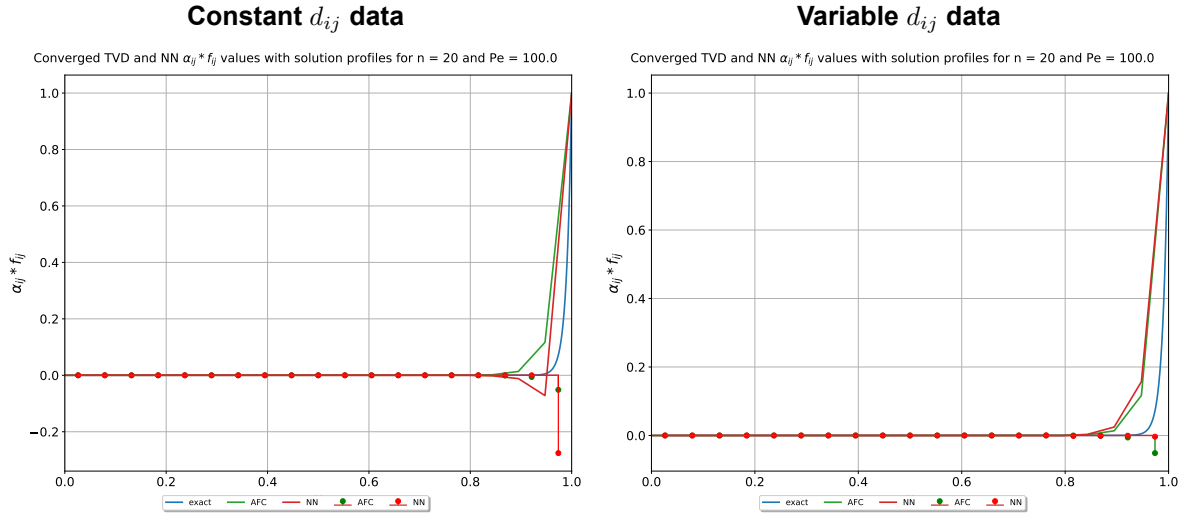


Figure 6.22: SpNN limited anti-diffusive flux results with the NN and AFC solution profiles against the exact solution for constant (left) and variable (right) d_{ij} data for $n = 20$ and $Pe = 100$.

A summary of the results for the tested problems is given in Table 6.6 for the SpNN. Based on the results, the problems have been categorized in three classes of accuracy as defined before.

Splitted Neural Network	Constant d_{ij} data	Variable d_{ij} data
$\pm$ BL problem	$\surd$	$\surd$
$\pm$ PK problem	$+/\surd$	$\surd$
$\pm$ varBL problem	$-$	$+$

$\surd =$ sufficiently accurate; $+ =$ moderately accurate; $- =$ not accurate;

Table 6.5: SpNN comparison table between the constant and variable d_{ij} data tested on the trained problems: boundary layer (BL) and peak (PK) problem, and the untrained problem: variation of boundary layer (varBL) problem in positive and negative velocity directions ($\pm$) for all meshes.

Overall, it can be observed that the SpNN architecture is suitable more for the variable diffusion data, so it might be better at learning a more complex relation/mapping than the FFNN architecture. In the section 6.4.4, it is investigated if the performance could improve by training on the above tested problems.

6.4.3. Physics-informed neural network

It must be taken into account that the training time of this neural network took considerably longer than the other network architectures, which resulted in less samples being optimized with Ray Tune than the previous network architectures. So a more optimized model could be developed if a longer training time is given to this architecture.

The predictions of PINN trained on the constant diffusion data are more accurate than those of the variable diffusion data based on the observations of the input and output mapping with the true values.

Likewise to the two previous network architecture, the errors of the PINN for the boundary layer problem is also around 10^{-3} for all meshes as shown in Figure 6.23, but based on the observations this network has the most symmetric error so far for both data sets for this problem. The solutions for both models are similar to AFC on fine meshes ($n = 50, 80$), therefore, sufficiently accurate, but on coarse mesh ($n = 20$) they are slightly overshooting AFC in the low Péclet range ($Pe \leq 50$) for variable diffusion data and in the high Péclet range ($Pe > 100$) for constant diffusion data, so are limiting less than AFC here.

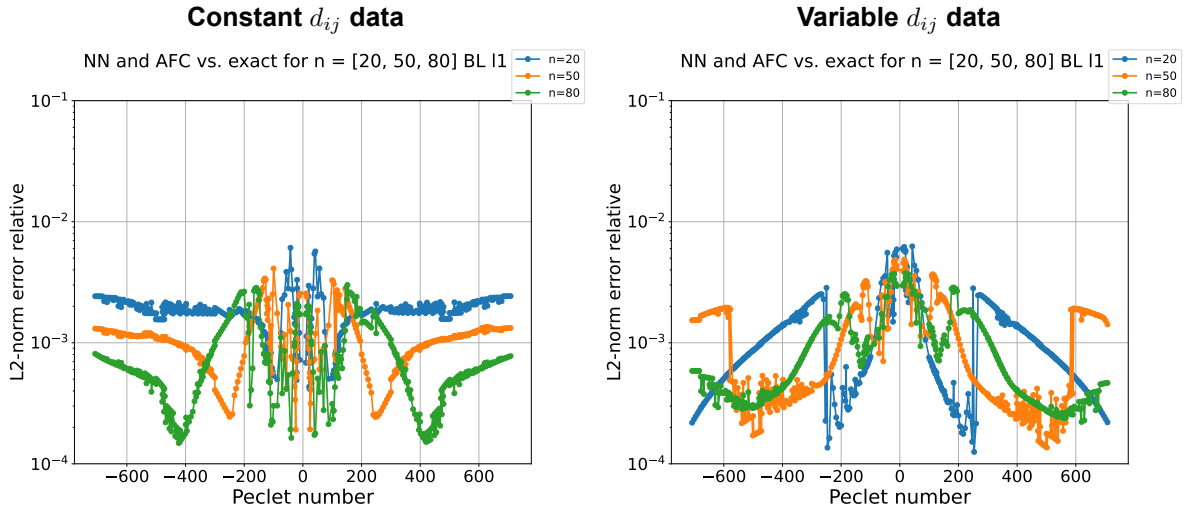


Figure 6.23: PINN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

Figure 6.24 present the error for the peak problems for both data sets. It is clear that the accuracy is higher on fine meshes than on a coarse mesh. Both models' solutions contain small overshoots on the coarse mesh for all Péclet values (see one example with solution differences in Figure 6.25), which start for constant diffusion data at high Péclet numbers ($Pe > 100$) and for variable diffusion data it starts at low Péclet numbers ($Pe > 50$) but decreases as the Péclet values increase. However, they outperform

AFC on the straight region of the solution like in the previous two neural network architectures on all meshes. On fine meshes the solutions are sufficiently accurate. Based on the coarse mesh, variable diffusion data would be more accurate for this problem.

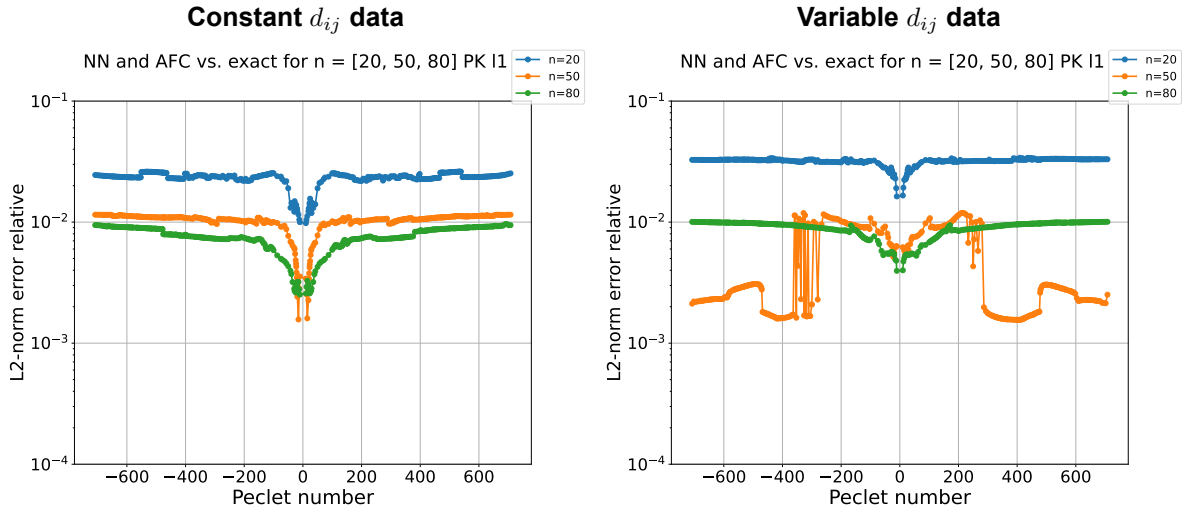


Figure 6.24: PINN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

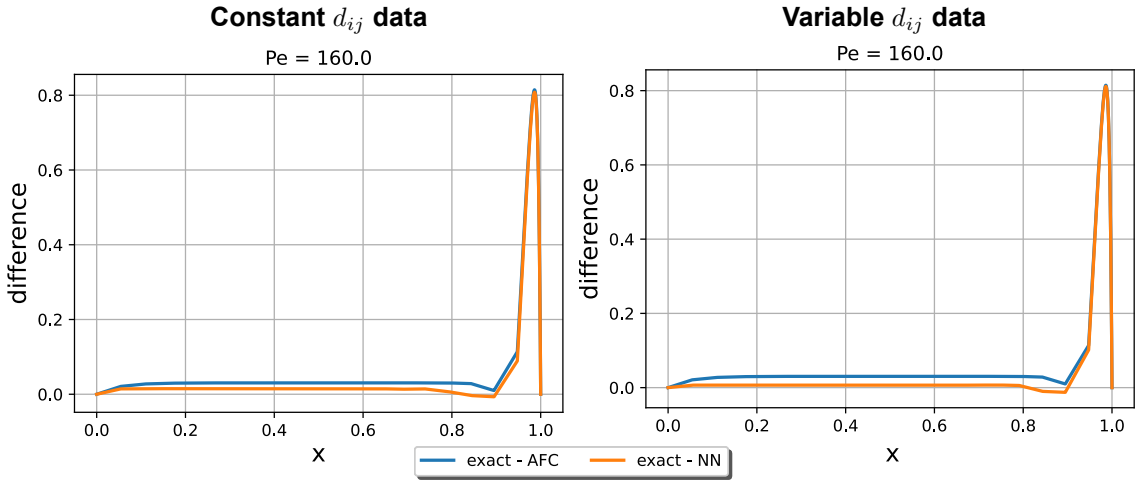


Figure 6.25: PINN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (left) and variable diffusion data (right).

For the variation of boundary layer problem the result is given in Figure 6.26, which shows that symmetry is preserved, but the model trained on constant diffusion data is more accurate than on variable diffusion data. Although, the solutions of both models contain overshoots on all meshes (see one example with solution differences in Figure 6.27). The constant diffusion model has an increase in small overshoots in the solutions (for $n = 20$ at $Pe \geq 50$, and for $n = 50, 80$ at $Pe \geq 100$), but they start to decrease when the Péclet number start to increase, therefore, limiting more. Also, for this data the overshoots gets larger as the meshes get finer, while, for the variable diffusion data it is the opposite, but are still larger than constant diffusion data. Overall, both data sets only produce accurate results for low Péclet numbers with this neural network unlike the previous two neural network architectures.

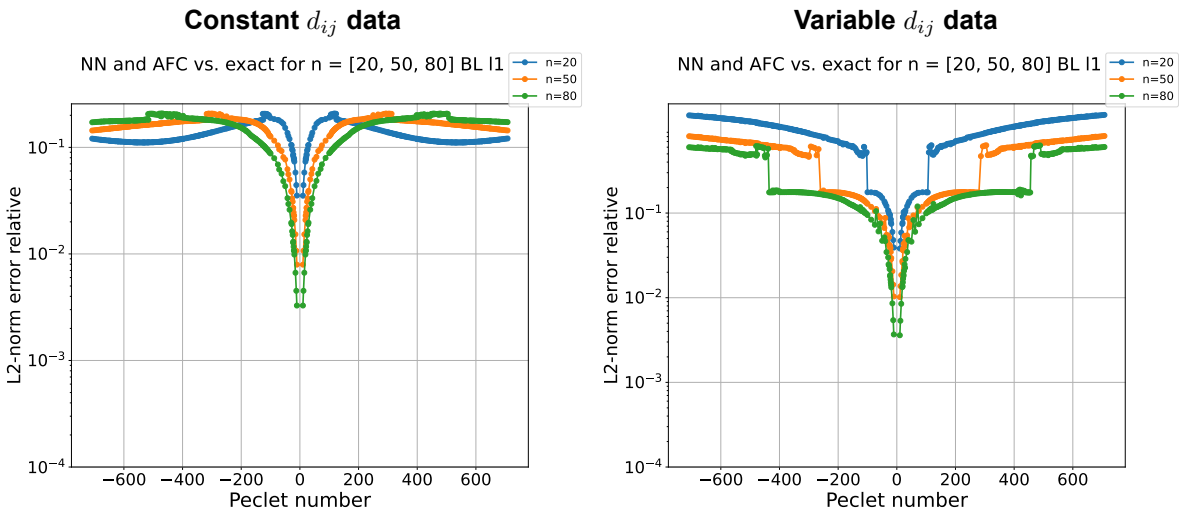


Figure 6.26: PINN L2 relative error for **variation of boundary layer problem (untrained)** with positive and negative Péclet numbers for $n = 20, 50, 80$ for constant diffusion data (left) and variable diffusion data (right).

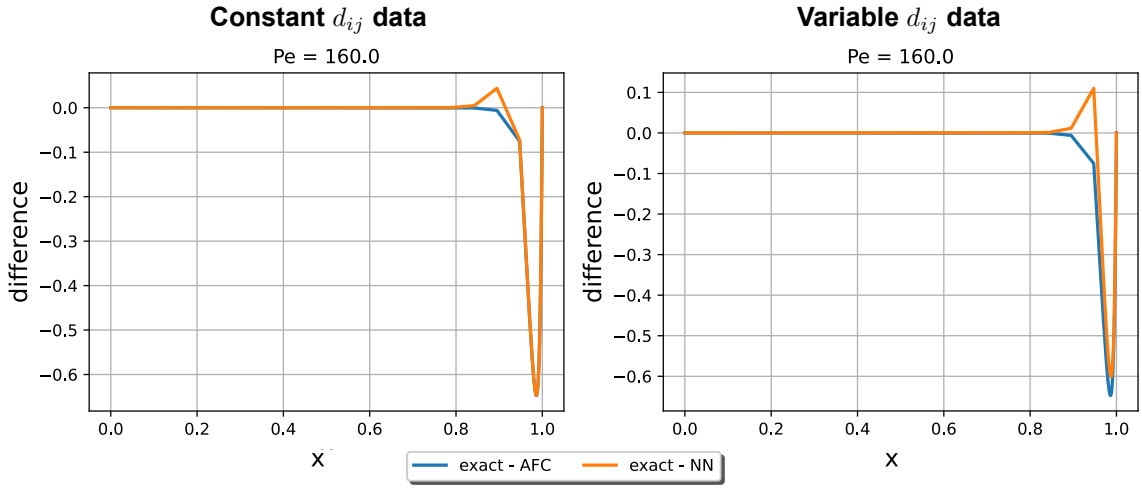


Figure 6.27: PINN pointwise differences plots for the variation of the boundary layer problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (left) and variable diffusion data (right).

A summary of the results for the tested problems is given in Table 6.6 for the PINN. Based on the results, the problems have been categorized in three classes of accuracy as defined before.

Physics-informed Neural Network	**Constant d_{ij} data**	**Variable d_{ij} data**
$\pm$ BL problem	$\surd$	$\surd$
$\pm$ PK problem	$+/\surd$	$+$
$\pm$ varBL problem	$-$	$-$

$\surd$ = sufficiently accurate; $+$ = moderately accurate; $-$ = not accurate;

Table 6.6: PINN comparison table between the constant and variable d_{ij} data tested on the trained problems: boundary layer (BL) and peak (PK) problem, and the untrained problem: variation of boundary layer (varBL) problem in positive and negative velocity directions ($\pm$) for all meshes.

Overall, the PINN is not generalizable with these two types of data because of the results of the untrained problem. In the section 6.4.4 it will be investigated if this improves when more data is used to train this network.

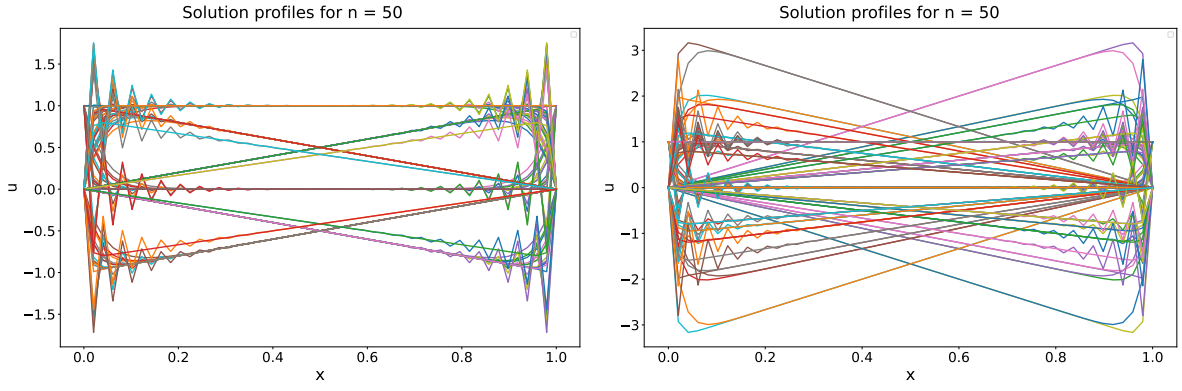
6.4.4. Improvements on previous results

For the three types of neural network architectures and the two data sets the previous results on the untrained problem, namely the variation of the boundary layer problem, were not all accurate. Therefore, this section will investigate if it is possible to improve those results. To achieve this, the solutions to the variation of the boundary layer problem are included in the previous constant and variable d_{ij} data sets to let the networks learn that problem, but also a variation of the peak problem is included, namely where the source of the problem is -1 instead of 1, which will invert the solutions as shown in Figure 6.28. The variation of the problems mirrors the previous mapping, which mainly consisted of non-zero α_{ij} values corresponding to the positive fluxes therefore also generating the mapping on the negative flux side, which results in **symmetric data** as shown in Figures 6.29 and 6.30 for constant and variable d_{ij} data respectively. The previous data will be called **asymmetric data** henceforth.

Moreover, the symmetric data is larger than the asymmetric data and challenges the models to learn when to use which side of the mapping. Therefore, they have to learn which solution profile requires which side of the mapping. Thus, this investigates if the models can learn the AFC limiter behaviour in both ways, so limit increasing and decreasing solutions.

To measure the accuracy and performance of the models similar results are produced as the previous results and also the hyperparameters are tuned with RayTune to optimize the models as shown in Table 6.7 and Table 6.8 for constant and variable diffusion data respectively.

To compare the models the results for the networks trained on asymmetric and symmetric data are shown. The results are also shown for the constant and variable diffusion separately. Finally, to gain more insight into the neural networks for these data sets the input and output mapping of predictions against AFC as target values are generated and can be found in the appendix B.3. Here the L^2 solution errors are also included for all neural networks and for all tested problems. Finally, some interesting results are shown in the results below and more cases can be found in the appendix B.4.



(a) constant diffusion coefficient $d_{ij} = 0.5$

(b) variable diffusion coefficient d_{ij}

Figure 6.28: The solution profiles for constant and variable d_{ij} symmetric data to compute the fluxes and corresponding alpha values to train the networks. The Galerkin and upwind solutions, both for peak (source=1), boundary layer ($u(0) = 1, u(1) = 0$) and variations of peak (source=-1) and boundary layer problems ($u(0) = 0, u(1) = 1$) (a) for Péclet number from ± 30 to ± 700. (b) for Péclet number from ± 50 to ± 700.

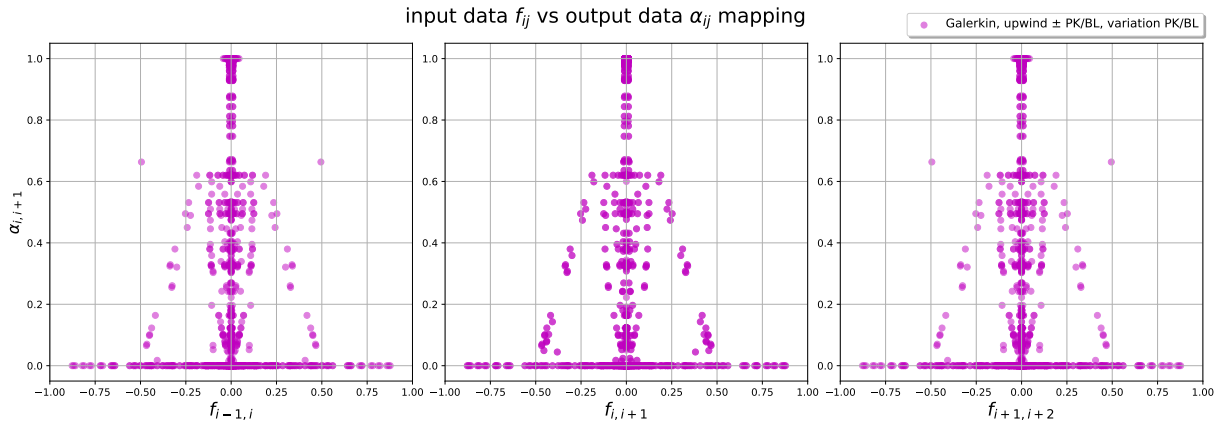


Figure 6.29: Input-output mapping to train the neural networks for constant diffusion coefficient $d_{ij} = 0.5$ constructed from Galerkin and upwind solutions for both for peak (source=1), boundary layer ($u(0) = 1, u(1) = 0$) and variations of peak (source=-1) and boundary layer problems ($u(0) = 0, u(1) = 1$) for fixed number of basis functions, $n = 50$, and Péclet numbers ranging from ± 30 to ± 700.

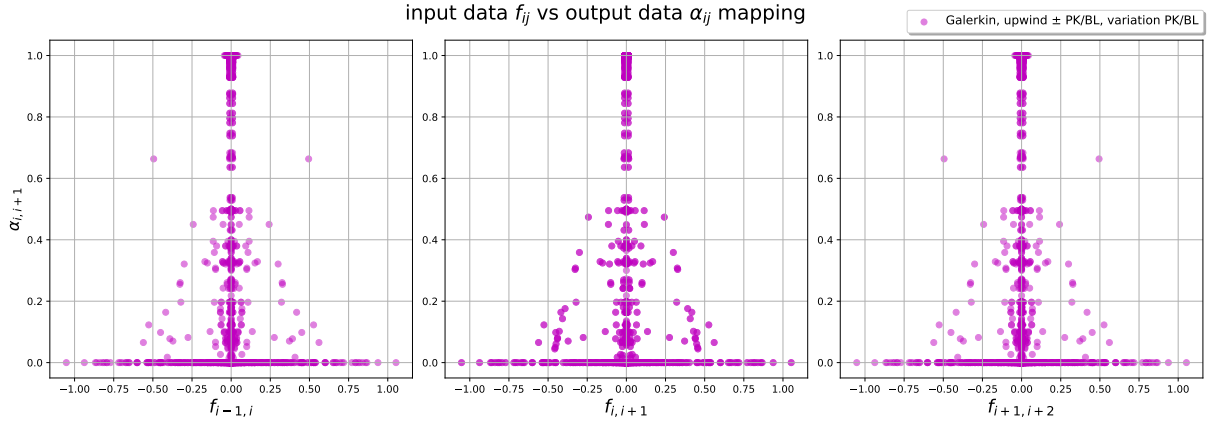


Figure 6.30: Input-output mapping to train the neural networks for variable diffusion coefficient d_{ij} constructed from Galerkin and upwind solutions for both for peak (source=1), boundary layer ($u(0) = 1, u(1) = 0$) and variations of peak (source=-1) and boundary layer problems ($u(0) = 0, u(1) = 1$) for fixed number of basis functions, $n = 50$, and Péclet numbers ranging from ± 50 to ± 700.

To measure the accuracy and performance of the models similar results are produced as the previous results and also the hyperparameters are tuned with RayTune to optimize the models as shown in Table 6.7 and Table 6.8 for constant and variable diffusion data respectively.

To compare the models the results for the networks trained on asymmetric and symmetric data are shown. The results are also shown for the constant and variable diffusion separately. Finally, to gain more insight into the neural networks for these data sets the input and output mapping of predictions against AFC as target values are generated and can be found in the appendix B.3. Here the L^2 solution errors are also included for all neural networks and for all tested problems.

Best hyperparameters	**FFNN**	**SpNN**	**PINN**
Neurons per HL	17,40,21,33,29	48,10,20,14,20	11,33,19,42,18
Learning rate	0.00413..	0.00396..	0.00292..
Epochs	1124	1446	1298
Batch size	58	30	42

Table 6.7: Constant d_{ij} symmetric data: best hyperparameter with Ray Tune for the neural networks: FFNN, SpNN, PINN with five hidden layers.

Best hyperparameters	FFNN	SpNN	PINN
Neurons per HL	19,19,25,49,21	40,20,10,34,12	13,50,13,19,42
Learning rate	0.00100..	0.00386..	0.00187..
Epochs	1467	1242	1019
Batch size	28	40	28

Table 6.8: Variable d_{ij} symmetric data: best hyperparameter with Ray Tune for the neural networks: FFNN, SpNN, PINN with five hidden layers.

Feedforward Neural Network

The predictions of FFNN trained with symmetric variable diffusion data are more accurate than those of the symmetric constant diffusion data based on the observations of the input and output mapping with the true values.

The model trained on symmetric constant d_{ij} data has for the boundary layer problem small observable differences on all meshes compared to asymmetric data, but for both sets the solutions are relatively close to AFC solutions. However, the former data set is significantly more accurate for low Péclet values. For the peak problem this data set performs more accurate on a coarse mesh for all Péclet numbers (for $n = 20$ see Figure 6.31), but on fine meshes the asymmetric data is better. Lastly, for the variation of the boundary layer problem the symmetric data achieves more accurate results than asymmetric data for all cases.

The model trained on symmetric variable d_{ij} data has been improved significantly by this data set for the variation of the boundary layer problem because the solutions have no overshoots exceeding the analytical solution on all meshes and Péclet values, see one case in Figure 6.32. For other problems the symmetric data results remained similar or were in some cases better than the asymmetric data results.

To conclude, the FFNN achieves the most accurate results with the symmetric variable d_{ij} data for all problems on most meshes and Péclet values compared to the constant d_{ij} data and asymmetric data.

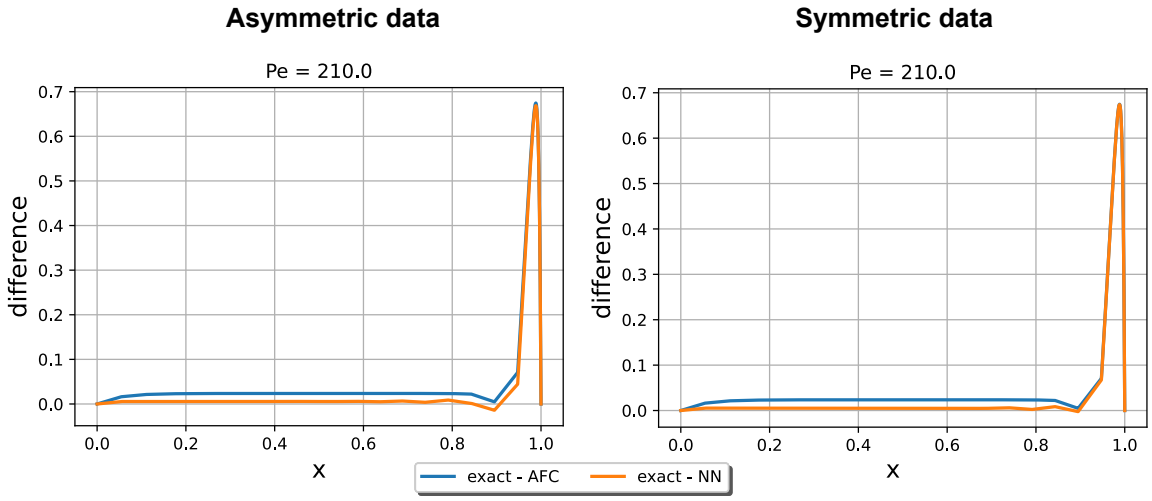


Figure 6.31: FFNN pointwise differences plot for $n = 20$ for constant d_{ij} data with asymmetric data (left) and symmetric data (right) for the peak problem for one of the Péclet values.

