Politecnico di Milano

MSc in Mathematical Engineering

École polytechnique fédérale de Lausanne

MSc in Computational Science and Engineering

Master Thesis

Controlling oscillations in high-order schemes using neural networks

Author:

Niccolò Discacciati
(ID No. 849777)

Supervisors:

Prof. Jan S. Hesthaven
Prof. Nicola Parolini
Dr. Deep Ray

Academic Year 2017-2018

POLITECNICO
MILANO 1863

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Abstract

High-order numerical solvers for conservation laws suffer from Gibbs phenomenon close to discontinuities, leading to spurious oscillations and a detrimental effect on the solution accuracy. A possible strategy to reduce their amplitude aims to add a suitable amount of artificial viscosity. Several models are available in the literature, which rely on the identification of a shock sensor, adding dissipation when the solution regularity is lost. The dependence on problem-specific parameters limits their performances. To solve this issue, in this thesis we propose a new technique based on artificial neural networks. In particular, we focus on the construction of multilayer perceptrons. Emphasis is given to the training phase, which is carried out using a robust dataset created using the classical models with optimal parameters. The online evaluation is then integrated in the numerical solver of the partial differential equation. Even though most of the effort is put in one-dimensional problems, an extension to a two-dimensional scenario is provided. Several numerical results are presented to demonstrate the capabilities of the network-based technique. Initially, we focus on one-dimensional scalar problems, where Burgers equation represents our first benchmark test. Then, we move to less simple cases, characterized by non-convex flux functions or involving multiple equations (compressible Euler equations). The same strategy is followed for multi-dimensional problems. In most of the cases, the proposed model is able to guarantee high accuracy in presence of smooth solutions and to capture discontinuities (shock and contact waves). In general, the results are comparable to (or better than) the classical models with properly tuned parameters. A final performance analysis is carried out.

Keywords: Conservation laws, Discontinuous Galerkin, Artificial viscosity, Artificial neural networks

Abstract (Italiano)

Schemi numerici di alto ordine per leggi di conservazioni sono soggetti al fenomeno di Gibbs in prossimità di punti di discontinuità, causando oscillazioni spurie e intaccando negativamente l'accuratezza della soluzione. Una possibile strategia per smorzarne l'ampiezza consiste nell'aggiungere un adeguato valore di viscosità artificiale. Diversi modelli sono presenti in letteratura, molti dei quali sono basati sull'identificazione di un sensore di discontinuità (discontinuity sensor), aggiungendo dissipazione dove la soluzione perde di regolarità. La dipendenza da parametri empirici limita le loro performances. Al fine di risolvere questo problema, in questa tesi viene introdotta una nuova tecnica basata su reti neurali artificiali (artificial neural networks). Più precisamente, viene presentata la costruzione di percettroni multistrato (multilayer perceptrons). Particolare attenzione è data alla fase di apprendimento (training), che viene effettuata usando un opportuno set di dati creato per mezzo dei modelli standard con parametri ottimali. L'applicazione della rete neurale è integrata all'interno del solutore numerico dell'equazione differenziale. Sebbene maggior enfasi è posta su problemi monodimensionali, vengono fornite opportune estensioni al caso bidimensionale. Sono presentati diversi risultati numerici, con lo scopo di dimostrare le potenzialità della tecnica proposta. Inizialmente, vengono considerati problemi scalari monodimensionali, dove l'equazione di Burgers rappresenta il primo caso test. Successivamente, si analizzano casi più complessi, caratterizzati da flussi non convessi o dalla presenza di più equazioni (equazioni di Eulero). La medesima strategia è perseguita per problemi multidimensionali. Nella maggior parte dei casi, il modello proposto garantisce elevata accuratezza in presenza di soluzioni regolari, così come la capacità di catturare diverse discontinuità (onde di shock e di contatto). In generale, i risultati sono comparabili con (o meglio de) i modelli classici con parametri opportunamente tarati. Infine, viene effettuata una analisi delle performances computazionali delle tecniche considerate.

Parole chiave: Leggi di conservazione, Discontinuous Galerkin, Viscosità artificiale, Reti neurali artificiali

Acknowledgements

I would like to sincerely thank Prof. Jan S. Hesthaven for the opportunity to work within the Chair of Computational Mathematics and Simulation Science (MCSS) at EPFL. His inspiring ideas, his continuous support and his positive attitude represented a continuous incentive and motivation throughout these months.
I wish to express my gratitude to Prof. Nicola Parolini, who accepted the role of thesis co-supervisor without even knowing me. I really appreciated his kindness and availability.
A special thanks goes to Dr. Deep Ray. The precious advice he gave me, together with his positive spirit, helped me a lot to fulfil my goals. The discussions we had during the last semester were extremely useful and inspirational.
Thanks to all the people within MCSS, with a particular mention to my office mates Boris and Nicolò. From the first day you made me feel as integral part of the group. I really enjoyed all the moments we shared together, especially the ones outside university.
Ringrazio i miei colleghi di Doppia Laurea (Alberto, Antonio, Riccardo e Santo) con i quali ho condiviso sia i periodi di disperazione che i momenti felici durante la nostra permanenza a Losanna. Anche quando il traguardo sembrava irraggiungibile, li ringrazio per aver reso questa esperienza indimenticabile.
Ringrazio anche tutti gli amici e colleghi "italiani", specialmente Jacopo e Matteo. Le belle giornate passate all'interno e al di fuori dell'università saranno per sempre parte della mia vita.
Un ringraziamento particolare va a Maria Cristina. Lei sa il perché.
Infine, un grazie di cuore alla mia famiglia (Paola, Tino, Fabio, Maria Grazia e Camilla) per avermi supportato emotivamente ed economicamente durante questo periodo a Losanna. Sapere di poter contare sempre su di voi mi ha aiutato a superare i momenti difficili vissuti in questo periodo. Ho sempre apprezzato l'affetto e lo spirito positivo che mi avete dimostrato ogni volta che tornavo a casa.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the context of computational sciences, numerical accuracy and precision are key properties demanded by practical applications, such as mechanics or fluid dynamics [1]. These requirements are translated in a strict limit on the error level of the associated discretization schemes. Thus, the last few decades have seen an increasing research activity in the development of high-order methods, among which Discontinuous Galerkin (DG) schemes [2] have risen in popularity. Geometrical flexibility, high order accuracy, conservation of physical properties and high parallelization potential are some of the advantages of using DG methods. A challenging class of problems is constituted by hyperbolic conservation laws, such as Euler equations or magnetohydrodynamics. It is well known that their solutions might be discontinuous even for smooth initial data [3]. Therefore, high-order schemes need to be corrected in the regions where regularity is lost in order to avoid the Gibbs phenomenon [4], which consists in the emergence of spurious numerical oscillations close to discontinuities and leads to inaccurate and unstable numerical results.
Several approaches have been proposed to tackle this issue [5], among which a popular family is flux limiting [6, 7, 8]. Although they guarantee the solution to be total variation diminishing (TVD), they are computationally expensive and have a detrimental effect on the solution accuracy. The difficulties in their generalization to a multidimensional framework and the dependence on empirical parameters are critical issues to be taken into account. An alternative technique is based on weighted essentially non-oscillatory (WENO) reconstruction [9, 10]. Despite keeping high order accuracy and being extendible to multidimensional scenarios, the computational cost might still be high. We recall that both these strategies are extensively based on the identification of the troubled cells, namely the mesh elements where the solution loses regularity [11].
Another family of stabilization methods consists in adding artificial dissipation to the problem. Their implementation is usually quite straightforward and can be easily extended to higher dimensions. Ideally, a locally varying amount of viscosity has to be added in each cell, possibly keeping high order accuracy in presence of smooth solutions. In this framework, several approaches have been proposed, with a common theme of constructing a sensor which measures the regularity of the solution, controlling the numerical viscosity to be injected. A first technique employs first-order differential operators, adding dissipation in regions of strong compressibility. The original version, proposed in [12, 13], is developed for compressible Navier-Stokes equations. In [14], the authors define a simple and more general derivative-based model, similar to the technique developed in [15]. A second idea relies on the estimation of the decay rate of the modal coefficients, analogous to a Fourier expansion. Here, the primary approach consists in evaluating the decay of the highest mode [16]. Although this method is quite popular, its suffers from drawbacks related to the lack of both a sense of scale and a monotone decay of the coefficients. Thus, an improved version, specifically designed for high polynomial orders, is proposed in [17]. A third strategy exploits the entropy production, using the local residual of the entropy equation to estimate the artificial viscosity [14, 18, 19]. Other artificial viscosity approaches, which are not considered in this work, construct the dissipation relying on the local residual of the equation [20], the solution jump at the element boundaries [21], or a suitably defined PDE [22].
The main drawback of these models is their dependence on empirical parameters, whose choice might have an impact on stability, accuracy and robustness of the numerical solution. Since there is no rule to estimate the best values, their tuning is usually done in a problem-dependent, time-consuming framework. In this work we propose an alternative technique based on artificial neural networks (ANNs). It can be interpreted as a parameter-free, universal black-box which predicts a (pseudo-)optimal amount of dissipation, having the further advantage of being computationally inexpensive.

In general, ANNs are computing models which are capable of approximating a function exhibiting high degrees of complexity and nonlinearity. They are based on simple computational units, named neurons, which process signals, similarly to their biological counterpart [23]. The neurons are assembled together, creating a network of connected nodes. A key feature of ANNs is their capacity to learn. After training them by means of a given dataset, they are able to predict the output for samples which are not included in the training set. One of the main advantages of ANNs lies in the low computational cost. For a given input, the model output is obtained at a cost which is comparable to some matrix-vector multiplications, independently of the degree of complexity of the underlying input-output mapping. For this reason, they increased their popularity in applications such as image processing, voice recognition, forecasting, medical diagnosis and so on. In the context of numerical analysis and computational sciences, possible applications relate to solution of partial differential equations [24], reduced order modeling [25] or identification of troubled cells for DG schemes [11].
The goal of this work is to explain the design, training and application of an ANN to predict artificial viscosity. Here, we focus on a simple yet effective ANN architecture, named multilayer perceptron (MLP). The neurons are grouped in layers, processing data from an input to an output layer. Despite their simplicity, they can be viewed as universal function approximators [26, 27], making them well suited for our purpose. The (expensive) offline training phase is performed using an appropriately created dataset. The (cheap) online evaluation of the trained network is carried out at each iteration of the time-integration scheme for the particular conservation law being solved.
This rest of this thesis structured as follows. In Chapter 2 we describe both the spatial (DG) and the temporal (Runge-Kutta) discretization we employ, as well as some standard artificial viscosity models, which are needed to assemble the training set and evaluate the performances of the proposed technique. After an overview of the main tools needed to build an ANN, in Chapter 3 we present the construction of the MLP-based artificial viscosity predictor. A few implementation aspects are discussed in Chapter 4, focusing mainly on some ANN-related algorithms. In Chapter 5 we present several numerical results to show the capabilities of the proposed model, along with a comparison with the classical artificial viscosity models. After a comment on computational performances, we make a few concluding remarks in Chapter 6.

Chapter 2

Mathematical framework

2.1 Numerical discretization

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and $T > 0$ be a fixed time instant. Consider a conservation law expressed by the following formulation:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{f} - \nabla \cdot \boldsymbol{g} = \boldsymbol{0}, \tag{2.1}$$

in $\Omega \times [0, T]$. The spatial coordinate will be denoted by $\boldsymbol{x} = (x_1, \dots, x_d)$. If a single dimension is involved, we simply set $x = x_1$, while in the two-dimensional case we often write $(x, y) = (x_1, x_2)$. In (2.1), $\boldsymbol{u} \in \mathbb{R}^{n\times 1}$ denotes the vector consisting of n conserved variables, while $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{u}) \in \mathbb{R}^{n\times d}$ is the convective flux and $\boldsymbol{g} \in \mathbb{R}^{n\times d}$ is the artificial viscous flux. Throughout this work, the latter takes the following form:

$$\boldsymbol{g} = \mu \boldsymbol{q}, \quad \boldsymbol{q} = \nabla \boldsymbol{u},$$

where μ denotes the artificial viscosity coefficient. Here, *artificial* means that no physical viscosity is present and, in the continuous limit, μ vanishes and the standard form of a first-order conservation law is recovered. Note that the convective flux can be decomposed in its d components as

$$\boldsymbol{f} = \sum_{i=1}^{d} \boldsymbol{f}^{x_i} \boldsymbol{e}_{x_i},$$

where $\boldsymbol{e}_{x_i} \in \mathbb{R}^{1\times d}$ are the unit (row) vectors of the canonical basis in $\mathbb{R}^d$, and the components $\boldsymbol{f}^{x_i}$ belong to $\mathbb{R}^{n\times 1}$. A similar decomposition holds for the viscous flux.
Problem (2.1) is completed by suitable boundary conditions on $\partial\Omega \times [0, T]$ and an initial condition $\mathbf{u}(\boldsymbol{x}, t = 0) = \mathbf{u}_0(\boldsymbol{x})$ in Ω.
For the sake of simplicity, we provide a detailed description of the finite-dimensional discretization in the one-dimensional scalar case, i.e. with $d = 1$ and $n = 1$. We describe the extension to the two-dimensional case, as well to systems of equations, in the dedicated Subsections.

2.1.1 Spatial discretization: definitions

The spatial discretization is performed using a Discontinuous Galerkin (DG) scheme. In the one-dimensional case, Ω is simply an interval, say $[a, b]$. Let us subdivide the domain into K elements, namely intervals, denoted by $D^k = [x_l^k, x_r^k]$. In particular, we have $x_l^1 = a$ and $x_r^K = b$. We also denote by $h^k = x_r^k - x_l^k$ the length of each interval. Moreover, let $m \geq 1$ be an integer representing the discretization degree, which we assume to be fixed a priori and constant in all the elements. In each interval we consider the finite-dimensional space made of polynomials up to degree m, defined as

$$V_h^k := P^m(D^K) = \left\{ p = p(x) : p = \sum_{i=0}^{m} \alpha_i x^i, \ x \in D^k \right\},$$

and we set the global space as

$$V_h \coloneqq \left\{ v \in L^2(\Omega) : v|_{D^k} \in V_h^k \right\}.$$

Note that we omit the dependence on the degree m in order to simplify the notation. In the spirit of standard Discontinuous Galerkin schemes, the solution is approximated by a piecewise polynomial function u_h, defined as the direct sum

$$u_h = \bigoplus_{k=1}^{K} u_h^k,$$

with $u_h^k \in V_h^k$ for all $k \in 1, \dots, K$. The dimension of V_h^k is given by $\binom{m+d}{m} = m+1$ and it is denoted by N or N_m, in case the dependence on m has to be stressed.
As in classical finite elements discretizations, it is helpful to introduce the idea of reference element. In particular, given any mesh interval, there exists an affine mapping Ψ between a suitably defined reference element, denoted by I, and the physical one. For the one-dimensional framework, we consider $I = [-1, 1]$. Thus, we can map any point r in I to a point x of a physical interval D^k as follows:

$$x = \Psi(r) = -\frac{r-1}{2} x_l^k + \frac{r+1}{2} x_r^k = x_l^k + \frac{r+1}{2} h^k.$$

Using this mapping, we are able to assemble all the building blocks of the scheme in the reference element. We consider two different possibilities of constructing a basis for the space of polynomials over I:

1. *Nodal basis*
   The basis is defined through Lagrange polynomials, denoted by $\{l_i\}_{i=0}^m$. Letting $\{r_i\}_{i=0}^m$ be a set of disjoint points in I, we define the basis as

   $$l_i(r) = \prod_{\substack{j=0 \\ j \neq i}}^{m} \frac{r - r_j}{r_i - r_j}, \tag{2.2}$$

   Such polynomials satisfy the following interpolation property:

   $$l_i(r_j) = \delta_{ij}. \tag{2.3}$$

   A good choice for $\{r_i\}$ is given by the Gauss-Legendre-Lobatto quadrature nodes [28], which include the extrema of each interval in the set of points.
   Thus, we can represent a polynomial function on I as linear combination of the basis functions, namely

   $$u_h(r) = \sum_{j=0}^{m} u_j l_j(r). \tag{2.4}$$

   Due to property (2.3), the coefficients u_j are the nodal values of the solution at the quadrature points, i.e. $u_j = u_h(r_j)$.

2. *Modal basis*
   An alternative strategy is to consider an orthonormal basis for the polynomial space. In the one-dimensional case, a good choice is provided by the Legendre polynomials, which can be introduced as a special case of the Jacobi polynomials [2, 28]. The latter have multiple equivalent definitions, one of them being the recursive relation

   $$r P_n^{(\alpha,\beta)}(r) = a_n P_{n-1}^{(\alpha,\beta)}(r) + b_n P_n^{(\alpha,\beta)}(r) + a_{n+1} P_{n+1}^{(\alpha,\beta)}(r), \qquad n \geq 1,$$

   where a_n, b_n are coefficients depending on α, β, n defined as

   $$a_n = \frac{2}{2n+\alpha+\beta} \sqrt{\frac{n(n+\alpha+\beta)(n+\alpha)(n+\beta)}{(2n+\alpha+\beta-1)(2n+\alpha+\beta+1)}},$$

$$b_n = -\frac{\alpha^2 - \beta^2}{(2n+\alpha+\beta)(2n+\alpha+\beta+2)}.$$

The initial values $P_0^{(\alpha,\beta)}$ and $P_1^{(\alpha,\beta)}$ are equal to

$$P_0^{(\alpha,\beta)}(r) = \sqrt{2^{-\alpha-\beta-1}\frac{\Gamma(\alpha+\beta+2)}{\Gamma(\alpha+1)\Gamma(\beta+1)}},$$

$$P_1^{(\alpha,\beta)}(r) = \frac{1}{2}P_0^{(\alpha,\beta)}(r)\sqrt{\frac{\alpha+\beta+3}{(\alpha+1)(\beta+1)}}\left((\alpha+\beta+2)x + (\alpha-\beta)\right).$$

With these definitions, the following orthonormality condition is satisfied:

$$\int_I P_i^{(\alpha,\beta)}(r)P_i^{(\alpha,\beta)}(r)(1-r)^\alpha(1+r)^\beta dr = \delta_{ij}.$$

The Legendre polynomials are simply obtained by setting $\alpha = 0$, $\beta = 0$ and are denoted by $\{\phi_i(r)\}_{i=0}^m$. This leads to an orthonormal basis with respect to the standard inner product in $L^2(I)$. Again, we write a polynomial function on I as linear combination of the basis functions, namely:

$$u_h(r,t) = \sum_{j=0}^{m} \hat{u}_j(t)\phi_j(r), \tag{2.7}$$

where each coefficient $\hat{u}_j$ can be interpreted as the $L^2(I)$-product between u_h and the j-th basis function.

We construct the corresponding bases in each physical element by composition with the geometric mapping Ψ. We adopt the same notation for the basis in the reference and physical element, unless a distinction is explicitly required. Thus, the physical nodal basis is defined as

$$l_i(x) = l_i^{phi}(x) = l_i^{ref}(\Psi^{-1}(x)) = l_i(r),$$

but the superscripts phi and ref are dropped in order to ease the notation. A similar expression holds for the modal basis.
It's worth noting that there exist a bijective linear mapping between the modal and the nodal coefficients. Indeed, it is enough to observe that

$$u_i = u_h(r_i) = \sum_{j=0}^{m} \hat{u}_j\phi_j(r_i) = \sum_{j=0}^{m} V_{ij}\hat{u}_j,$$

or in a vectorial form

$$\mathbf{u} = \mathbf{V}\hat{\mathbf{u}},$$

where we introduced the Vandermonde matrix $\mathbf{V}$ defined as $\mathbf{V}_{ij} = \phi_j(r_i)$.