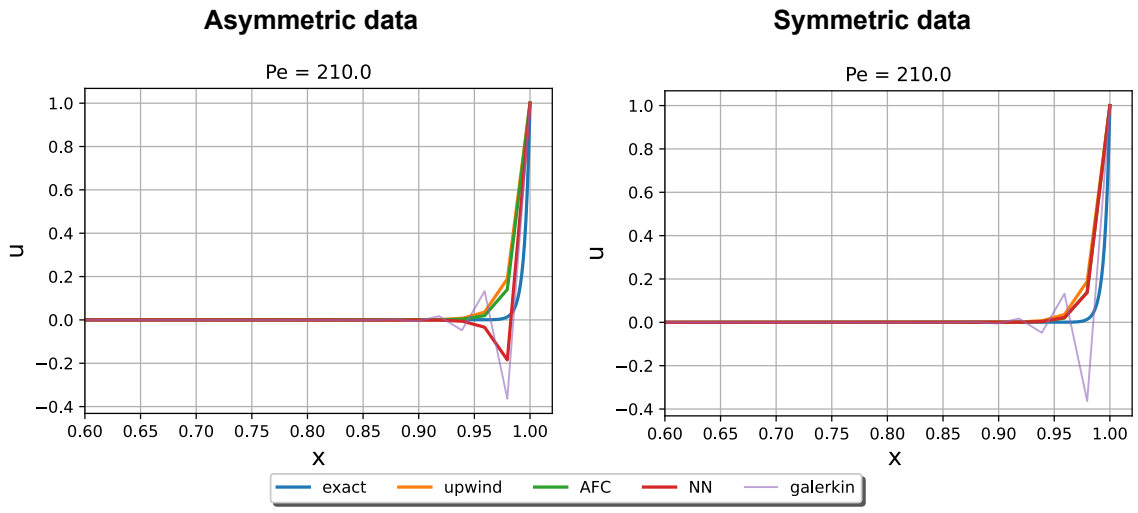


Figure 6.32: FFNN solution profiles for $n = 50$ for variable d_{ij} data with asymmetric data (left) and symmetric data (right) for the variation of the boundary layer problem for one of the Péclet values.

Splitted Neural Network

The predictions of SpNN trained with symmetric variable diffusion data are more accurate than those of the symmetric constant diffusion data based on the observations of the input and output mapping with the true values.

The model trained on symmetric constant d_{ij} data has been improved and is now more accurate for the variation of the boundary layer problem compared to the asymmetric d_{ij} data for all meshes and Péclet values. For example, the solution as shown in Figure 6.33 have no overshoots present. For the peak problem, the results have also slightly improved as well since the solutions have become closer to AFC compared to the asymmetric data results, but this leads to the fact that it does not outperform AFC anymore for all cases. The boundary layer problem are similar for both data sets.

The model trained on symmetric data variable d_{ij} data performs slightly more accurate on the variation of the boundary layer problem for all cases than for the asymmetric data, as shown for example for $n = 50$ in Figure 6.34. The same has been observed in the $n = 20, 80$ solution results for this problem. For the peak problem the solutions remained similar for fine meshes, but on a coarse mesh they are slightly closer to AFC on the straight region, which also affects the outperformance. The boundary layer problem results are similar for both data sets.

Both constant and variable d_{ij} data sets performed similar for boundary layer and its variation problems with this neural network architecture. However, based on the peak problem on all meshes the variable d_{ij} data results were in general more accurate than AFC and therefore also more accurate than the constant d_{ij} data results, which solutions appear to be closer to the AFC solutions.

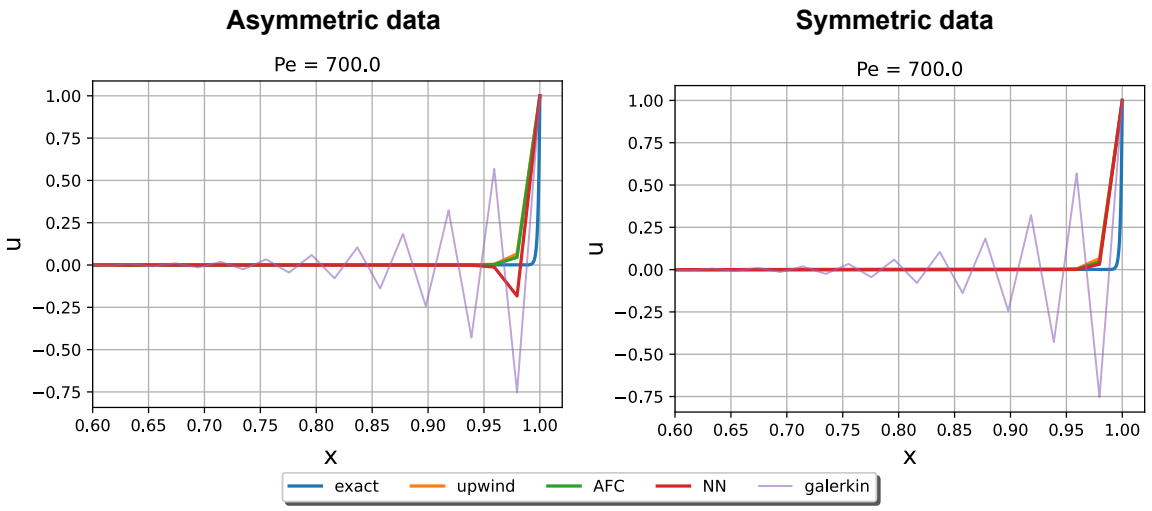


Figure 6.33: SpNN solution profiles for $n = 50$ for constant d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem for one of the Péclet values.

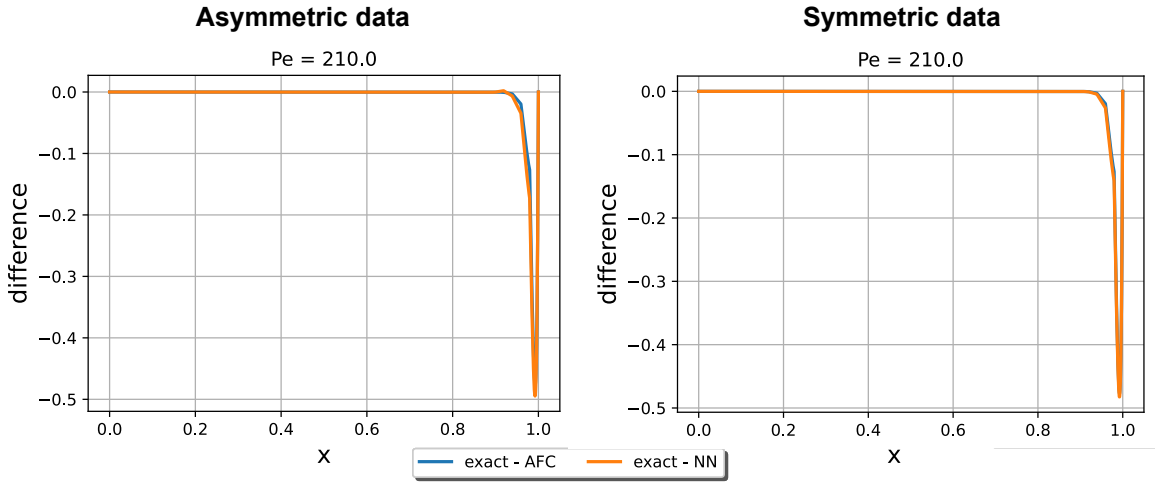


Figure 6.34: SpNN differences plot for $n = 50$ for variable d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem for one of the Péclet values.

Physics-informed Neural Network

The predictions of PINN trained with symmetric variable diffusion data are more accurate than those of the symmetric constant diffusion data based on the observations of the input and output mapping with the true values.

The results for the constant d_{ij} data are also improved by the symmetric data for the variation of the boundary layer problem for fine meshes and all Péclet values, see an example for $n = 50$ in Figure 6.35. However, on a coarse mesh the solutions have small noticeable under-and overshoots present, therefore, this is inconsistent limiting. The results on the peak problem have also been observably improved on and on the boundary layer problem remained similar to the asymmetric data, only for the latter problem it losses performance on a coarse mesh for low Péclet values.

With symmetric variable d_{ij} data the results of the model also performs more accurate on fine meshes for the variation of the boundary layer problem than with the asymmetric data see in Figure 6.36 an example. However, on a coarse mesh the results are less accurate since the solutions undershoots the AFC solutions at some Péclet values. For the peak problem on a coarse mesh the solutions have become further away from AFC at the straight region and also exceeds the analytical solutions compared to the asymmetric data results. On fine meshes the results are more accurate. For the boundary

layer problem, the results are also noticeably more accurate than the asymmetric data.

Comparing the results for symmetric data, the variable d_{ij} data is more accurate than the constant d_{ij} data for the boundary layer problem. Based on the peak problem results for high Péclet values on all meshes, variable d_{ij} data is also more noticeably accurate than the constant d_{ij} data, because the latter data set solutions consists of small oscillations and overshoots, even though, the former one exceeds the analytical solutions this a trade-off that has to be considered. Lastly, for most meshes and Péclet values the results for the variation of the boundary layer problem are most accurate with variable d_{ij} data. To conclude, the symmetric variable d_{ij} data is on all problems better than the asymmetric data on most meshes and Péclet values.

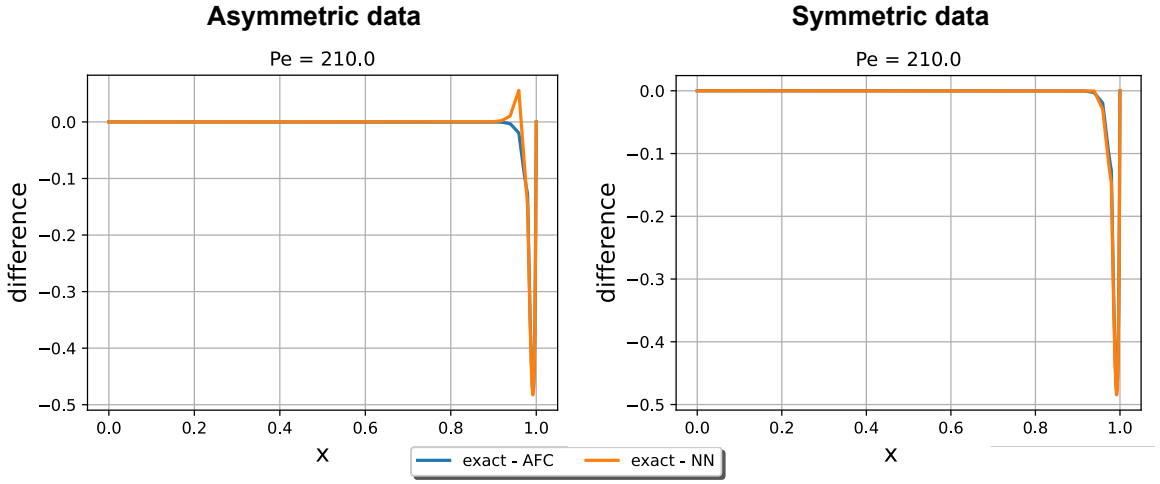


Figure 6.35: PINN differences plot for $n = 50$ for constant d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem for one of the Péclet values.

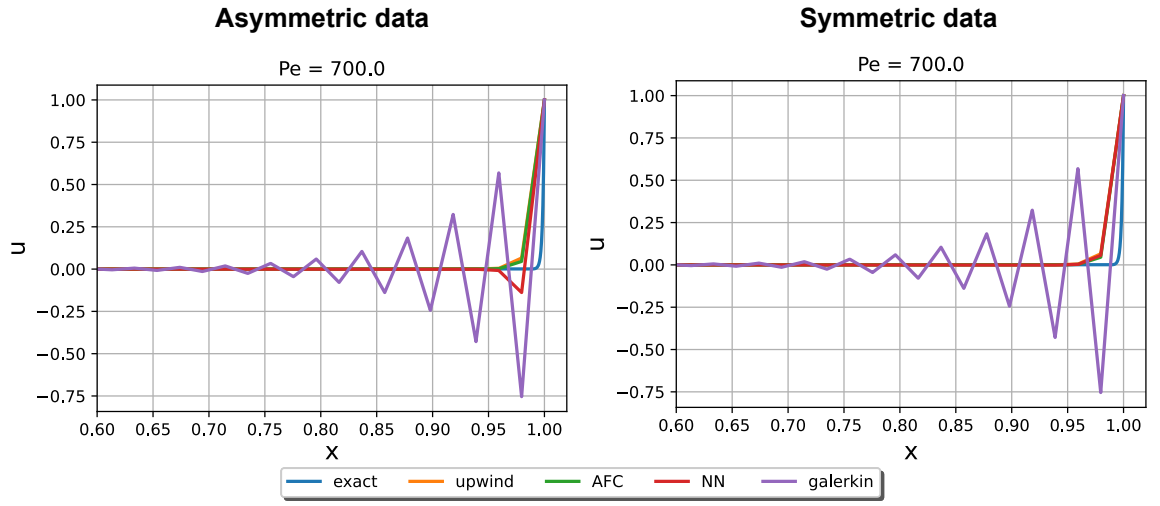


Figure 6.36: PINN solution profiles for $n = 50$ for variable d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem for one of the Péclet values.

6.5. Analysis

In this section, the results of the presented three neural network architectures, feedforward neural network (FFNN), splitted neural network (SpNN) and physics-informed neural network (PINN), are discussed and analysed based on the tested problems: boundary layer, peak and variation of the boundary layer problems. These neural networks were trained on two different data sets, namely constant and variable diffusion d_{ij} data, which were generated by the Galerkin and low-order upwind solutions of the boundary layer and peak problems. The types of data sets are called asymmetric data and a summary of the most accurate results of the asymmetric data is provided in Table 6.9. To train the FFNN and SpNN the mean-square error (MSE) is taken between the target α_{ij} from AFC and the predicted α_{ij} of the network. For the PINN the loss function is based on the AFC limiter.

NNs	± **BL problem**		± **PK problem***		± **varBL problem**	
	Data	Meshes (n) + Pe	Data	Meshes (n) + Pe	Data	Meshes (n) + Pe
FFNN	both data sets	on most meshes and Pe	constant d_{ij} data	on fine meshes $\forall Pe$	constant d_{ij} data	on most meshes and Pe (overall overshoot/limit↓)
			variable d_{ij} data	on most meshes and Pe	variable d_{ij} data	on fine meshes $\forall Pe \leq 100$
SpNN	both data sets	on most meshes and Pe^{**}	both data sets	on most meshes and Pe^{**}	constant d_{ij} data	on fine meshes $\forall Pe \leq 160$
					variable d_{ij} data	on most meshes and Pe (overall undershoot/limit↑)
PINN	constant d_{ij} data	on most meshes and Pe^{**}	constant d_{ij} data	on most meshes and Pe^{**}	both data sets	on fine meshes $\forall Pe \leq 100$
	variable d_{ij} data	on most meshes and Pe	variable d_{ij} data	on most meshes and Pe		

$* =$ outperforms AFC observably; $** =$ more accurate than FFNN on all meshes and $\forall Pe$ based on a coarse mesh ($n = 20$)

Table 6.9: Summary of the best obtained asymmetric data results for three neural networks (NNs): feedforward neural network (FFNN), splitted neural network (SpNN) and physics-informed neural network (PINN) trained on two fine meshed ($n = 50$) data sets: constant and variable diffusion d_{ij} data and tested on the trained boundary layer (BL), peak (PK) and untrained variation of boundary layer (varBL) problems on coarse ($n = 20$) and fine ($n = 50, 80$) meshes for different positive and negative ($\pm$) Péclet values (Pe).

Data analysis: constant d_{ij} data vs variable d_{ij} data

The variable diffusion data has more spread in the input-output mapping compared to the constant diffusion data, which allows the α_{ij} values to be converging in a larger search space corresponding to the flux values. Whereas, the constant diffusion data forms a more confined search space for the α_{ij} values. This mapping difference influences the ability of neural networks to mimic the behaviour of the AFC limiter. For example, Table 6.9 shows that FFNN is more accurate with constant diffusion data on most problems, but SpNN is more accurate with variable diffusion data, which even outperforms AFC for the trained problems. This might be because FFNN is able to learn the AFC limiter better with a simple mapping and SpNN with a more complex mapping, which represents a more general limiting behaviour. However, PINN achieves the most accurate results for the trained problems with the constant diffusion data, which shows that this architecture does not need a complex mapping to learn the AFC limiter.

Neural network architecture analysis

Due to the architectures of neural networks, they are able to interpret the constant and variable diffusion data sets differently and consequently learn for one data set better. Despite the fact that they all are feedforward neural networks, they have different architectures, activation functions, hyperparameters and loss functions.

The FFNN and PINN have relatively simple neural network architectures compared to the SpNN. However, the PINN consists of the AFC limiter equations in the loss function that enables it to have prior knowledge about the function to approximate, which increases the complexity of backpropagation and optimization of the network parameters based on the loss function. Consequently, the best results for the trained boundary layer and peak problems were obtained by the SpNN and PINN, which both outperform AFC on both problems, compared to FFNN. In short, these complex neural network architectures are most suitable to mimic the behaviour of the AFC limiter on these problems with a relatively small data set compared to other applications of neural networks in science and engineering.

Outperformance: AFC vs neural networks

To outperform AFC, the neural networks' solutions are better than the AFC solutions and therefore are closer to the analytical solution (and not exceeding it). From Table 6.9, it can be concluded that all neural network architectures are outperforming AFC for the peak problem on most meshes and Péclet values. This is caused by the fact that for the boundary layer problem less limiting is required compared to the peak problem, therefore, with sufficient limiting, but less limiting than AFC has implemented for this problem the neural networks become more accurate. This is indicative that the input and output data relationship plays a significant role. Furthermore, the advantage of training on two different problems enables the neural network to be a more general stabilization method and therefore it is recommended to train on different problems.

Generalizability analysis

From Table 6.9 it can be concluded that the most generalizable neural network architectures are FFNN and SpNN, because both are able to perform accurately on the untrained variation of boundary layer problem for most cases with one of the diffusion data sets. Moreover, the PINN fails to generalize well to untrained problems on most meshes and Péclet because it is more difficult to train due to the complicated loss function than the other two architectures. This training difficulty is due to the nature of including the fluxes in the loss function, therefore, there is an added complexity in the back-propagation and optimization. Other optimization algorithms might be used to make this more efficient in training and possibly improve the performance.

Improvements: symmetric data vs asymmetric data

To improve the asymmetric data results on the untrained variation of the boundary layer problem, all neural networks were trained for this problem and the variation of the peak problem, as described and shown in subsection 6.4.4. This created symmetric data for both the constant and variable diffusion data, which is a larger data set than the asymmetric data.

For FFNN more accurate results were achieved with symmetric data on all problems than with asymmetric data. This was acquired with variable diffusion data, which entails that this neural network architecture learns this complex mapping better with a larger data set. Similar results for the PINN were achieved, but not for the peak problem as the coarse mesh solutions exceed the analytical solutions, which was not the case for the asymmetric data. Furthermore, for SpNN the variable diffusion data obtained in general more accurate results than AFC and therefore outperforms AFC and the other neural networks in all cases. However, it is recommended that other untrained problems are tested with the symmetric data for all neural networks to test the performance, accuracy and generalizability. For example, by changing the boundary conditions of the problems or applying them to other partial differential equations.

Other possibilities to improve

Another possible way to improve the results in general for FFNN and SpNN could be by changing the loss function, for instance by taking the MSE of limited anti-diffusive fluxes as done in the previous approach. For PINN a more efficient algorithm to compute the loss function is possibly required to optimize the learning process.

7 Conclusion

In this thesis project, the one-dimension stationary convection-diffusion equation is solved with finite element methods and a stabilization method, the algebraic flux correction (AFC), is applied to limit the non-physical oscillations in the solution. With the knowledge of the concept of feedforward and physics-informed neural network and machine learning techniques, two alternative approaches were developed to mimic the AFC limiter behaviour: learning-based flux limiting and a learning-based surrogate model of AFC limiting.

In the learning-based flux limiting approach, the extra computations performed by the defect corrections scheme in AFC to compute the AFC limiter in each iteration are circumvented by the neural network. For this approach, four feedforward neural network models have been proposed:

- solution model
- solution + fluxes model
- gradient model
- gradient + fluxes model

where the name of the models represents the type of input data is used. Each model has been trained on two different data sets, namely single and multi-mesh data. These data sets are generated by the low-order upwind solutions of the boundary layer problem and also trained on three different loss functions, namely the alpha loss function, the limited anti-diffusive fluxes loss function and the limited anti-diffusive correction loss function. All the models have been tested on the trained boundary layer problem and untrained peak problem. From the results, it follows that the gradient-based models are most suitable and generalizable compared to the solution-based models to mimic the AFC limiter, because they are less influenced by the accuracy of the numerical method. Moreover, the gradient input values provide more insight into the AFC limiting behaviour, because the relationship or mapping between the input and output data represents the underlying nonlinear behaviour that the neural network has to approximate. Thus the mapping plays a significant role in training the neural network.

Furthermore, it has been observed that the type of loss functions has a large impact on the performance of the neural network because it follows that for AFC the minimization of the limited anti-diffusive flux is more important than the minimization of the alpha values. Consequently, the limited anti-diffusive flux loss function gives more accurate results for all models than other loss functions. Moreover, the data based on fine meshes is recommended as more accurate results were achieved due to the numerical accuracy, so multi-mesh data is recommended. Lastly, outperforming AFC is possible on the untrained problem making them generalizable.

For the learning-based surrogate model of AFC limiting approach, the neural network replaces the AFC limiter in the defect correction scheme and solves the limited anti-diffusive flux correction in each iteration. For this approach, three neural network architectures:

- feedforward neural network (FFNN)
- splitted neural network (SpNN)
- physics-informed neural network (PINN)

were developed and applied. These neural networks were trained on diffusion data, which were generated by the Galerkin and low-order upwind solutions of the boundary layer and peak problems. These types of data sets are called asymmetric data. Moreover, these neural networks were tested on three problems: boundary layer, peak and a variation of the boundary layer problems.

It can be concluded, that complex neural network architecture, such as SpNN and PINN, are most suitable to mimic the AFC limiter and for this type of application. Both models perform very good on trained problems with constant or variable diffusion data and even outperform AFC. However, PINN is less generalizable than SpNN due to it achieving less accurate results for the untrained problem. FFNN is more accurate with constant diffusion data in general for all cases.

The results were improved for all neural networks by training them on symmetric data, which was created by the variation of the boundary layer and the peak problems. All neural networks were more

accurate with the variable diffusion data, which entails that this complex limiting mapping is learned better with a larger data set. However, it is recommended to test the symmetric data model for other untrained problems to be able to determine the generalizability capacity. Finally, training on two different problems enables to be a more general stabilization method and therefore it is recommended to train on different problems.

It can be concluded that promising results are produced with regards to replacing the AFC limiter with machine learning and some suitable neural network architectures and data sets are achieved to do this. The generalizability of the neural network approaches has been tested on untrained problems whereby it performed good and in some cases it outperformed AFC. However, within the time frame of the project, the generalizability to higher dimensions and/or complex geometries have not been tested and therefore are not known. Thus, this is a good opportunity to do more research on this type of application in the future.

References

[1] R. Babuska and J. Kober. *Lecture slides of knowledge-based control systems(SC42050)*. 2021. URL: Tu%20Delft.

[2] J.A. Cottrell, T.J.R. Hughes, and Y. Basilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, Ltd., 2009.

[3] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* Volume 9 of JMLR (2012).

[4] S.K. Godunov. "Finite difference method for numerical computation of discontinous solutions of the equations of fluid dynamics". In: *Mat. Sbornik* 47:271–306 (1959).

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] A. Jaeschke. "Isogeometric Analysis for Compressible Flows with Application in Turbomachinery". Delft University of Technology, 2015.

[7] A. Jaeschke and M. Möller. "High-Order Isogeometric Methods for Compressible Flows. I. Scalar Conservations Laws". In: *van Brummelen H., Corsini A., Perotto S., Rozza G. (eds) Numerical Methods for Flows. Lecture Notes in Computational Science and Engineering.* Vol. 132 (2020). DOI: https://doi.org/10.1007/978-3-030-30705-9_3.

[8] J. van Kan, A. Segel, and F. Vermolen. *Numerical Methods in Scientific Computing*. 2nd ed. Delft Academic Press/ VSSD, 2014.

[9] D. Kuzmin. "Algebraic flux correction for finite element discretizations of coupled systems". In: *Computational Methods for Coupled Problems in Science and Engineering* 2 (2007), pp. 653–656.

[10] D. Kuzmin, R. Löhner, and S. Turek. *Flux-Corrected Transport. Principles, Algorithms, and Applications*. 2nd ed. City, State or Country ....: Springer, 2012.

[11] D. Kuzmin and M. Möller. *Algebraic Flux Correction I. Scalar Conservation Laws. In: Kuzmin D., Löhner R., Turek S. (eds) Flux-Corrected Transport. Scientific Computation*. 1st ed. Berlin, Heidelberg: Springer, 2005.

[12] D. Kuzmin and M. Möller. *Algebraic Flux Correction II. Compressible Euler equations. In: Kuzmin D., Löhner R., Turek S. (eds) Flux-Corrected Transport. Scientific Computation*. 1st ed. Berlin, Heidelberg: Springer, 2005.

[13] D. Kuzmin and S. Turek. "Flux correction tools for finite elements". In: *Journal of computational physics* 175, 525-558 (2002).

[14] D. Kuzmin and S. Turek. "High-resolution FEM-TVD schemes based on a fully multidimensional flux limiter". In: *Journal of Computational Physics* (2004).

[15] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[16] The Ray Team. *Key Concepts of Ray Tune*. 2022. URL: https://docs.ray.io/en/latest/tune/key-concepts.html (visited on 12/20/2021).

[17] C. Vuik et al. *Numerical methods for ordinary differential equations*. 2nd ed. ISBN:97890-6562-3737. Delft: Delft Academic Press/VSSD, 2016.

A Learning-based flux limiting

A.1. Solution model approach

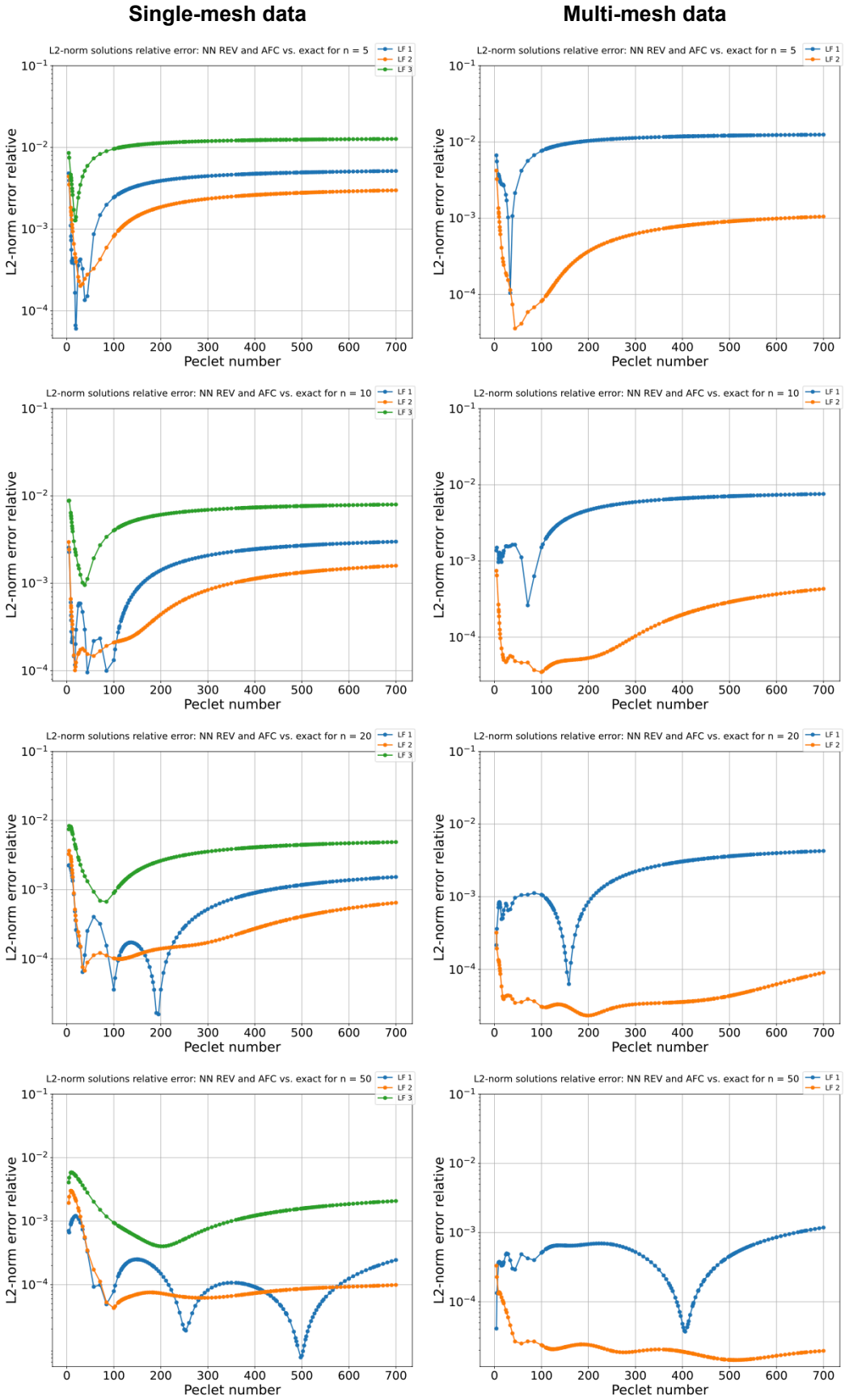


Figure A.1: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

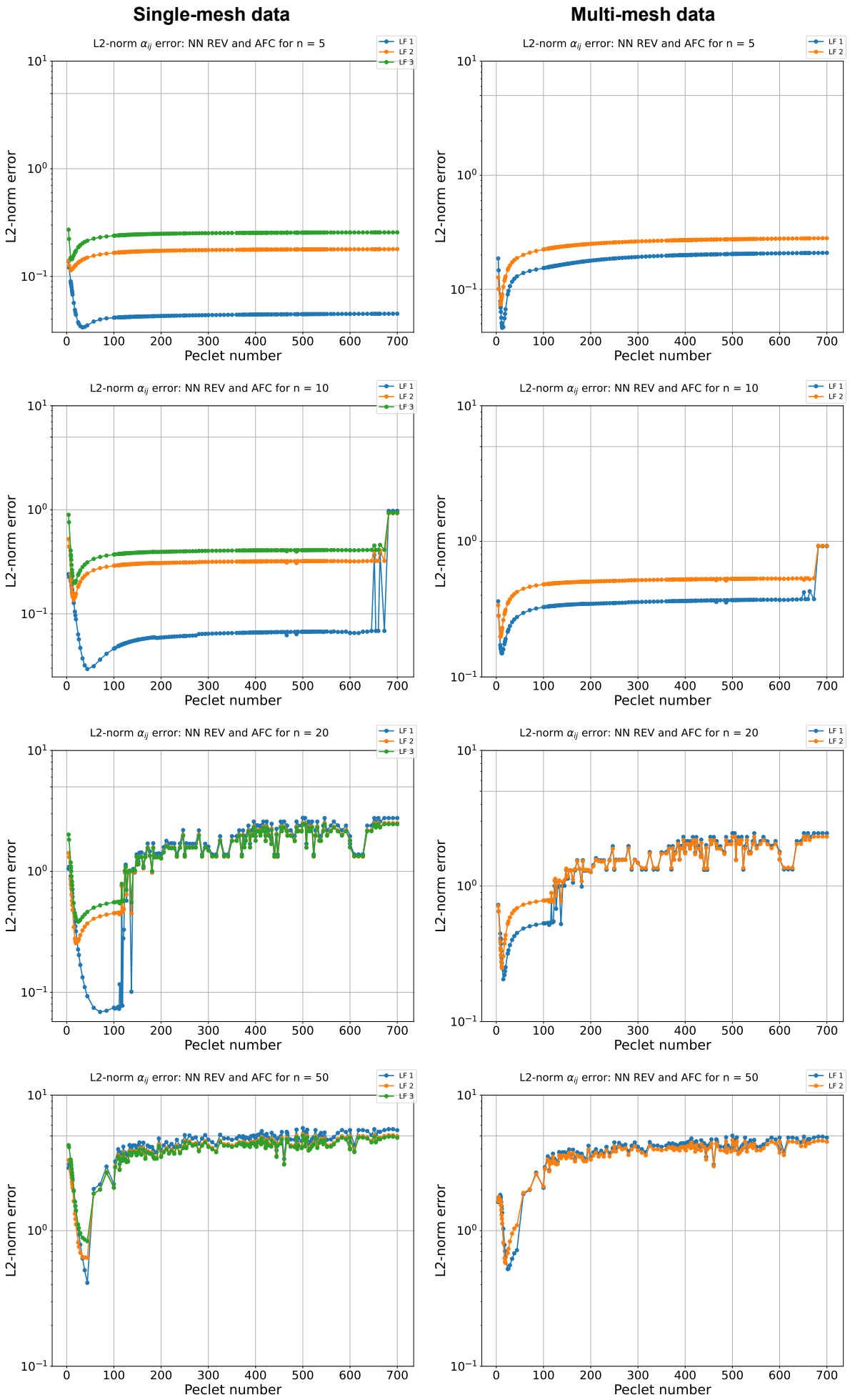


Figure A.2: The L^2-norm error of the **(trained) boundary layer problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

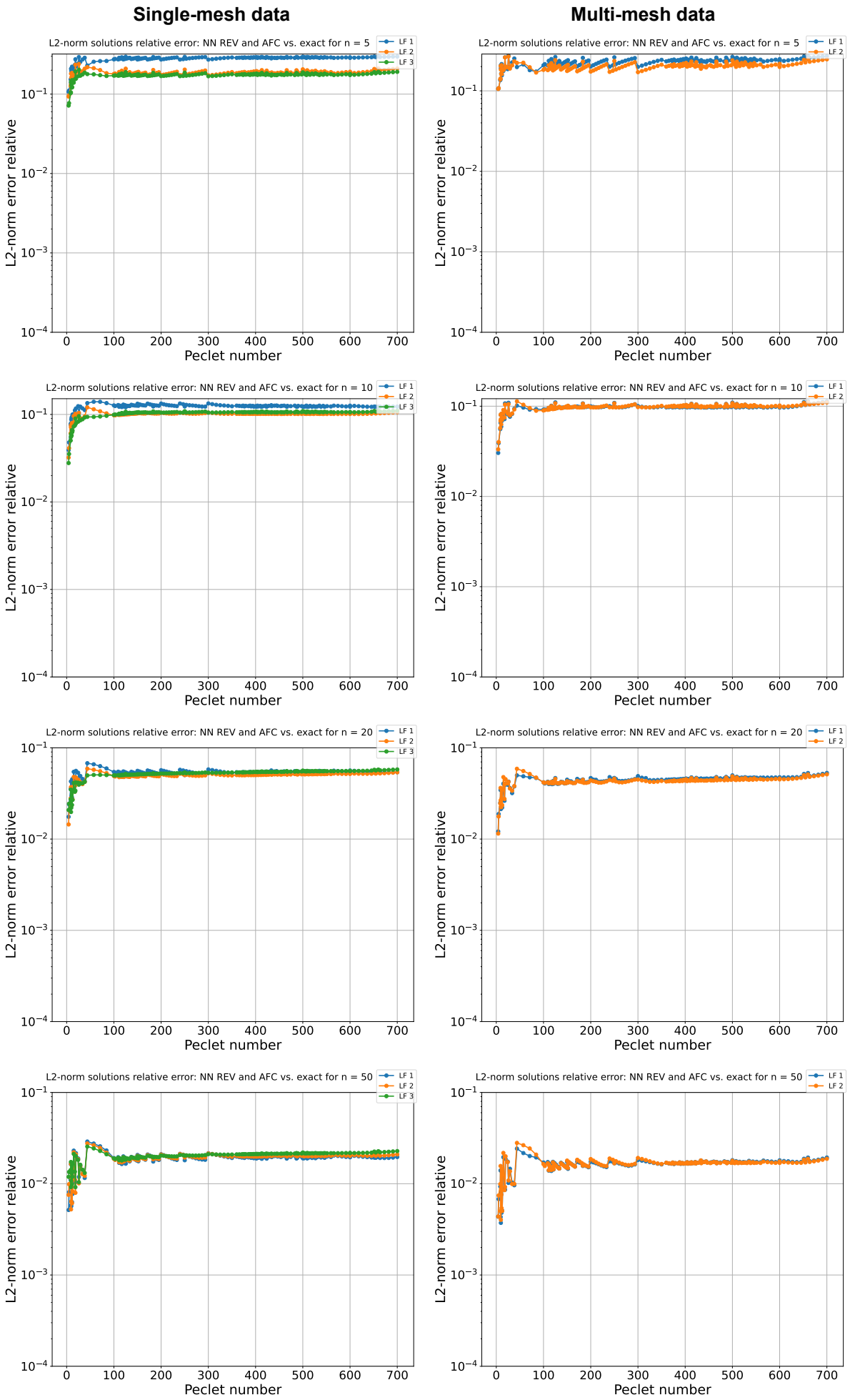


Figure A.3: The L^2-norm relative error of the **(untrained) peak problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

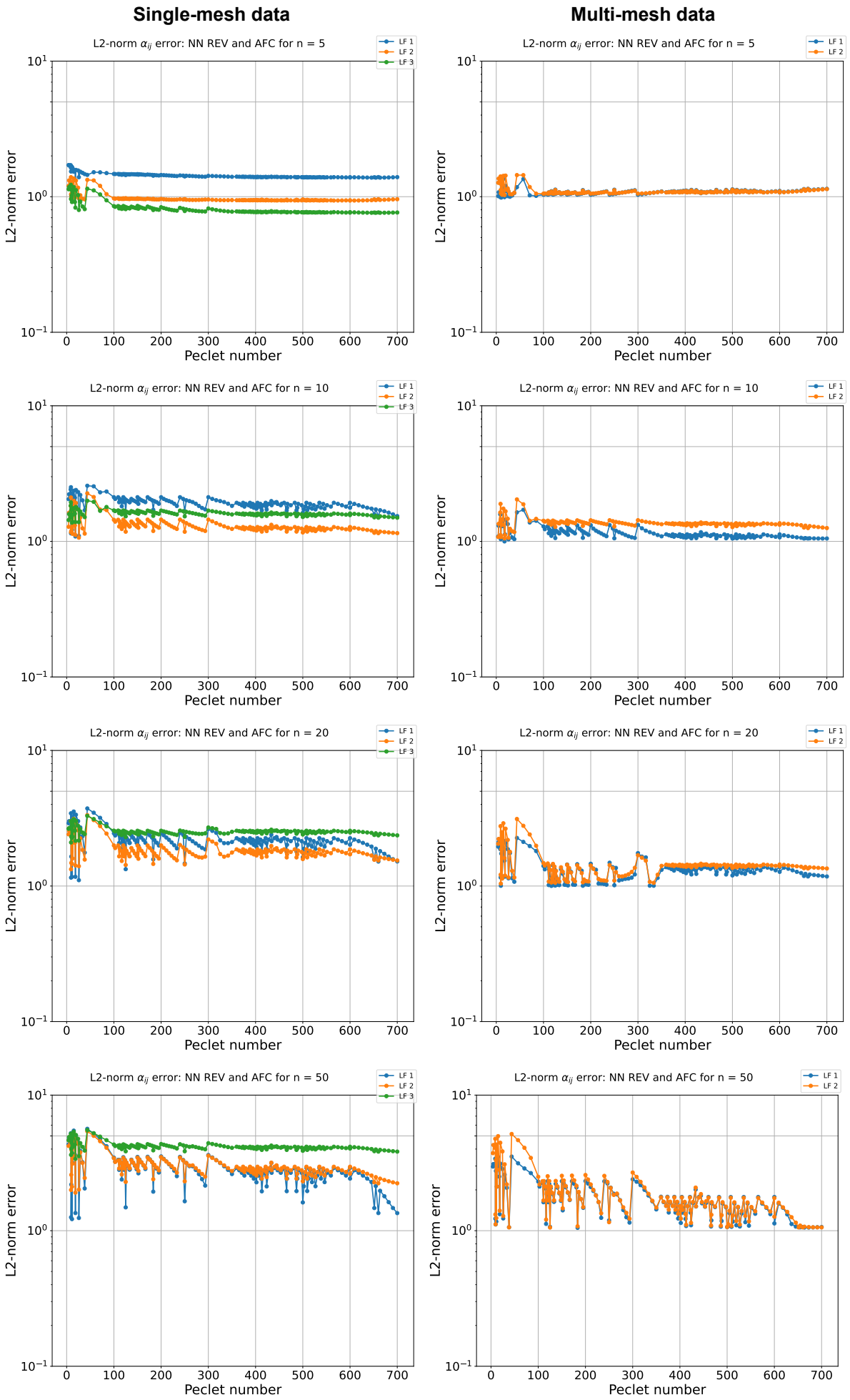


Figure A.4: The L^2-norm error of the **(untrained) peak problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

A.2. Solution + fluxes model approach

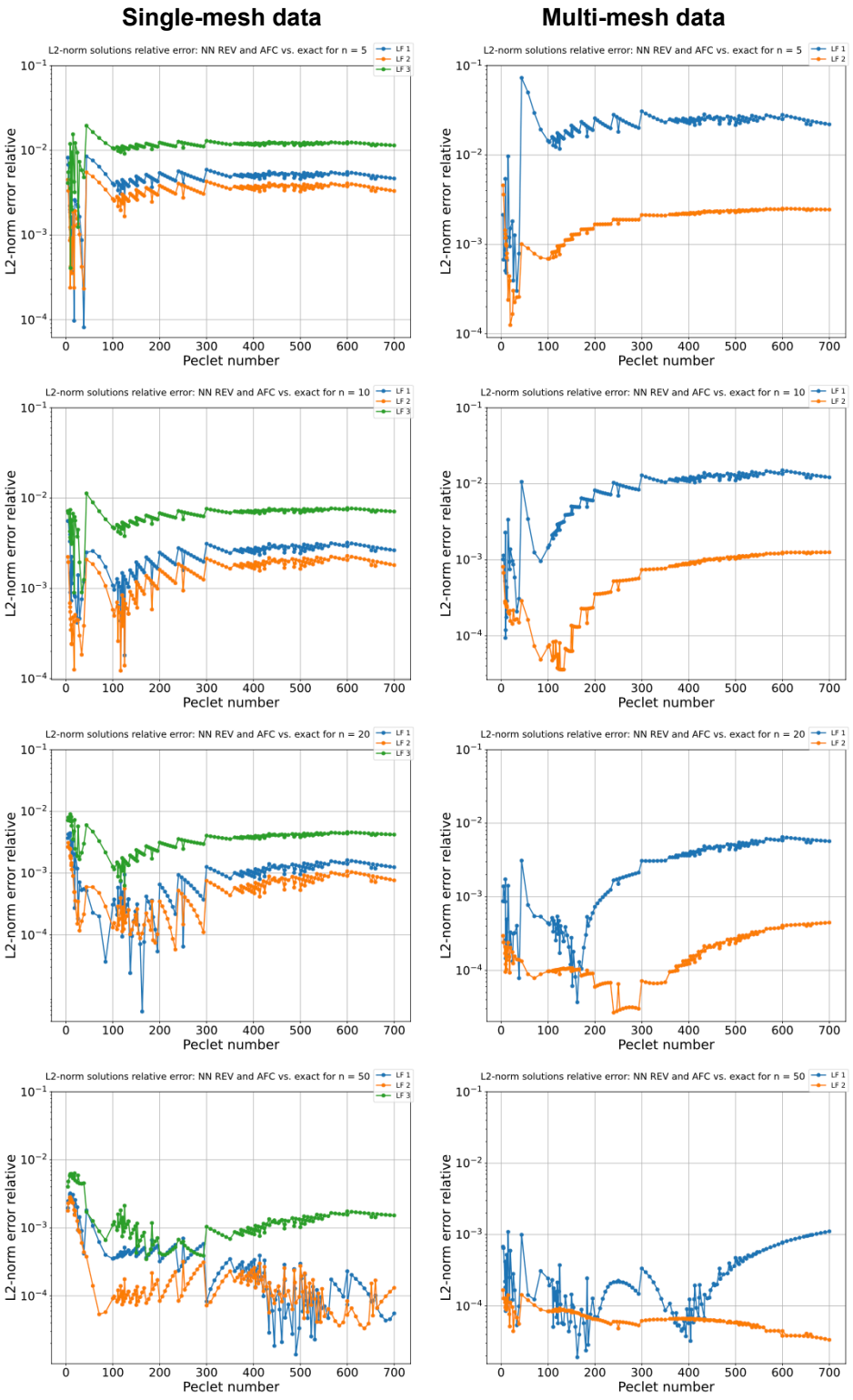


Figure A.5: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

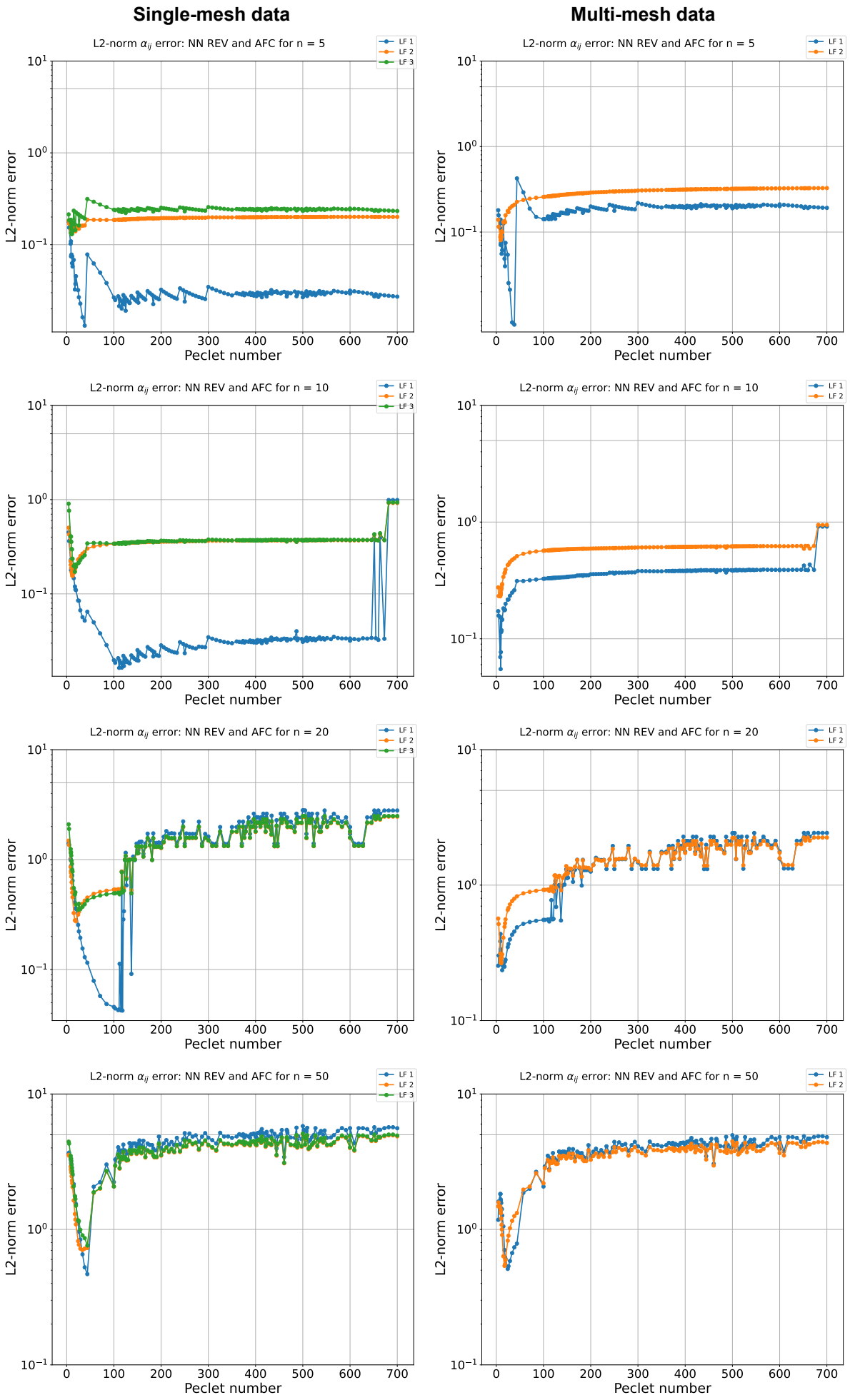


Figure A.6: The L^2-norm error of the **(trained) boundary layer problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

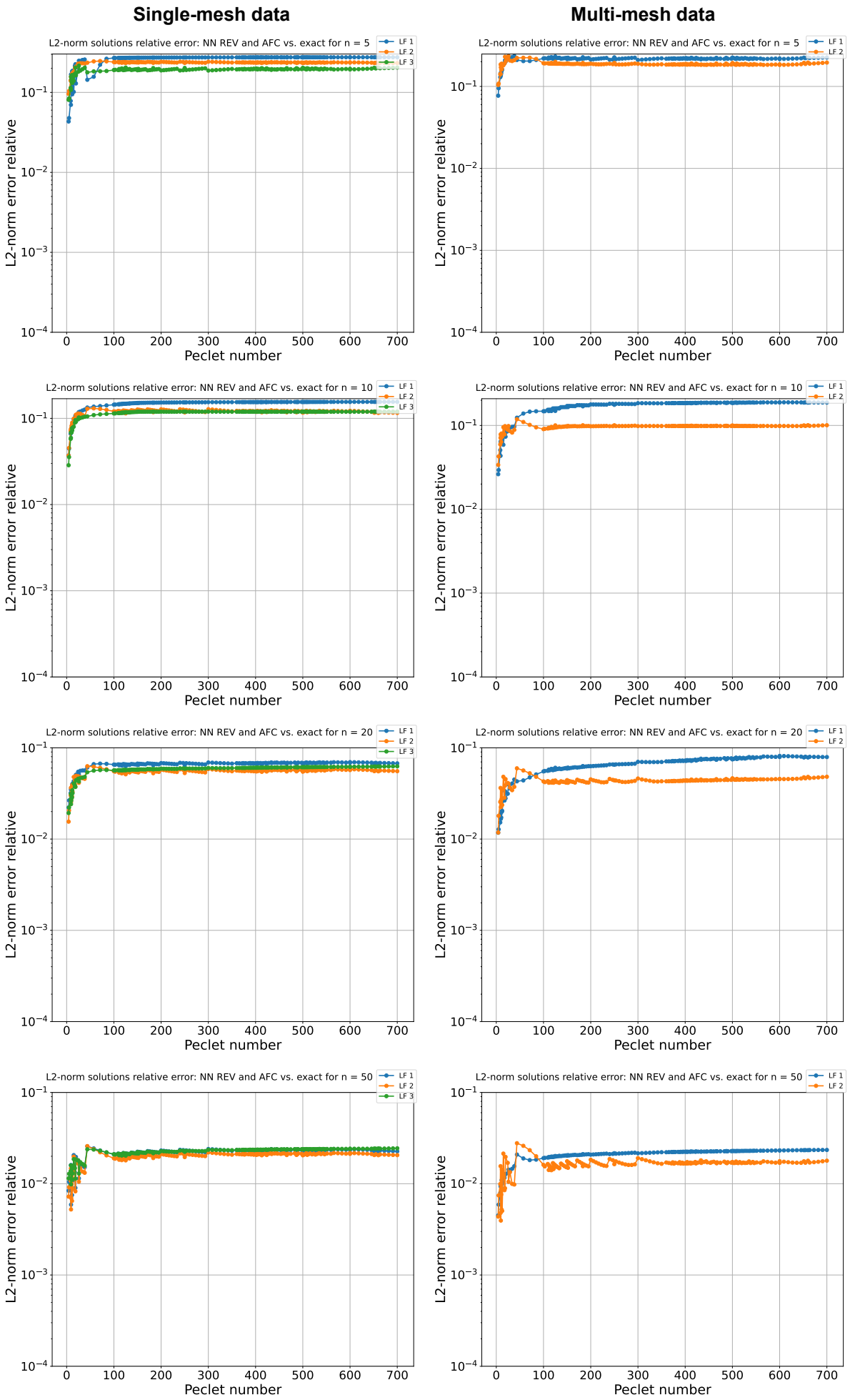


Figure A.7: The L^2-norm relative error of the **(untrained) peak problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

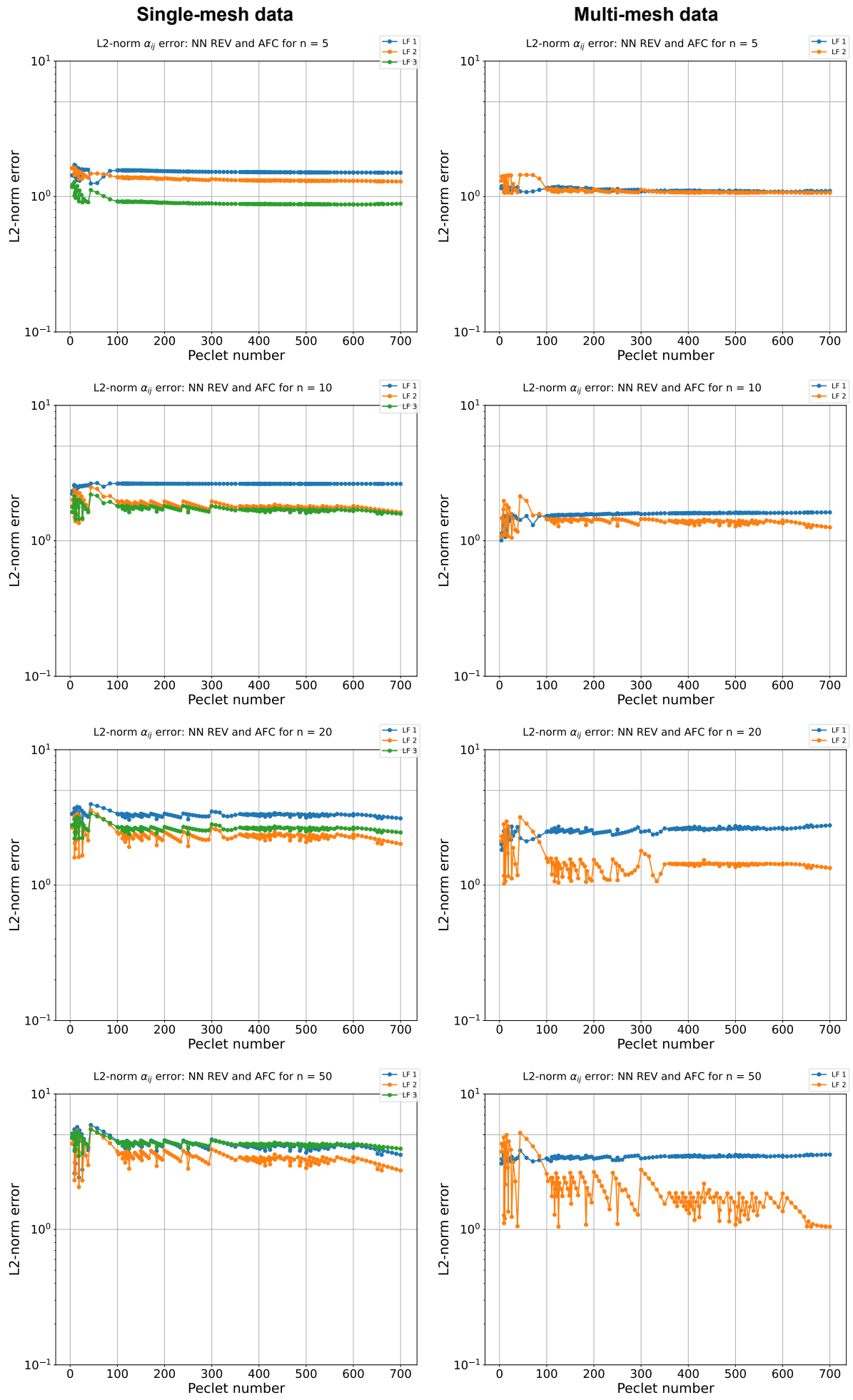


Figure A.8: The L^2-norm error of the **(untrained) peak problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the solution + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