2.1.2 Discontinuous Galerkin approximation

In order to perform the spatial discretization, we consider the Lagrange basis functions, leading to the so-called *nodal Discontinuous Galerkin* approximation. We refer to the expansion coefficients in equation (2.4) as the (local) degrees of freedom, since it can be easily proven that the unknown polynomial function u_h^k is uniquely determined by assigning its value at such points. We recall that our starting point is the following conservation law:

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} - \frac{\partial g}{\partial x} = 0, \tag{2.8a}$$

$$g = \mu q,$$

$$q - \frac{\partial u}{\partial x} = 0. \tag{2.8b}$$

Due to the local nature of the finite dimensional space, we can focus on a single interval, say D^k, dropping the superscript k in all the involved terms, unless explicitly required. Thus, we consider the local nodal approximation for both the solution and the fluxes, namely

$$\begin{aligned} u_h(x,t) &= \sum_{j=0}^{m} u_j(t) l_j(x), \quad u_j(t) = u_h(x_j,t), \\ f_h(x,t) &= \sum_{j=0}^{m} f_j(t) l_j(x), \quad f_j(t) = f_h(x_j,t), \\ g_h(x,t) &= \sum_{j=0}^{m} g_j(t) l_j(x), \quad g_j(t) = g_h(x_j,t), \\ q_h(x,t) &= \sum_{j=0}^{m} q_j(t) l_j(x), \quad q_j(t) = q_h(x_j,t), \end{aligned} \tag{2.9a}$$

and we define the residuals associated to equations (2.8a) and (2.8b) respectively as:

$$\begin{aligned} R_{h,1} &= \frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - \frac{\partial g_h}{\partial x}, \\ R_{h,2} &= q_h - \frac{\partial u_h}{\partial x}. \end{aligned}$$

We remark that, even though we seek u_h^k belonging to the polynomial space V_h^k, the convective flux does not necessarily belong to the same space. Thus, equation (2.9a) implies a projection of the flux function on this space, defined as the element in V_h^k minimizing the error at the nodal points. A similar reasoning holds for the (artificial) viscous term.
The Discontinuous Galerkin approximation is defined by imposing the residuals to vanish locally, i.e. within the element D^k, in a Galerkin sense. In other words, we require the residuals to belong to $(V_h^k)^\perp$, by imposing orthogonality with respect to each of the Lagrange basis functions as

$$\int_{D^k} R_{h,1}(x,t) l_i(x) dx = 0, \quad \int_{D^k} R_{h,2}(x,t) l_i(x) dx = 0, \quad \forall i = 0, \dots, m = N-1 \quad \forall t \in [0,T].$$

Integrating both equations by parts, we obtain the Discontinuous Galerkin (weak) formulation:

$$\int_{D^k} \left(\frac{\partial u_h}{\partial t} l_i - f_h \frac{\partial l_i}{\partial x} + g_h \frac{\partial l_i}{\partial x} \right) + \int_{\partial D^k} (f_h^* \; n \; l_i - g_h^* \; n \; l_i) \; = 0, \tag{2.11a}$$

$$\int_{D^k} \left(q_h l_i + u_h \frac{\partial l_i}{\partial x} \right) - \int_{\partial D^k} u_h^* \; n \; l_i \; = 0, \tag{2.11b}$$

which is completed by

$$g_h = \mu \; q_h.$$

Note that in (2.11a) and (2.11b) n denotes the outward unit vector in element D^k. In the one-dimensional case it is simply equal to 1 (resp. -1), depending whether we consider the right (resp. left) boundary. In principle, such notation is redundant for the one-dimensional scenario, since one could simply write

$$\int_{\partial D^k} v'(x) dx = v(x_r^k) - v(x_l^k),$$

for any smooth enough function $v(x)$, without the need to introduce the vector n. However, this formulation is more compact and makes the extension to higher dimensions easier.
Moreover, the flux functions evaluated at the boundary are replaced by suitable numerical fluxes. This is mandatory in Discontinuous Galerkin (and finite volumes) schemes, in order to have a uniquely defined flux at the interfaces. Practically, such terms are the only ones which link the neighboring elements, since the internal contributions are, by definition, local. The choice of these fluxes plays a key role in terms of, e.g. consistency, stability and accuracy of the numerical scheme. In general, they depend on the solution in both the elements sharing the same boundary. To properly define them, consider two neighboring elements, denoted by D^+ and D^-, having normal vectors n^+ and

n^- respectively. Suppose that such elements share a boundary e. Given a generic function v, whose restriction on $e \in D^+, D^-$ is denoted by v^+, v^- respectively, we define the average and jump operators across e as follows:

$$\{v\} \coloneqq \frac{1}{2}\left(v^+ + v^-\right), \quad [v] \coloneqq v^+ n^+ + v^- n^-.$$

Then, we adopt the following numerical fluxes for the solution and viscous flux:

$$u_h^* = u_h^*(u_h^+, u_h^-) = \{u_h\}, \quad g_h^* = g_h^*(g_h^+, g_h^-) = \{g_h\},$$

which are known as *centered* fluxes. More challenging is the choice for the convective flux, since f_h can have a nonlinear dependence on the degrees of freedom. A good choice for the corresponding numerical flux is the following:

$$f_h^* = f_h^*(u_h^+, u_h^-) = \{f(u_h)\} + \frac{\Lambda}{2}[u_h], \tag{2.12}$$

where Λ is a large enough stabilization parameter, usually defined as $\Lambda \coloneqq \max |f'(u_h)|$. The maximum is either taken over the whole domain Ω or locally within the elements sharing the considered face. The first choice leads to the so-called *Lax-Friedrichs* flux, while the second one is generally known as *Local Lax-Friedrichs* or *Rusanov* flux. Being less dissipative, we adopt the latter choice. An interpretation of the two terms on the right-hand-side of equation (2.12) is the following. The first one, which is the same as the centered flux, guarantees the accuracy of the scheme, while the second term is responsible of stability, penalizing the jumps of the solution. We also point out that, in general, f and f^*, as well as their discrete counterparts, may depend on space and time.

2.1.3 Algebraic formulation

Now, we define a few algebraic operators with the aim of clarifying how to practically compute the terms required by a DG discretization.
Firstly, we point out that the operator definition is independent of time, and most of the computations can be done in the reference element. This is a huge performance advantage, since their assembly is performed once in the preprocessing step. Secondly, we observe that such operators are defined locally in each physical element, as the Discontinuous Galerkin formulation is, by definition, local. Thirdly, a key role is played by the geometric mapping Ψ or, more precisely, by its jacobian matrix J, its determinant and its inverse. Since Ψ is affine, these quantities are constant within D^k and in one spatial dimension they are scalar values equal to $\frac{h^k}{2}$, $\frac{h^k}{2}$ and $\frac{2}{h^k}$ respectively.
The key operators we employ are the following:

- Internal mass matrix:

$$\boldsymbol{M}_{ij} = \int_{D^k} l_j(x) l_i(x)\, dx = |\det J| \int_I l_j(r) l_i(r)\, dr = \frac{h_k}{2} \int_I l_j(r) l_i(r)\, dr, \tag{2.13}$$

  where $i, j = 0 \dots N-1$.

- Advection matrix:

$$\boldsymbol{S}_{ij} = \int_{D^k} l_i(x) \frac{d}{dx}(l_j(x))\, dx = |\det J| \int_I l_i(r) J^{-1} \frac{d}{dr}(l_j(r))\, dr = \int_I l_i(r) \frac{d}{dr}(l_j(r))\, dr,$$

  where $i, j = 0 \dots N-1$.

- Boundary mass matrix:

$$\boldsymbol{M}^{\sigma}_{e,ij} = \int_{\sigma_e} l_j(x) l_i(x)\, dx = |\det J^{bd}| \int_{I_e} l_j(r) l_i(r)\, dr = \left(\delta_{i1}\delta_{j1}\delta_{e1} + \delta_{iN}\delta_{jN}\delta_{e2}\right),$$

  where $i, j = 0 \dots N-1$, while $e \in \{1, 2\}$ denotes the left (resp. right) boundary.

We refer to [2] for a detailed description of such matrices and their practical assembly. Now, we can write the following algebraic semi-discrete form:

$$\boldsymbol{M}\frac{d\boldsymbol{u}}{dt} - \boldsymbol{S}^T\boldsymbol{f} + \boldsymbol{S}^T\boldsymbol{g} + \sum_{e=1}^{2}\boldsymbol{M}_e^{\sigma}\boldsymbol{f}^* n_e - \sum_{e=1}^{2}\boldsymbol{M}_e^{\sigma}\boldsymbol{g}^* n_e = \boldsymbol{0}, \tag{2.14a}$$

$$\boldsymbol{M}\boldsymbol{q} + \boldsymbol{S}^T\boldsymbol{u} - \sum_{e=1}^{2}\boldsymbol{M}_e^{\sigma}\boldsymbol{u}^* n_e = \boldsymbol{0}, \tag{2.14b}$$

where we introduced the finite-dimensional vectors $\boldsymbol{u},\ \boldsymbol{f},\ \boldsymbol{g},\ \boldsymbol{q}$ collecting the (local) degrees of freedom, namely the nodal values, of $u_h,\ f_h,\ g_h,\ q_h$ respectively. The system is completed by the equation

$$\boldsymbol{g} = \mu\boldsymbol{q} \quad (\text{or} \quad \boldsymbol{g} = \boldsymbol{\mu}\odot\boldsymbol{q}),$$

which identifies the point-wise multiplication between the nodal values of μ, collected in the vector $\boldsymbol{\mu}$, and the degrees of freedom of $\boldsymbol{q}$. We recall that the symbol $\odot$ denotes the Hadamard product, defined as $(\boldsymbol{a}\odot\boldsymbol{b})_i = \boldsymbol{a}_i\boldsymbol{b}_i$ for two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.
Moreover, due to our choice of the flux u_h^* we observe that, knowing the values of u_h, we can easily compute q_h in equation (2.11b) and inject it into (2.11a). Indeed, by definition u_h^* does not depend on q_h.
Collecting the local contributions from all the elements, we can formally assemble a global algebraic formulation:

$$\frac{d\boldsymbol{u}}{dt} = \mathcal{A}(\boldsymbol{u},\boldsymbol{f},\boldsymbol{g}), \tag{2.15a}$$

$$\boldsymbol{g} = \mu\boldsymbol{q}, \tag{2.15b}$$

$$\boldsymbol{q} = \mathcal{B}\boldsymbol{u}, \tag{2.15c}$$

by means of finite-dimensional operators $\mathcal{A}$ and $\mathcal{B}$ which already take into account the inversion of the mass matrix. Note that $\mathcal{B}$ is linear. Finally, equations (2.15a), (2.15b) and (2.15c) can be cast a single equality as

$$\frac{d\boldsymbol{u}}{dt} = \mathcal{A}(\boldsymbol{u},\boldsymbol{f}(\boldsymbol{u}),\mu\mathcal{B}\boldsymbol{u}) = \mathcal{F}(t,\boldsymbol{u}), \tag{2.16}$$

which is the standard form of an ordinary differential equation.

2.1.4 Time discretization

The final step is the choice of time integrating scheme, leading to the definition of the fully discretized version of the problem. In principle, any (stable) method could be employed. However, we do not want the time discretization error to have a (large) impact on the global solution accuracy. Therefore, a high-order time integrator should be implemented. In the context of a DG discretization, a standard choice is the class of (explicit) Runge-Kutta schemes. Here we focus on the so-called *five-stage fourth-order low-storage Runge-Kutta scheme*.
We briefly recall its main aspects, referring to [29] for more details. The algorithm originates from a modification of standard Runge-Kutta schemes, designed in order to implement the method employing two storages (i.e. vectors) only, denoted by $U,\ V$. Considering a suitable time step Δt and given the numerical solution $u_h(t^n)$ at time t^n, the scheme computes $u_h(t^n+\Delta t)$ as follows:

$$U \leftarrow u_h(t^n), \tag{2.17a}$$

$$V \leftarrow A_j U + \Delta t\ \mathcal{F}(t^n + c_j\Delta t, U), \quad \forall j = 1\dots M,$$

$$U \leftarrow U + B_j V, \quad \forall j = 1\dots M,$$

$$u_h(t^n+\Delta t) \leftarrow U, \tag{2.17b}$$

for some given coefficients A_j, B_j, c_j, while M is the number of stages. Note that the coefficients c_j are the same values appearing in the standard Butcher's tableau [28]. In order to have a fourth order scheme, we need to choose five stages, i.e. $M = 5$. Moreover, the algorithm is both self-starting and

explicit, since both A_1 and c_1 are equal to zero. The unique initialization of the storages has to be performed at the first step by means of the initial condition, setting $U = u_0$ and $V = 0$. In other words, there is no need to re-initialize the storages at each time iteration. Thus, equations (2.17a) and (2.17b) are, at a computational level, redundant.
The coefficients A_j, B_j, c_j we employ are the ones denoted as 'Solution 3' in [29], whose rational form is reported in Table 2.1.

	A_j	B_j	c_j
$j=1$	0	$\frac{1432997174477}{9575080441755}$	0
$j=2$	$-\frac{567301805773}{1357537059087}$	$\frac{5161836677717}{13612068292357}$	$\frac{1432997174477}{9575080441755}$
$j=3$	$-\frac{2404267990393}{2016746695238}$	$\frac{1720146321549}{2090206949498}$	$\frac{2526269341429}{6820363962896}$
$j=4$	$-\frac{3550918686646}{2091501179385}$	$\frac{3134564353537}{4481467310338}$	$\frac{2006345519317}{3224310063776}$
$j=5$	$-\frac{1275806237668}{842570457699}$	$\frac{2277821191437}{14882151754819}$	$\frac{2802321613138}{2924317926251}$

TABLE 2.1: Coefficients used for the time integration scheme.

Since the time-marching algorithm is explicit, an upper bound on the time step is required to make the scheme stable. Such stability condition has to take into account both the physical and the viscous fluxes, and it can be viewed as an extension of the standard Courant-Friedrichs-Levy (CFL) condition. Therefore, we require

$$\Delta t = C \min \left\{ \frac{1}{|f'(u_h)|\frac{m^2}{h} + \mu \frac{m^4}{h^2}} \right\}, \tag{2.18}$$

where C, which is usually of order one, depends weakly on m, since the influence of the polynomial degree is already taken into account in the denominator. In this work, an *adaptive* time step is chosen. At each temporal iteration, condition (2.18) is evaluated, so that Δt is chosen according to the current solution features. Clearly, if at the n-th iteration the value of $t^n + \Delta t$ exceeds the final simulation time T, the time step is corrected setting $\Delta t = T - t^n$.

2.1.5 Boundary conditions

Imposing boundary conditions is a key step in this framework. Indeed, we need to guarantee that their application does not have an impact on the high-order accuracy of the problem. They are imposed weakly, computing fluxes between the physical boundary elements and suitably defined ghost elements. Given an element k_0 having at least one boundary point (edge), say $e \subset \partial D^{k_0}$, we want to compute the fluxes through e. Denoting with u_h^- the numerical solution restricted to e, we need to appropriately define the quantity u_h^+. A similar reasoning holds for the auxiliary variable g_h. Throughout this work, we consider three types of boundary conditions:

- Dirichlet b.c.
  Let $G = G(t)$ be a prescribed function on e. At a continuous level, we aim to impose $u = G$ on e. In this case we simply define $u_h^+ = -u_h^- + 2G(t)$. Concerning the auxiliary variable g, we impose $g_h^+ = g_h^-$. With such choices we have that $\{u_h\} = G$ and $[\![g_h]\!] = 0$.
- Neumann b.c.
  Only homogeneous Neumann, namely zero-gradient boundary conditions, are needed in this work. Thus, we define $u_h^+ = u_h^-$ and $g_h^+ = -g_h^-$, leading to $\{g_h\} = 0$ and $[\![u_h]\!] = 0$. Essentially, we are swapping the signs with respect to Dirichlet conditions.
- Periodic b.c.
  Imposing periodicity can be interpreted as the absence of physical boundaries. Let D^1 and D^K be the first and the last mesh interval. Thus, periodic conditions are imposed as $u_h|^+_{D^K} = u_h|^-_{D^1}$ and $u_h|^+_{D^1} = u_h|^-_{D^K}$. Concerning the auxiliary variable, we impose that $g_h^+ = g_h^-$ on the involved boundaries, similarly to Dirichlet conditions.

Similar definitions hold for systems of equations.

2.1.6 Extension to two-dimensional problems

The extension of the previous framework to two-dimensional problem is rather straightforward from the numerical point of view, although it is worth emphasizing the main changes.
We seek the discrete solution in the space of multivariate polynomials having degree at most m, i.e.

$$V_h^k := P^m(D^K) = \left\{ p = p(x,y) : p = \sum_{\substack{i,j=0 \\ i+j \leq m}}^{m} \alpha_{ij} x^i y^j, \ (x,y) \in D^k \right\}.$$

For the two-dimensional case, its dimension is $N = \frac{(m+1)(m+2)}{2}$.
We consider triangular meshes only, and the definition of the reference element can be extended from the one-dimensional case. In this framework, I is the triangle whose vertexes are $P_1 = (-1,-1)$, $P_2 = (1,-1)$ and $P_3 = (-1,1)$, and the affine mapping is consequently defined as

$$\boldsymbol{x} = (x,y) = \Psi(r,s) = -\frac{r+s}{2}\boldsymbol{v}_1 + \frac{r+1}{2}\boldsymbol{v}_2 + \frac{s+1}{2}\boldsymbol{v}_3,$$

where $\{\boldsymbol{v}_i\}_{i=1}^3$ denotes the set of vertices of the physical triangle. A graphical representation is provided in Figure 2.1. Different choices for the grid spacing h^k can be considered. Throughout this work, as

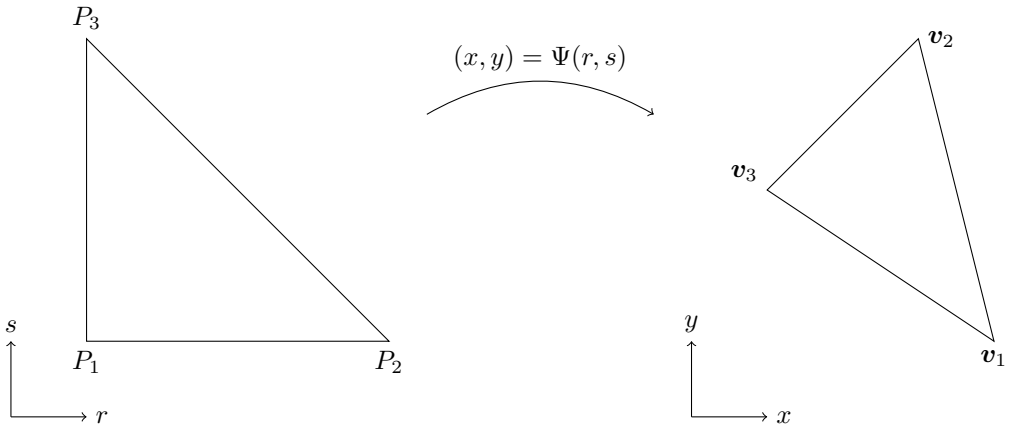


FIGURE 2.1: Graphical representation of the two-dimensional mapping Ψ.

characteristic length we adopt the diameter of the triangle, defined as the length of its longest edge. The definitions of the nodal and modal bases can be easily extended. A key role is played by the choice of quadrature points, since no trivial definition is provided for triangular elements. In [30], the authors end up with a set of quadrature points satisfying a minimum energy solution to an electrostatics problem on a reference equilateral triangle. We exploit such points, after mapping them from the equilateral triangle to I. Note that with this choice we recover the one-dimensional Gauss-Legendre-Lobatto points along each of the edges. Concerning the nodal basis, property (2.3) still holds, but there is no explicit expression similar to (2.2). On the other hand, the modal basis exploits again the one-dimensional Jacobi polynomials. In particular, the functions $\{\phi_j\}_{j=0}^{N-1}$ are defined as

$$\phi_j = \sqrt{2} Q_i^{(0,0)}(a) Q_l^{(2i+1)}(b)(1-b)^i,$$

where j is related to (i,l) as $j = l + (m+1)i - \frac{1}{2}i(i-1)$, with $(i,l) \geq 0$ and $i + l \leq m$. Here, (a,b) denote the coordinate system of the square $[-1,1]^2$, transformed from the coordinates (r,s) of the reference triangle, i.e.

$$a = 2\frac{1+r}{1-s} - 1, \quad b = s.$$

Obviously, if $s = 1$ such mapping is singular, since the top edge of the square collapses into a single point of the triangle. Again, the change of basis is represented by the Vandermonde matrix $\boldsymbol{V}_{ij} = \phi_j(r_i, s_i)$.

The Discontinuous Galerkin weak form can be written as

$$\int_{D^k}\left(\frac{\partial u_h}{\partial t}l_i - \boldsymbol{f}_h\cdot\nabla l_i + \boldsymbol{g}_h\cdot\nabla l_i\right) + \int_{\partial D^k}(\boldsymbol{f}_h^*\cdot\boldsymbol{n}\ l_i - \boldsymbol{g}_h^*\cdot\boldsymbol{n}\ l_i)\ = 0,$$
$$\int_K(\boldsymbol{q}_h l_i - u_h\nabla l_i) - \int_{\partial D^k} u_h^*\ \boldsymbol{n}\ l_i\ = 0,$$

where the inner products act on the unit vectors $\boldsymbol{e}_{x_i}$ of the canonical basis in $\mathbb{R}^2$. Defining the x and y advection matrices as

$$\begin{aligned}\boldsymbol{S}^x_{ij} &= \int_{D^k} l_i(x,y)\frac{\partial}{\partial x}(l_j(x,y))\\ &= |\det J|\left(\int_I l_i(r,s)(J^{-1})_{11}\frac{\partial}{\partial r}(l_j(r,s)) + \int_I l_i(r,s)(J^{-1})_{12}\frac{\partial}{\partial s}(l_j(r,s))\right)\\ &= |\det J|\frac{\partial r}{\partial x}\left(\int_I l_i(r,s)\frac{\partial}{\partial r}(l_j(r,s))\right) + |\det J|\frac{\partial s}{\partial x}\left(\int_I l_i(r,s)\frac{\partial}{\partial s}(l_j(r,s))\right),\end{aligned}$$

$$\begin{aligned}\boldsymbol{S}^y_{ij} &= \int_{D^k} l_i(x,y)\frac{\partial}{\partial y}(l_j(x,y))\\ &= |\det J|\left(\int_I l_i(r,s)(J^{-1})_{21}\frac{\partial}{\partial r}(l_j(r,s)) + \int_I l_i(r,s)(J^{-1})_{22}\frac{\partial}{\partial s}(l_j(r,s))\right)\\ &= |\det J|\frac{\partial r}{\partial y}\left(\int_I l_i(r,s)\frac{\partial}{\partial r}(l_j(r,s))\right) + |\det J|\frac{\partial s}{\partial y}\left(\int_I l_i(r,s)\frac{\partial}{\partial s}(l_j(r,s))\right),\end{aligned}$$

the algebraic form can be cast as

$$\boldsymbol{M}\frac{d\boldsymbol{u}}{dt} - (\boldsymbol{S}^x)^T\boldsymbol{f}^x - (\boldsymbol{S}^y)^T\boldsymbol{f}^y + (\boldsymbol{S}^x)^T\boldsymbol{g}^x + (\boldsymbol{S}^y)^T\boldsymbol{g}^y + \sum_{e=1}^{3}\boldsymbol{M}_e^\sigma\boldsymbol{f}^*\cdot\boldsymbol{n}_e - \sum_{e=1}^{3}\boldsymbol{M}_e^\sigma\boldsymbol{g}^*\cdot\boldsymbol{n}_e = \boldsymbol{0},$$

$$\boldsymbol{M}\boldsymbol{q_x} + (\boldsymbol{S}^x)^T\boldsymbol{u} - \sum_{e=1}^{3}\boldsymbol{M}_e^\sigma\boldsymbol{u}^* n_e^x = \boldsymbol{0}, \quad \boldsymbol{M}\boldsymbol{q_y} + (\boldsymbol{S}^y)^T\boldsymbol{u} - \sum_{e=1}^{3}\boldsymbol{M}_e^\sigma\boldsymbol{u}^* n_e^y = \boldsymbol{0},$$

where the boundary and internal mass matrices are defined in a way similar to the one-dimensional case.