A.3. Gradient model approach

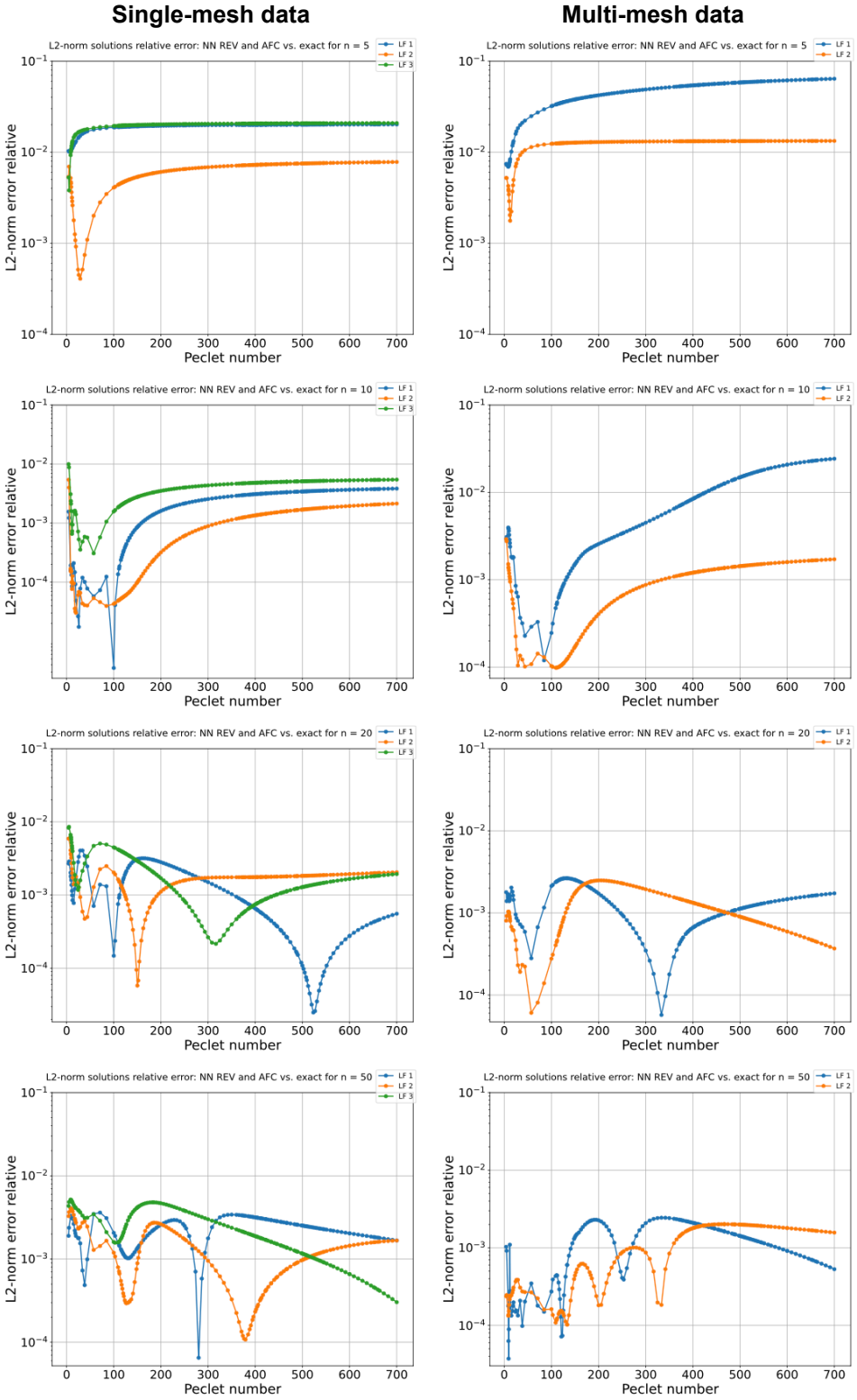


Figure A.9: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

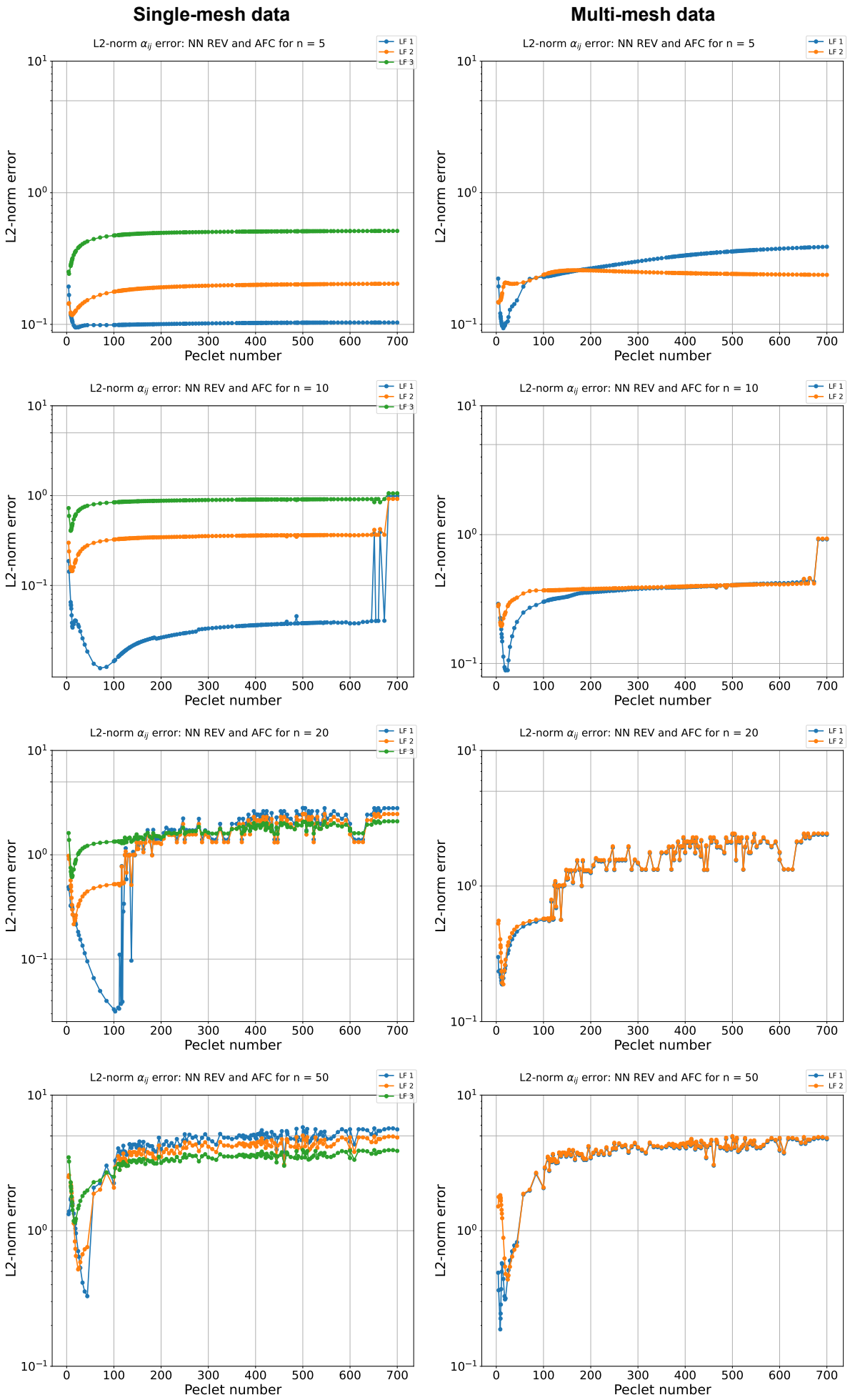


Figure A.10: The L^2-norm error of the **(trained) boundary layer problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

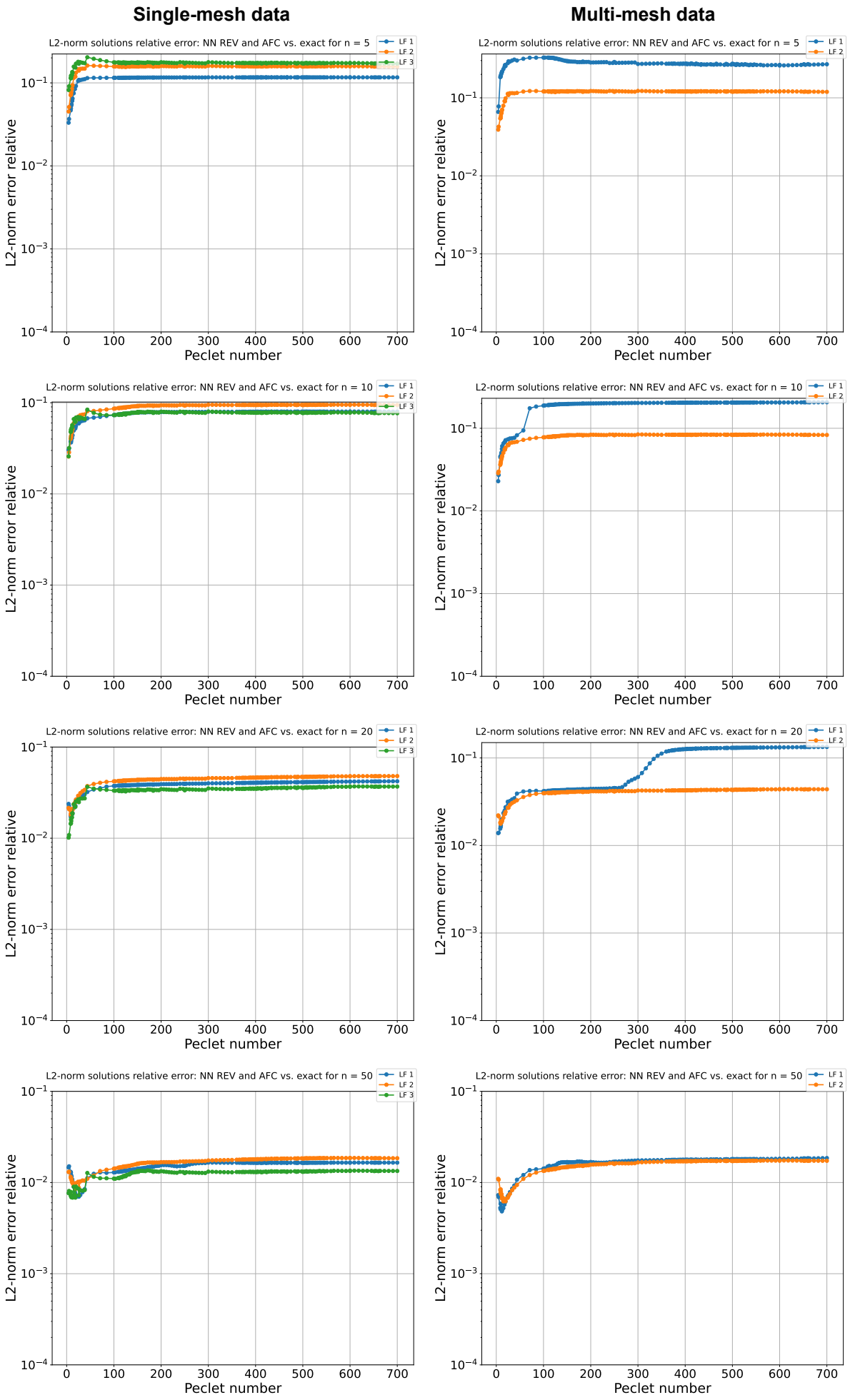


Figure A.11: The L^2-norm relative error of the **(untrained) peak problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

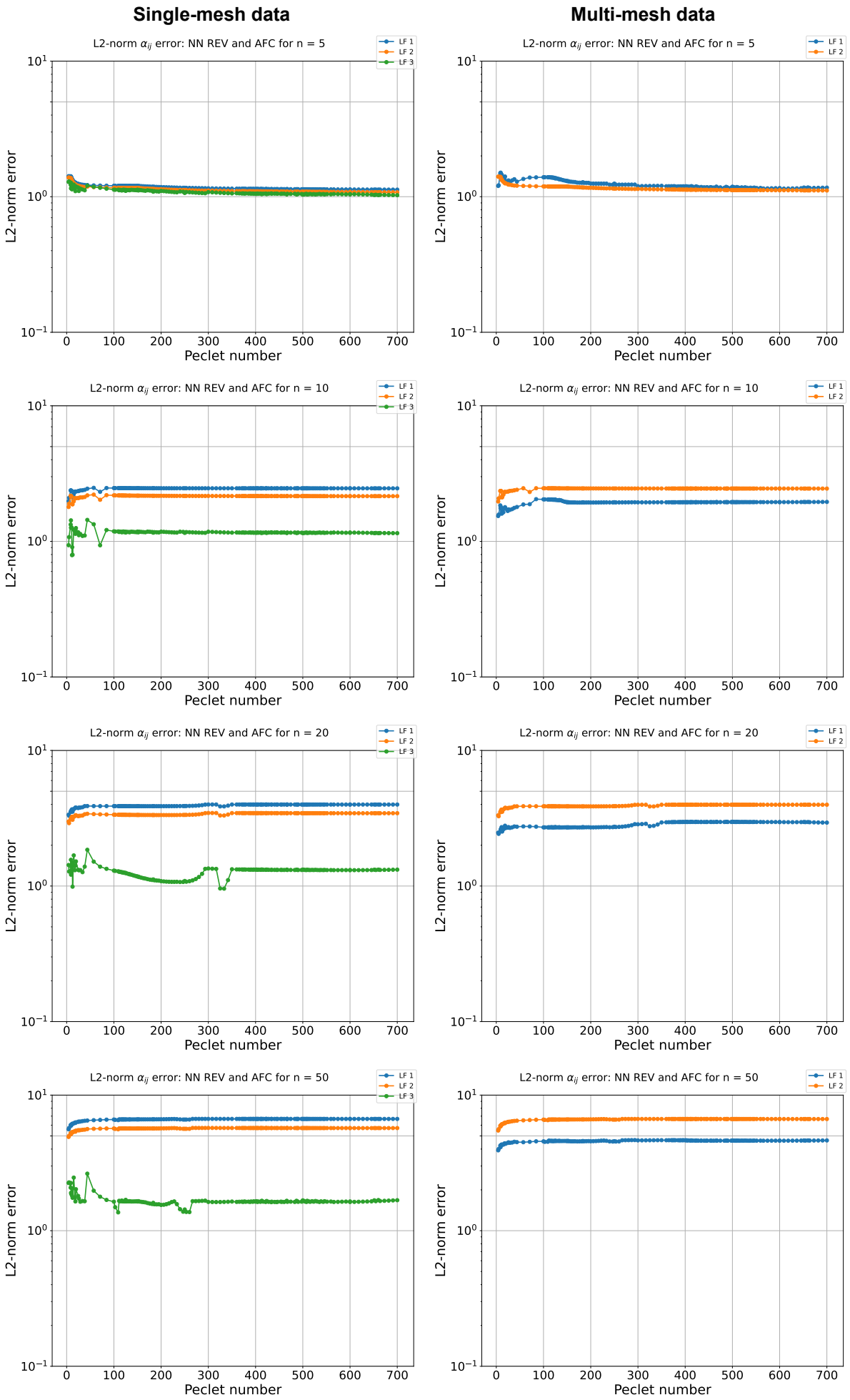


Figure A.12: The L^2-norm error of the **(untrained) peak problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

A.4. Gradient + fluxes approach

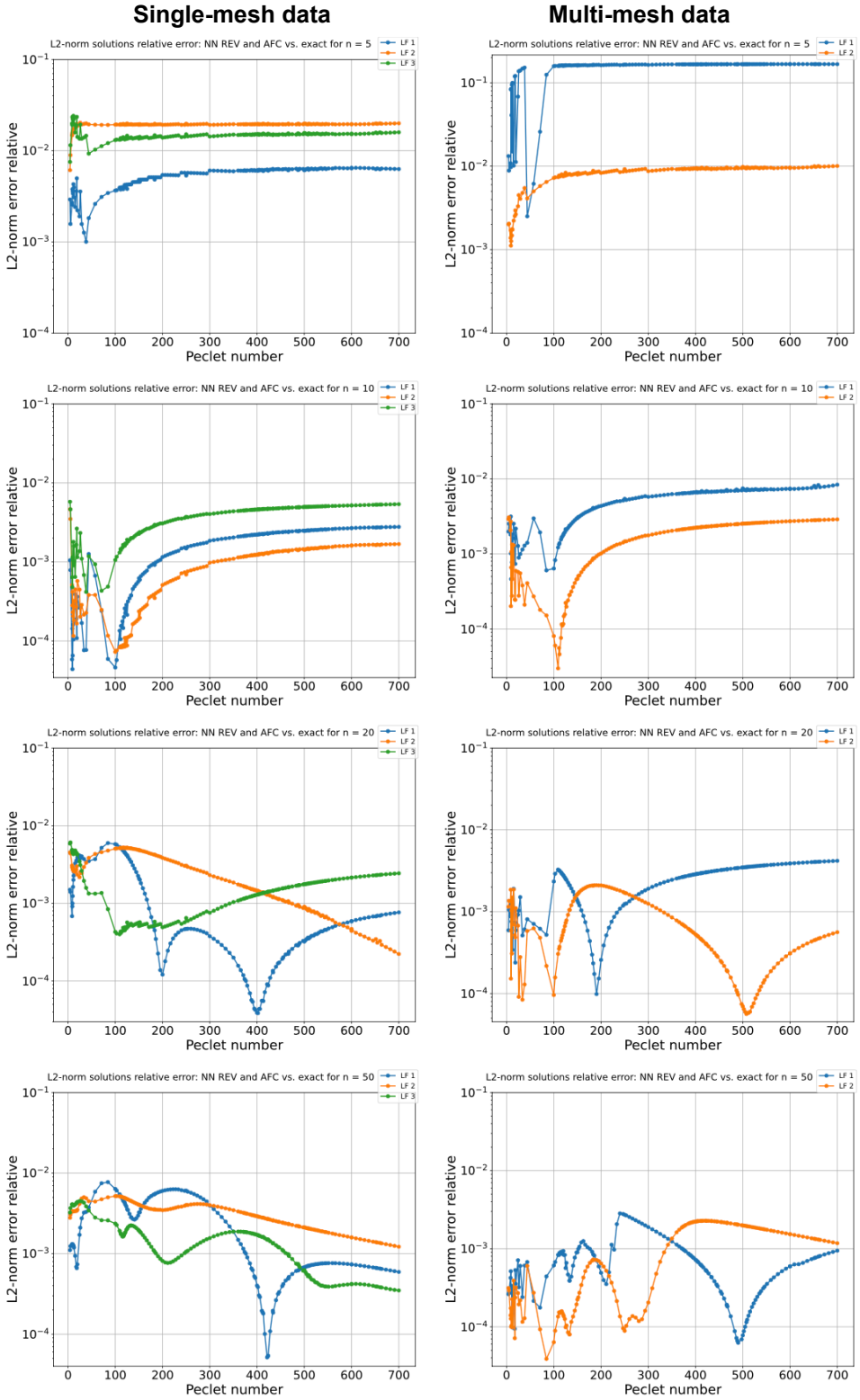


Figure A.13: The L^2-norm relative error of the **(trained) boundary layer problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

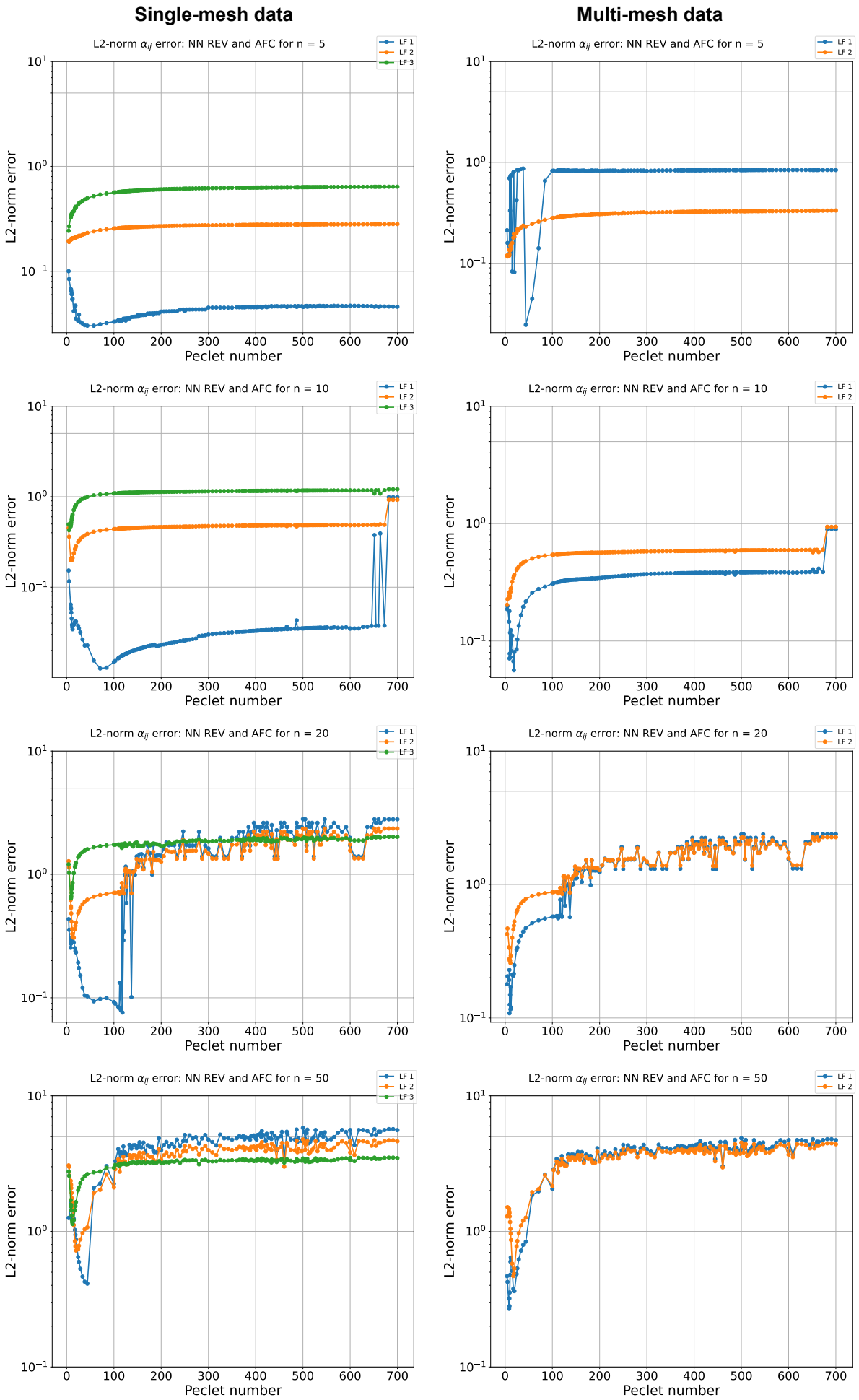


Figure A.14: The L^2-norm error of the **(trained) boundary layer problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

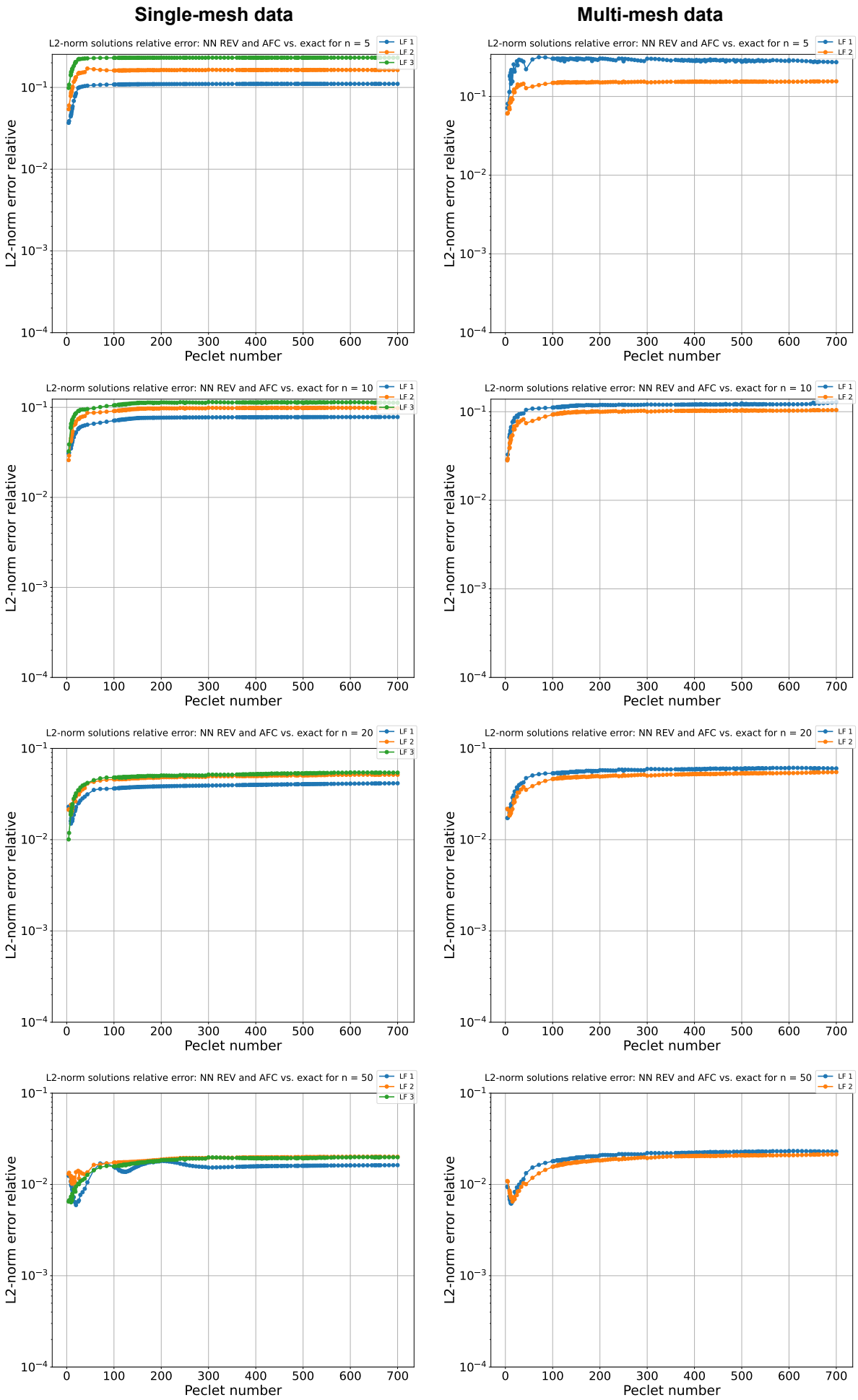


Figure A.15: The L^2-norm relative error of the **(untrained) peak problem** solutions for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

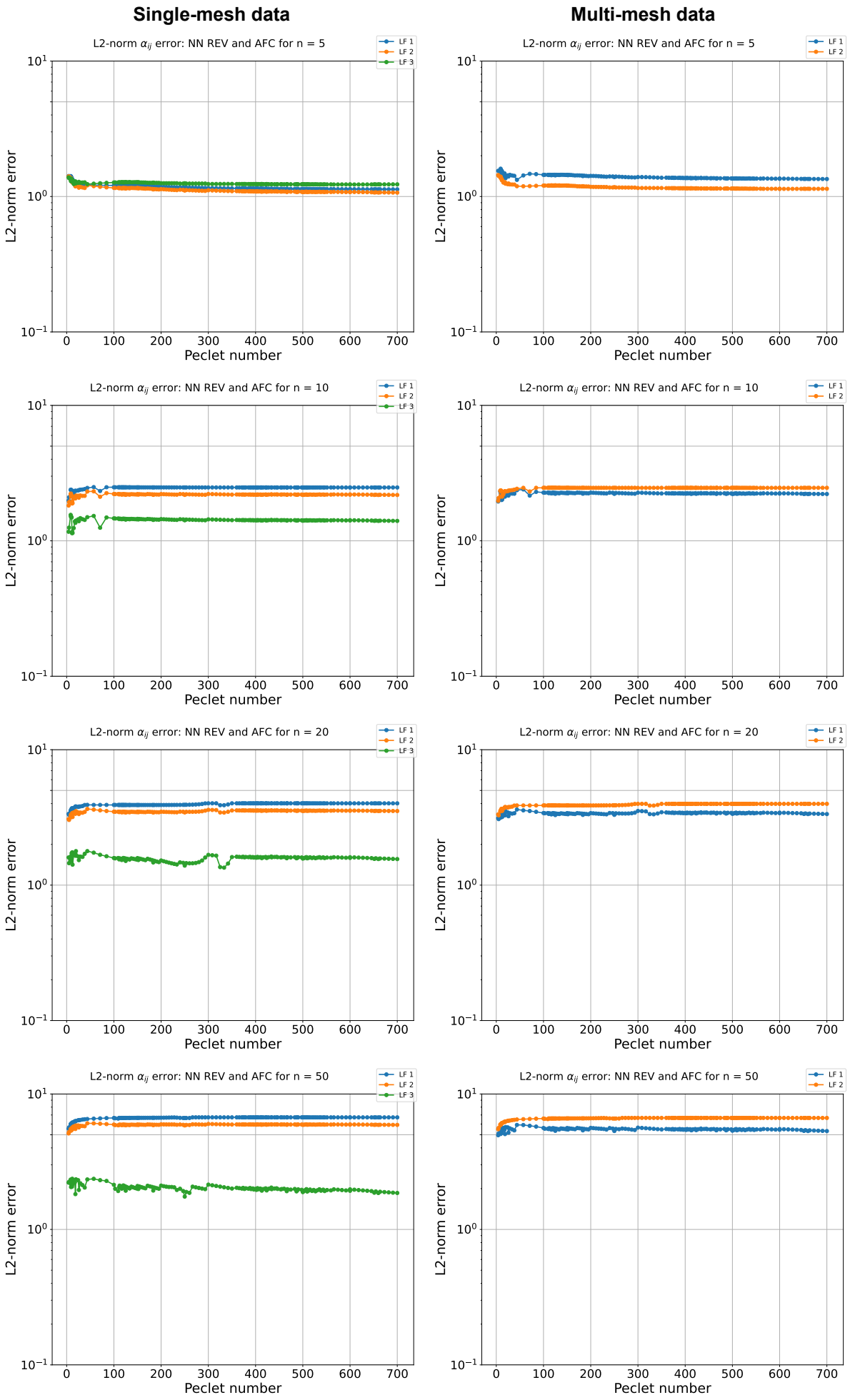


Figure A.16: The L^2-norm error of the **(untrained) peak problem** α_{ij} values for different meshes ($n = 5, 10, 20, 50$, from top to bottom) for the gradient + fluxes model approach trained on single-mesh data (left) and multi-mesh data (right) and with loss functions (LFs): alpha LF1 (blue), limited anti-diffusive fluxes LF2 (orange) and limited anti-diffusive correction LF3 (green).