2.1.7 Extension to systems of equations

So far we focused on the scalar case, i.e. when the unknown variable u is a scalar quantity. In principle, extending the previous framework to systems is quite straightforward. Roughly speaking, considering separately each equation, the same reasoning holds. Thus, the local solution $\boldsymbol{u}_h$ belongs to the space $(V_h^k)^n$.
We make a simple remark concerning the physical flux. Considering the one-dimensional case for simplicity, for scalar problems there is no ambiguity in defining $f'(u)$, which is a scalar quantity. For systems we continue to adopt the same notation, but it is worth observing that it is actually a matrix, defined as

$$[\boldsymbol{f}'(\boldsymbol{u})]_{ij} = [J_{\boldsymbol{f}}(\boldsymbol{u})]_{ij} = \frac{\partial f_i}{\partial u_j},$$

Moreover, $\max|f'(u)|$ is given by the maximum absolute eigenvalue of the jacobian matrix of the flux function and represents the maximum wave speed.

2.1.8 Test cases

In this work, we consider different conservation laws. Each problem is characterized, at a continuous level, by the choice of the conserved variables, flux function, initial and boundary conditions, and a

final time.
For one dimensional problems, we deal with:

- linear advection equation, which describes, e.g., the transport of a scalar quantity u by a field $\beta = \beta(x,t)$. The problem is then defined choosing $f(u) = \beta u$.
- Burgers equation, representing a simplified version of Navier-Stokes equations. It is defined by choosing $f(u) = \frac{u^2}{2}$.
- an equation with a degree-4 flux function, defined by $f(u) = \frac{u^4}{4}$, in order to have a non quadratic (but still convex) flux.
- Buckley-Leverett problem, which describes the water saturation in a mixture of oil and water [31]. It is defined by choosing the non-convex flux function

$$f(u) = \frac{u^2}{u^2 + 0.5(1-u)^2}.$$

- Euler system, representing the conservation of mass, momentum and energy for a compressible inviscid flow. The physical variables are density (ρ), velocity (v) and pressure (p), while the conserved variables are density (ρ), momentum (ρv) and energy (E). Thus, we define

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho v \\ E \end{pmatrix}, \quad \boldsymbol{f}(\boldsymbol{u}) = \begin{pmatrix} \rho v \\ \rho v \otimes v + p\mathbb{I} \\ v(E+p) \end{pmatrix} = \begin{pmatrix} \rho v \\ \rho v^2 + p \\ v(E+p) \end{pmatrix}.$$

  The system is closed by the ideal gas law as

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho|v|^2\right), \tag{2.22}$$

  γ being a fluid dependent constant which we take to be $\gamma = 7/5$, as it is typical for atmospheric gasses. Other important variables are the speed of sound

$$c = \sqrt{\frac{\gamma p}{\rho}},$$

  and the Mach number

$$Ma = \frac{|v|}{c}. \tag{2.23}$$

In a two dimensional framework, we focus on the following problems:

- linear advection equation, defined choosing $\boldsymbol{f}(u) = f^x \boldsymbol{e}_x + f^y \boldsymbol{e}_y = (\beta^x u, \beta^y u)$.
- a possible two-dimensional extension of Burgers equation, obtained by setting $\boldsymbol{f}(u) = f^x \boldsymbol{e}_x + f^y \boldsymbol{e}_y = (\frac{u^2}{2}, \frac{u^2}{2})$.
- The KPP rotating wave problem [18], obtained by setting non-convex x and y fluxes

$$\boldsymbol{f}(u) = f^x \boldsymbol{e}_x + f^y \boldsymbol{e}_y = (\sin(u), \cos(u)).$$

- Euler system, representing the conservation of mass, momentum and energy for a compressible inviscid flow. Compared to the one-dimensional case, we have to take into account both the x and y component of the velocity, defined as $\boldsymbol{v} = (v_x, v_y)$. Thus, we have

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ E \end{pmatrix}, \quad \boldsymbol{f}(\boldsymbol{u}) = \begin{pmatrix} \rho \boldsymbol{v} \\ \rho \boldsymbol{v} \otimes \boldsymbol{v} + p\mathbb{I} \\ \boldsymbol{v}(E+p) \end{pmatrix} = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + p \\ \rho v_x v_y \\ v_x(E+p) \end{pmatrix} \boldsymbol{e}_x + \begin{pmatrix} \rho v_y \\ \rho v_y v_x \\ \rho v_y^2 + p \\ v_y(E+p) \end{pmatrix} \boldsymbol{e}_y,$$

  and

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho|\boldsymbol{v}|^2\right) = (\gamma - 1)\left(E - \frac{1}{2}\rho(v_x^2 + v_y^2)\right), \tag{2.24}$$

while the definitions of the speed of sound and the Much number remain the same.

2.2 Artificial viscosity models

In this Section we present the artificial viscosity models adopted in this work. Such models will play a central role in the construction of the training samples, and they will represent the benchmark cases in order to evaluate the performances of the new technique based on artificial neural networks. Unless explicitly required, in this Section we drop both the subscript h and the superscript k to denote the finite-dimensional local variables. However, we note that these models are defined at a discrete level only and they employ local variables to compute the viscosity.

2.2.1 Overview

Before going into the details of each model, we remark that the main aspect to be taken into account in adding the artificial viscosity is the spatial *locality*, which results in a different amount of dissipation in each mesh element. As a consequence, the viscosity profile is globally *nonlinear*. More precisely, one wants to inject dissipation only close to discontinuities, while if the solution is regular enough the inviscid scheme should not be altered. A related aspect is the choice of an *optimal* viscosity amount. Indeed, large values would result in over-dissipative results, and significant features of the solution could not be captured. Vice versa, the artificial viscosity has to be large enough to smoothen spurious numerical oscillations.
The strategy followed by all the shock-capturing models is similar and can be summed up as follows:

- In each cell of the domain
  - Estimate a maximum amount of viscosity by choosing a characteristic velocity and length. We choose these as the (maximum) local wave speed $\max_{D^k} |\boldsymbol{f}'(\boldsymbol{u})|$ and the subcell grid size $\frac{h}{m}$ respectively. Thus, we define

$$\mu_{max} = c_{max} \frac{h}{m} \max_{D^k} |\boldsymbol{f}'(\boldsymbol{u})|, \tag{2.25}$$

    where c_{max} is a problem-dependent global constant. Note that in the high-resolution limits $h \to 0$ and/or $m \to \infty$ the numerical dissipation vanishes.
  - Identify a shock sensor $\mathcal{S}$, namely a quantity which estimates the smoothness of the solution and determines the amount of artificial viscosity to be added. In general, $\mathcal{S}$ depends nonlinearly on the solution $\boldsymbol{u}$ and its choice characterizes the different models.
  - Based on the sensor, estimate a value for the viscosity, say $\mu_{\mathcal{S}}$. Depending on the model, we obtain either a pointwise or a constant value for the numerical dissipation.
  - Require μ to be not greater than μ_{max} by defining

$$\mu = \min \{\mu_{\mathcal{S}}, \mu_{max}\}.$$

- If needed, perform a global smoothing of the viscosity. This is justified by two main arguments. Firstly, it helps to further reduce the numerical oscillations. As an example, if we consider a model which gives a piecewise constant dissipation, the global viscosity profile might be discontinuous across the element boundaries. This causes additional undesirable spurious oscillations, especially for high discretization degrees and/or when the jump in the viscosity is sufficiently large. We refer to [22] for a broader discussion on this topic and a more concrete example. Secondly, the viscosity sub-cell resolution is enhanced. In other words, after the smoothing, a different value of μ is added in each node, letting the dissipation vary locally within each cell.
  In this thesis we consider a simple C^0 smoothing, performed only for the models giving a piecewise constant profile for the artificial viscosity. It is carried out in three steps:

1. Consider the set of nodal points corresponding to a given degree P. Conduct averaging with all the cells sharing the same node, in order to have a uniquely defined viscosity at the boundary points.
2. Compute the coefficients of the degree-P interpolating polynomial in each element.
3. Evaluate such a polynomial on the nodal points required by the discretization order m.

For most of practical applications, choosing $P = 1, 2$ is enough. In this work we always pick $P = 1$, except in the one-dimensional scalar case, where $P = 2$ is considered. A graphical representation of the global viscosity profile is reported in Figure 2.2 for the one-dimensional case. We recall that more sophisticated techniques can be employed to perform the smoothing as

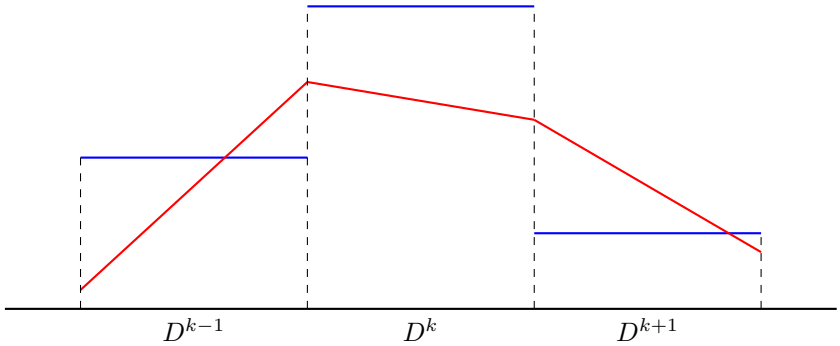


FIGURE 2.2: A graphical representation of the smoothed viscosity profile using $P = 1$ in a one-dimensional framework. The blue lines denote the piecewise constant values, while the (continuous) red line represents the final profile, after the smoothing process.

in, e.g., [22]. However, despite its simplicity, our algorithm guarantees good results, as well as a low computational cost.

A final aspect that needs to be considered is the *frequency* at which the artificial viscosity has to be estimated. Indeed, in theory the viscosity might change every time each Runge-Kutta substep updates the solution vector U. Since a five-stage algorithm is used, we would need to estimate μ for $5N_{iter}$ times. However, for our purposes it is enough to assume that the viscosity changes only at the beginning of the internal loop. Thus, we estimate μ for N_{iter} times only. This choice can be viewed as a good compromise between computational performances and *precision* in estimating the viscosity.

2.2.2 Derivative-based (DB) model

The simplest shock sensor is based on first-order differential operators, which identify the regions where the solution exhibits an abrupt variation. At a first glance, discontinuities (in particular shocks) can be interpreted as high jumps in the solution located in small spatial regions. Thus, derivative operators represent a good tool to estimate artificial viscosity. In a broad sense, the amount of dissipation will be proportional to the *intensity* of the variation, measured in a suitable norm. A different treatment for scalar equations and the (Euler) system is considered.
In the first scenario, we define the viscosity to be proportional, up to scaling factors, to the $\mathbb{R}^d$ euclidean norm of the gradient as:

$$\mu_\beta = c_\beta \left(\frac{h}{m}\right)^2 \|\nabla u\| . \tag{2.26}$$

Indeed, the gradient is a good indicator of rapid variations of the solution and it can be computed easily, since all the discrete derivative operators are assembled in the pre-processing phase. Practically, its implementation mainly consists of matrix-vector multiplications. As an analogy with the field of image processing, we can interpret the gradient norm as a simple edge detector [32]. In the context of conservation laws, edges are essentially the discontinuities in the solution.
Although it is a rather simple model, its main drawback lies in the fact that μ is at most of order 2. Indeed, in equation (2.26) a second power of the grid spacing h is present. For this reason, the

accuracy of the viscous Discontinuous Galerkin scheme is limited to at most second order. In other words, high order accuracy cannot be achieved, limiting the applicability of this viscosity model, at least for scalar problems.
In the second scenario, a possible strategy consists in repeating the same reasoning by choosing u as a suitable scalar variable. However, restricting our attention to Euler system, a better indicator is built using the divergence of the velocity field:

$$\mu_\beta = c_\beta \left(\frac{h}{m}\right)^2 |\nabla \cdot \boldsymbol{v}| \,. \tag{2.27}$$

It achieves large absolute values in regions exhibiting strong compression or expansion. In particular, the former is a good indicator of the presence of shocks. A similar indicator is built in [15], where the authors propose to further add a sigmoid function in order to take into account compression only. However, in this work we employ the standard formulation defined in (2.27). Note that this model, and in particular equation (2.27), is a simplified version of the artificial bulk model presented in [13]. For systems, the second-order accuracy issue is still present. However, for solutions exhibiting low compressibility, higher orders can be achieved. We refer to Chapter 5 for more concrete examples of such a behavior.
The final value of the artificial viscosity is determined as

$$\mu_{DB} = \min\left\{\mu_\beta, \mu_{max}\right\}.$$

For both scalar equations and systems, we end up with a pointwise value for μ_{DB}. Thus, within each element, we already have an in-built sub-cell resolution. In order to better demonstrate the sub-cell nature of the model, we do not perform the C^0 smoothing. As shown in Chapter 5, this choice does not have a detrimental effect on the results, which are mainly affected by the second order accuracy of the method. Note that the same choice was made in [14], at least for one-dimensional problems.

2.2.3 Highest modal decay (MDH) model

This model, presented in [16], identifies the decay of the modal expansion coefficients as a good indicator for discontinuous solutions. We describe it first in the one dimensional scalar case. The starting point is the following result on Fourier series [33]:
Theorem 1. *If $f(x)$ is a continuous, T-periodic function on $[-T/2, T/2]$ with a piecewise continuous first derivative, then the complex Fourier coefficients $\left\{\hat{f}_k\right\}_{k\in\mathbb{Z}}$ of $f(x)$ satisfy*

$$|\hat{f}_k| \leq \frac{C}{k^2},$$

and the sequence of partial Fourier sums

$$f_n(x) = \sum_{|k|\leq n} \hat{f}_k \; e^{\frac{2\pi}{T} ikx},$$

converges uniformly to $f(x)$.
More generally, if $f(x) \in C^n([-T/2, T/2])$ $(n \geq 0)$ with a piecewise $(n+1)$-th derivative, the coefficients $\left\{\hat{f}_k\right\}_{k\in\mathbb{Z}}$ satisfy

$$|\hat{f}_k| \leq \frac{C}{k^{n+2}}.$$

Now we rely on the modal expansion of the solution (see (2.7)) and we define the truncated representation $\tilde{u}$ as the expansion containing only the first $N_{m-1} = m$ terms. Comparing them, we have

$$u = \sum_{j=0}^{m} \hat{u}_j \phi_j, \quad \tilde{u} = \sum_{j=0}^{m-1} \hat{u}_j \phi_j,$$

and we define the shock sensor as the fraction of energy of u contained in the highest mode, namely

$$S_k = \frac{(u-\tilde{u}, u-\tilde{u})^2_{L^2(D^k)}}{(u,u)^2_{L^2(D^k)}} = \frac{\|u-\tilde{u}\|^2_{L^2(D^k)}}{\|u\|^2_{L^2(D^k)}} = \frac{|\hat{u}|^2_m}{\sum_{j=0}^{m}|\hat{u}|^2_j}, \tag{2.28}$$

where the last equality holds thanks to the orthonormality of the Legendre polynomials. We now assume that the modal coefficients satisfy a result which is analogous to the Fourier ones, explained in Theorem 1. Therefore, for continuous solutions, we expect the indicator S_k to behave as $1/m^4$, i.e. $S_k \simeq C/m^4$. In a logarithmic scale, such a requirement is cast as $s_k = \log S_k \simeq -c_A - 4\log m = s_0$. Then, if the solution is more regular, a higher decay rate is expected, and a smaller amount of viscosity needs to be added. Vice versa, if the solution coefficients exhibit low decay, discontinuities are present and smoothing is necessary.
Thus, according to the previous reasoning, we should add either zero dissipation in case $s_k < s_0$ or a prescribed maximum viscosity in the opposite case. However, a suitable set of empirical parameters is introduced in order to make the jump less abrupt, so that the final viscosity introduced by this model is set as follows:

$$\mu_{MDH} = \mu_{max} \begin{cases} 0 & \text{if } s_k < s_0 - c_\kappa, \\ \frac{1}{2}\sin\left(1 + \frac{\pi(s_k - s_0)}{2c_\kappa}\right) & \text{if } s_0 - c_\kappa \le s_k < s_0 + c_\kappa, \\ 1 & \text{if } s_0 + c_\kappa \le s_k, \end{cases} \tag{2.29}$$

where μ_{max} is defined in equation (2.25), and c_A, c_κ, together with c_{max}, are problem-dependent parameters. As a side remark, we observe that the name of the model originates from the fact that S_k relies only on the behavior of the highest modal coefficient.
The extension to Euler system is carried out by applying the previous framework using a *representative* variable of the problem. This could be again an empirical choice, and in [14, 16] the authors rely on density. A justification of this choice is that ρ is discontinuous across both contact and shock waves, unlike velocity or pressure.
Finally, the previous reasoning can be ported to two dimensional problems. In this scenario, the numerator in (2.28) is made of all the expansion coefficients of degree-m polynomials, but the analogy with Fourier series still holds, observing that

$$S_k = \frac{\sum_{j=N_{m-1}}^{N_m-1}|\hat{u}|^2_j}{\sum_{j=0}^{N_m-1}|\hat{u}|^2_j} \sim \left(\frac{2}{m^4} + \sum_{\substack{k_1,k_2=1 \\ k_1+k_2=m}}^{m-1} \frac{1}{k_1^4 k_2^4}\right),$$

at least for separable functions. An alternative scaling argument, more similar to the one-dimensional setting, is

$$S_k \sim \frac{1}{m^4}.$$

Both the approaches give similar results, so that we adopt the latter, which still depends on a few empirical parameters.
We remark that the model estimates a constant viscosity value μ_{MDH} in each physical element, resulting in a piecewise constant profile. Thus, we perform the C^0 global smoothing.

2.2.4 Averaged modal decay (MDA) model

This model, which can be interpreted as an improved version of the MDH, is proposed in [17]. We focus again on the one-dimensional case first. Considering the analogy with Fourier modes, we assume that the coefficients of the modal expansion decay as $|\hat{u}_j| \simeq Cj^{-\tau}$, converted in a logarithmic scales as

$$\log|\hat{u}_j| \simeq \log C - \tau \log j, \quad j = 1, \dots, N_m - 1 = m,$$

where both C and τ need to be estimated. A simple strategy consists in determining the best parameters in a least-squared sense by solving the following minimization problem:

$$\min_{C,\tau} \sum_{j=1}^{m} \left(\log|\check{u}_j| - (\log C - \tau \log j)\right)^2, \tag{2.30}$$

where $|\check{u}_j|$ is a suitable modification of the modal coefficients $|\hat{u}_j|$, since setting $\check{u}_j = \hat{u}_j$ might lead to an overestimation of the decay rate τ [17]. The modified modal coefficients can be obtained following a two-step process:

- First step: add the *sense of scale* to the model. The objective function (2.30) ignores the scale of the function, which is taken into account by the first modal coefficient ($j = 0$). If we consider a scenario where a constant function is perturbed by white noise at a much smaller scale, the previous model captures only the oscillations and predicts a low decay rate, resulting in a high dissipation. Since this behavior is undesirable, we re-add the sense of scaling to the model by considering the modified coefficients defined as

$$|\check{u}_j|^2 = |\hat{u}_j|^2 + \|u\|^2_{L^2(D^k)} |b_j|^2, \quad |b_j| = \frac{j^{-m}}{\sqrt{\sum_{l=1}^{m} l^{-2m}}}.$$

- Second step: *Skyline pessimization*. This procedure is needed to recover a monotone decay. Indeed, if we consider a scenario where there exist i, n such that $|\check{u}_i| \gg |\check{u}_n|$, then the smaller coefficient is likely to be spurious and should be eliminated. This motivates the following definition

$$|\check{u}_j| = \max_{i=\min(j,m-1),\dots,m} |\check{u}_i|$$

  for the coefficients appearing in (2.30).

The solution of the unconstrained minimization problem can be easily found by solving a linear system in $(\log C, \tau)^T$ arising from the first order optimality conditions. Once the decay rate is known, we define the final viscosity of the model as

$$\mu_{MDA} = \mu_{max} \begin{cases} 1 & \text{if } \tau < 1, \\ 1 - \frac{\tau - 1}{2} & \text{if } 1 \le \tau < 3, \\ 0 & \text{if } 3 \le \tau, \end{cases} \tag{2.31}$$

where μ_{max} is defined in equation (2.25). In this way, we add the maximum (resp. minimum) dissipation in presence of discontinuous (resp. C^1) solutions characterized by a decay rate $\tau = 1$ (resp. $\tau = 3$), as stated by Theorem 1.
It is worth observing that this model is designed for high discretization degrees. In particular, one can verify that the optimization problem is not well posed when $m = 1, 2$ are considered. Thus, in this work it is employed for $m \ge 3$ only.
Extension to the Euler system is again performed by applying the previous framework using density only.
In the two-dimensional scanario, a simple yet effective way to extend the previous reasoning is proposed in [14] and can be summed up in three steps:

1. Extract the nodal values for a prescribed scalar quantity (e.g. the discrete solution in case of scalar problem) on each edge of the triangular element.
2. Apply the one-dimensional reasoning on each edge, estimating three decay rates τ_e ($e = 1, 2, 3$) for the three edges.
3. Find the minimum decay rate $\tau = \min_e \tau_e$ and estimate the viscosity using (2.31).

Again, since this model predicts a piecewise constant viscosity, we need to apply the smoothing technique.