B Learning-based surrogate model of AFC limiting

B.1. Asymmetric data: other results

B.1.1. Feedforward neural network

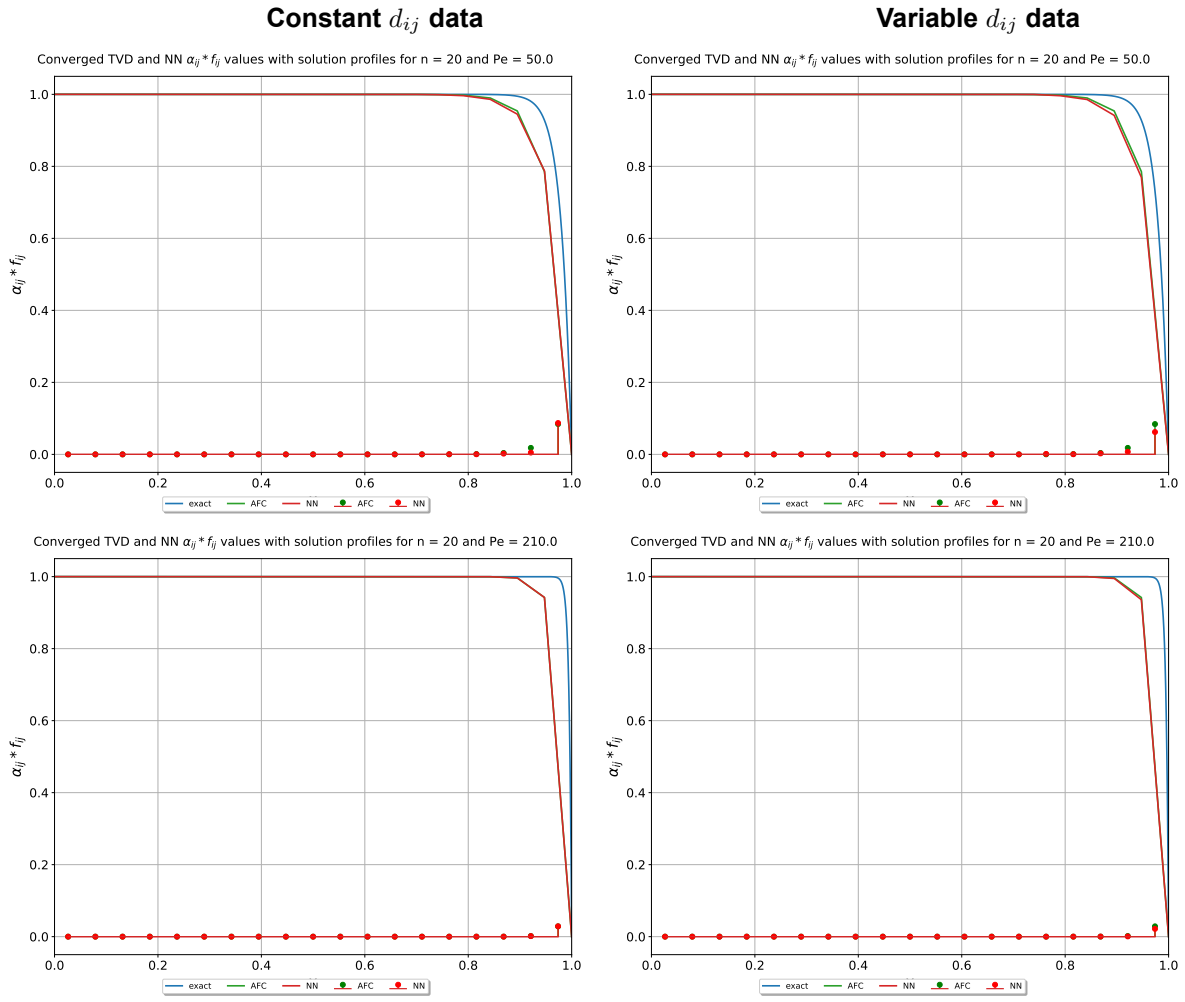


Figure B.1: FFNN limited anti-diffusive flux results with the NN and AFC solution profiles against the exact solution for constant (left) and variable (right) d_{ij} data for $n = 20$ and $Pe = 50, 210$ for the boundary layer problem.

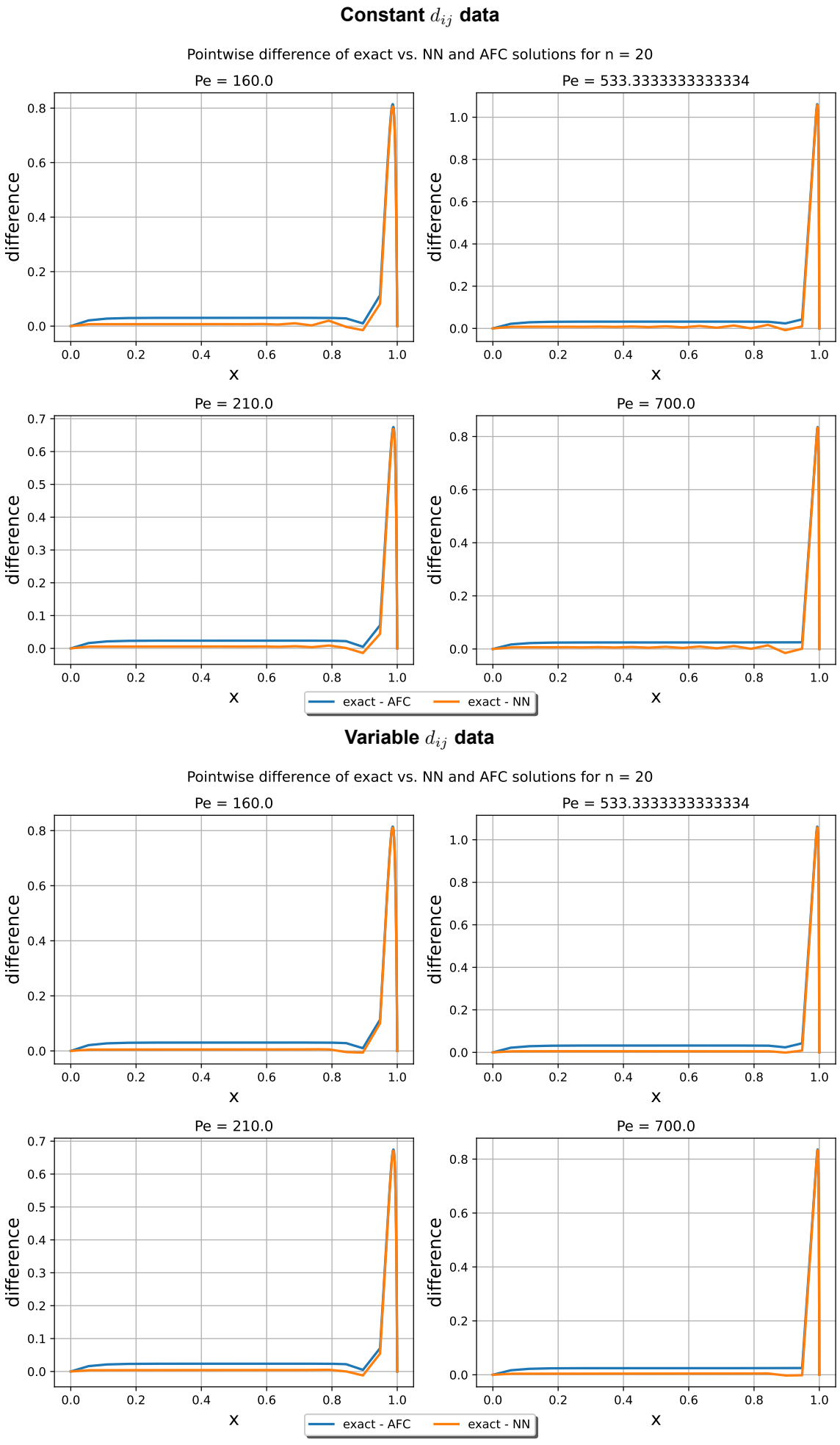


Figure B.2: FNN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (upper) and variable diffusion data (lower).

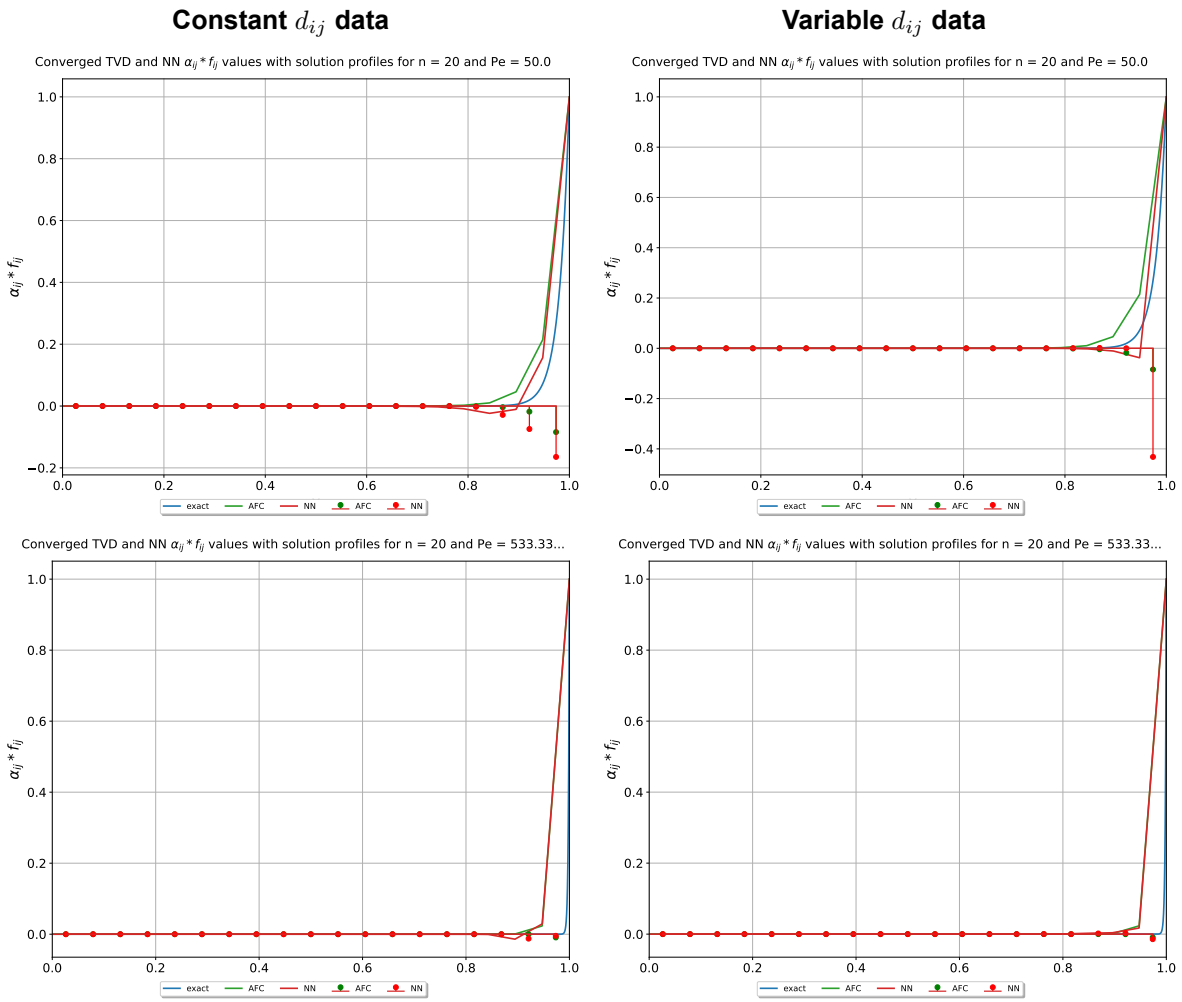


Figure B.3: FFNN limited anti-diffusive flux results with the NN and AFC solution profiles against the exact solution for constant (left) and variable (right) d_{ij} data for $n = 20$ and $Pe = 50, 533.33$ for the untrained problem.

B.1.2. Splitted neural network

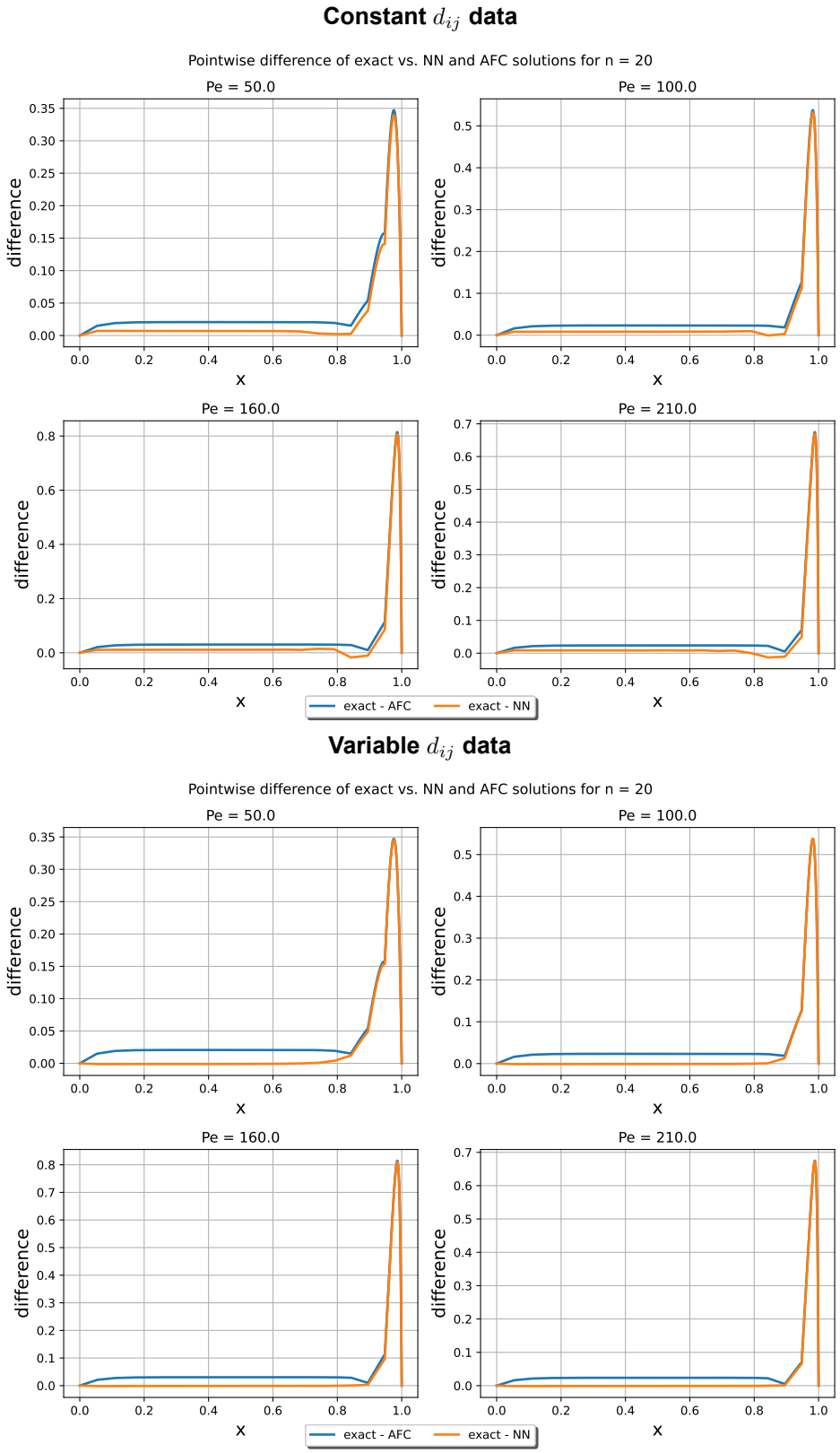
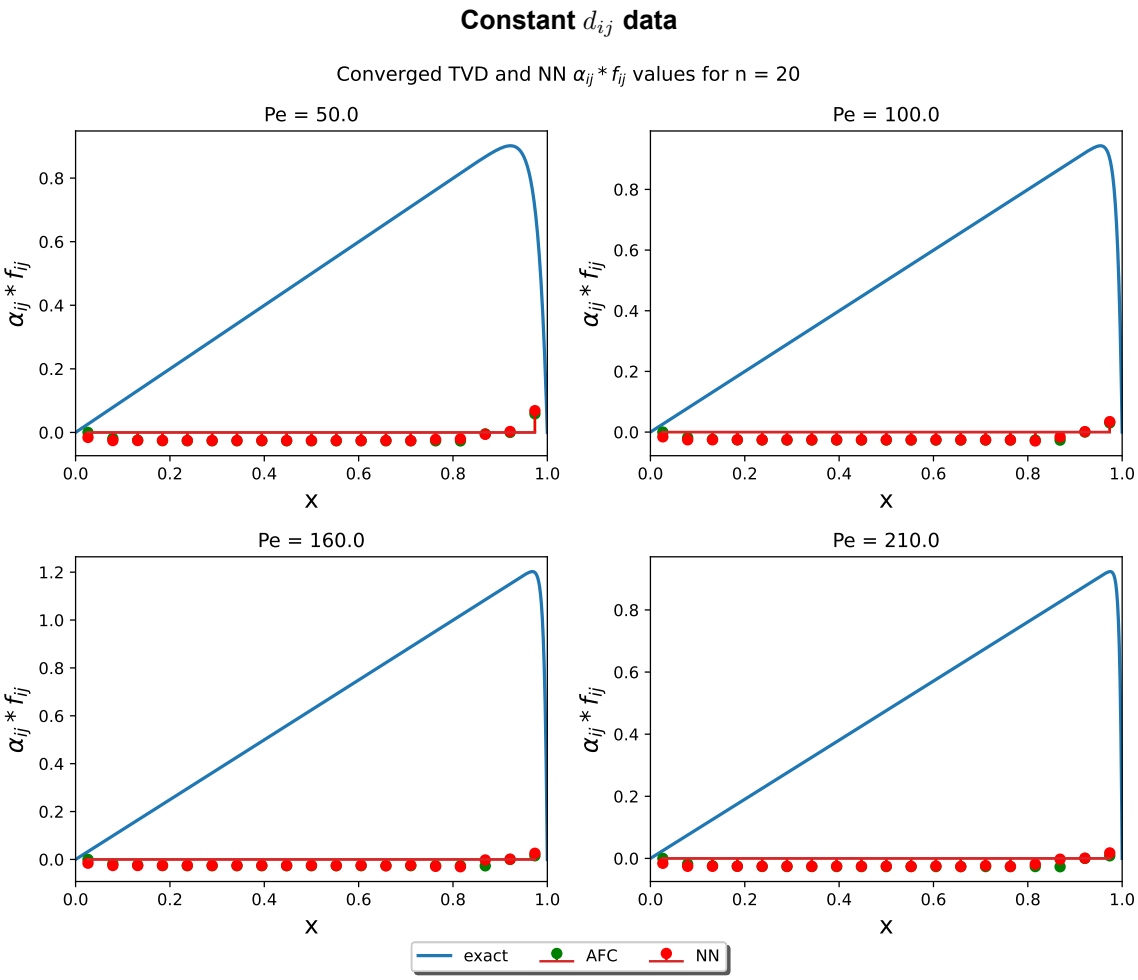


Figure B.4: SpNN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (upper) and variable diffusion data (lower).



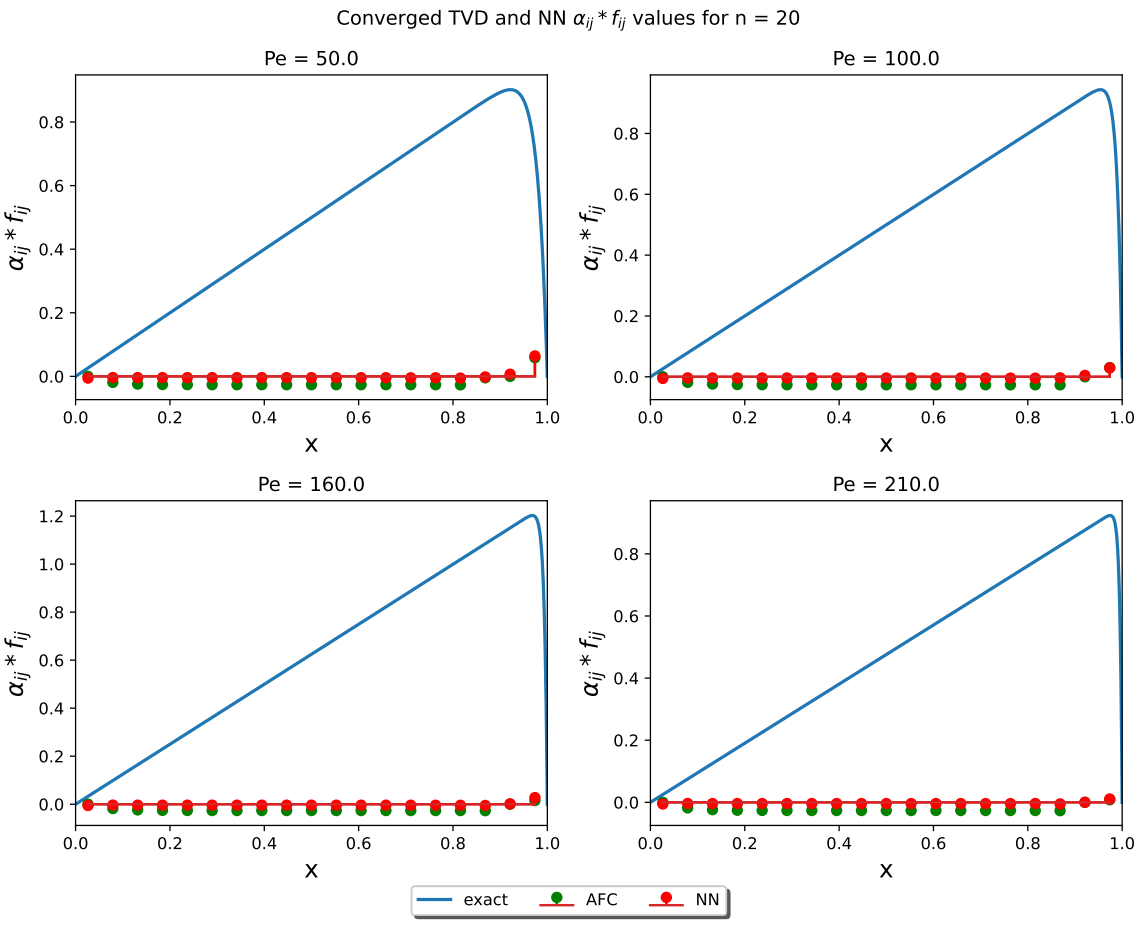


Figure B.5: SpNN limited anti-diffusive flux results with the exact solution for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (upper) and variable diffusion data (lower).