2.2.5 Entropy viscosity (EV) model

The last model we consider, presented in [18, 19], is based on the entropy behavior. Let $(E(\boldsymbol{u}), \boldsymbol{F}(\boldsymbol{u}))$ be an entropy pair for the inviscid continuous problem. It satisfies the following *entropy inequality*

$$\frac{\partial E}{\partial t} + \nabla \cdot \boldsymbol{F} \leq 0,$$

for a physically relevant weak solution of the conservation law [34]. Moreover, for smooth solutions, the equality holds. Therefore the entropy residual is a good shock sensor, since viscosity can be added in a way proportional to the entropy production. In [19], the authors note that the entropy residual is a reliable and robust choice, in the sense that in the continuous limit $h \to 0$, it converges to δ-distributions supported in the shocks.
More precisely, we define

$$\mu_E = c_E \left(\frac{h}{m}\right)^2 B, \tag{2.32}$$

where c_E is an empirical parameter and B is defined as

$$B = \frac{1}{A} \max \left(\max_{D^k} |R(\boldsymbol{u})|, \max_{\partial D^k} |H(\boldsymbol{u})| \right),$$

and consists of two contributions:

$$R = \frac{\partial E}{\partial t} + \nabla \cdot \boldsymbol{F} \simeq \frac{E(u^n) + E(u^{n-1})}{\Delta t} + \frac{\nabla \cdot F(u^n) + \nabla \cdot F(u^{n-1})}{2}, \tag{2.33a}$$

$$H = \left(\frac{h}{m}\right)^{-1} [\![\boldsymbol{F}]\!] \cdot \boldsymbol{n}. \tag{2.33b}$$

The former takes into account the entropy residual and the time derivative is discretized using a Crank-Nicolson scheme. Since the right-hand-side of equation (2.33a) is based on the solution at the current and the previous time step, the residual R can be computed explicitly. Following [19], R is set to zero for the first temporal iteration. The latter introduces the effect of the jump of the entropy flux along the element boundary. Finally, the scalar A acts as a normalization factor, which restores the correct physical dimensions for the viscosity. Unless specified otherwise, it is set as

$$A = \max_{\Omega} \left| E - \frac{1}{|\Omega|} \int_{\Omega} E d\Omega \right|.$$

The final viscosity is obtained as

$$\mu_{EV} = \min \{\mu_E, \mu_{max}\} \tag{2.34}$$

and smoothing the result.
As observed in [19], the (entropy) viscosity μ_E scales as $\mu_E \sim h^2(\Delta t^2 + h^m)$. Indeed, the Crank-Nicolson scheme is a second-order algorithm, while the accuracy of the divergence of the entropy flux depends on the discretization degree. The first factor h^2 comes from definition (2.32). As shown in Chapter 5, this might limit the accuracy of the scheme to fourth order, which is anyway better compared to the DB scheme. This issue could be overcome by using a higher order discretization for the above-mentioned time derivative.
The last remark is devoted to the choice of the entropy pair. For scalar cases we select

$$E = \frac{u^2}{2}, \quad F_i = \int f_i'(v) E'(v) dv \quad (i = 1, \ldots, d),$$

while for Euler system we consider

$$E = -\frac{\rho}{\gamma - 1} \log \left(\frac{p}{\rho^\gamma}\right), \quad \boldsymbol{F} = \boldsymbol{v} E.$$

Chapter 3

Artificial neural networks

3.1 Background

In this Chapter we present the building blocks of our technique. We start by emphasizing the key features that have to be taken into account and the motivation behind the use of artificial neural networks. More precisely we aim to:

- estimate a relation exhibiting *high degrees of complexity and nonlinearity*. We want to approximate the (optimal) local viscosity μ, whose exact dependence on local features cannot be explicitly written. In particular, there is no practical rule to estimate the optimal amount of viscosity, i.e. there is no clear one-to-one relation between the solution and the dissipation value.
- find a *non-parametric* model. One of our main goals is to eliminate the dependence on the empirical parameters whose tuning might represent a bottleneck in the application of the standard viscosity models. By definition, neural networks involve parameters, but they are tuned a priori and their value is not changed when applying the technique.
- have a computationally *simple and fast* model. Since the estimation of the artificial viscosity has to be carried out at least once per time step, it is desirable to develop a technique that is both accurate and computationally inexpensive.
- construct a *universal method*. In principle, we want to build a 'black box' which is problem-independent, aiming to apply the same model for different conservation laws.

The artificial neural networks are able to fulfil most of these requirements, by shifting the complexity (and the computational cost) to an offline stage, which is done only once. On the contrary, the online phase mainly consists of low-cost matrix-vector multiplications.

3.1.1 The model

Artificial neural networks (ANNs, or more simply NNs) are computational models which are able to process information, learning from observational data. The name is inspired by their biological counterpart, usually called *biological neural networks* [35]. Naively, the latter are connected sets of a very large number of cells, called *biological neurons*, transmitting information by continuous communication. A graphical representation of such cells is provided in Figure 3.1. Each neuron receives signals from other cells by means of the *dendrites*. Such special connections among neurons are named *synapses*. All the inputs are collected in a single signal, where each connection contributes differently to the aggregate. In other words, the neural cell receives strong or weak *stimuli*. Thus, the synapse can be interpreted as a weighted connection among neurons. Once the accumulated signal exceeds a certain threshold, the neuron sends a pulse from its *nucleus* via the *axon* to other receptors. By continuously repeating the same process, impulses are transmitted throughout the body. A neuron gains *knowledge* by a *training* process, in which the synaptic connections are created or modified according to the different environmental situations they are subjected to.
The structure of an artificial network is built mimicking the biological one and it can be described by the triplet $(\mathcal{N}, \mathcal{V}, \mathcal{W})$. Here, $\mathcal{N}$ denotes the set of all artificial neurons in the network, while $\mathcal{V}$ is the set of all directed connections (i, j), $i, j \in \mathcal{N}$. Throughout this work, i denotes the sending neuron,

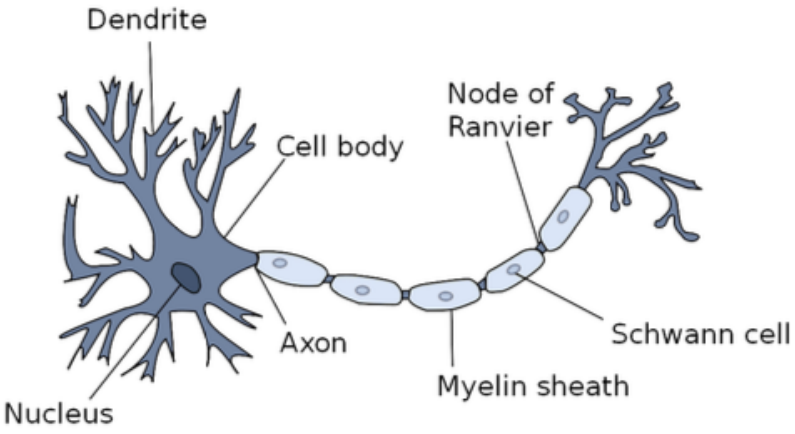


FIGURE 3.1: A simplified model of the biological neuron. The image has been taken from [35].

while j is the receiving neuron. Finally, since the connections are weighted, the set $\mathcal{W}$ is the collection of the weights, denoted by $w_{i,j}$.
The way information is processed in a single artificial neuron is rather simple, coherently with our requirements for an artificial viscosity model. To describe it, let us focus on a single unit, denoted by the subscript j. Suppose that it receives k signals $x_{s_1}, \dots, x_{s_k}$ from a set $\{s_1, \dots, s_k\}$ of sending neurons. They are processed by means of a *propagation function*, which transforms a vectorial input into a scalar one, usually named *net(work) input*, performing a combination with the corresponding weights. Since it is common to adopt a weighted linear combination of the incoming signals, in this work we adhere to this choice. Mathematically, we may thus write

$$q_j = f_{prop}(x_{s_1}, \dots, x_{s_k}, w_{s_1,j}, \dots, w_{s_k,j}) = \sum_{i=1}^{k} w_{s_i,j} x_{s_i}.$$

Then, the neuron transmits a single pulse, provided that the net input exceeds a certain threshold $-b_j$, usually named *bias*. Note that the minus sign in front of the term b_j is purely conventional. More precisely, we can define an *activation function* such that

$$a_j = f_{act}(q_j, b_j) = f_{act}(q_j + b_j),$$

where a_j is usually named *activation state*. Different choices are available for f_{act}, the most popular being the Heaviside function or the hyperbolic tangent. We refer to Section 3.2 for a discussion on such a topic, explaining the choices for our model. It is worth observing that f_{act} has to be nonlinear with respect to its input, in order to avoid the model from collapsing to a linear mapping.
Finally, the resulting output is defined as a manipulation of the activation state a_j by means of an *output function*. Being the standard choice the identity function, we adopt the same convention, ending up with

$$y_j = f_{out}(a_j) = a_j.$$

Thus, no distinction will be made between the output and the activation state, unless explicitly required. A graphical representation of a neuron is reported in Figure 3.2. This mechanism is repeated in (almost) all the neurons in the net, as the output y_j is processed by some receiving neurons. Within a network there exist also two special types of units, namely the *input* (*source*) and the *output* neurons. The latter behave similarly to most of the neural units, but the corresponding output y_j is not processed. Indeed, it will constitute part of the global output of the network. The former do not process any information, and their role is to supply input signal to the network [23]. Their output y_j is equal to x_j, i.e. propagation and activation functions are the identity operators.

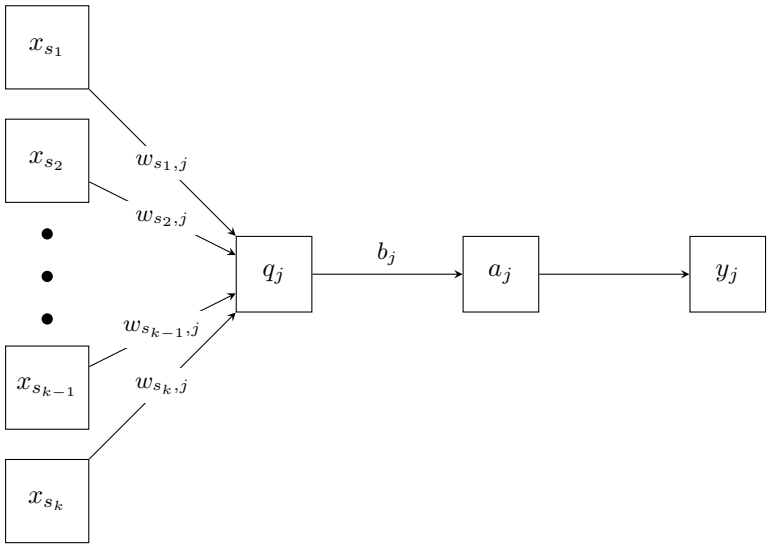


FIGURE 3.2: A graphical representation of a single neuron j receiving k input signals $\{x_{s_i}\}_{i=1}^{k}$ with output y_j.

3.1.2 Network topology

The way in which neurons are arranged and connected defines the topology of the network, namely its architecture. Several design strategies have been proposed in the literature [35, 36]. Here, we focus on *feedforward* neural networks, where the connections between the nodes do not form a cycle (i.e. no recurrence is present) and both the input and the output are fixed. Among them, *perceptrons* are the most common examples, in which the neurons are grouped in layers. The first one, made of N_I source neurons, is named *input layer*. By definition of input neurons, data are not processed in this first phase. It is followed by L *hidden layers* made of N_H^l neurons ($l = 1 \dots L$) each, which are invisible from the outside. The last *output layer* is made of N_O output neurons. Each layer can be composed of a different number of neural units, i.e. N_I, N_H^l, N_O can differ, and their choice is actually part of the architectural design. Clearly, the connections are always directed from one layer to the next one, towards the output. Therefore, perceptrons can be viewed as a map from the input to the output space, say

$$\boldsymbol{f} : \mathbb{R}^{N_I} \to \mathbb{R}^{N_O}, \quad \boldsymbol{x} \mapsto \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}). \tag{3.1}$$

Furthermore, they can be classified based on the number of hidden layers, or equivalently by the trainable weight layers. Single layer perceptrons (SLPs) have no hidden layers. The connection, which is directly from the input to the output, is made by a single layer of trainable weights. Although they constitute a simple model, they are not suited for concrete applications, since they are only capable of representing linearly separable data [35]. Vice versa, multilayer perceptrons (MLPs) allow multiple hidden layers and, in a broad sense, they are a composition of SLPs. This allows the model to have more degrees of freedom, i.e. weights and biases, compared to a simple SLP.
MLPs can be regarded as *universal function approximators*. Intuitively, since a SLP can classify linearly separable sets, a MLP with one hidden layer classifies convex polygons, being a combination of different SLPs. Arguing in a similar way, a network with at least two hidden layers can classify any set. The following theorem characterizes more rigorously the approximation properties of such class of networks [26, 27]:

Theorem 2 (Cybenko)**.** *The following statements hold:*

- *MLPs with one hidden layer and continuous and differentiable activation functions can approximate any continuous function in a compact domain.*
- *MLPs with two hidden layers and continuous and differentiable activation functions can approximate any continuous function.*

The main drawback of such a theorem lies in the fact that it does not provide any information on how many neurons should the network be made of. Thus, for practical applications, it is common to employ more than two layers. Estimating the number of hidden layers and neurons is an application-dependent

task, which is usually performed using a bottom-up approach.
A pictorial representation of a MLP with $N_I = 2$ input neurons, $L = 2$ hidden layers made of $N_H^1 = N_H^2 = 5$ neurons each, and an output layer with $N_O = 3$ neurons is provided in Figure 3.3.

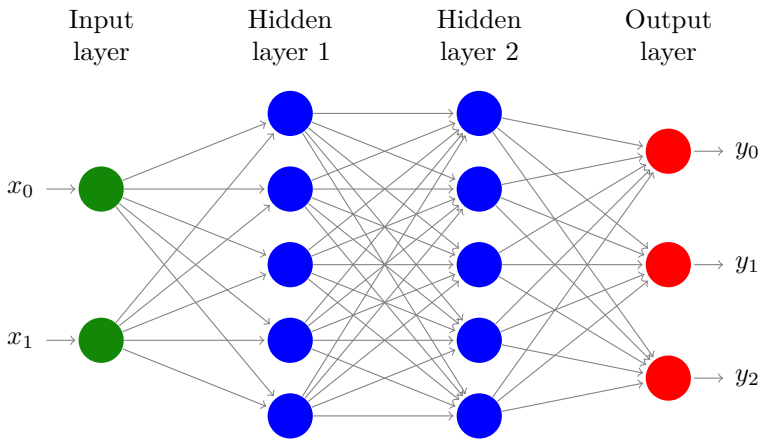


FIGURE 3.3: A graphical representation of a MLP with $N_I = 2$ input neurons, $L = 2$ hidden layers made of $N_H^1 = N_H^2 = 5$ neurons each and an output layer with $N_O = 3$ neurons.

3.1.3 Training the network

In a nutshell, the result of the network design is the mapping (3.1) from the input to the output signal, both identified as vectors. Such function depends on some parameters, named *weights* and *biases*, corresponding to the biological synaptic connection weights and thresholds. The training procedure is an algorithm, usually iterative, which tunes the network parameters in order to accurately predict responses within a given dataset, usually called *training set*. We denote it by $\mathbb{T}$, referring to the number of training samples as $N_{\mathbb{T}} = |\mathbb{T}|$. Its generation is far from being an easy task, since no theoretical results are known in order to estimate its size. Practically, on one hand it has to be sufficiently rich in order to capture the variability of the physical phenomenon of interest. In other words, the input space has to be properly sampled, possibly guaranteeing the samples to be independent or unbiased. On the other hand, collecting data might be rather expensive. Moreover, the number of training samples has a large influence on the training time, which can turn out to be too large for concrete applications.
Several training strategies are known, depending on the available data. For regression problems, i.e. when $\boldsymbol{y}$ varies continuously within its space, a good paradigm is named *supervised learning*. For this strategy, the exact output for all the input samples is known. Thus, $\mathbb{T}$ is made of both input and output data. Thus, in this framework the algorithm is designed in order to minimize the prediction error, i.e. the error between the response of the network and the exact output.
Assume that the exact input-ouput relation is described as

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}),$$

while the network maps the input values to the output ones as in equation (3.1), namely

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}; (\boldsymbol{w}, \boldsymbol{b})),$$

where in the second equality we emphasized the dependence on the trainable weights and biases. Thus, the training set is defined as

$$\mathbb{T} = \left\{ (\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i) \ : \ \hat{\boldsymbol{y}}_i = \hat{\boldsymbol{f}}(\boldsymbol{x}_i) \right\}_{i=1}^{N_{\mathbb{T}}}$$

Consider now a suitable measure of the error made when approximating $\hat{\boldsymbol{y}}$ with $\boldsymbol{y}$, denoted by $\mathcal{C} = \mathcal{C}(\boldsymbol{y}, \hat{\boldsymbol{y}})$. Typical choices of $\mathcal{C}$ are p-norms for regression problems or cross-entropy functions for classification tasks [36].

Taking into account all the samples in the training set, we define the cost function as an average of the single errors, namely as

$$\mathcal{C}_{tot} = \frac{1}{N_{\mathbb{T}}} \sum_{i=1}^{N_{\mathbb{T}}} \mathcal{C}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \frac{1}{N_{\mathbb{T}}} \sum_{i=1}^{N_{\mathbb{T}}} \mathcal{C}(\boldsymbol{f}(\boldsymbol{x}_i; (\boldsymbol{w}, \boldsymbol{b})), \hat{\boldsymbol{f}}(\boldsymbol{x}_i)).$$

The goal is to solve an (unconstrained) optimization problem, aiming to find the parameters giving the minimum cost

$$(\boldsymbol{w}^*, \boldsymbol{b}^*) = \arg\min_{(\boldsymbol{w}, \boldsymbol{b})} \mathcal{C}_{tot}.$$

This minimization problem is usually solved via iterative algorithms, among which the (stochastic) gradient descent and its variants are quite popular ones. According to this rule, the weights are updated towards the steepest descent direction as

$$w_{i,j}^{n+1} = w_{i,j}^n + \Delta w_{i,j}, \quad \Delta w_{i,j} = -\eta \frac{\partial \mathcal{C}_{tot}}{\partial w_{i,j}}, \tag{3.2}$$

where η is a parameter known as *learning rate*. It has to be tuned in order to be small enough to guarantee stability and not to miss possible minimum points, and high enough not to drastically slow down the training time. We also note that if $\mathcal{C}_{tot}$ has vanishing gradients, the update $\Delta w_{i,j}$ can be minimal, leading to slow convergence. An update rule, equivalent to (3.2), holds also for the biases. Linearity of the derivative operators implies that, in order to to estimate the gradient for the whole dataset, one has to compute the gradients associated with a single sample, combining them together. For large training sets, repeating this procedure at each iteration has a high computational cost [36]. Therefore, it is a good practice to randomly shuffle $\mathbb{T}$ and divide it in *mini-batches* at each iteration. Then, each batch is used to perform one iteration of the optimization algorithm. Note that if the size of such batches is (much) smaller than $N_{\mathbb{T}}$, we might not estimate perfectly the steepest descent direction, but computing the total gradient is not time-consuming. When the training set is exhausted, the dataset is reshuffled and the process is repeated. One pass over all the training samples is called an *epoch*. The overall procedure is usually stopped after a certain number of epochs and/or by means of a *validation set*, generated independently of the training set. We go back to these criteria in the following Section.

3.2 A neural network to predict artificial viscosity

In the previous Section we provided a general overview of the concept of neural networks. Now we adapt it to the artificial viscosity estimation problem, considering multilayer perceptrons only. First, we focus on one-dimensional scalar problems, leaving extensions to both systems of conservation laws and a multidimensional framework to the dedicated Sections. In this context, the prototype problem is represented by the Burgers equation. It constitutes a good test case, due to nonlinearities in the flux function. Both rarefaction and shock waves may develop, even with a smooth initial condition. In order to enhance the capabilities of the neural network in capturing contact waves, as well as to estimate the accuracy of the scheme, the linear advection problem is also considered. In the spirit of most of the standard artificial viscosity models we described in Section 2.2, our strategy is to predict a local (constant) artificial viscosity and perform a global smoothing afterwards.

3.2.1 A family of neural networks

We build a *family* of neural networks, i.e. we design one network for each discretization degree m. This approach exploits all the available degrees of freedom in a mesh element. Thus, all the solution features are taken into account by the model. Note that this choice is different from the one made in, e.g., [11], where a single network is constructed. However, dealing with regression problems, we believe that more precision and variability for both input and output are required as compared to a classification task. The drawback of the adopted technique lies in the higher offline cost, since different networks have to be built. However, the training process is performed only once.

In this work we focus on the cases $m \in \{1, \dots, 4\}$, but extending the technique to higher orders is quite straightforward.

3.2.2 Choice of input and output

The primary idea is to estimate the viscosity values using the nodal values of a particular variable. For instance, dealing with (one-dimensional) scalar problems a natural choice is given by the solution values at the quadrature nodes. However, the following problem arises. Suppose that we provide as input (resp. output) the nodal values of the numerical solution (resp. the artificial viscosity in the corresponding nodes) for different mesh sizes h. Running some tests, we observe that the dependence on h is lost, in the sense that the network is not capable of capturing the variability in the mesh resolution. Indeed, for a given length h we have many different solution values, so that the network grasps the variability of the latter, leaving only a *weak* dependence on h. Since one of the major goals of neural networks is their ability to generalize their applicability range to samples not included in the training set, this naive approach does not work. A similar issue is present even if h is added to the input data, since no variability in the mesh resolution is captured.
However, we keep the idea of using as input and output some nodal values, but a *scaling* for all the values has to be performed, motivated by two arguments. Firstly, we state the following result, which is related to scaling arguments.

Proposition 1. *The MDH, MDA, EV models predict an artificial viscosity μ which scales as*

$$\mu \sim h \ \max |f'(u)|. \tag{3.3}$$

Proof. Let us start by the MDH model. Then, $\mu_{\max}$ clearly satisfies (3.3). By definition, the model adds a dissipation proportional to μ_{max}, with the exact amount depending on the estimated highest modal decay. Therefore, the scaling argument holds.
Similarly, the result is true even for the MDA model. Indeed, it predicts a viscosity which scales again as μ_{max}. The only difference lies in the way the exact amount is estimated, since this model predicts it according to an averaged decay rate.
Concerning the EV model, we observe that the normalization factor A scales as the entropy E. This is expected, due to physical dimensions arguments. Moreover, the entropy flux scales as $F \sim Ef'(u)$. Therefore, recalling equations (2.33a) and (2.33b) we find that

$$R \sim \frac{1}{T} = \frac{f'(u)}{h}, \quad H \sim \frac{f'(u)}{h},$$

according to the stability condition (2.18). Therefore, both μ_E (equation (2.32)) and μ_{EV} (equation (2.34)) satisfy the scaling property. □

We remark that the DB model does not satisfy the previous result, unless Burgers equation is considered. Indeed, μ_β scales linearly with the solution, which is consistent with equation (3.3) if and only if $f'(u) = u$. Thus, we do not rely on this model when constructing the training set. However, as we noted in Section 2.2, the DB model limits the solution accuracy to a second order. In other words, for scalar problems the DB model is never the best one, since it is usually more dissipative compared to the others. Therefore, ignoring it in the construction of the required datasets is not an issue.
Secondly, we make a comment related to neural networks and optimization algorithms. It is common practice to provide the input values in a standardized range. If this is not the case, large and small values can have significantly different impacts on the results. Moreover, the optimization algorithms converge faster if all the input data lie within the same range. As an example, consider the gradient descent procedure. Dealing with different scales, the range of the step sizes $\Delta w_{i,j}$ in equation (3.2) might be very wide, making the algorithm slower and/or forcing a low learning rate in order to obtain convergence. We can interpret the scaling of the samples as a way to reduce the curvature of the error surface, forcing the gradient to point towards the optimal direction [37]. The standardization process is usually a problem-dependent task. However, in our framework a consistent way is to scale any sample with its maximum absolute value, resulting in a normalized input vector lying in $[-1, 1]$.
These observations motivate our strategy, which is described in the following. Consider two variables

U_1, U_2, or better their pointwise values. Then, the scaled version of U_1 is provided as input to the network, which returns a pointwise estimation of the non-dimensional scaled viscosity $\mu_0 := \frac{\mu}{h \max |U_2|}$. In principle, the models we use to construct the dataset provide a constant viscosity value in each cell, thus one could also prescribe a single constant output, instead of the pointwise values. However, to make it more general, we prefer to adopt the latter choice. When one of the standard models returns a constant dissipation coefficient, we assume μ to be equal to such constant in all the nodes. For a given simulation, the target output is the viscosity predicted by the best model, chosen according to both qualitative and quantitative criteria, as the estimation of the amplitude of the residual oscillations (see Subsection 3.2.7). Note that the values are collected before the global smoothing is carried out. Then, a double inverse scaling, i.e. a multiplication by h and $\max |U_2|$ is applied to get the dimensional dissipation value in the corresponding nodes. A pictorial representation is provided in Figure 3.4. Therefore, the final outcome is a pointwise viscosity value. In principle sub-cell resolution is already

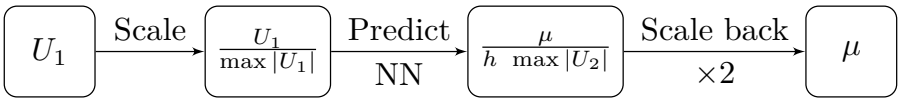


FIGURE 3.4: A graphical representation of the strategy used to build the network.

partly embedded, since there is no guarantee that the all the output neurons return the same value. However, in all the cases we considered, the variability in output nodal values is very low, resulting in a constant viscosity, up to a small tolerance. Thus, the C^0 smoothing is performed, using as elementwise value for μ the maximum among the nodal values within a given cell. A pictorial representation is provided in Figure 3.5.

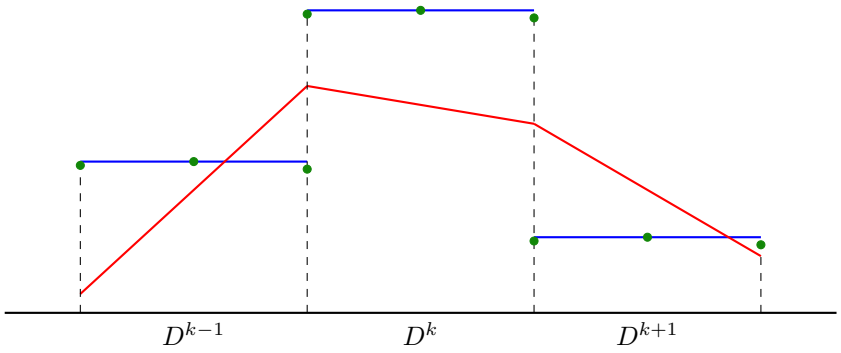


FIGURE 3.5: A graphical representation of the smoothed viscosity profile using $P = 1$ and $m = 2$ in a one-dimensional framework using the ANN. The green dots denote the pointwise values obtained with the network. The blue lines denote the piecewise constant values, obtained as the maximum value in each cell. The (continuous) red line represents the final profile, after the smoothing process.

A key step is the choice of the variables U_1 and U_2. We consider three different strategies, which are also reported in Table 3.1, where advantages and disadvantages are highlighted.

1. $U_1 = u$, $U_2 = u$. This is the easiest and the most naive approach. The advantage is that, beyond its simplicity, both input and output are scaled by the same quantity. The main drawback is the inconsistency with Proposition (1), since we would need to scale by $\max |f'(u)|$. However, for parabolic flux functions, and particularly for Burgers equation, the argument holds.

2. $U_1 = f'(u)$, $U_2 = f'(u)$. This version is coherent with Proposition (1) and scales input and output by the same quantity. We observed that in most of the cases, the numerical solution is too smoothed compared to other strategies. This holds true in particular for (two-dimensional) systems. Another issue is present when $f'(u)$ is constant, i.e. with the linear advection problem. For a constant transport field β, the input to the network would be always the same, making the model ignorant of the solution features.

3. $U_1 = u$, $U_2 = f'(u)$. Here, we are able to exploit the potential of the previous strategies, i.e. to extract solution features and to satisfy the scaling arguments. The fact that input and output are scaled by different quantities does not constitute an issue. Indeed, there are no results stating

that the same scaling has to be applied. For instance, in [11] the authors force the input to lie in $[-1, 1]$, while no scaling is performed at the output. As shown in Chapter 5, this approach appears to be the best among the proposed ones.

	U_1	U_2	Pros	Cons
1^{st} str.	u	u	captures u-variability, same I/O scaling	Prop. 1 does not hold
2^{nd} str.	$f'(u)$	$f'(u)$	same I/O scaling, ok with Prop. 1	too dissipative, linear advection
3^{rd} str.	u	$f'(u)$	captures u-variability, ok with Prop. 1	-

TABLE 3.1: Table which summarizes the different choices for the variables U_1 and U_2.

Coherently with the notation adopted in Figure 3.3, from here on we denote with $\boldsymbol{x} = (x_0, \dots, x_m)$ and $\boldsymbol{y} = (y_0, \dots, y_m)$ the input and the output of the network. In all the scenarios we have $N_I = N_O = m+1$, since the whole local cell information is provided.

3.2.3 Cost function

Dealing with a regression problem, natural choices for $\mathcal{C}$ are p-norms, among which $p = 1, 2$ are the most common ones. We consider the latter.
Moreover, we add a *regularization* contribution to the cost function [38], to avoid *overfitting* the training set. Indeed, it may happen that the model performs brilliantly on the training set but fails to generalize well to other data. Such regularization term is usually expressed as a penalty for the weights, while its practical effect is to create oscillations in the error decay as the algorithm proceeds. Thus, the final form of the cost function is:

$$\mathcal{C}_{tot} = \mathcal{C}_{err} + \mathcal{C}_{reg} = \frac{1}{2N_{\mathbb{T}}} \sum_{i=1}^{N_{\mathbb{T}}} \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_{l^2}^2 + \frac{\beta}{2} \|\boldsymbol{w}\|_{l^2}^2 = \frac{1}{2N_{\mathbb{T}}} \sum_{i=1}^{N_{\mathbb{T}}} \sum_{j=0}^{N_O-1} (y_{ij} - \hat{y}_{ij})^2 + \frac{\beta}{2} \sum_{(i,j)\in\mathcal{V}} w_{i,j}^2, \quad (3.4)$$

where $\beta \geq 0$ is a constant parameter which has to be properly tuned.

3.2.4 Activation functions

In order to choose proper activation functions, two aspects have to be recalled. Firstly, iterative optimization algorithms might suffer when gradients vanish. In other words, the update (3.2) could be very small when the corresponding gradients have low magnitude. By the chain rule, it can be observed that the update of the parameters involves the gradients of the cost and the activation functions [35]. Secondly, nonlinear activation functions are mandatory to ensure that the network does not collapse to a linear input-output mapping.
For MLPs, it is common practice to choose the same activation function for all the neurons belonging to the same layer. In this work we adopt this strategy, choosing two types of activation functions. The first is applied to all the hidden layers, while the second relates to the output neurons only. Note that the input neurons are source neurons, therefore no activation function has to be specified.
Concerning the hidden layers, we use the *leaky rectified linear unit (leaky ReLU)* function [39], defined as

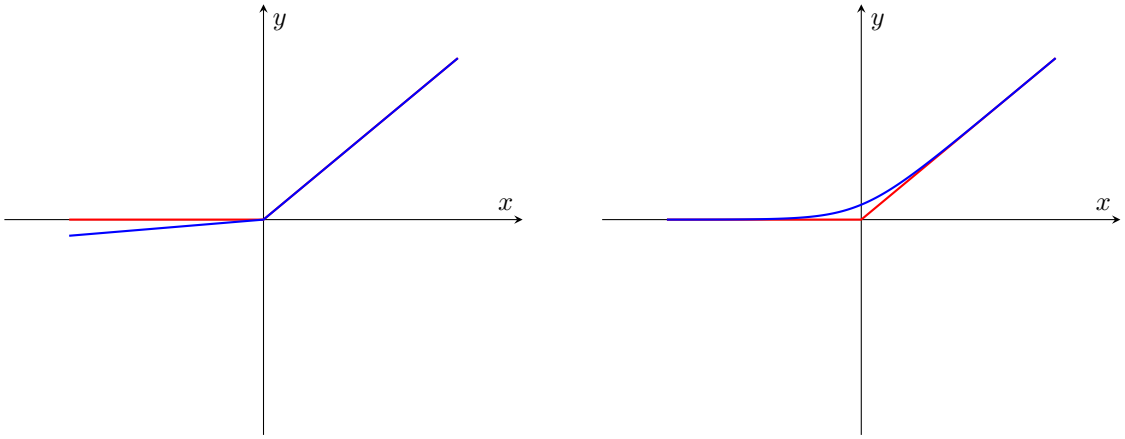
$$f_{LReLU}(t) = \max\{x, 0\} - \alpha \max\{-x, 0\} = \begin{cases} x & \text{if } x \geq 0, \\ \alpha x & \text{if } x < 0, \end{cases}$$

where α is a (small) nonnegative coefficient. If $\alpha = 0$ the function is simply called *rectified linear unit (ReLU)*. Its main advantage is the fast computation of its derivative, which is always nonzero. This avoids the problem of vanishing gradients and *dying neurons* [11, 39], which are critical issues when using, e.g. logistic function, hyperbolic tangent or ReLU functions.
The leaky ReLU is not a viable option for the output layer. Indeed, the output of the network is by definition proportional to the amount of dissipation. Thus, it has to be a nonnegative value. Even

though the training set does not contain negative values, we have no guarantee that for any input data the network predicts a nonnegative coefficient, unless we explicitly enforce it. This goal is achieved by adopting a nonnegative activation function for the output layer. A possible strategy would be the ReLU function, but we experienced the dying neuron problem. Consequently, the predicted dissipation value was very low, as verified by some numerical tests. A better option is the *softplus (SP)* function [40], defined as

$$f_{SP}(t) = \log(1 + e^x),$$

which can be interpreted as a smooth positive approximation of the ReLU function. Figure 3.6 shows a graphical representation of the activation functions. Finally, we refer to, e.g., [41] for a broader discussion on their choice. To sum up, the activation functions we adopted are the leaky ReLU



(a) Leaky ReLU function (b) Softplus function

FIGURE 3.6: Graphical representation of the activation functions used in this work. The blue line represents the activation function itself, while the red one corresponds to the ReLU function, which the both the LReLU and the SP approximate.

($\alpha = 0.001$) and the softplus for the hidden and output layers respectively.

3.2.5 Hyperparameters

The so-called *hyperparameters* are parameters which define higher level concepts about the model such as complexity or capacity to learn [42]. Practically, they affect both the structure of the network and the training process. They differ from the network parameters, i.e. weights and biases, since hyperparameters are fixed a priori and cannot be learned via the dataset. Their optimal value is problem-specific, thus they have to be properly tuned before the training process. Here, we focus on the design of the hidden layers and the value of the regularization parameter β in equation (3.4).
Based on Theorem 2, it is good practice to start with two hidden layers and increase their number as long as good estimated values are obtained. We recall that in this work both the input and the output layers are made of $m + 1$ neurons, where m varies between 1 and 4. However, we can assume that the number of hidden layers and the neurons therein is independent of m. Thus, both values can be treated as constants for all the discretization degrees. There is no optimal rule for the choice of the hyperparameters and, in a broad sense, our only practical criterion is provided by the accuracy of the numerical simulations obtained with the ANN-based viscosity estimator. Specifically, we choose $L = 5$ hidden layers made of $N_H^l = 10$ ($l = 1, \dots, L$) neurons each. Again, there is no theoretical motivation behind such choice, but we observe that $m+1 \leq N_H^l$, i.e. the number of hidden neurons in each layer is always greater than the input and output size. This guarantees that the input information is processed by a large enough number of neurons, independent of the discretization degree we consider. At the same time, we believe that $N_H^l = 10$ is a value which is not too large to slow down both the training time and the computational cost. In general, we having too many hidden neurons does not necessarily improve the efficiency of the network. We have no guarantee that the same strategy continues to hold when higher discretization degrees are considered, even though we believe that no issues should arise for $m \lesssim 7$.

Finally, a choice for the regularization parameter β has to be made. Recalling the definition of the cost function (3.4) an evident trade-off is present. If β is too large, in order to minimize $\mathcal{C}_{tot}$ the algorithm will force the weights to be close to zero, ending up with incorrect estimations, i.e. *underfitting*. On the other hand, if β is too small, not enough regularization is added and overfitting might still be present. We observe that the range for $\mathcal{C}_{err}$ is similar to the values of c^2_{max} appearing in equation (2.25). Typical order of magnitudes are $\mathcal{O}_1 = 10^{-2}$. The total number of weights is of order 10^3, so that $\mathcal{C}_{reg}$ has order (at most equal to) $\mathcal{O}_2 = \beta \cdot 10^3$. Since we do not want the regularization term to dominate over the error term we impose

$$\mathcal{O}_2 \leq \mathcal{O}_1 \implies \beta \leq 10^{-2-3} = 10^{-5}.$$

Finally, for the whole family of networks we choose $\beta = 10^{-5}$.

3.2.6 Optimization algorithm

A key role is played by the optimization algorithm. Good minimization schemes are based on the gradient descent method, with random data shuffling. Improved versions exploit, e.g., second order derivatives. For our purposes we rely on the *Adam optimizer* proposed in [43], a first-order gradient-based optimization algorithm based on adaptive estimates of lower-order moments. A suitable value for the *learning rate* η has to be picked. In this work, we adopt the value of $\eta = 10^{-3}$, as proposed in [43]. Weights and biases are randomly initialized, selecting their values according to a normal distribution with zero mean and unit variance. This is mandatory in order to break possible symmetries. For instance, a zero initialization for all the variables might result in no change for the weights.
Vice versa, the stopping criterion is based on the need to both guarantee convergence and avoid overfitting. Thus, it is built by controlling the:

1. *Maximum number of epochs*
   We set up a maximum number of epochs, denoted by N_{epochs}, like in every standard iterative algorithm. Clearly, if the batch size is fixed, this is equivalent to set a maximum number of iteration of the optimization scheme. Usually, N_{epochs} lies between 1000 and 2000.

2. *Validation error*
   Similarly to the training set, we create a *validation set* $\mathbb{V}$, independent from the former, but following the same philosophy. After each epoch, we compute the cost function using the validation data, defined as in (3.4), after replacing the samples in the training with the ones from the validation set. Then, we compare it with respect to its value at the previous epoch. If it increases, it is an indicator that the network is losing its generalization properties. However, the error decay is very oscillatory, thus we decide to allow the validation error to increase, stopping the algorithm only when this phenomenon emerges for $R = 10$ consecutive times. Usually, the (smoothed) behavior of the training and validation errors are as in Figure 3.7. We observe that overfitting is present for a large number of iterations, and an early stopping criterion is implemented.

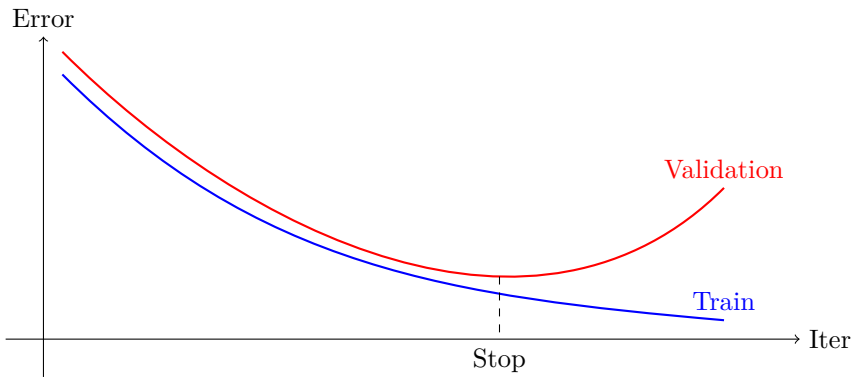


FIGURE 3.7: Typical behavior for the train and validation errors with respect to the number of iterations.

Practically, for most of the networks we trained, the algorithm stopped because the maximum number of epochs has been reached.

3.2.7 Training and validation sets

A key role is played by the construction of the training and the validation sets. In particular, we need to guarantee that the network can capture many different solution features, keeping good generalization properties. The procedure is summed up as follows:

1. Select a specific problem which has to be solved, by assigning the simulation setup. In particular, both the boundary value problem and the discretization parameters have to be specified. They include, for instance, the convective flux, the initial condition and the final simulation time, as well as the grid spacing and the discretization degree.
2. At this point, an optimization with respect to the models and parameters has to be done. Firstly, the models are analyzed separately, by selecting the parameters which give the *best* solution at the final time. In a second phase we compare the best solution obtained by each model, ending up with an optimal viscosity model with the best parameters.
   The optimization is performed according to the following criteria:
   - Overshoots and undershoots with respect to a reference solution have to be smoothed. Note that small oscillations are still acceptable, since none of the models guarantees the solution to be non oscillatory. The comparison is firstly done visually, and it is benchmarked by numerically evaluating the amplitude of such oscillations.
   - The solution must not be over dissipated, if compared to the reference solution. In most of the cases, this is evaluated visually.

   Note that the reference solution is either the exact analytical one (if available), a pseudo-analytical one (using the method of the characteristics by numerically solving nonlinear equations) or a numerical solution computed with one of the viscosity models using high order polynomials and a very fine mesh. In this last scenario, the role played by the parameters is marginal. Since we are approaching the continuous limit, the amount of dissipation is very small and a slight variation of the parameters does not significantly alter the solution. Moreover, in most of the cases the training samples are chosen from rather simple problems and initial conditions, so that the first and the second options are used.
3. Once the optimal model is picked, we collect the required data from it. In particular, we track the solution values u, the flux function values $f(u)$ and the corresponding derivative $f'(u)$, the grid spacing h and the artificial viscosity values μ for all the time steps. Here, we assume that the optimal model and parameters do not vary in time. This makes the samples generation easier.
4. Repeat steps from 1 to 3 for different grid spacings h and initial conditions u_0 in order to teach the network several possible configurations and solution features. For instance, we need to capture smooth solutions, discontinuities, points of non-differentiability and so on. In particular, for a fixed initial condition we consider different mesh sizes in order to take into account the spatial variability. For one-dimensional problems, we consider at least three different values of h. Then, the initial condition is varied and the same process is carried out.
5. Repeat step 4 for different problems, i.e. conservation laws. This is needed in order to guarantee that different wave types are captured by the network. In particular, we need it to be able to keep track of, and possibly distinguish between, rarefaction, shock, and contact waves. In order to capture rarefactions and shocks, we rely on Burgers equation. On the other hand, we use the linear advection problem as prototype for linear waves, i.e. contact discontinuities.
6. Repeat step 5 for different discretization degrees m, in order to build the required datasets for all the networks we need to train.