B.1.3. Physics-informed neural network

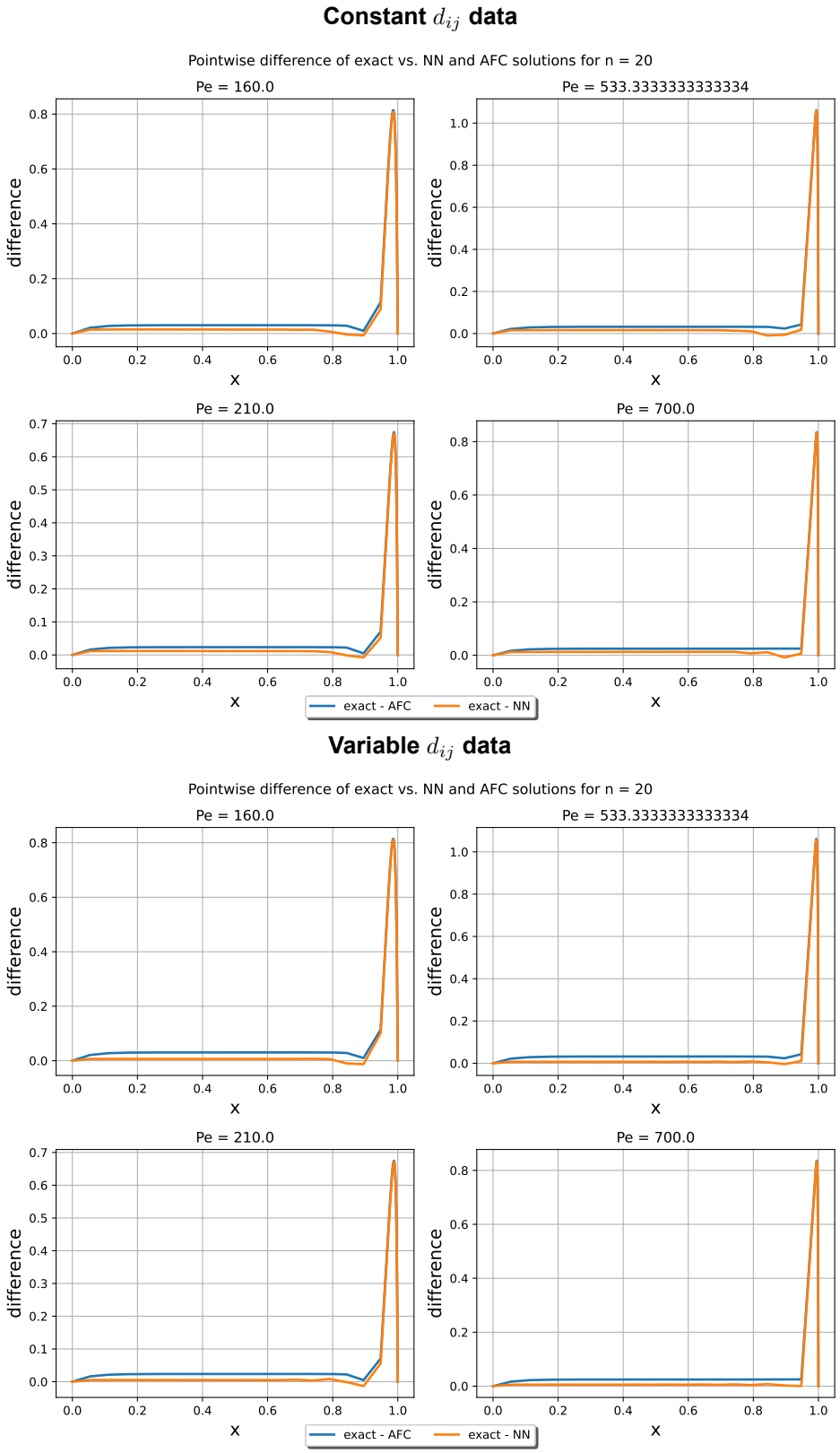


Figure B.6: PINN pointwise differences plots for the peak problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (upper) and variable diffusion data (lower).

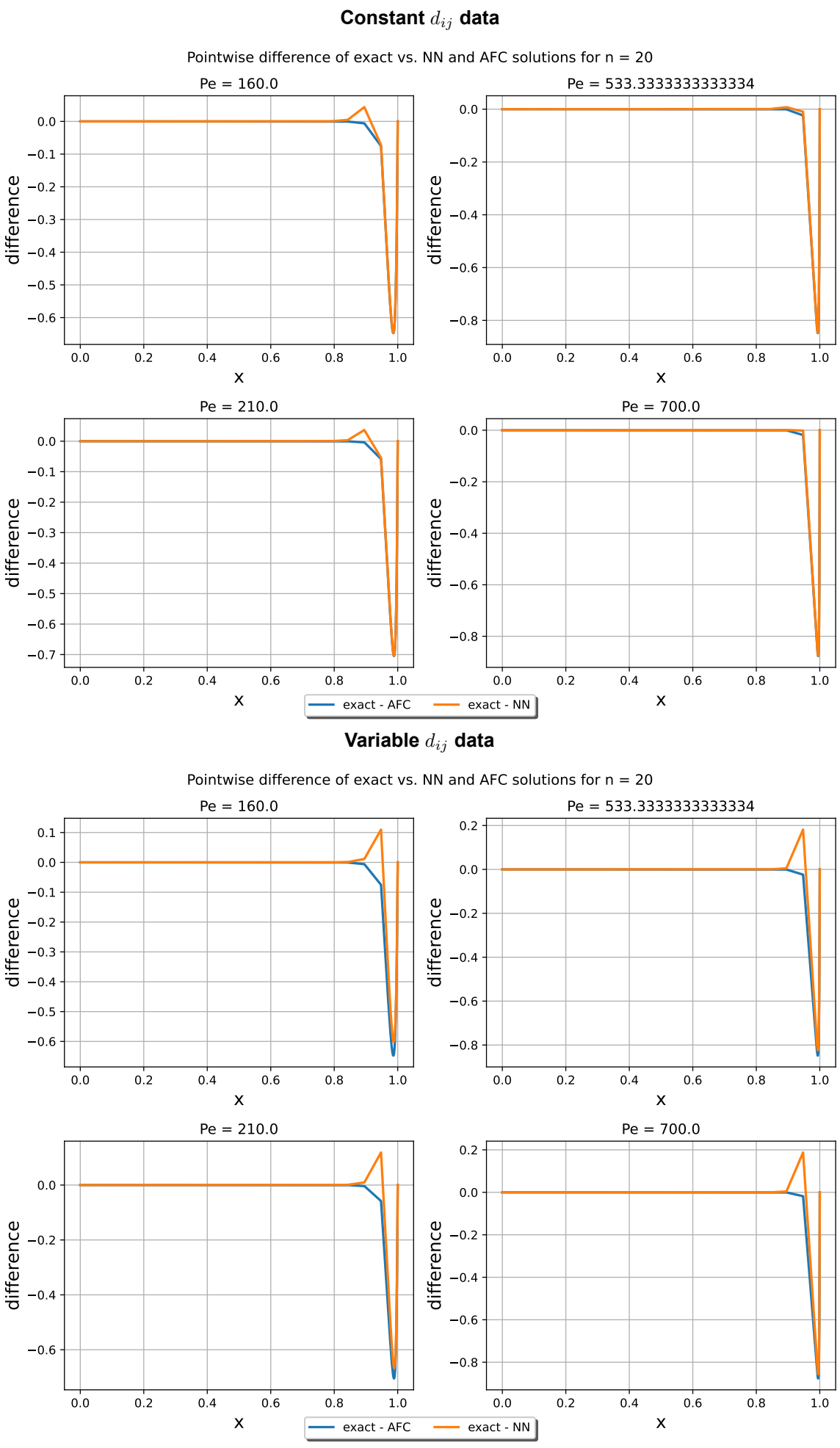


Figure B.7: PINN pointwise differences plots for the variation of the boundary layer problem with positive high Péclet numbers for $n = 20$ trained with constant diffusion data (upper) and variable diffusion data (lower).

B.2. Asymmetric data: input-output mapping

The predictions for each trained neural network for the constant and variable diffusion constant data is given in an input and output mapping against the true data in the following sections. Moreover, both data sets have 3528 and 2940 data points respectively, where one data point corresponds to the three input fluxes f_{ij} and the corresponding output α_{ij} value. The data is divided into three sets, such as: 70% (2469 data points for constant, 2057 for variable) is the training data, 22.5% (794 data points for constant, 662 for variable) is the validation data and 7.5% (265 data points for constant, 221 for variable) is the test data. The results are given for the training and validation sets.

B.2.1. Feedforward neural network

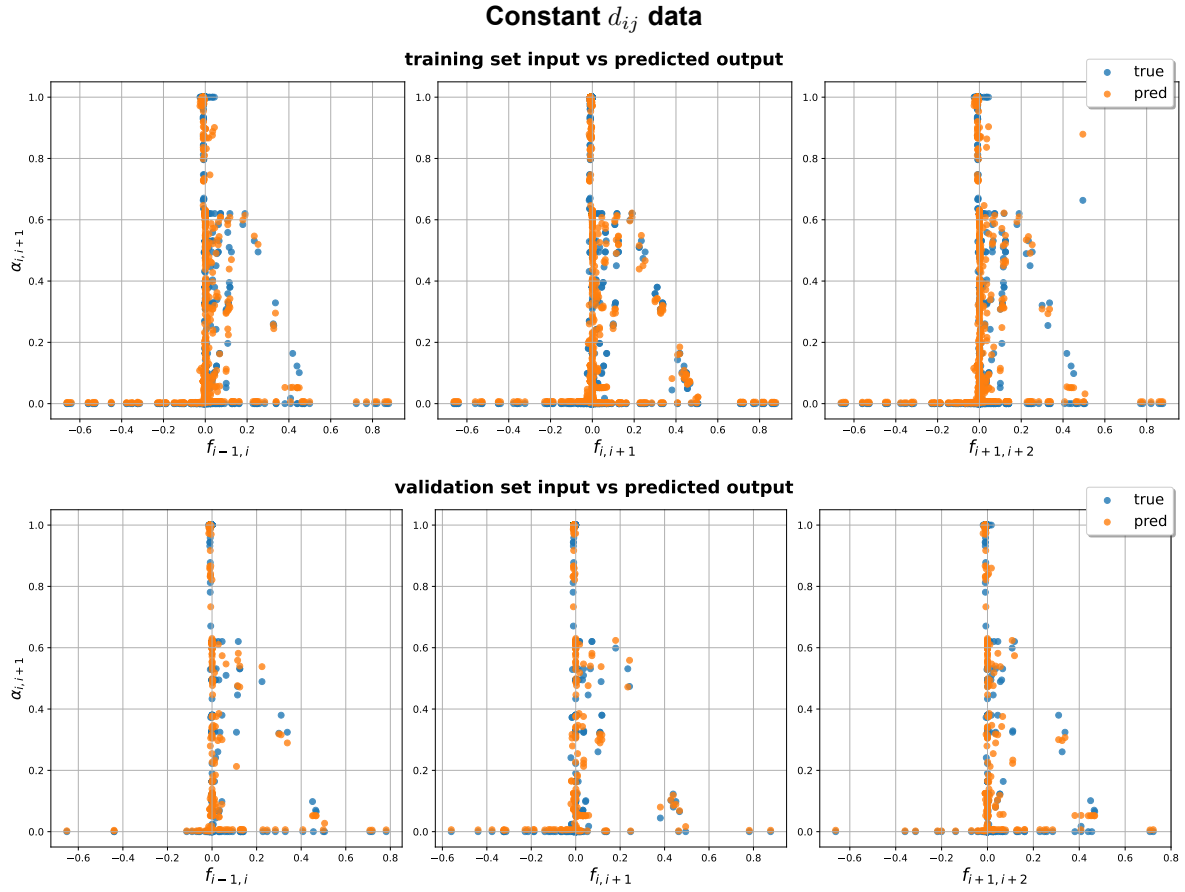


Figure B.8: FNN Input-output mapping for the training and validation data sets with asymmetric data for constant diffusion data.

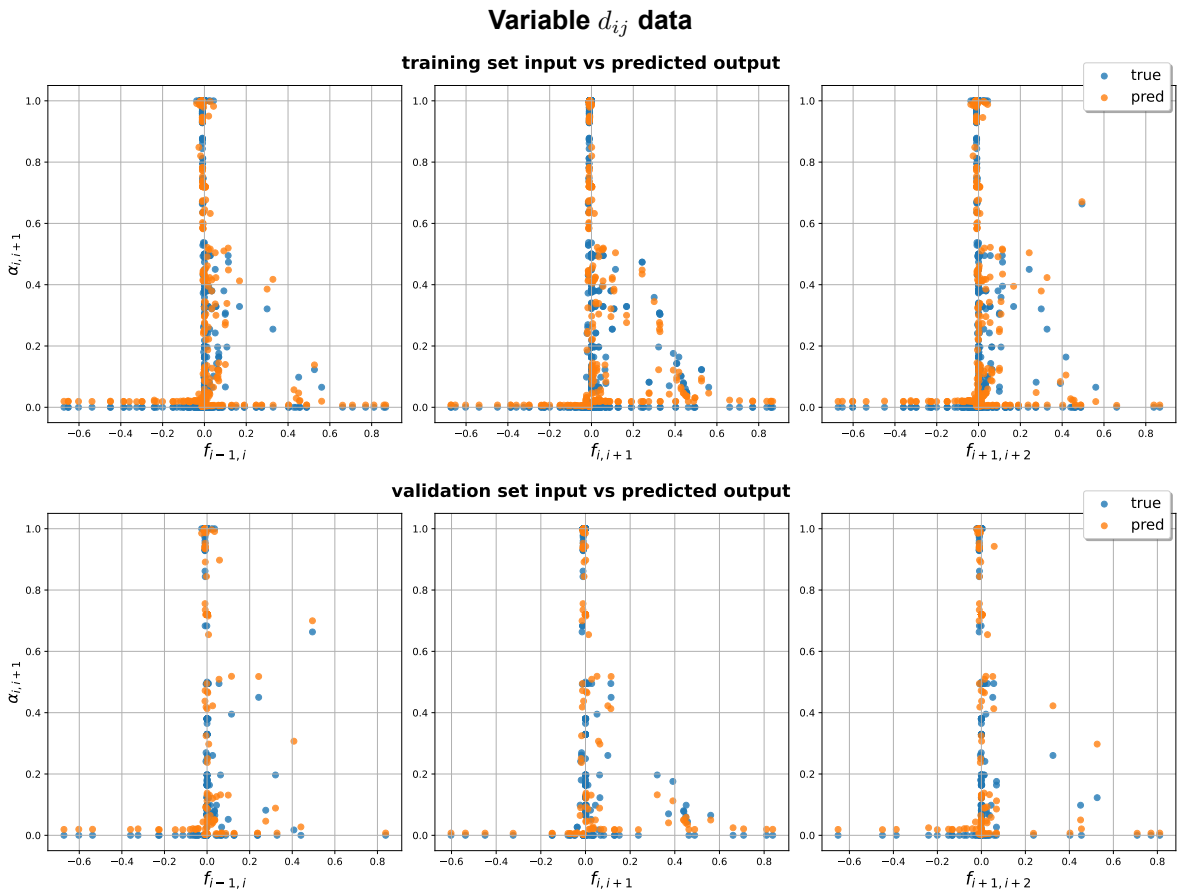


Figure B.9: FNN Input-output mapping for the training and validation data sets with asymmetric data for variable diffusion data.

B.2.2. Splitted neural network

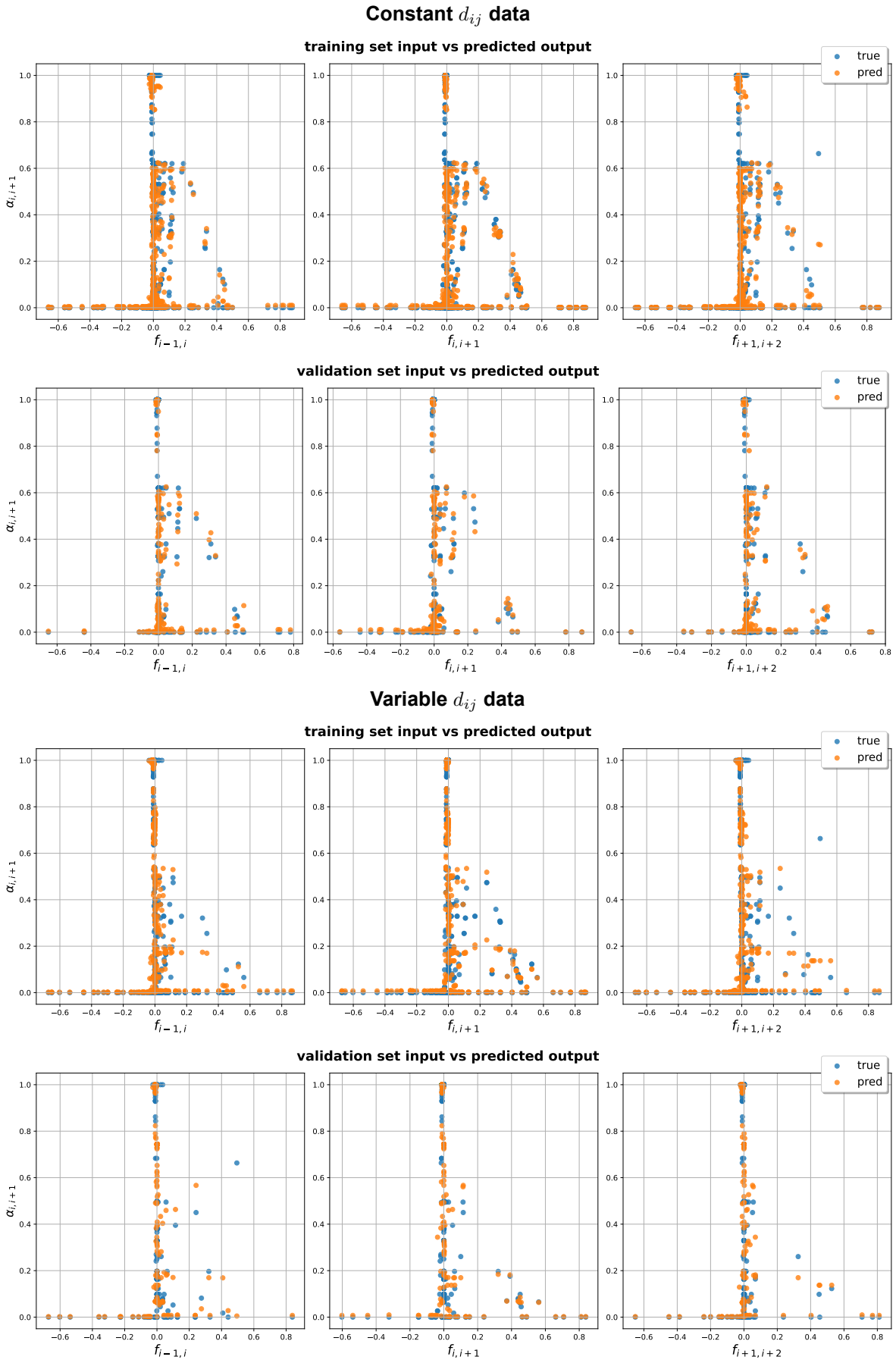


Figure B.10: SpNN Input-output mapping for the training and validation data sets with asymmetric data for constant diffusion data (upper) and variable diffusion data (lower).

B.2.3. Physics-informed neural network

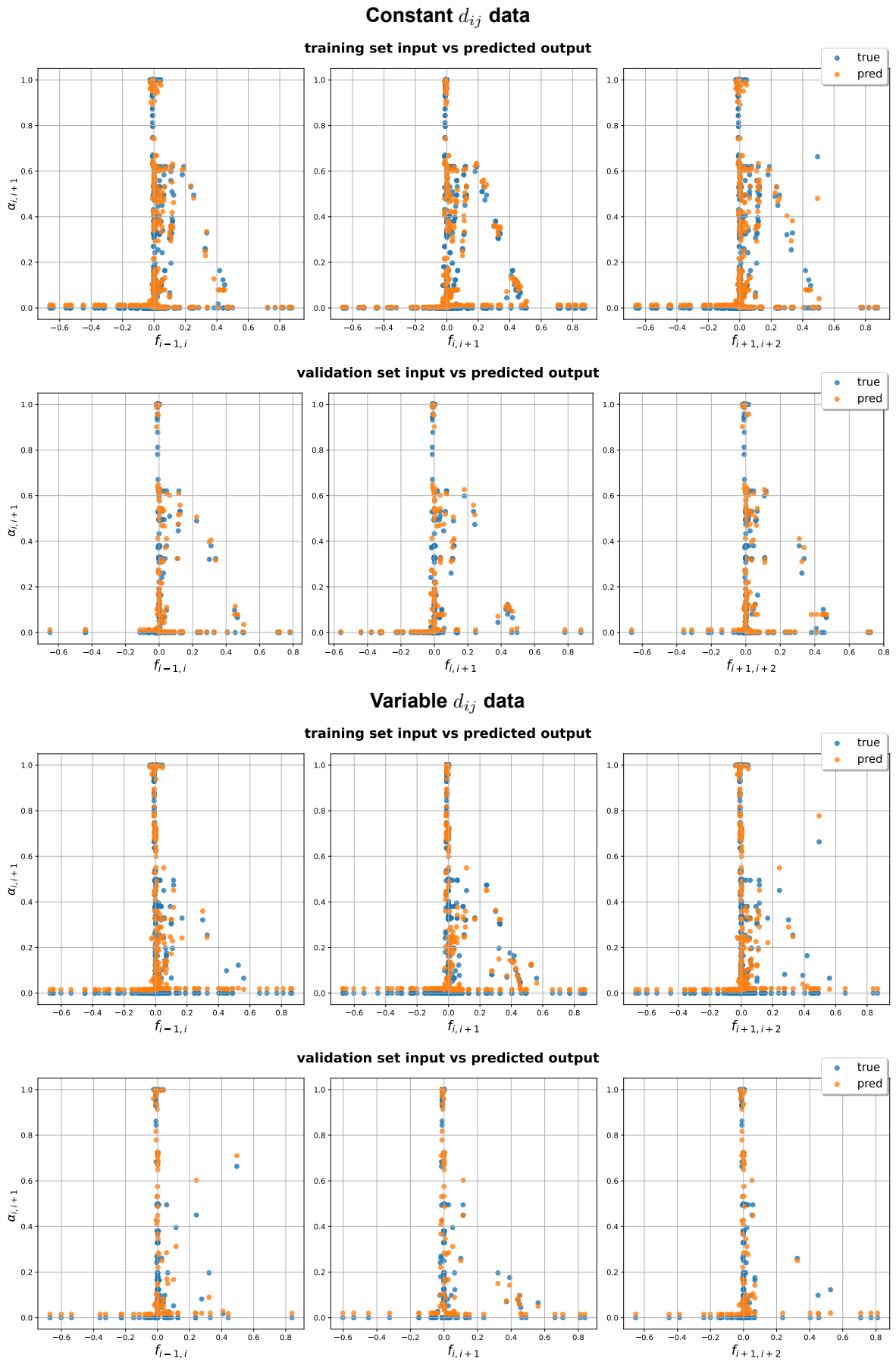


Figure B.11: PINN Input-output mapping for the training and validation data sets with asymmetric data for constant diffusion data (upper) and variable diffusion data (lower).

B.3. Symmetric data results

The predictions for each trained neural network for the constant and variable diffusion constant data is given in an input and output mapping against the true data in the following sections. Moreover, both data sets have 7056 and 7840 data points respectively, where one data point corresponds to the three input fluxes f_{ij} and the corresponding output α_{ij} value. The data is divided into three sets, such as: 70% (4939 data points for constant, 5487 for variable) is the training data, 22.5% (1587 data points for constant, 1764 for variable) is the validation data and 7.5% (530 data points for constant, 589 for variable) is the test data. The results are given for the training and validation sets. The results are given for the training and validation sets. Furthermore, the relative error results for the boundary layer, peak and variation of the boundary layer problems are also given for each model trained on the constant and variable diffusion data sets.

B.3.1. Feedforward neural network

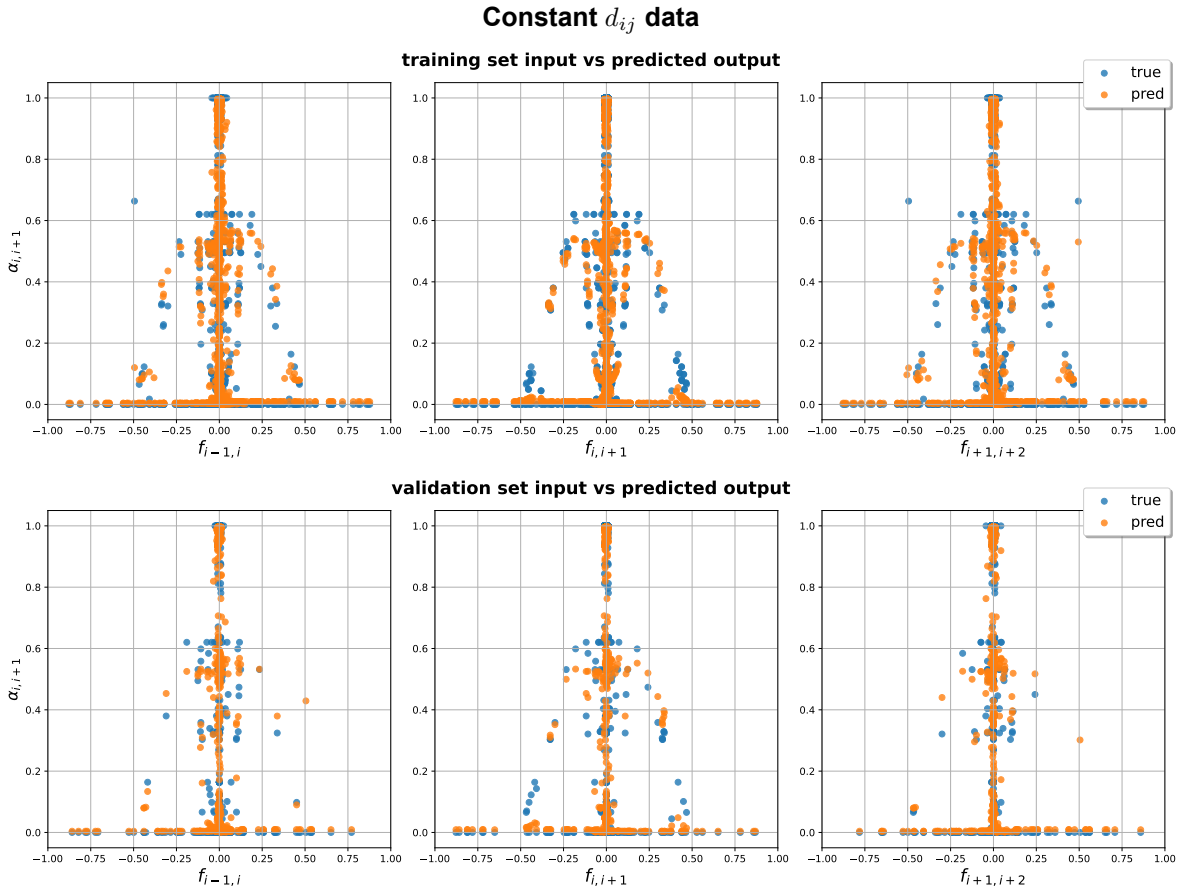


Figure B.12: FNN Input-output mapping for the training and validation data sets with symmetric data for constant diffusion data.

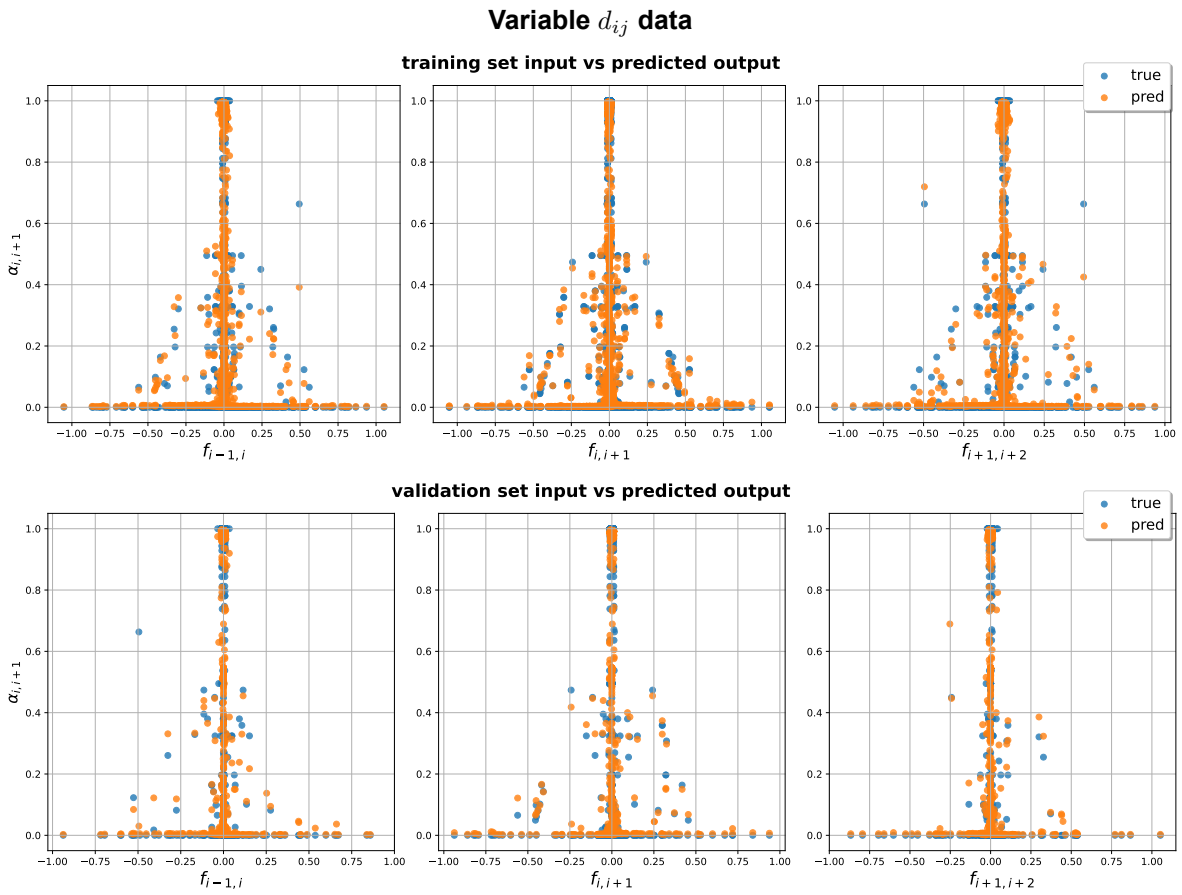


Figure B.13: FNN Input-output mapping for the training and validation data sets with symmetric data for variable diffusion data.