After this procedure, we end up with a different dataset for each degree m. However, a post-processing phase is required in order to construct the final training and validation sets. This procedure, carried out separately for each m, can be summed up in three further steps:

- First, we need to create a *balanced* dataset. Indeed, suppose that we run two simulations using two different element sizes, say $h_1 = h$ and $h_2 = h/2$. Keeping all the data from both scenarios, a rather strong bias towards the second simulation is present. Suppose that the sizes of the above-mentioned dataset are N_1 and N_2 respectively. Each time step, the second simulation provides twice the number of samples due to the larger number of mesh elements. Moreover, the time step is directly proportional to the grid size, so that in the second scenario we approximately perform twice the number of time iterations with respect to the first one. Consequently, we have $N_2 \simeq 4N_1$. This behavior is clearly undesirable, since the contributions from the fine meshes dominate.
  Thus, to solve this issue we define h_{min} as the size of the coarsest mesh we use for a certain initial condition. Then, we define a *shrinking* factor as

$$S = S(h) = \text{round}\left(\frac{h_{min}}{h}\right)^2,$$

  and we ignore the solutions at time steps which are not integer multiples of S.
  Moreover, a further shrinkage is performed. Due to the different range of the solutions, the time step my vary by orders of magnitude. Indeed, suppose that we run two simulations with Burgers equation, whose initial condition ranges lie in $U_1 = [0, 1]$ and $U_2 = [0, 10]$ respectively. Then, $\frac{\Delta t_1}{\Delta t_2} \simeq \frac{U_2}{U_1} \simeq 10$ and the dataset will be biased towards the second simulation. Moreover, due to different final times, the number of temporal iterations could be different, too. Thus, for each initial condition we might need to remove some data to guarantee a good balancing. This is performed by randomly shuffling the data and deleting a certain percentage of them. In general this step might not be strictly necessary, but it helps in order to improve balancing.
  As a side remark, we note that the first shrinkage is performed among the solutions having the same initial condition and different mesh sizes. Vice versa, the second one involves an overall inspection of the collected data. A more concrete example is provided in the following Subsection, to which we refer for further details.
- Then, we compute the scaled variables used for both input and output.
- Finally, we check for data *consistency*. If two (scaled) samples $(\boldsymbol{x}_1, \boldsymbol{y}_1)$, $(\boldsymbol{x}_2, \boldsymbol{y}_2)$ have the same input, then the output must be the same. That is, $\boldsymbol{x}_1 = \boldsymbol{x}_2$ implies $\boldsymbol{y}_1 = \boldsymbol{y}_2$. If this is not the case, we claim the dataset to be inconsistent, and the training algorithm may fail to converge properly. This issue is simply solved by setting the corresponding output to be the average along the data having the same input. That is, if $\boldsymbol{y}_1 \neq \boldsymbol{y}_2$ then we set $\boldsymbol{y}_{1,2} \leftarrow \frac{\boldsymbol{y}_1 + \boldsymbol{y}_2}{2}$. We remark that this check has to be performed with the scaled variables, i.e. the ones that are concretely used for training. Moreover, the presence of different repeated inputs is not an issue. It can be even seen as a desirable feature, since it forces the network to learn from these data. For instance, a reasonable percentage of the data is given by *constant* samples, say $\boldsymbol{x} = (1, \dots, 1)^T$, characterized by a corresponding zero viscosity coefficient. Having many constant inputs, we force the network to learn this behavior.

Ultimately, the training and validation sets generated. For each m, we first randomly shuffle all the post-processed samples. Then, we assign the first 70% of the data to the training and the remaining 30% to the validation set.

3.2.8 An example

We provide a detailed example on the whole dataset generation procedure in the case $m = 3$. Obviously, the strategy is similar for the other degrees.
The conservation laws we employed are Burgers and linear advection only. For simplicity, here we focus on the former. We select different initial conditions $u_0(x)$ and final simulation times T. Then, for each case we consider different values of h, by varying the number of elements. We collect the samples from the best model and we perform the post-processing discussed in the previous Subsection. In Table 3.2 we list the simulations we consider, highlighting the number of training samples we obtain from each scenario before the post-processing phase. The definitions of the corresponding initial conditions is provided later in this Subsection.

Test nb.	T	h	Best model	Nb. samples
1	0.15	2/40	MDH: $c_A = 2.5,\ c_\kappa = 0.4,\ c_{max} = 0.6$	6560
		2/80	MDH: $c_A = 2.5,\ c_\kappa = 0.5,\ c_{max} = 0.6$	26240
		2/120	MDH: $c_A = 2.4,\ c_\kappa = 0.4,\ c_{max} = 0.5$	58920
		2/200	MDH: $c_A = 2.2,\ c_\kappa = 0.4,\ c_{max} = 0.5$	163400
2	0.08	1/40	EV: $c_E = 1.5,\ c_{max} = 0.5$	4560
		1/80	EV: $c_E = 1.2,\ c_{max} = 0.4$	18480
		1/120	EV: $c_E = 1.0,\ c_{max} = 0.3$	41760
3	0.07	2/40	EV: $c_E = 1.8,\ c_{max} = 0.5$	19360
		2/80	EV: $c_E = 1.4,\ c_{max} = 0.5$	78720
		2/120	EV: $c_E = 1.5,\ c_{max} = 0.4$	178080
		2/200	EV: $c_E = 1.0,\ c_{max} = 0.6$	497000
4	0.07	1/40	EV: $c_E = 2.0,\ c_{max} = 1.0$	40600
		1/80	EV: $c_E = 1.8,\ c_{max} = 0.8$	162400
		1/120	EV: $c_E = 1.6,\ c_{max} = 0.8$	365400
5 (multiple)	0.03	1/40	MDH: $c_A = 2.5,\ c_\kappa = 0.4,\ c_{max} = 0.8$	30600
		1/80	MDH: $c_A = 2.0,\ c_\kappa = 0.4,\ c_{max} = 0.6$	122160
		1/120	MDH: $c_A = 2.0,\ c_\kappa = 0.4,\ c_{max} = 0.4$	274680
6	0.3	1/40	EV: $c_E = 1.8,\ c_{max} = 0.5$	17280
		1/80	EV: $c_E = 1.6,\ c_{max} = 0.6$	69200
		1/120	EV: $c_E = 1.6,\ c_{max} = 0.4$	155880
7	0.08	1/40	MDH: $c_A = 2.0,\ c_\kappa = 0.4,\ c_{max} = 0.6$	4640
		1/80	MDH: $c_A = 2.2,\ c_\kappa = 0.4,\ c_{max} = 0.5$	18560
		1/120	MDH: $c_A = 2.0,\ c_\kappa = 0.4,\ c_{max} = 0.4$	41760

TABLE 3.2: A summary on how to generate the datasets for Burgers equation.

Some comments have to be made. Firstly, we observe that in the test cases we considered the best model is always either the MDH or the EV. The DB model is always too dissipative (second order accuracy), and does not satisfy the scaling arguments (see Proposition 1). At the same time, the MDH model is better than its improved version (the MDA model) since the latter is specifically designed for high order schemes. We believe that $m = 3$ is not high enough for the the MDA to show its potential. Secondly, we note that the parameter values are not very sensitive to a change in the number of mesh elements. However, it is important to capture small variations, in order to have a better precision on the corresponding viscosity values which will constitute part of the training set. Vice versa, the parameters show important variations when different initial conditions and/or discretization degrees are considered. Thirdly, for most of the cases the final simulation times are generally small. This choice is mainly dictated by practical reasons, since dealing with large times would result in a very large dataset and higher training costs.
We provide now the analytical expression and a graphical representation of the test cases we considered.

- Test case nb. 1. It combines a `rect` function, a straight line and a quadrant of a circle.

$$u_0(x) = \begin{cases} 1.5 & \text{if } 0.1 \le x < 0.25, \\ x & \text{if } 0.5 \le x < 1, \\ 0.5 + \sqrt{\frac{1}{4} - (x-1)^2} & \text{if } 1 \le x < 1.5, \\ 0.5 & \text{otherwise in } [0, 2]. \end{cases}$$

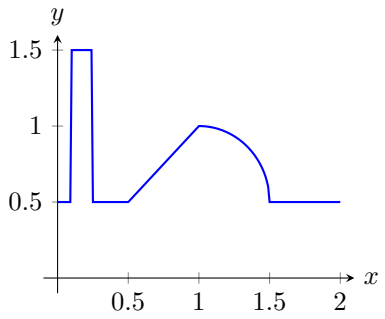


FIGURE 3.8: I.c. for test case nb 1.

- Test case nb. 2. It is simply a gaussian function with a reasonably low variance. It is required to have negative solution values. Note that initially the solution is continuous, while a shock appears after $t \simeq 0.05$.

$$u_0(x) = \begin{cases} -e^{-400(x-0.5)^2} & \text{if } 0.3 \leq x < 0.7, \\ 0 & \text{otherwise in } [0,1]. \end{cases}$$

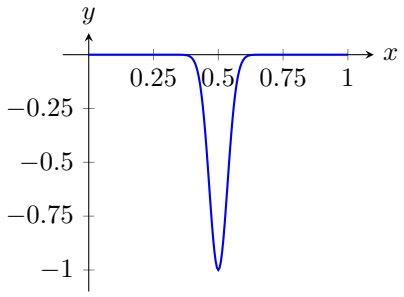


FIGURE 3.9: I.c. for test case nb 2.

- Test case nb. 3. This hat function is needed to teach the network with data arising from points of non-differentiability.

$$u_0(x) = \begin{cases} 20\ (0.5 - |x - 0.5|) & \text{if } 0 \leq x < 1, \\ 0 & \text{otherwise in } [0,2]. \end{cases}$$

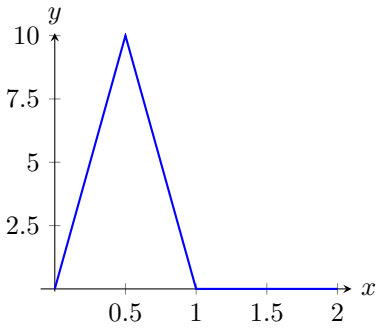


FIGURE 3.10: I.c. for test case nb 3.

- Test case nb. 4. It is a combination of step functions which eventually collapse in a single shock [11].

$$u_0(x) = \begin{cases} 10 & \text{if } 0 \leq x < 0.2, \\ 6 & \text{if } 0.2 \leq x < 0.4, \\ 0 & \text{if } 0.4 \leq x < 0.6, \\ -4 & \text{if } 0.6 \leq x < 1.0. \end{cases}$$

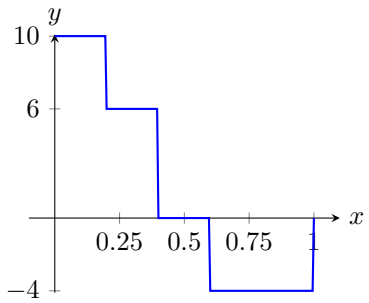


FIGURE 3.11: I.c. for test case nb 4.

- Test case nb. 5. It is a `rect` function, which develops both a rarefaction and a shock wave. We considered three different set of parameters (α, β) which define the maximum and the minimum of the function respectively.

$$u_0(x) = \begin{cases} \alpha & \text{if } 0.25 \leq x < 0.75, \\ \beta & \text{otherwise in } [0,1]. \end{cases}$$

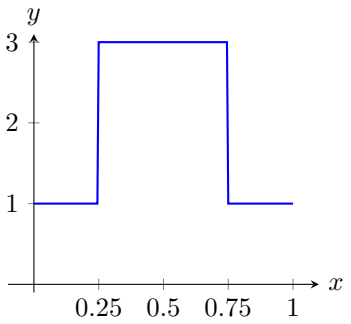


FIGURE 3.12: I.c. for test case nb 5.

- Test case nb. 6. It is one period of a sine wave. A shock develops around $t \simeq 0.15$.

$$u_0(x) = \sin(2\pi x)\mathbb{1}_{[0,1]}(x)$$

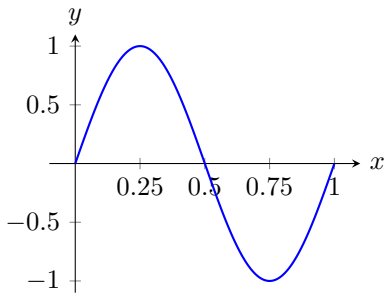


FIGURE 3.13: I.c. for test case nb 6.

- Test case nb. 7. Similarly, it is a combination of sine waves having different frequencies. In order to avoid shocks forming at the domain boundary, we assume that the initial condition is compactly supported, with the support lying well within the computational domain.

$$u_0(x) = \begin{cases} \sin(4\pi x) & \text{if } 0.25 \leq x < 0.5, \\ \sin(8\pi x) & \text{if } 0.5 \leq x < 0.75, \\ 0 & \text{otherwise in } [0,1]. \end{cases}$$

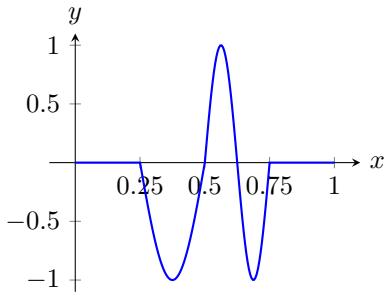


FIGURE 3.14: I.c. for test case nb 7.

The final phase is related to the post-processing of the data, which mainly consists in the balancing of the dataset. The first step is responsible of eliminating data collected from finer meshes. Then, we remove samples from the simulations having larger solution ranges or final times. Table 3.3 sums up the procedure. Finally, note that the input-output consistency is already guaranteed. We believe that it is related to floating point arithmetic.

A similar procedure is carried out for the linear advection equation, where both smooth (e.g. sine waves) and discontinuous initial conditions (e.g. `rect` functions) are used, creating the final dataset.

Test nb.	h	Nb. samples	Shrink 1	Shrink 2
1	2/40	6560	6560	6560
	2/80	26240	6560	6560
	2/120	58920	6547	6547
	2/200	163400	6536	6536
2	1/40	4560	4560	4560
	1/80	18480	4620	4620
	1/120	41760	4640	4640
3	2/40	19360	19360	9680
	2/80	78720	19680	9840
	2/120	178080	19787	9894
	2/200	497000	19800	9900
4	1/40	40600	40600	10150
	1/80	162400	40600	10150
	1/120	365400	40600	10150
5 (multiple)	1/40	30600	30600	7650
	1/80	122160	30540	7635
	1/120	274680	30520	7630
6	1/40	17280	17280	17280
	1/80	69200	17300	17300
	1/120	155880	17320	17320
7	1/40	4640	4640	4640
	1/80	18560	4640	4640
	1/120	41760	4640	4640

TABLE 3.3: A summary on how to post-process the data for Burgers equation.

3.3 Improved versions

In the previous Section we presented the building blocks of our new model for artificial viscosity. This technique works quite well, at least for the test cases we analyzed, so that one could consider only this model. However, in this Section we propose two different improvements.
The main drawback of the approach we described relates to the convergence rate for smooth problems. We recall that the network predicts a scaled viscosity μ_0, while the final value is obtained by multiplying it with the grid spacing and a local wave speed. Here, the issue is contained in the former. Indeed, we end up with a viscosity which scales almost linearly with h, since one can verify that the dependence of μ_0 and $\max|f'(u)|$ on the mesh spacing is rather weak. This could represent a problem, since high order accuracy is never obtained, even for smooth problems. We propose two different approaches to fix it.

3.3.1 Two coupled neural networks

Consider the amount of dissipation added by the MDH model, defined in equation (2.29). Essentially, the model adds an order-1 dissipation, unless the problem resolution and/or the solution smoothness are high enough. In this scenario, no dissipation is added and high order accuracy is retained. Thus, the model has an in-built regularity detector, in the sense that, provided that a certain regularity condition is satisfied, no dissipation is injected.
In this spirit, we make use of a *troubled-cell indicator*. Its role is to identify the cells where the solution loses regularity, and consequently dissipation has to be added. Ideally, such indicator has to be:

- Parameter-free. We recall that one of our goals is to obtain a non-parametric artificial viscosity method. Thus, it makes no sense to add an indicator which depends on tunable variables.
- Computationally fast. Essentially, we do not want to create a performance bottleneck in order to evaluate the troubled cells. Since the identification has to be done at least once per time step, it has to be reasonably fast. As a rule of thumb, the computational time has to be comparable to the cost required to estimate artificial viscosity.

Well-known methods are based on the *minmod* [8] or the *minmod-type TVB* [44] limiters. Despite being popular choices, the former flags more cells than necessary, while the latter depends on a problem-dependent parameter [45]. Therefore, both schemes are not suitable for our purposes. A good alternative is the technique proposed in [11], where an ANN is used as troubled-cell indicator. Since it satisfies both the above-mentioned requirements, we rely on this method.
Its application is rather simple. For a given mesh element D^k, the input of this network is defined as:

$$\boldsymbol{x} = \left(\bar{u}_{k-1}, \bar{u}_k, \bar{u}_{k+1}, u^+(x_l^k), u^-(x_r^k)\right)^T,$$

where the overbar denotes the cell-average in the corresponding element. In particular, the solution averages in the k-th element and its two neighbors have to be provided, as well as the pointwise values at the cell boundaries for the k-th element. The output of the ANN-based indicator is:

$$\boldsymbol{y} = (y_1, y_2)^T,$$

where both y_1 and y_2 lie in $[0, 1]$. In particular, the former is the probability of the k-th element to be a troubled cell. Vice versa, the latter is simply $y_2 = 1 - y_1$ and can be interpreted as the probability of the element to be a good cell. Therefore, we flag the k-th element as troubled cell when

$$y_1 \geq 0.5. \tag{3.5}$$

Practically, we first compute the artificial viscosity for all the mesh elements. Then, we set it to zero in the cells where condition (3.5) is not met. Note that the application of the first network is done only at the beginning of each temporal iteration. As already explained, this is motivated by performance reasons. However, the ANN-based troubled-cell indicator is applied every time the solution vector is updated, i.e. even in the Runge-Kutta internal loop. Indeed, we numerically verified it to be the best technique. Good results can also be obtained by applying both networks once per time iteration. Of course, computational efficiency turns out to be slightly affected by our choice. We refer to [11] for further details and the numerical validation of the ANN technique.

3.3.2 A different scaling

In principle, the first strategy works well and smooth areas are correctly identified by the indicator. However, we would like to let our ANN be able to automatically detect the regions where the solution is smooth, without relying on an external tool. To achieve this goal, we need to find a different scaling factor, say $\mathcal{H}$, which satisfies the following property:

$$\mathcal{H} = \begin{cases} \mathcal{O}(h^{m+1}) & \text{if } u \text{ is smooth,} \\ \mathcal{O}(h^{\alpha}) & \text{otherwise,} \end{cases} \tag{3.6}$$

for a certain α included in the interval $[0, 1]$. The value $\alpha = 1$ is the one we obtain with the standard artificial viscosity models, coherently with the definition of the maximum dissipation μ_{max} (2.25). Then, the final viscosity value is given by $\mu = \mu_0 \mathcal{H} \max |f'(u)|$, since $\mathcal{H}$ replaces the linear scaling h.
A good choice, in compliance with the requirement (3.6), is provided by the solution jump. In particular, one may choose

$$\tilde{\mathcal{H}} = \max_{e \in \partial D^k} (|[\![u_h]\!]_e|) = \max\left(|u_h^-(x_r^{k-1}) - u_h^+(x_l^k)|, |u_h^-(x_r^k) - u_h^+(x_l^{k+1})|\right),$$

where we emphasized the fact that the jump is computed using the discrete solution u_h. The requirement (3.6) is then satisfied with $\alpha = 0$. Indeed,

$$[\![u_h]\!] = u_{h,l} - u_{h,r} = u_l + \mathcal{O}(h^q) - u_r - \mathcal{O}(h^q).$$

For smooth solutions, $q = m + 1$ and $u_l = u_r$ so that the optimal accuracy is restored. On the other hand, for discontinuous solutions, $q < m + 1$ and the zero order term $u_l - u_r$ dominates.
Since we would like to be compliant with (3.6) with $\alpha = 1$ and with the standard models, we define the final scaling factor as

$$\mathcal{H} = \min\left\{C \cdot \tilde{\mathcal{H}}, h\right\}, \tag{3.7}$$

where C is set equal to 1 (in the correct physical units) throughout this work. Note that the choice (3.7) does not influence the output of the ANN-based viscosity estimator. Thus, there is no need to retrain the network using the improved scaling factor.

3.3.3 A remark

Throughout this work, we consider the above-mentioned improvements in an independent way. In other words, we use either two ANNs keeping the standard scaling or a single ANN with the modified scaling. We believe that the explanations of our results are clearer by keeping the improved versions separated. At the same time, using the scaling-related improvement, we still get significant results, even if no troubled-cell identification is carried out. However, we remark that these two approaches can be combined. In this case, one may expect further improvements in the quality of the numerical results. Implementing and validating this new approach represents one of the possible extensions of this work.

3.4 Extension to systems

So far we described the strategy for one-dimensional scalar problems. Now, we extend the procedure to systems of conservation laws and higher-dimensional problems. The former is quite straightforward, at least from a theoretical point of view. Basically, the reasoning is rather similar to the MDH and MDA models.
Let us start by observing that, since multiple equations are involved, one may think of adding a different amount of dissipation in each of the equations. This approach treats each equation independently. For instance, in the context of Euler system one may use the degrees of freedom of the density to estimate the amount of dissipation to be added in the mass conservation equation. Similarly, one computes μ for the momentum and energy equations using ρv and E respectively. Theoretically, one may expect it to be the best strategy, but this is not observed in practice. In particular, some issues appear to be present in terms of:

- Solution quality. The numerical results are not as good as expected. As a simple test case, which is not reported here, one may consider the Sod problem for Euler equations [46]. It turns out that the profile for density and energy is smoothed enough, while the momentum (or equivalently the velocity) seems to be too oscillatory.
- Computational performances. This approach requires the application of the neural network for $n \cdot N_{iter}$ times, with n denoting the number of equations. Thus, the increase in computational cost is not negligible.

Our reasoning is coherent with [21], where the authors claim that "the shock-capturing technique proved to be more robust and accurate if the same viscosity coefficient is used for every component of the solution vector". Therefore, we apply the same amount of dissipation to all the equations of the system.
To pursue this goal, a possible idea would be to construct a separate family of networks to deal with systems. In particular, for Euler equations, in the ideal scenario one should use density, momentum and energy (or, similarly, density, velocity and pressure) as predictors for the artificial viscosity, i.e. as input to the ANN. The design of such network should not be different from the scalar case. The advantage lies in the fact that this technique grasps features from all the variables. For instance, it would be able to correctly distinguish between shocks and contact waves, which we identify as one of the main issues when dealing with systems. However, the main drawback is that we would end up with a a problem-dependent network. It could not be applied to, e.g., shallow water equations and another one should be constructed. Even though we believe that this approach gives optimal results for Euler equations, we cannot rely on such a technique, at least in the general case. Indeed, we remind that one of our main goals is to develop a universal black box to predict artificial dissipation.
Therefore, we mimic the idea of the MDH and MDA models, which apply the technique of scalar equations using, in the context of Euler system, density. However, recalling the strategies identified in Table 3.1, we note that both u and $f'(u)$ are present. In case of Euler equations, the latter is easily

replaced by the maximum absolute eigenvalue of the flux jacobian matrix, which turns out to be

$$|f'(u)| = |v| + c.$$

The safest choice for the former is the density, as it is done in the standard models based on modal decay. Another viable option is the Mach number (2.23). Indeed, such variables are discontinuous across both shocks and contact waves. Thus, the network is capturing these discontinuities, adding dissipation in the system. We numerically observed that velocity (or pressure) might be used in place of density. This approach provides better results when at least one shock wave is present in the problem, but fails when a single contact wave is present. Again, since we aim to provide a general technique, we stick to density and Mach. We believe that any variable which is discontinuous across a contact wave could be used as predictor.
To provide an example of practical implementation, consider the third strategy presented in Table 3.1. We give as input the scaled values of density ($U_1 = \rho$), while the final viscosity value, to be applied to all the equations is given by $\mu = \mu_0 h \max(|v| + c)$, obtained by setting $U_2 = |v| + c$.
The improved versions of the ANN can be easily generalized to systems. For Euler equations, the ANN to detect the troubled cells is applied three times, using density, velocity and pressure separately. A certain cell is flagged if condition (3.5) is met with at least one of these three variables. On the other hand, the new version of the scaling that we denoted by $\mathcal{H}$ is simply obtained by using the maximum absolute value of the jump of the predictive variable, i.e. density.

3.5 Extension to two dimensional problems

The last extension covers two-dimensional problems. Again, let us start by considering scalar equations. Throughout this work, we consider two different strategies:

1. Rely on the one-dimensional family of networks.
2. Construct and train another family of networks, specifically designed for two-dimensional problems.

The former technique is inspired by the MDA model. In the two-dimensional scenario, the one-dimensional framework is applied along each boundary and an element-wise constant viscosity coefficient is estimated. Thus, our first technique consists of applying the previously described network along each edge. This results in computing three different coefficients for each triangle. Then, the element-wise viscosity is defined as an average of such values. Finally, the double inverse scaling and the global smoothing are carried out.
To be more rigorous, consider the nodal values of a given variable U_1 along each edge, denoted by $U_{1,e}$ $(e = 1, \dots, 3)$. Then, using the one-dimensional network, the scaled viscosity, say $\mu_{0,e}$ $(e = 1, \dots, 3)$, is estimated along each edge. The scaled element-wise constant viscosity coefficient μ_0 has to be a suitable combination of the boundary values. A good definition is the following:

$$\mu_0 = \texttt{wmean}(\mu_{0,e}) = \sum_{e=1}^{3} w_e \cdot \mu_{0,e} = \sum_{e=1}^{3} \left(\frac{\mu_{0,e}}{\sum_{i=1}^{3} \mu_{0,i}} \right) \mu_{0,e}.$$

This is simply a weighted average of the boundary coefficients, where the weights are designed in such a way that higher values make a greater contribution in the average. A similar option consists of setting μ_0 as the maximum of the edge coefficients. On the other hand, we have numerically observed that taking the standard arithmetic mean seems not to add enough dissipation. The final element-wise viscosity is then computed by scaling μ_0 as

$$\mu = \mu_0 \mathcal{H} \Lambda,$$

where Λ is the maximum wave speed, obtained by taking into account both the x and the y fluxes. Again, the standard version is simply obtained by setting $\mathcal{H} = h$, but the low-order accuracy issue is still present. To restore high-order precision, only one of the two presented improvements can be generalized to two-dimensional problems. Indeed, at the moment we write this thesis, there is no available two-dimensional version of the ANN as troubled-cell indicator. One could rely on the standard detectors, but the same issues of the one-dimensional case are present. Thus, we consider

only the new scaling $\mathcal{H}$ based on the jump, which can be easily extended to any spatial dimension. Defining again

$$\tilde{\mathcal{H}} = \max_{e \in \partial D^k} \left(|[\![u_h]\!]_e| \right),$$

we set

$$\mathcal{H} = \min \left\{ C \cdot \tilde{\mathcal{H}}, h \right\},$$

where C is set equal to 1.
As shown in Chapter 5, for most of the considered test cases the approach based on the one-dimensional ANN provides good results and captures most of the solution features. The main drawback of such a technique lies in the fact that only the boundary degrees of freedom are taken into account. The internal solution values are not used to predict the viscosity, which might represent a limitation for high discretization degrees. As an example, consider $m = 4$, which is the highest degree employed in this work. Then, the basis cardinality is $N = 15$ and only 12 degrees of freedom belong to the boundary and are taken into account. The three remaining ones are ignored by the scheme. Since the number of internal nodes is equal to

$$N_{int} = N - N_{bd} = \frac{(m+1)(m+2)}{2} - 3m = \frac{(m-1)(m-2)}{2},$$

the problem might become significant for higher orders. However, recalling the analysis with the MDA model and its performances for high orders [14], we believe that this issue is rather controlled. Clearly, this is not a concern when $m = 1$ or $m = 2$ are considered, since no internal nodes are present. A second, less critical, issue relates to computational performances. Indeed, we need to apply the network three times to estimate the local viscosity. Although this process is computationally efficient, this multiple evaluation of the network might slow down the whole algorithm.
For these reasons, it makes sense to switch to the second strategy, designing another family of networks. Even though the basic reasoning does not significantly differ from the one-dimensional case, further issues arise:

- The complexity of the problem increases. Since two spatial variables are involved, the solution can exhibit a more complex behavior. Waves propagating in different directions can interact, resulting in finer structures which cannot exist for one-dimensional problems. As a concrete case, consider a two-dimensional Riemann problem for the Euler system. As verified in [47], there exist at least 15 different admissible configurations. Each of them is characterized by different solutions features [48], which should be captured by the network.

- The mesh structure, as well its elements, exhibit more variability. For one-dimensional problems, the elements are simply intervals, which are essentially characterized only by their length h. Increasing the spatial dimension, the complexity increases. An obvious example is the orientation of the triangles. Elements having the same shape may differ by the way they are rotated. This is taken into account by the affine mapping Ψ from the reference element. A second, less trivial, property is the element aspect ratio, defined as the ratio between the longest and the shortest edge. It is a measure of the stretching of the element, and it is directly linked to mesh regularity [49].
  Note that for general cases one has no control on how the mesh is created. This usually depends on the problem that has to be solved, since some features might have to be highlighted. A clear example is the enhancing of boundary layers, where the mesh could be refined and/or stretched close to this region. Thus, as in the one-dimensional scenario, the network should not rely on a specific mesh structure, otherwise losing its *generalization* properties. In Figure 3.15 some examples of different elements having the same diameter h are shown.

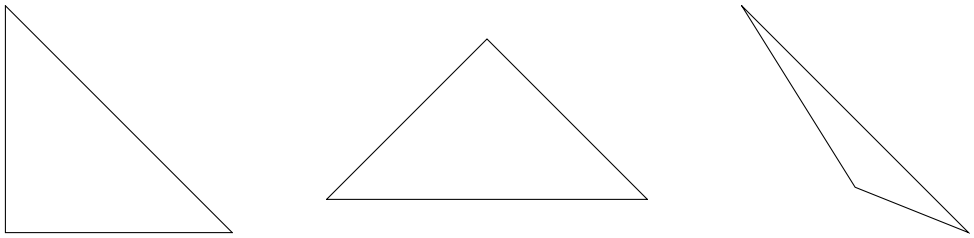


FIGURE 3.15: Graphical representation of different triangles having the same diameter h.

- Another issue is related to *node orientation*, shown in Figure 3.16. Even if the node ordering in the reference element is fixed, their mapping into each physical triangle might be different. In particular, for a given mesh element, the nodes can be stored in six different configurations. Here, we assume that a counterclockwise ordering of the vertexes, reducing the possible orientations to three. Thus, the vector collecting the degrees of freedom is defined up to a node permutation. Ideally, the way the neural network is built has to be *invariant* under such permutations, which cannot be controlled by the user. Here, invariant means that the network should give similar results independently of the orientation. Constructing an ANN whose input is the same even if a permutation is applied would probably be more challenging, especially if nodal values have to be the input variable. A possible way to tackle this issue is the following. At every time step, each triangle gives three different samples to the dataset. Each of them corresponds to a possible node orientation, with corresponding nodal viscosity values. Clearly, this phenomenon is not present in one-dimensional contexts, since nodes are always ordered from the left to the right of the interval.

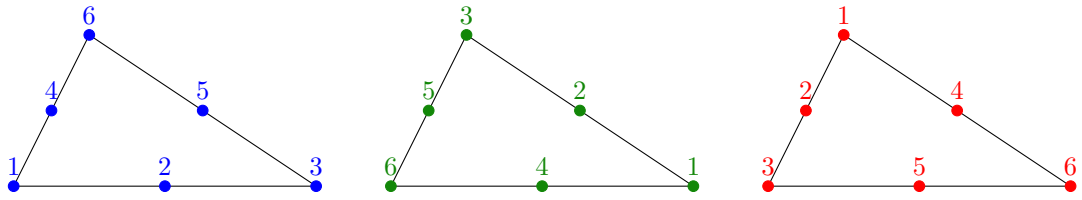


FIGURE 3.16: A graphical representation of the orientation issue using $m = 2$. Even if the shape is equal, the way the degrees of freedom are stored is different.

- Finally, the construction of the training set might suffer from memory issues. Consequently, training times increase. As an example, consider a square domain whose edges are divided into N intervals. Suppose that the associated (structured) mesh is obtained by dividing quadrilateral elements into two triangles. Therefore, the total number of elements is $2N^2$. Thus, for a frozen time step, we collect data from $2N^2$ elements, in comparison with the N generated in the one-dimensional case with the same edge resolution. This results in a larger dataset and consequently higher training times. In terms of order of magnitude, the dimension of the training set has order of tens of Gigabytes and the corresponding training time has order of few days, unless some further shrinkage is carried out.

We remark that, motivated by the reasoning illustrated in Section 3.4, the viscosity estimation for two-dimensional systems is carried out using the networks trained for scalar problems. A representative scalar variable has to be identified, acting as a predictor. In case of Euler equations, density is used.

3.5.1 How to build a two-dimensional network

The strategy to construct a two-dimensional family of neural networks resembles the one-dimensional one. Again, we consider scalar equations only.
The general structure is similar. In particular, we still rely on MLPs where input and output are the scaled solution and artificial viscosity respectively. Since nodal values are still employed, their

size is equal to $N_I = N_O = \frac{(m+1)(m+2)}{2}$. The same reasoning holds for cost and activation functions, since there is no obvious reason to change them. Therefore, the former consists of the l^2-norm of the difference between predicted and exact output, to which we add a regularization term. The latter are the Leaky ReLU and the softplus functions for the hidden and output layer respectively. Particular attention has to be given to the choice of the hyperparameters, with a focus on the hidden layers. As in the one-dimensional case, our main goal is to have a high enough number of neurons processing the signals, possibly guaranteeing computational efficiency. For high orders, say $m = 3, 4$, the input size is $N = 10, 15$ and exceeds the number of neurons in a single hidden layer, which was set to $N_H^l = 10$ in the one-dimensional framework. We numerically find that increasing the number of hidden units is necessary. Therefore, we set $N_H^l = 20$ neurons per hidden layer. Note that $L = 5$ is still a sufficient number of hidden layers.
Again, a significant role is played by the generation of the training set. In principle, the strategy is equal to the one-dimensional case, but memory issues arise. We report a summary of the main steps, comparing them with the one-dimensional framework.

- Select a specific problem which has to be solved.
- Choose the best model. It is done in a way similar to the one-dimensional case, even though the required cost is much higher. Several two-dimensional simulations have to be run, so that obtaining very precise values for the parameters turns out to be more challenging and time-consuming.
- Collect the data, storing the required values at each time iteration. As explained in the previous Subsection, for each element D^k we add three samples to the dataset, which differ because of the node permutation. Thus, for each simulation we collect $3N_{iter}K$ samples.
- In principle, we need to repeat the previous steps for different mesh sizes and initial conditions. The second procedure is compulsory to capture several solution patterns. The first one takes care of the spatial variability, but, using different mesh resolutions, the number of samples dramatically increases, ending up with a huge dataset. Thus, we generate the data relying on a single unstructured mesh. We believe that most of the solution features can still be captured, consequently taking into account the spatial variability. Therefore, data are collected from simulations run with the same mesh, whose elements have different shapes. A pictorial representation is provided in Figure 3.17.

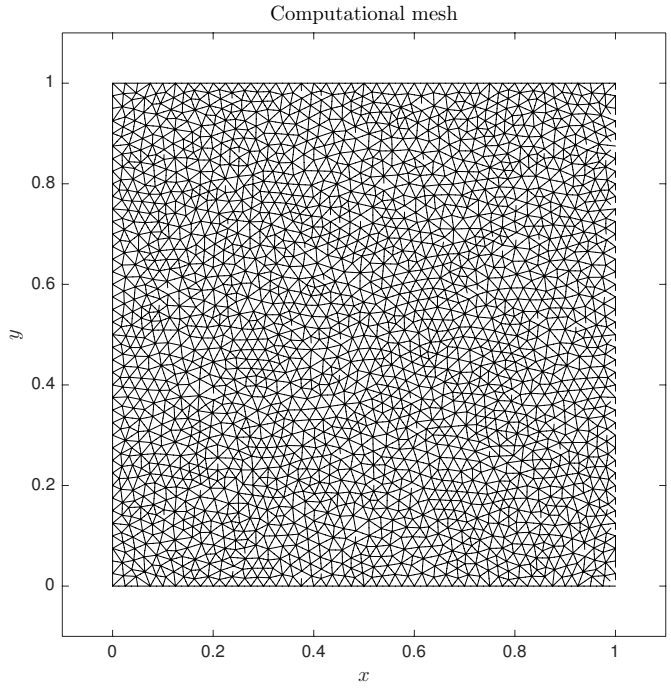


FIGURE 3.17: A graphical representation of the unstructured mesh we used to generate the two-dimensional dataset.

- The previous steps are again repeated using different conservation laws. In this two-dimensional framework, they are the extended version of Burgers equation and the linear advection problem. Again, all the major types of waves can develop from suitably defined initial conditions.
- Repeat the procedure for different discretization degrees m.

The post-processing phase does not significantly change from the one-dimensional framework. We make a single remark concerning the data balancing described in Subsection 3.2.7. Since we deal with a single mesh, there is no need to carry out the first shrinkage. In other words, all the data resulting from the simulations are kept in the training set. At the same time, even the second shrinkage is not mandatory. Indeed, we deal with few simulations, so that guaranteeing a balanced dataset is much easier and can be controlled by choosing final simulation times which do not affect the balancing itself. However, due to memory issues, we need to get rid of some samples. Keeping all the data, we would end up with a huge dataset, which is hard to handle without strong computing power. Thus, for a given simulation, we simply shuffle the data and keep only a certain percentage, which decreases as m is increased.

3.5.2 A two-dimensional example

To clarify the reasoning explained in the previous Subsection, we provide a quick example. Again, we choose $m = 3$ and we focus on the two-dimensional Burgers equation only. A summary of both the dataset generation and the post-processing step is reported in Table 3.4. Note that we keep only a randomly selected 10% of the data. The considered tests are characterized as follows.

Test nb.	T	Best model	Nb. samples	Shrink
1 (multiple)	0.12	MDH: $c_A = 2,\ c_\kappa = 0.5,\ c_{max} = 0.5$	19313712	1931371
2	0.12	EV: $c_E = 1,\ c_{max} = 0.5$	16763328	1676333
3	0.06	EV: $c_E = 1.5,\ c_{max} = 0.5$	6036336	603634
4	0.12	MDH: $c_A = 2.5,\ c_\kappa = 0.5,\ c_{max} = 1$	18416592	1841659

TABLE 3.4: A summary on how to generate the datasets for Burgers equation in a two-dimensional scenario.

- Test case nb. 1. It is a simple sine wave. Different frequencies ω are considered.

$$u_0(x, y) = \sin(\pi\omega x)\sin(\pi\omega y)\mathbb{1}_{[0,1]^2}(x, y)$$

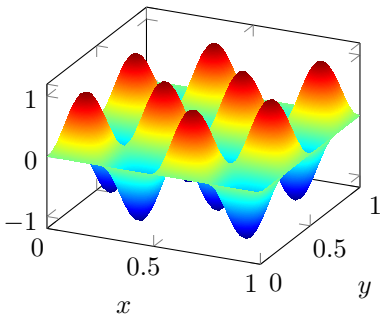


FIGURE 3.18: I.c. for test case nb 1.

- Test case nb. 2. A piecewise constant initial condition is considered.

$$u_0(x,y) = \begin{cases} 0 & \text{if } 0.5 < x < 1,\ 0.5 < y < 1, \\ 1 & \text{if } 0 < x < 0.5,\ 0.5 < y < 1, \\ 2 & \text{if } 0 < x < 0.5,\ 0 < y < 0.5, \\ 1 & \text{if } 0.5 < x < 1,\ 0 < y < 0.5. \end{cases}$$

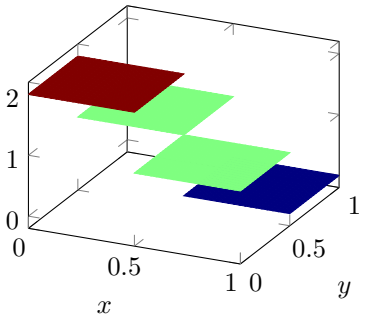


FIGURE 3.19: I.c. for test case nb 2.

- Test case nb. 3. In a way similar to the one-dimensional case, a function containing points of non-differentiability is considered here.

$$u_0(x,y) = 3 + 6(0.25 - |y - 0.5|)\mathbb{1}_{[0.25,0.75]}(y)$$

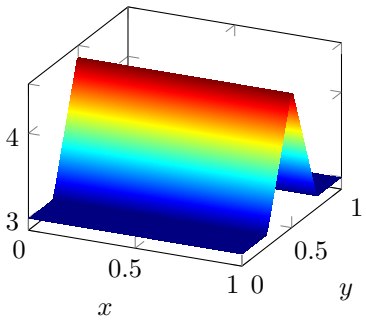


FIGURE 3.20: I.c. for test case nb 3.

- Test case nb. 4. It is a (negative) cosine bell which eventually forms a shock. It is needed in order to have negative samples.

$$u_0(r) = -2\left(1 + \cos\left(\frac{\pi}{0.5}r\right)\right)\mathbb{1}_{[0,0.5]}(r)$$

where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$.

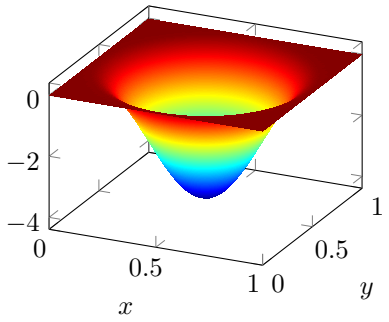


FIGURE 3.21: I.c. for test case nb 4.

Chapter 4

Practical implementation

In this Chapter we discuss the main algorithms needed to implement the technique described in Chapter 3. We provide the pseudo-codes, in order to make the strategy more understandable. Note that this Chapter adds no significant improvements to this work, and it can be skipped without any loss in the discussion.
The main procedure is given in Algorithm 4.1, which computes the numerical solution using the five-stage fourth-order low-storage Runge-Kutta scheme we described in Chapter 2. Note that the code can be split into three parts. The first, named *pre-processing* phase, is dedicated to the construction of the operators or storages which are constant in time. Examples are the mesh assembly (e.g. connectivity matrix, identify the neighboring elements and degrees of freedom, determine the boundary elements, ...) and the computation of some numerical operators, namely matrices (e.g. mass matrix, advection matrices, lifting operator, ...). The second, the *computation* step, contains the implementation of the time-advancing scheme. Most of the effort is spent in the computation of the right-hand-side, carried out as described in equations (2.14a) and (2.14b) (or in a compact form in equations (2.15a), (2.15b) and (2.15c)) for the one-dimensional case. Again, we observe that the artificial viscosity is estimated outside the Runge-Kutta internal loop, while the troubled-cell indicator is applied at each substep. The last phase, named *post-processing* step, is dedicated to the qualitative and quantitative analysis of the numerical solution. Since the implementation is similar to most of the time-dependent problems, we deliberately skip most of the details, which can be found in, e.g., [2].

Algorithm 4.1 Compute numerical solution.

Input: Problem definition (e.g. flux function $\boldsymbol{f}(\boldsymbol{u})$, initial condition $\boldsymbol{u}_0(\boldsymbol{x})$, boundary conditions, ...), discretization parameters (grid spacing h, final time T, discretization degree m,...).
Output: Numerical solution $\boldsymbol{u}$ at time T.

Mesh construction.
Compute the constant-in-time discretization operators.
Initialize $\boldsymbol{u}$ by means of the initial condition.
while $t < T$ **do**
  Compute artificial viscosity $\boldsymbol{\mu}$ according to a model $\mathcal{M}$.
  for $s \leftarrow 1, \dots, S_{RK} = 5$ **do**
    (Set $\mu = 0$ in the genuine - not troubled - cells).
    $\boldsymbol{v} \leftarrow A_s \boldsymbol{u} + \Delta t$ `compute_RHS`$(t + c_s \Delta t, \boldsymbol{u}, \boldsymbol{\mu})$.
    $\boldsymbol{u} \leftarrow \boldsymbol{u} + B_s \boldsymbol{v}$.
  end for
  $t = t + \Delta t$.
end while
Quantitative analysis of the numerical solution.

Instead, we focus on the algorithms related to artificial neural networks. Let us start by recalling how to generate the dataset. The whole procedure is summed up in Algorithm 4.2, which relies on Algorithm 4.3 to select the best artificial viscosity model for a given test case. The same strategy is repeated for all the discretization degrees $m \in \{1, \dots, 4\}$.

Once we construct the training sets, the network for each m is trained according to Algorithm 4.4. It is worth mentioning that the the implementation is carried out using TensorFlow, an open-source software

Algorithm 4.2 Generation of training and validation set for a given degree m.