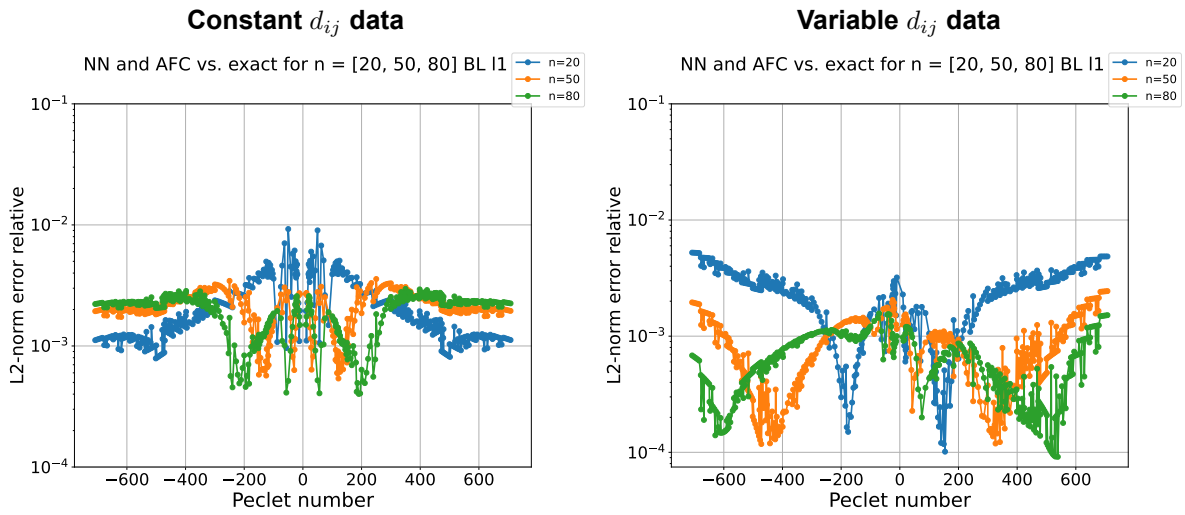


Figure B.14: FFNN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

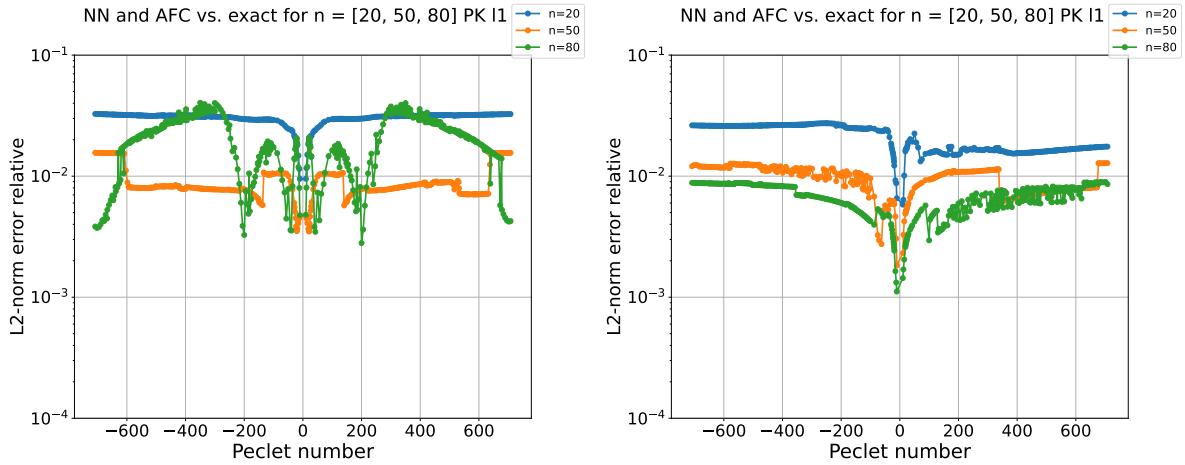


Figure B.15: FFNN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

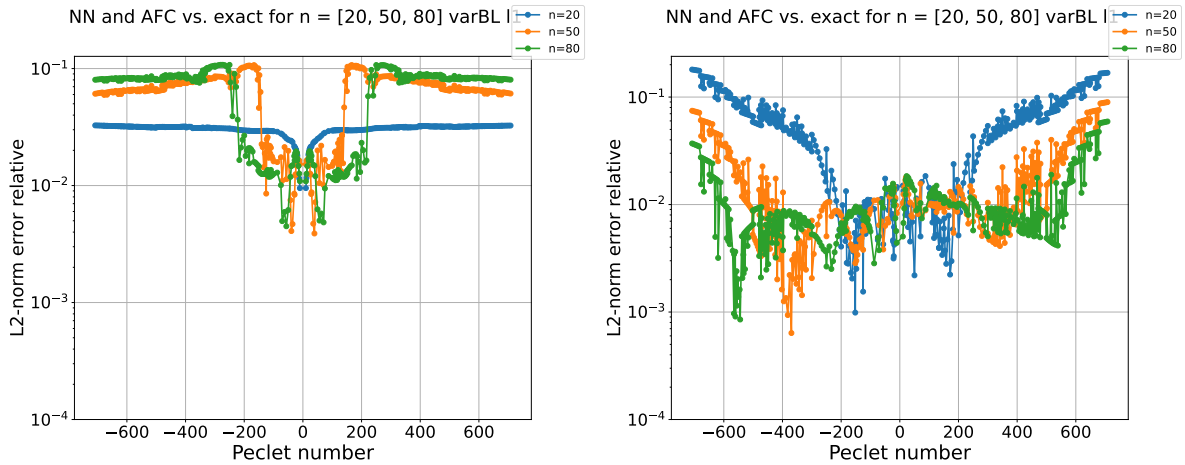


Figure B.16: FFNN L2 relative error for **variation of boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

B.3.2. Splitted neural network

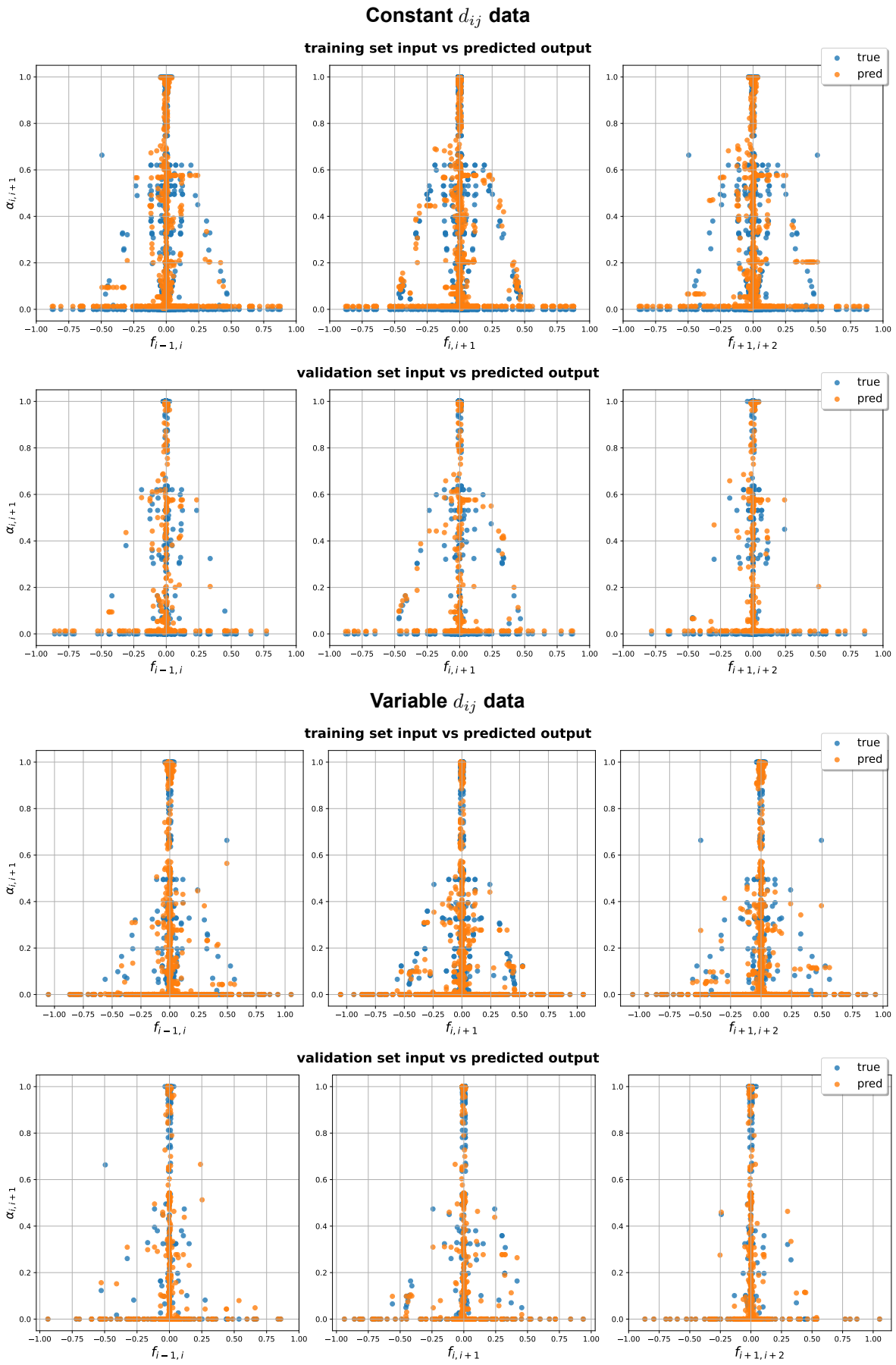


Figure B.17: SpNN Input-output mapping for the training and validation data sets with symmetric data for constant diffusion data (upper) and variable diffusion data (lower).

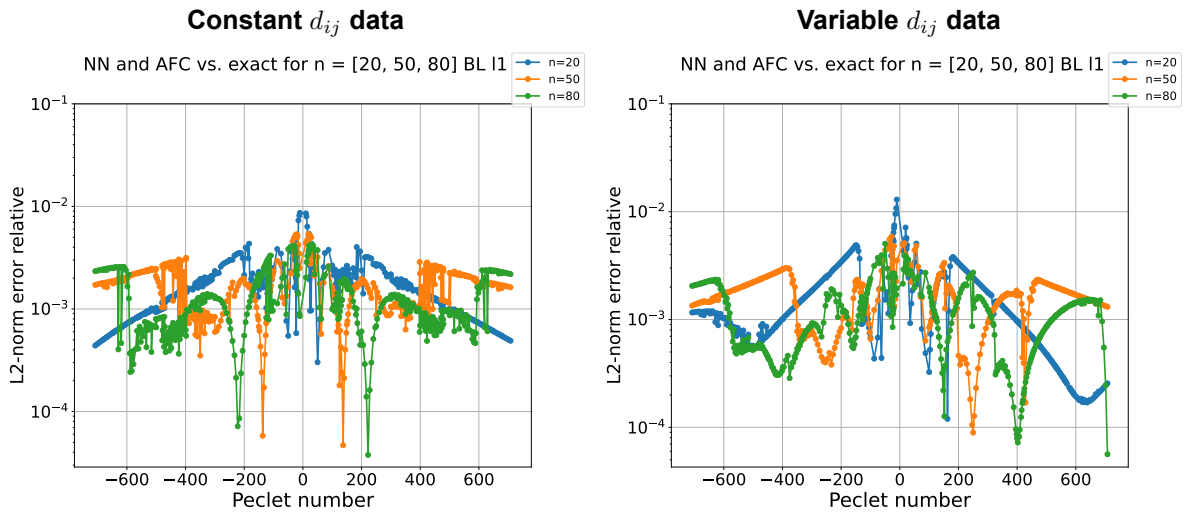


Figure B.18: SpNN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

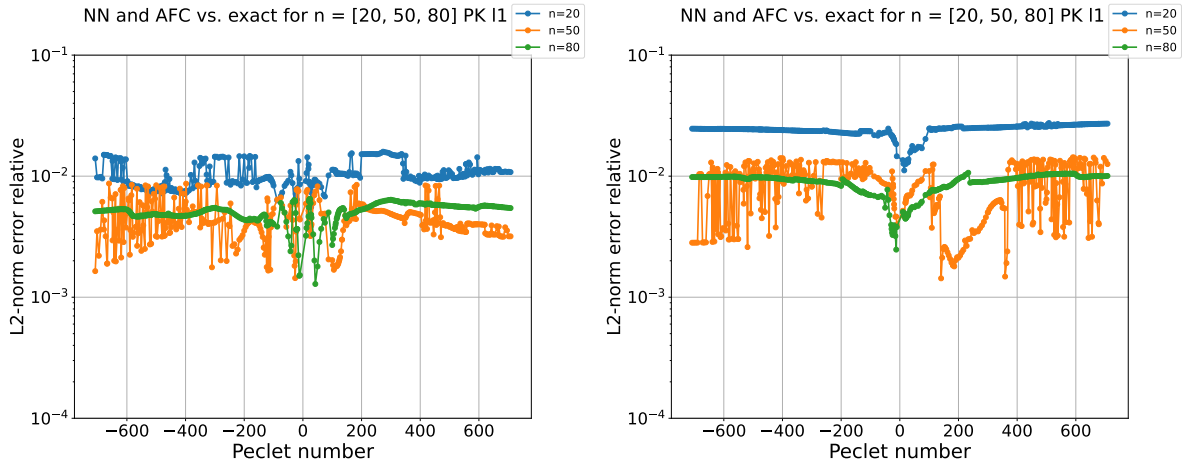


Figure B.19: SpNN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

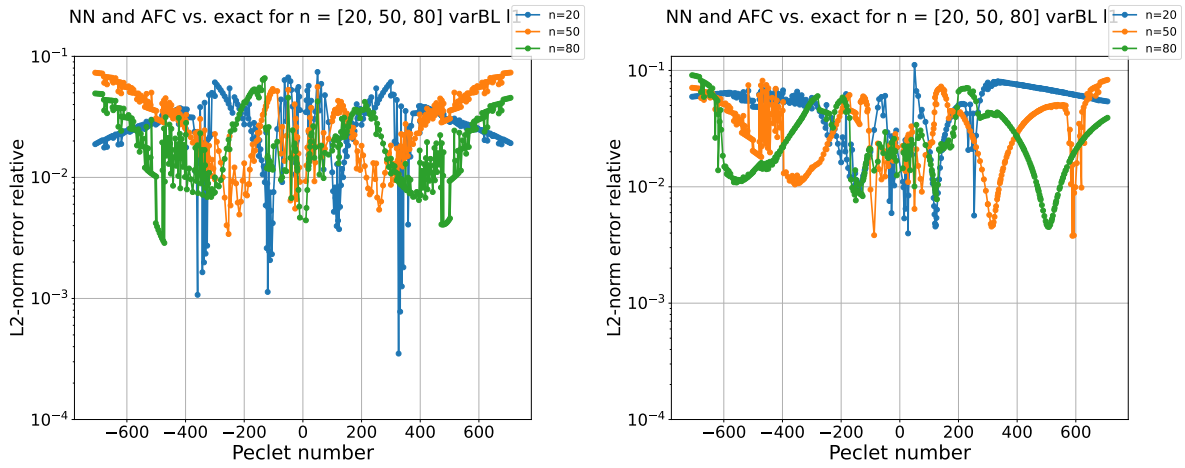


Figure B.20: SpNN L2 relative error for **variation of boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

B.3.3. Physics-informed neural network

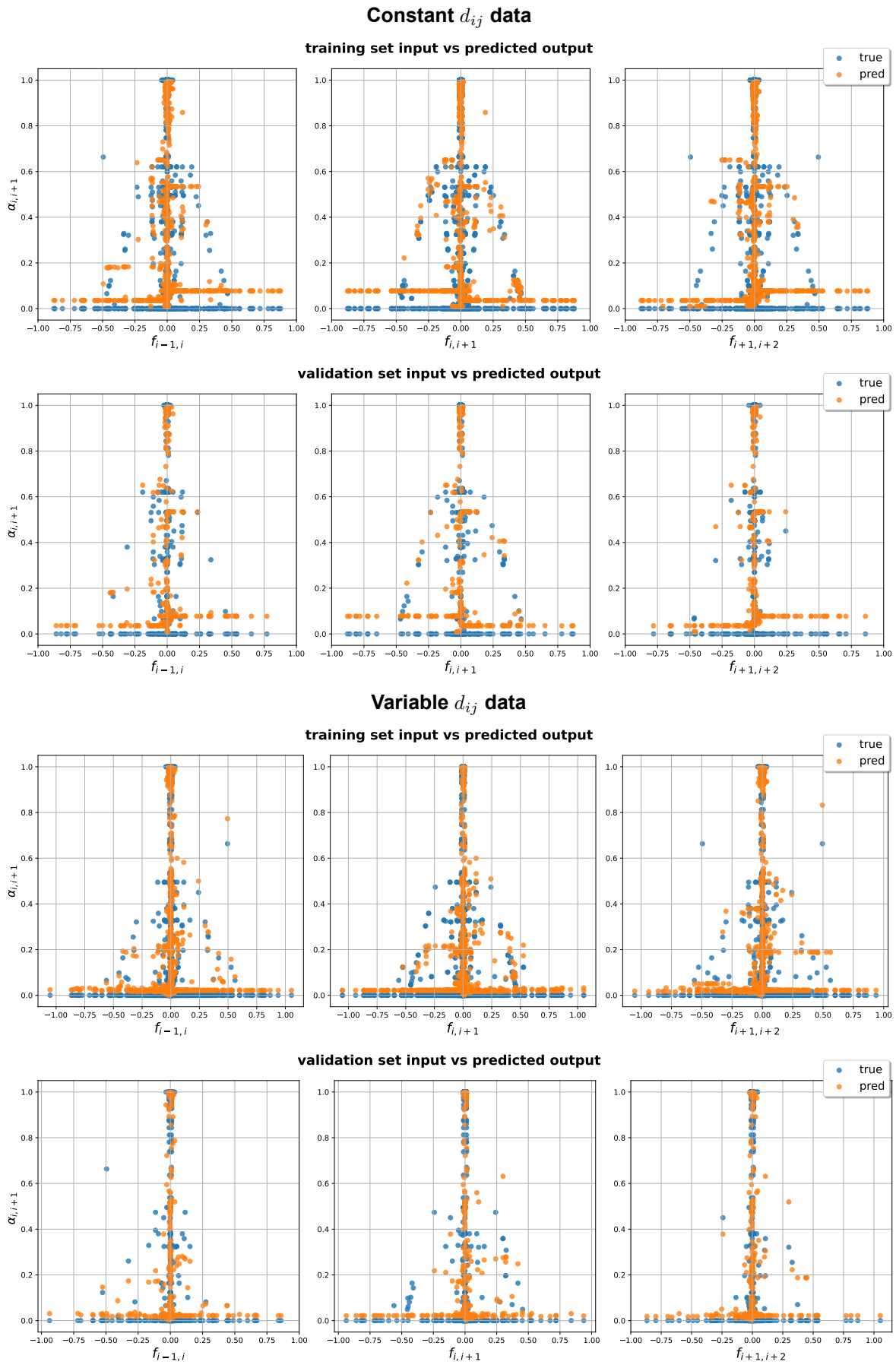


Figure B.21: PINN Input-output mapping for the training and validation data sets with symmetric data for constant diffusion data (upper) and variable diffusion data (lower).

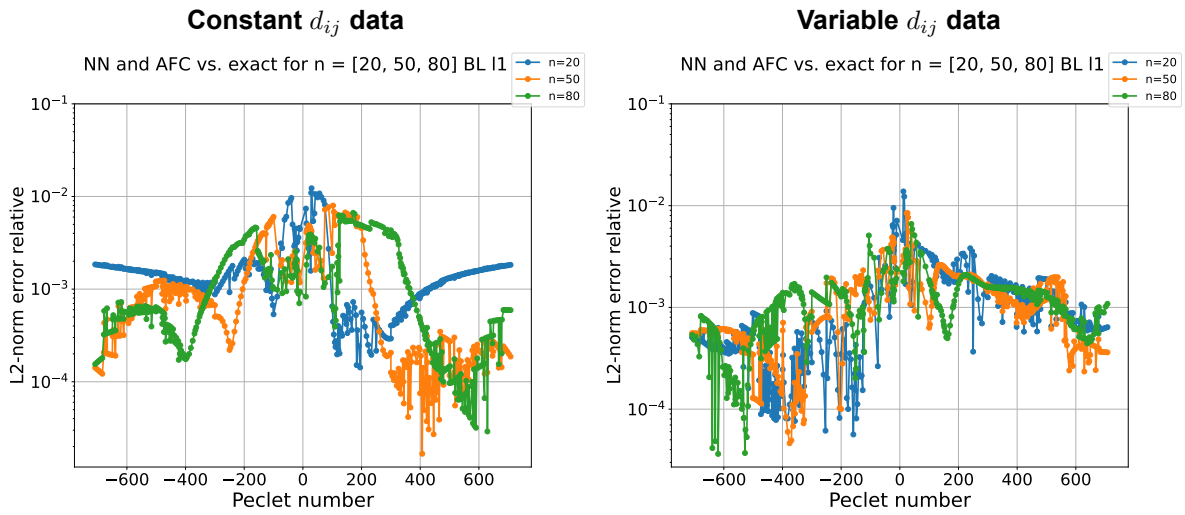


Figure B.22: PINN L2 relative error for **boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

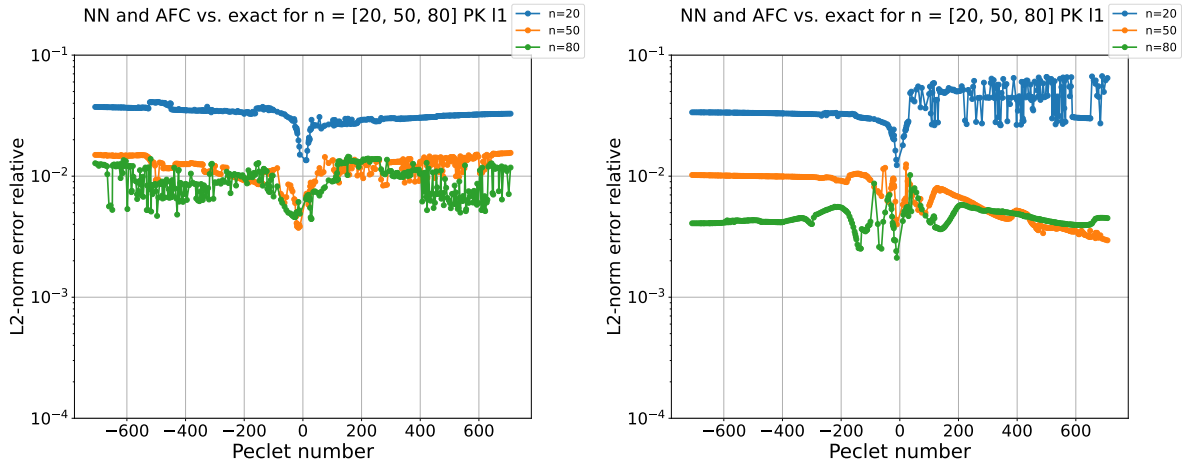


Figure B.23: PINN L2 relative error for **peak problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

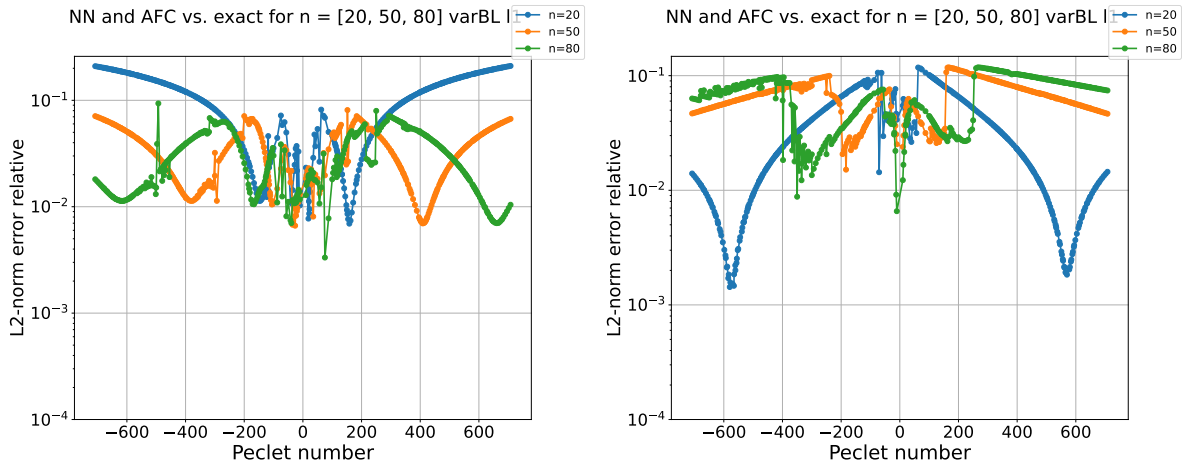


Figure B.24: PINN L2 relative error for **variation of boundary layer problem** with positive and negative Péclet numbers for $n = 20, 50, 80$ with symmetric data for constant diffusion data (left) and variable diffusion data (right).

B.4. Improvements: symmetric data vs asymmetric data

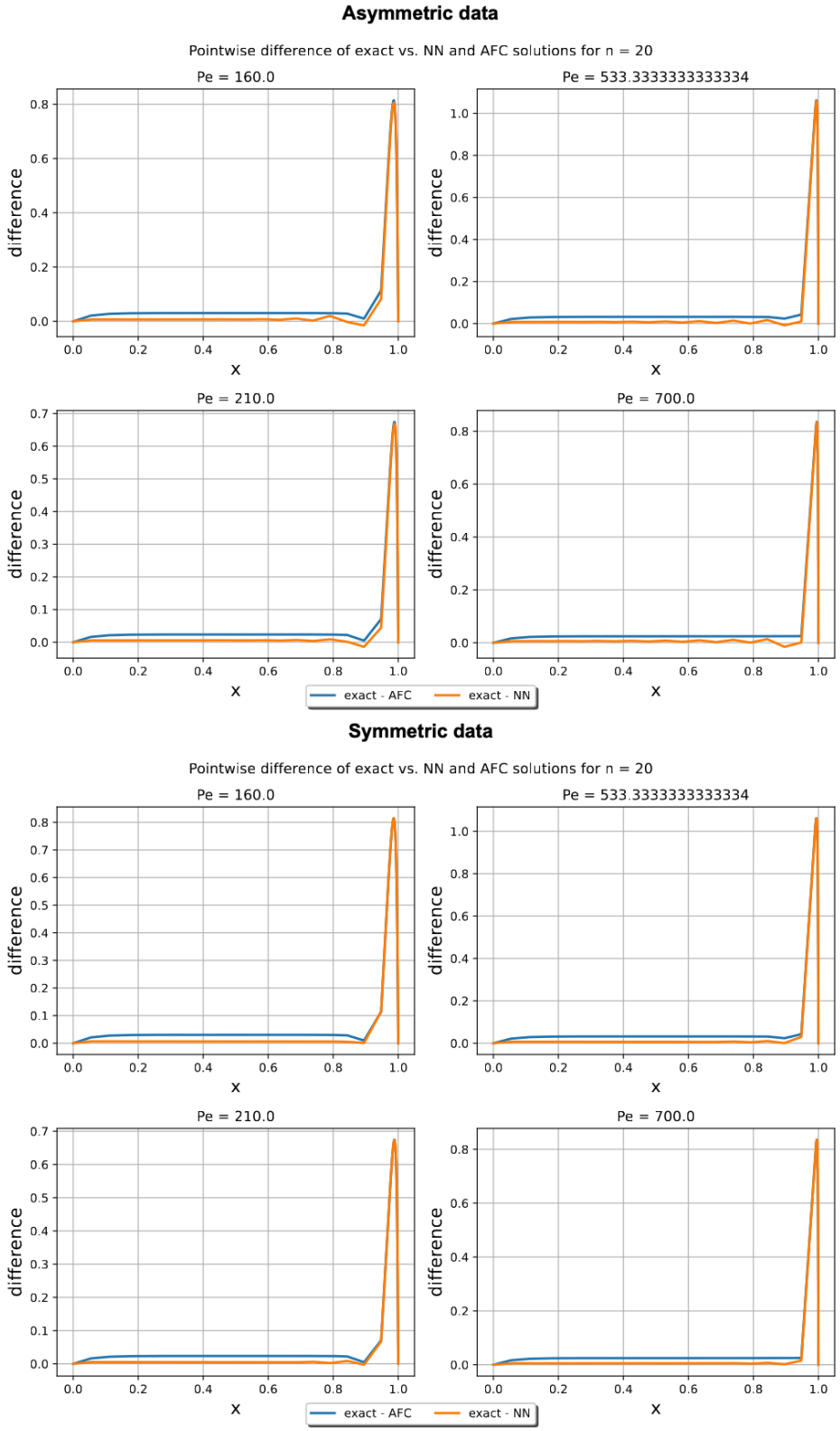


Figure B.25: FFNN differences plot for $n = 20$ for constant d_{ij} data with asymmetric data and symmetric data for the peak problem.

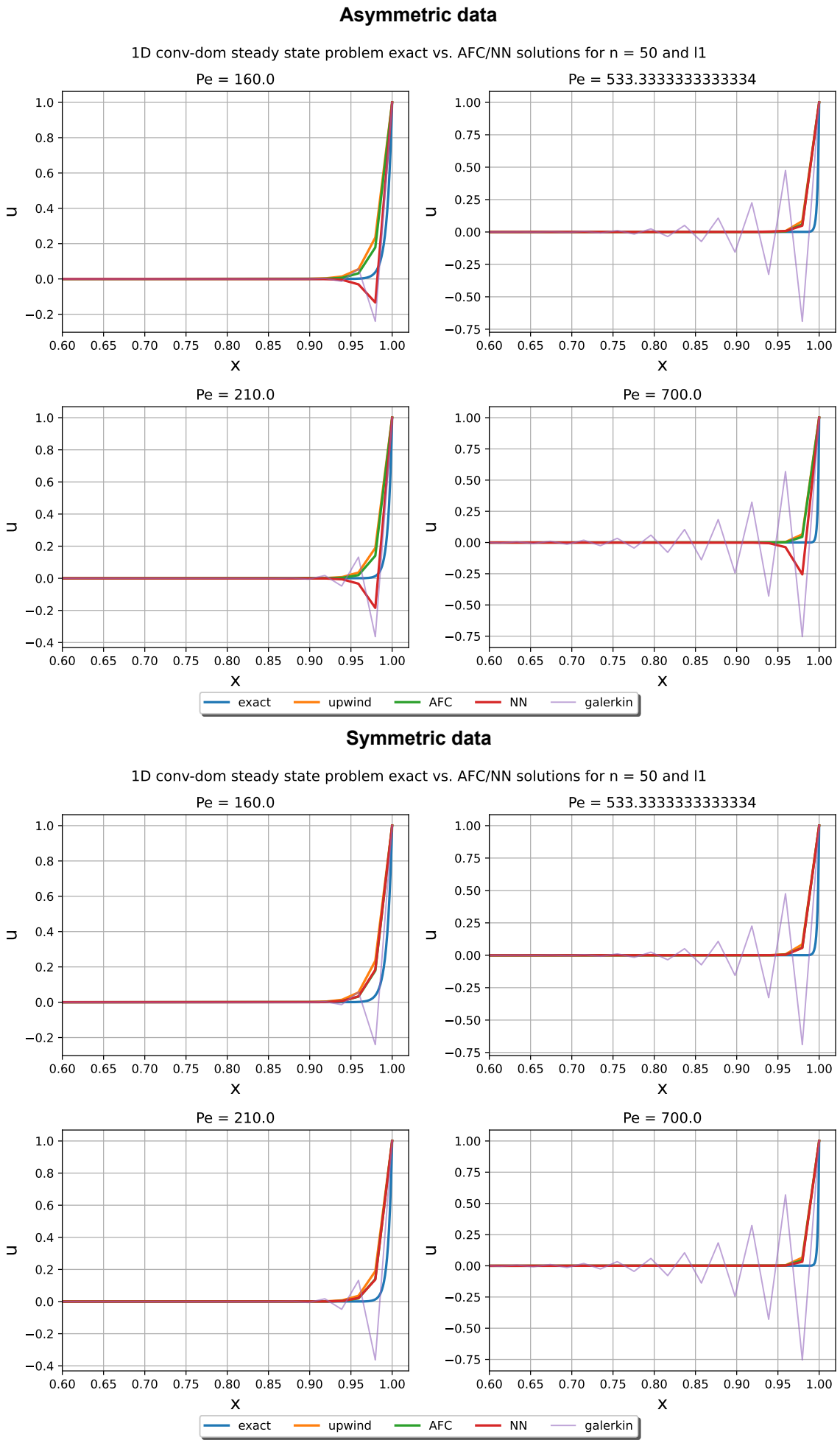


Figure B.26: FFNN solution profiles for $n = 50$ for variable d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem.

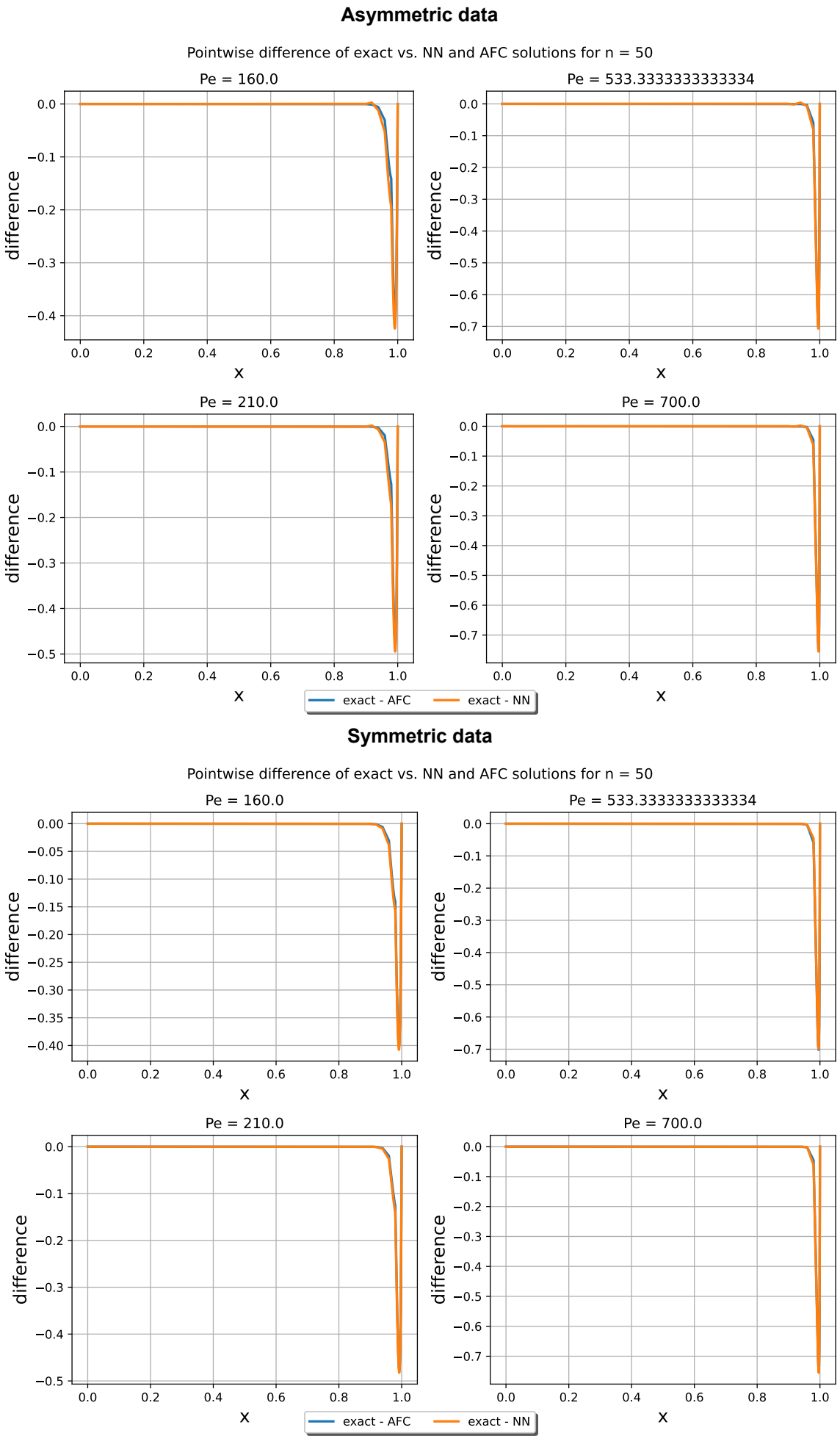
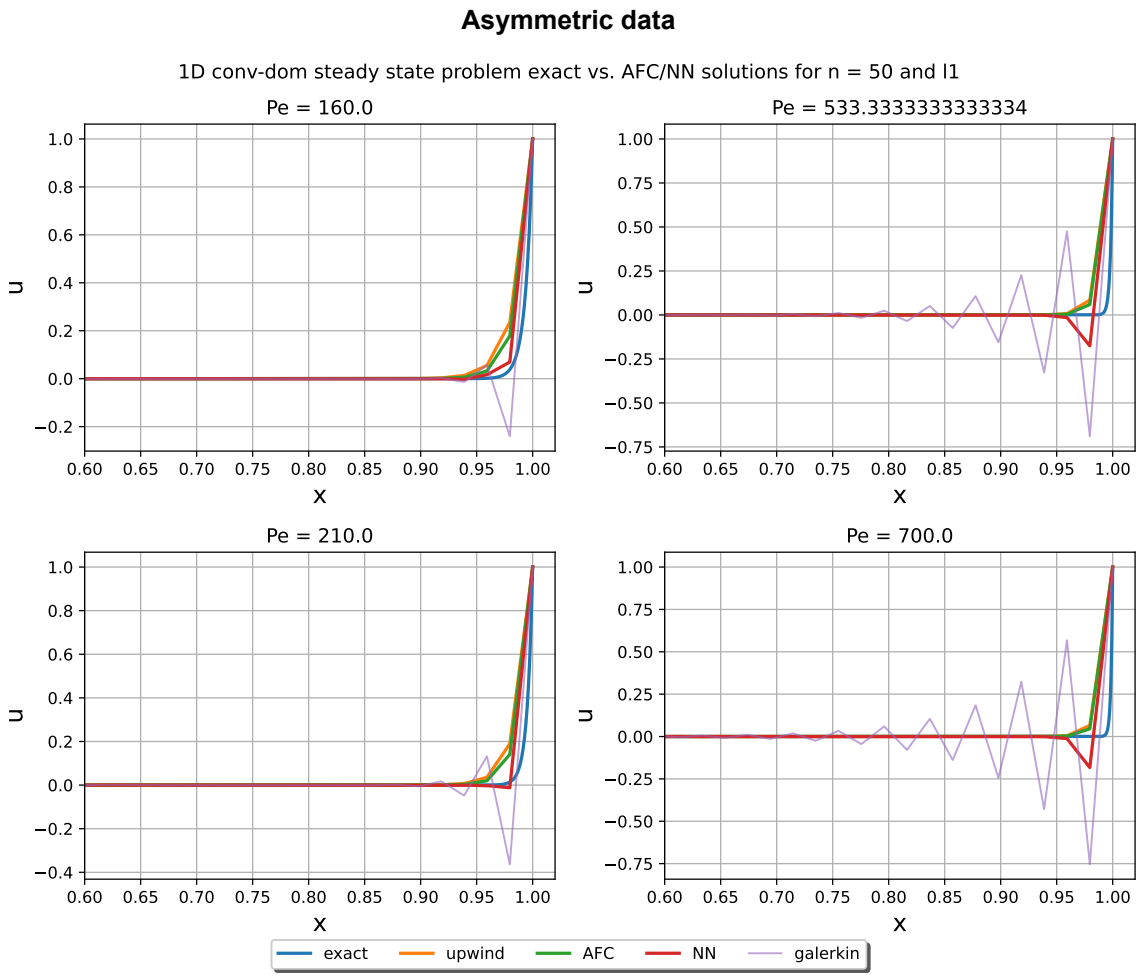


Figure B.27: SpNN differences plot for $n = 50$ for variable d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem.



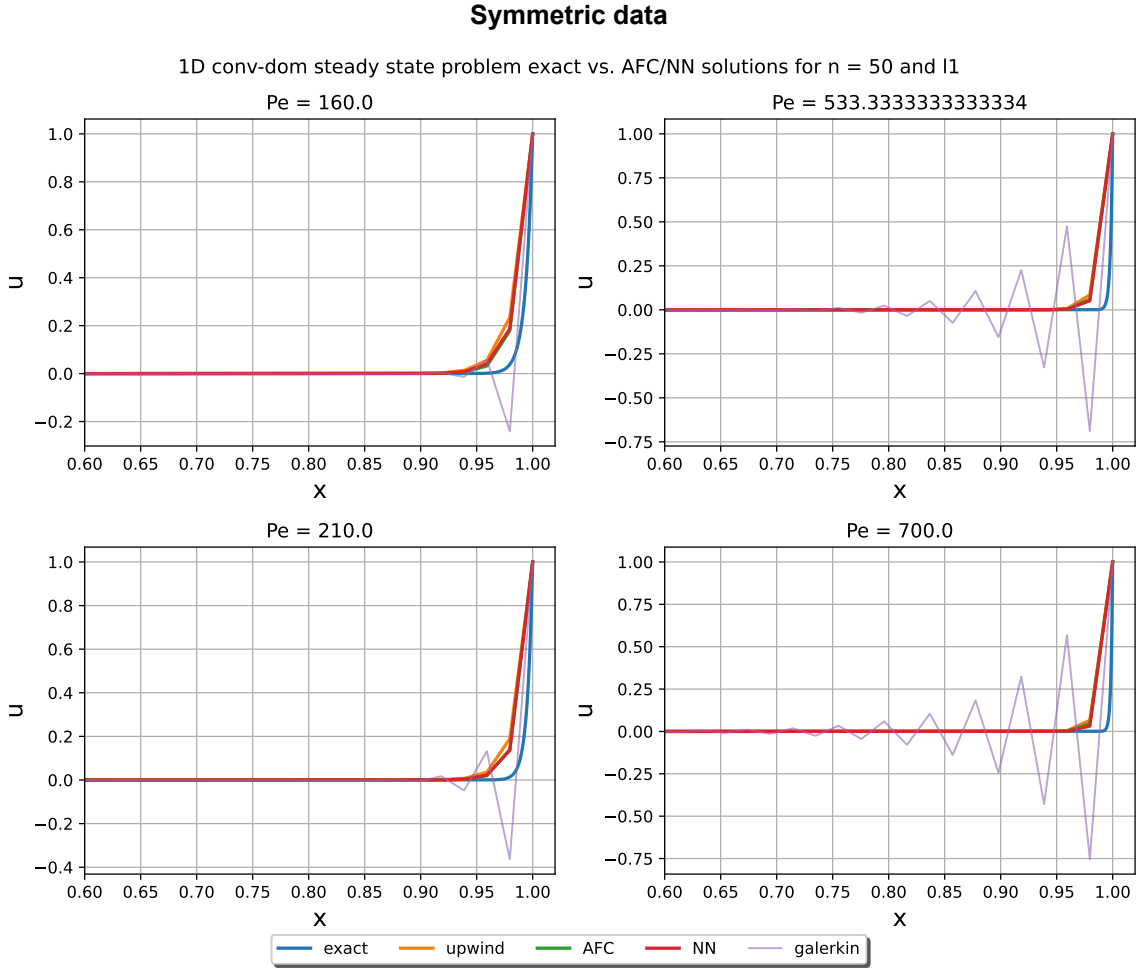


Figure B.28: SpNN solution profiles for $n = 50$ for constant d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem.

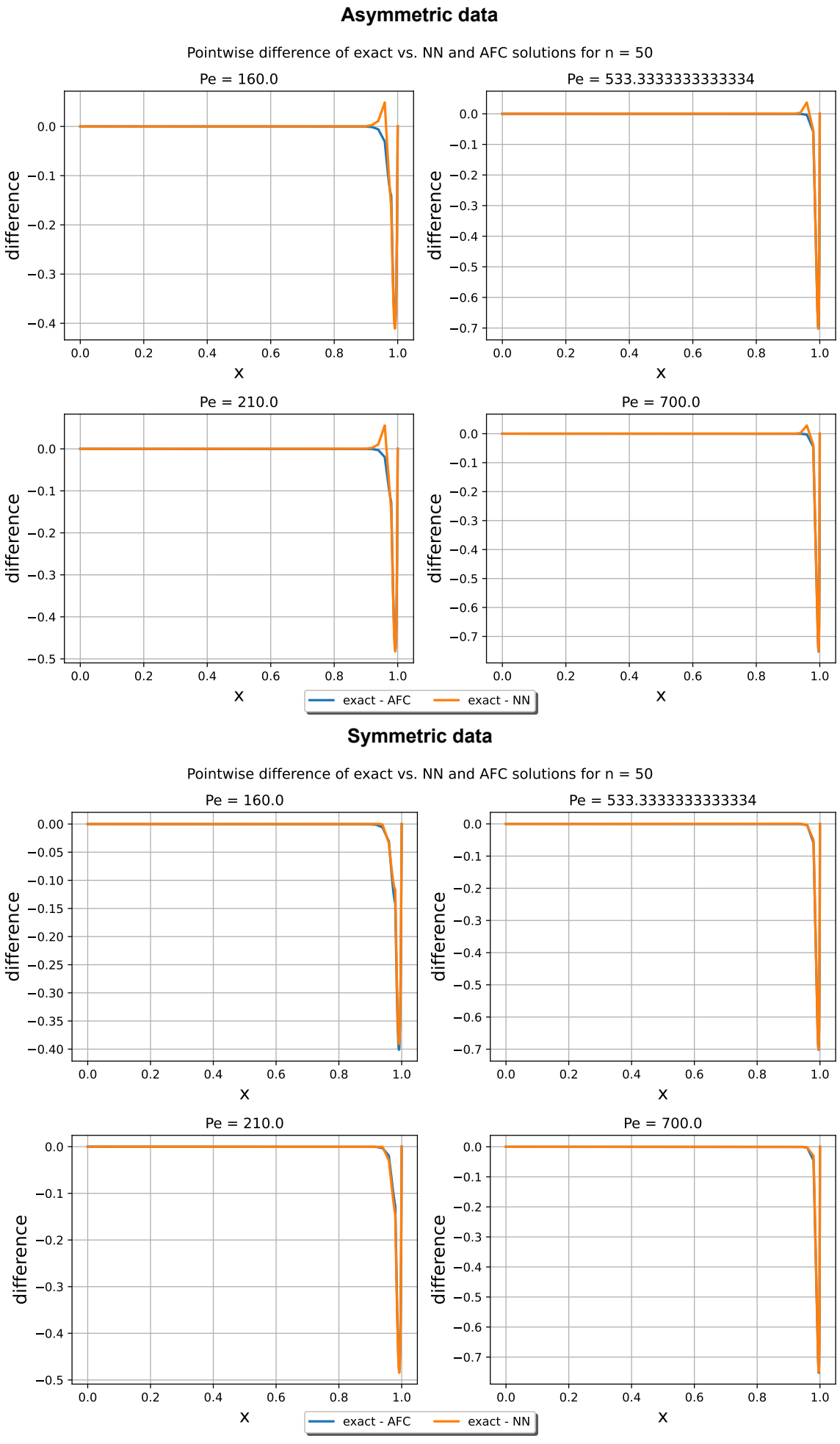
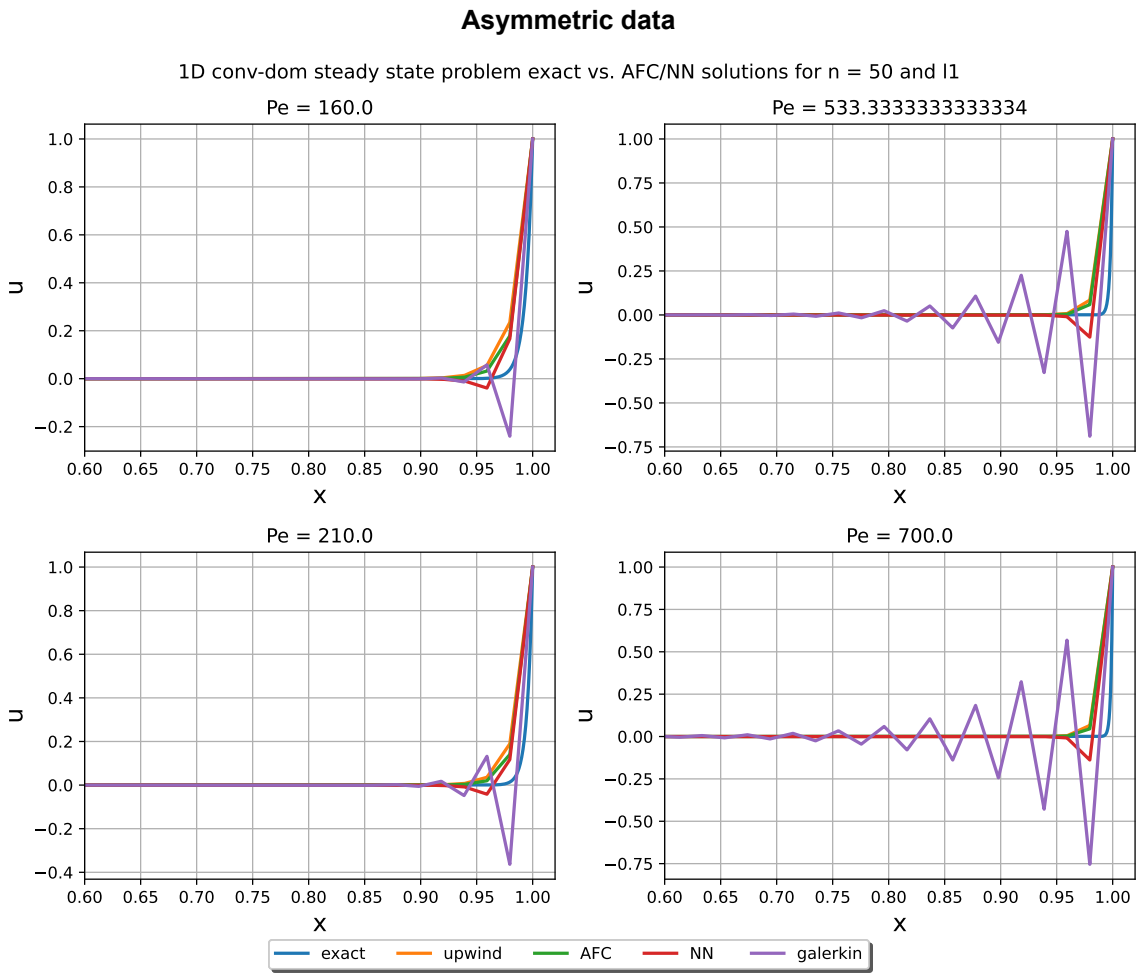


Figure B.29: PINN differences plot for $n = 50$ for constant d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem.



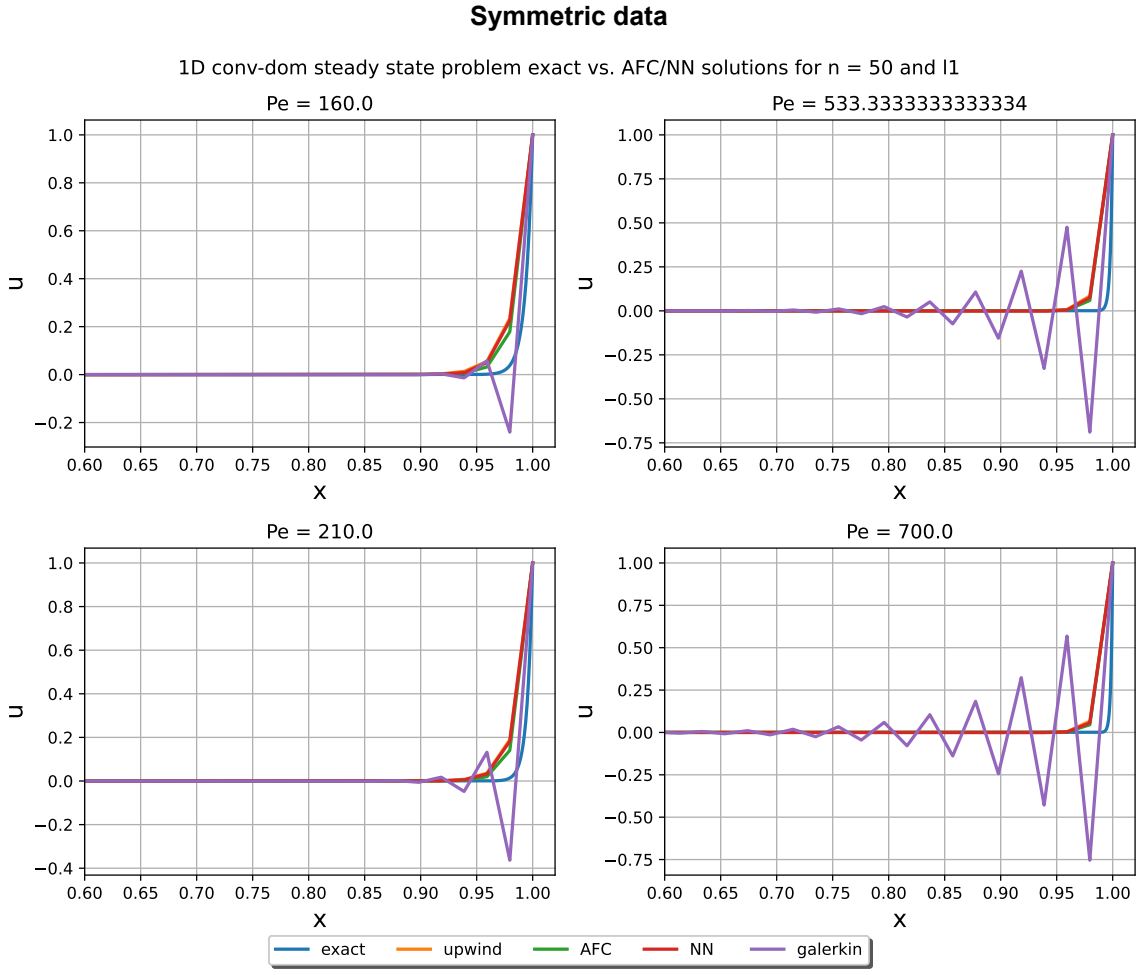


Figure B.30: PINN solution profiles for $n = 50$ for variable d_{ij} data with asymmetric data and symmetric data for the variation of the boundary layer problem.