Input: -
Output: Training set $\mathbb{T}$, Validation set $\mathbb{V}$.

Set $\mathbb{S} = \emptyset$.
for each type of equation **do**
  for each initial condition $u_0(x)$ **do**
    for each grid spacing h, final time T **do**
      Select the best artificial viscosity model (Algorithm 4.3).
      Run the simulation and add collected data in $\mathbb{S}$.
    end for
  end for

  Post-process the data in $\mathbb{S}$.
  Randomly shuffle $\mathbb{S}$ and assign the first 70% elements to $\mathbb{T}$ and the remaining 30% to $\mathbb{V}$.
end for

Algorithm 4.3 Select the best artificial viscosity model.

Input: Type of equation, initial condition, grid spacing, final time.
Output: Best model $\mathcal{M}_{best}$ and parameters $\mathcal{P}_{best}$.

$\mathcal{M}_{best} \leftarrow$ `NaN`, $\mathcal{P}_{best} \leftarrow$ `NaN`, $u_{\mathcal{M}_{best}} \leftarrow$ `NaN`.
for $\mathcal{M} \in$ {DB, MDH, MDA, EV} **do**
  $u_{\mathcal{M}} \leftarrow$ `NaN`, $\mathcal{P}_{\mathcal{M}} \leftarrow$ `NaN`.
  for each parameter $\mathcal{P}$ **do**
    Run the simulation with the selected model and parameters, obtaining $u_{\mathcal{M},\mathcal{P}}$.
    if $u_{\mathcal{M},\mathcal{P}} \succeq u_{\mathcal{M}}$ **then**
      $\mathcal{P}_{\mathcal{M}} \leftarrow \mathcal{P}$, $u_{\mathcal{M}} \leftarrow u_{\mathcal{M},\mathcal{P}}$
    end if
  end for
  if $u_{\mathcal{M}} \succeq u_{\mathcal{M}_{best}}$ **then**
    $\mathcal{M}_{best} \leftarrow \mathcal{M}$, $\mathcal{P}_{best} \leftarrow \mathcal{P}_{\mathcal{M}}$
  end if
end for

library for machine learning [50], which makes usage of *automatic differentiation* [51] to *differentiate the network*. In the context of machine learning, the technique used to calculate the gradients with respect to the weights or biases is known as *backpropagation*. Practically, it is a simple application of the standard multivariate *chain rule* to compute derivatives. More precisely, in order to compute the derivatives at a layer l, only the quantities at the following layer $l+1$ are needed. Thus, for a given input one simply computes the network output (*forward* pass) and updates the weights at all layers by using the error term, which is propagated backwards in the network (*backward* pass). Since we did not directly implemented this procedure, we do not provide all the details. Further information can be found in any machine learning text, such as [36, 52].

Algorithm 4.4 Train the Neural Network.

Input: Neural network $(\mathcal{N}, \mathcal{V})$, learning rate η, LReLU parameter α, cost function $\mathcal{C}$, training set $\mathbb{T}$, validation set $\mathbb{V}$, maximum number of epochs N_{epochs}, stopping parameter M, mini-batch size S_{batch}, number of restarts R.
Output: Optimal weights $\boldsymbol{W}_{opt}$ and biases $\boldsymbol{b}_{opt}$.

$V_{err}^{opt} = +\infty$.
for $r \leftarrow 1, \dots, R$ **do**
  $V_{err}^{r} = +\infty$, $l = 0$, $n = 1$.
  Randomly initialize $\boldsymbol{W}$ and $\boldsymbol{b}$ and set $V_{err}^{old} = +\infty$.
  while $n \leq N_{epochs}$, $l < L$ **do**
    Update weights and biases according to Algorithm 4.5.
    Evaluate validation error, say V_{err}^{new}.
    if $V_{err}^{new} > V_{err}^{old}$ **then**
      $l \leftarrow l + 1$.
    else
      $l \leftarrow 0$.
    end if
    $V_{err}^{old} \leftarrow V_{err}^{new}$, $n \leftarrow n + 1$.
  end while
  Evaluate final validation error, say V_{err}^{r} using weights and biases at epoch $n - l$.
  if $V_{err}^{r} < V_{err}^{opt}$ **then**
    Set $(\boldsymbol{W}_{opt}, \boldsymbol{b}_{opt})$ equal to weights and biases at epoch $n - l$.
  end if
end for

Algorithm 4.5 Minibatch optimization.

Input: Neural network $(\mathcal{N}, \mathcal{V})$, learning rate η, LReLU parameter α, cost function $\mathcal{C}$, training set $\mathbb{T}$, mini-batch size S_{batch}, number of restarts R, weights and biases to be updated $(\boldsymbol{W}, \boldsymbol{b})_{old}$.
Output: Updated weights and biases $(\boldsymbol{W}, \boldsymbol{b})_{new}$.

Randomly shuffle $\mathbb{T}$ and set $ids = 0$.
while $ids < |\mathbb{T}|$ **do**
  Construct a minibatch $\mathbb{B} = \mathbb{T}(ids + 1 : \min\{ids + S_{batch}, |\mathbb{T}|\})$.
  Perform an iteration of the optimization algorithm and update weights and biases, using data from $\mathbb{B}$.
  $ids \leftarrow ids + S_{batch}$.
end while

So far we focused on the *offline* phase only. The dataset generation, as well as the training procedure, is carried out only once, even if it is quite time-consuming. Once the optimal weights and biases are stored, the final user can exploit the trained network as a black box. The final Algorithm 4.6 describes the *online* stage, also known as *network evaluation*. It coincides with a single forward pass of the network for a given input which is not necessarily in the training set. We observe that the scaling for input and output variables is performed outside the algorithm, and it is not reported here.

At this point, is worth discussing the computational time of the ANN-based technique. As we already clarified, two stages are involved, with different costs. The offline phase requires a considerably large

Algorithm 4.6 Apply the Neural Network.

Input: Pointwise values for a variable x, in matrix form $\boldsymbol{X} \in \mathbb{R}^{m+1,K}$. Network weights $\{\boldsymbol{W}_i\}_{i=1}^{L+1}$ and biases $\{\boldsymbol{b}_i\}_{i=1}^{L+1}$. Parameter α for the LReLU function.
Output: Predicted values for a variable y, in matrix form $\boldsymbol{Y} \in \mathbb{R}^{m+1,K}$.

$\boldsymbol{X} = \boldsymbol{X}^T$.
for $i \leftarrow 1, \dots, L$ **do**
  $\boldsymbol{X} \leftarrow \boldsymbol{X}\boldsymbol{W}_i + \boldsymbol{b}_i$
  $\boldsymbol{X} \leftarrow \text{LReLU}(\boldsymbol{X}, \alpha)$
end for
▷ $i = L + 1$ corresponds to the output layer.
$\boldsymbol{X} \leftarrow \boldsymbol{X}\boldsymbol{W}_{L+1} + \boldsymbol{b}_{L+1}$
$\boldsymbol{Y} \leftarrow \text{Softplus}(\boldsymbol{X})$
$\boldsymbol{Y} = \boldsymbol{Y}^T$.

time. Suppose that the computational complexity to evaluate the updates Δw and Δb for weights and biases for a single mini-batch is denoted by θ. It depends on the optimization algorithm, the cost function, the mini-batch size and so on. The total complexity for a single epoch is therefore $\mathcal{O}(\theta N_{\mathbb{T}}/S_{batch})$, plus an additional cost to compute the error function using the validation set. Since several gradient computations are involved and $N_{\mathbb{T}}/N_{\mathbb{V}} = 7/3 > 2$, the former term dominates. Supposing that the stopping criterion is the maximum number of epochs N_{epochs} and the algorithm is restarted R times, the total offline cost is given by

$$\texttt{Offline\_cost} \sim \mathcal{O}\left(RN_{epochs}\theta\frac{N_{\mathbb{T}}}{S_{batch}}\right),$$

On the other hand, the online phase is computationally cheap, since mainly matrix-vector multiplications are performed. We recall that the cost required to multiply a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ by a vector $\boldsymbol{v} \in \mathbb{R}^n$ is $m\,(2n-1)$, while summing two vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ has a lower cost, which is neglected here. Thus, the cost for the propagation function is $\mathcal{O}(mn)$, and it has to be repeated for all the hidden layers plus the output layer. The activation functions have to be applied, too. Their cost can be estimated as mn and $mn\,(c_{exp} + 1 + c_{log})$ for the Leaky ReLU and Softplus respectively. Here, c_{exp} and c_{log} denote the computational cost to compute the exponential and the logarithmic functions respectively. Therefore the global cost of the online phase is

$$\begin{aligned}\texttt{Online\_cost} \sim\ & K((m+1)\cdot 10 + (m+1) + 4\cdot(10\cdot 10 + 10) \\ & + 10\cdot(m+1) + (m+1)\cdot(c_{exp} + 1 + c_{log})) \\ =\ & \mathcal{O}(K\cdot 6\cdot(10\cdot 10 + 10)) \\ =\ & \mathcal{O}(10^3\cdot K),\end{aligned}$$

where we consider $m + 1 = \mathcal{O}(10)$, as well as c_{exp} and c_{log}. Good values for c_{exp} and c_{log} are 40 and 20 flops respectively [53]. Clearly, it is way lower than the offline computational time. Finally, we note that the cost is linear in the number of mesh elements K.

Chapter 5

Numerical results

In this Chapter we provide several numerical results to show the capabilities of the proposed techniques. Both one-dimensional and two-dimensional problems are considered. In the first case, a comparison with a reference solution is shown. As described in Section 3.2, it is generated using the exact solution (if available) or a numerical one (using a very fine mesh and a high polynomial degree). The pseudo-analytical solution is not used, unless explicitly stated. The final Section of this Chapter is dedicated to a performance analysis.

5.1 One-dimensional scalar problems

Firstly, consider one-dimensional scalar problems. They represent a significant way to validate the performances of the proposed viscosity models. Basically, most of the techniques give good results for the considered problems.

5.1.1 A smooth problem

The first test we propose is used to assess the ability of the artificial viscosity models to maintain the expected solution accuracy. Consequently, a comparison among the standard techniques and the ANN-based one can be conducted. Let us consider the linear advection problem with a constant transport field $\beta = 1$. We choose $\Omega = [0, 1]$ as the computational domain, while the initial condition is a single period of a sine wave, i.e.

$$u_0(x) = B + A\sin(2\pi x),$$

where A and B are real numbers chosen equal to 1 and 2 respectively. The problem is completed by periodic boundary conditions. By following the characteristic method, one easily finds that the exact solution is given by

$$u(x,t) = u_0(x - \beta t) = B + A\sin(2\pi(x - t)).$$

To validate the scheme, we introduce a possible way to compute the discretization error. Several options are available, but for simplicity we focus on the spatial $L^2(\Omega)$-norm of the error vector evaluated at the final simulation time. Thus, let

$$\epsilon = \|u_h(\cdot, T) - u(\cdot, T)\|_{L^2(\Omega)}.$$

Since u has an analytical expression, ϵ can be easily computed. Given an element D^k, let $\boldsymbol{u}_h$, $\boldsymbol{u}$ be the finite dimensional vectors collecting the local degrees of freedom for the numerical and the (V_h^k-projection of the) analytical solution respectively. We again omit the superscript k to ease the

notation. Therefore,

$$\epsilon^2 = \|u_h(\cdot,T) - u(\cdot,T)\|^2_{L^2(\Omega)} = \sum_{k=1}^{K} \|u_h(\cdot,T) - u(\cdot,T)\|^2_{L^2(D^k)} = \sum_{k=1}^{K} \int_{D^k} (u_h|_k - u|_k)^2$$
$$= \sum_{k=1}^{K} \int_{D^k} \sum_{i,j=0}^{m} (u_{h,i} - u_i) l_i (u_{h,j} - u_j) l_j = \sum_{k=1}^{K} (\boldsymbol{u}_h - \boldsymbol{u})^T \boldsymbol{M} (\boldsymbol{u}_h - \boldsymbol{u}),$$

where $\boldsymbol{M}$ is the (internal) mass matrix defined in (2.13). The last equality holds thanks to linearity of the integral operator. Theoretical results for hyperbolic problems [5, 30] suggest that the following optimal estimate holds:

$$\epsilon \leq C_1 h^{m+1} (1 + C_2 T), \tag{5.1}$$

provided that u is sufficiently smooth. The constants C_1, C_2 depend on the discretization degree m. Note that the estimate (5.1) holds for the inviscid problem, i.e. with $\mu = 0$. Here, we are instead adding an extra dissipation term. Theoretically, we would like to keep the same optimal convergence rate by adding a dissipative term which preserves the scaling. As we have already observed in Section 2.2, that is not always the case. In order to numerically estimate the convergence rate, we compute the discretization error using two different meshes with characteristic mesh sizes h_1, h_2 (or equivalently with number of elements equal to K_1, K_2). Then, the estimated order, denoted by p, is computed as

$$p \simeq \frac{\log(\epsilon_1) - \log(\epsilon_2)}{\log(h_1) - \log(h_2)} = \frac{\log(\epsilon_1) - \log(\epsilon_2)}{\log(K_2) - \log(K_1)}.$$

We report the results obtained at time $T = 0.2$ for different mesh sizes with degrees $m = 1$ (Table 5.1), $m = 2$ (Table 5.2), $m = 3$ (Table 5.3) and $m = 4$ (Table 5.4). The parameters we adopt here are not meant to be the optimal ones for this specific problem. They affect the magnitude of the error, but the behavior is not drastically altered by the parameter values, at least from a qualitative point of view.

K	Inviscid		DB		EV		MDH	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.3386e−2	-	2.8271e−1	-	3.1848e−1	-	1.9595e−1	-
20	3.3576e−3	1.99	1.1451e−1	1.30	8.9574e−2	1.83	1.1561e−1	0.76
40	8.3953e−4	2.00	3.7073e−2	1.63	1.3176e−2	2.77	7.1512e−2	0.69
80	2.0987e−4	2.00	1.0390e−2	1.83	2.1677e−3	2.60	4.2553e−2	0.75
160	5.2465e−5	2.00	2.6940e−3	1.94	3.0662e−4	2.82	2.4201e−2	0.81
320	1.3116e−5	2.00	6.8108e−4	1.98	4.1720e−5	2.88	1.3184e−2	0.88

TABLE 5.1: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models. Linear advection problem, $m = 1$.

K	Inviscid		DB		EV		MDH	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.0519e−3	-	1.2734e−1	-	6.2039e−2	-	1.0519e−3	-
20	1.3298e−4	2.98	3.8678e−2	1.72	3.8533e−3	4.00	1.3298e−4	2.98
40	1.6664e−5	3.00	1.0582e−2	1.87	2.9406e−4	3.71	1.6664e−5	3.00
80	2.0844e−6	3.00	2.7157e−3	1.96	1.9935e−5	3.88	2.0844e−6	3.00
160	2.6059e−7	3.00	6.8357e−4	1.99	1.3089e−6	3.92	2.6059e−7	3.00
320	3.2575e−8	3.00	1.7119e−4	2.00	8.7767e−8	3.81	3.2575e−8	3.00

TABLE 5.2: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models. Linear advection problem, $m = 2$.

As expected, the DB method has maximum second order accuracy. For coarse meshes, the first order term dictated by μ_{max} comes into play, so that orders lower than two are found. Thus, the expected order of convergence is obtained asymptotically as $h \to 0$. The EV scheme is able to to achieve high accuracy orders, even though the numerical error is obviously larger with respect to the inviscid problem. Some issues seem to arise for $m > 3$, possibly due to the Crank-Nicolson discretization of the

K	Inviscid		DB		EV		MDH		MDA	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	3.1021e−5	-	6.3532e−2	-	3.4710e−4	-	3.1021e−5	-	8.3751e−2	-
20	2.2845e−6	3.76	1.8334e−2	1.79	1.3637e−5	4.67	2.2845e−6	3.76	4.3327e−2	0.95
40	1.5260e−7	3.90	4.7945e−3	1.94	4.9179e−7	4.79	1.5260e−7	3.90	2.2260e−2	0.96
80	9.3750e−9	4.02	1.2131e−3	1.98	1.7742e−8	4.79	9.3750e−9	4.02	1.1304e−2	0.99
160	5.8609e−10	4.00	3.0420e−4	2.00	7.5682e−10	4.55	5.8609e−10	4.00	5.6980e−3	0.99
320	3.6631e−11	4.00	7.6108e−5	2.00	4.0744e−11	4.22	3.6631e−11	4.00	2.8609e−3	0.99

TABLE 5.3: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models. Linear advection problem, $m = 3$.

K	Inviscid		DB		EV		MDH		MDA	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	9.9474e−7	-	3.8582e−2	-	1.4982e−4	-	9.9474e−7	-	9.9474e−7	-
20	3.1481e−8	4.98	1.0581e−2	1.87	3.6170e−6	5.37	3.1481e−8	4.98	3.1481e−8	4.98
40	1.0073e−9	4.97	2.7157e−3	1.96	1.1971e−7	4.92	1.0073e−9	4.97	1.0073e−9	4.97
80	3.3036e−11	4.93	6.8358e−4	1.99	5.7002e−9	4.39	3.3036e−11	4.93	3.3036e−11	4.93
160	1.0925e−12	4.92	1.7119e−4	2.00	3.3235e−10	4.10	1.0925e−12	4.92	1.0925e−12	4.92

TABLE 5.4: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models. Linear advection problem, $m = 4$.

time derivative in (2.33a). The MDH model has an interesting behavior. For low degrees ($m = 1$), the scheme has order less than one. Indeed, it adds a viscosity which scales linearly with h, thus preventing higher orders. On the other hand, for high orders ($m \geq 2$), the shock sensor detects the solution as regular enough not to add dissipation. Thus we return to the inviscid scheme, which has optimal accuracy. A similar reasoning holds for the MDA model.
We focus now on the estimated convergence rates using the ANN-based method. The results are reported in Table 5.5 for $m = 1$, Table 5.6 for $m = 2$, Table 5.7 for $m = 3$ and Table 5.8 for $m = 4$. They are obtained using the third strategy defined in Table 3.1, which turns out to be the most efficient one for general cases. The first one produces an inconsistency, since no scaling with the local wave speed is carried out, while the second one completely ignores the solution features, generating a uniform-in-space viscosity value. We will provide a more concrete example in Subsection 5.1.4, which completely clarifies the reason for our choice. We report the values obtained using the original version and the two proposed improvements, namely both using an ANN as troubled-cell indicator and using a different scaling.

K	Inviscid		NN (orig.)		NN ($\times$ 2)		NN (scaling)	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.3386e−2	-	1.1163e−1	-	3.3173e−2	-	4.8874e−2	-
20	3.3576e−3	1.99	5.0214e−2	1.10	3.3576e−3	3.30	1.1627e−2	2.07
40	8.3953e−4	2.00	2.2317e−2	1.22	8.3953e−4	2.00	3.4402e−3	1.76
80	2.0987e−4	2.00	8.3278e−3	1.42	2.0987e−4	2.00	5.7626e−4	2.58
160	5.2465e−5	2.00	1.9344e−4	5.43	5.2465e−5	2.00	5.2756e−5	3.45
320	1.3116e−5	2.00	2.6979e−5	2.84	1.3116e−5	2.00	1.3126e−5	2.00

TABLE 5.5: L^2 convergence errors and estimated rate in the inviscid case and using the ANN-based method. Linear advection problem, $m = 1$.

The original version of the ANN method is not able to get high order accuracy, due to the linear scaling in the element size. For high orders, its asymptotic convergence rate is one. Spurious effects are present in the linear case $m = 1$, possibly due to insufficient mesh resolution. Note that for $m \geq 2$ the absolute value of the error is usually much larger compared to the inviscid problem, at least with a sufficiently fine mesh. The proposed improvements seem to overcome these issues. Theoretically, the ANN-based troubled-cell indicator should flag no elements, since the solution is smooth. This is the observed behavior, even though for very low resolutions some cells might be flagged as troubled. Finally, the new scaling relying on the solution jump gives error values which are almost equal to the inviscid problem, together with an optimal rate. The estimated rate for $m = 1$ approaches the optimal

K	Inviscid		NN (orig.)		NN ($\times$ 2)		NN (scaling)	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.0519e−3	-	8.8847e−3	-	1.0519e−3	-	1.0586e−3	-
20	1.3298e−4	2.98	1.5650e−3	2.42	1.3298e−4	2.98	1.3383e−4	3.02
40	1.6664e−5	3.00	3.1335e−4	2.40	1.6664e−5	3.00	1.6684e−5	3.00
80	2.0844e−6	3.00	1.0179e−4	1.62	2.0844e−6	3.00	2.0854e−6	3.00
160	2.0659e−7	3.00	4.4963e−5	1.18	2.0659e−7	3.00	2.6069e−7	3.00
320	3.2575e−8	3.00	2.1500e−5	1.06	3.2575e−8	3.00	3.2587e−8	3.00

TABLE 5.6: L^2 convergence errors and estimated rate in the inviscid case and using the ANN-based method. Linear advection problem, $m = 2$.

K	Inviscid		NN (orig.)		NN ($\times$ 2)		NN (scaling)	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	3.1021e−5	-	2.8484e−3	-	3.1021e−5	-	3.1263e−5	-
20	2.2845e−6	3.76	5.4942e−4	2.37	2.2845e−6	3.76	2.2864e−6	3.77
40	1.5260e−7	3.90	2.0007e−4	1.46	1.5260e−7	3.90	1.5268e−7	3.90
80	9.3750e−9	4.02	6.8944e−5	1.54	9.3750e−9	4.02	9.3778e−9	4.03
160	5.8609e−10	4.00	2.8957e−5	1.25	5.8609e−10	4.00	5.8621e−10	4.00
320	3.6631e−11	4.00	1.2258e−5	1.29	3.6631e−11	4.00	3.6687e−11	4.00

TABLE 5.7: L^2 convergence errors and estimated rate in the inviscid case and using the ANN-based method. Linear advection problem, $m = 3$.

K	Inviscid		NN (orig.)		NN ($\times$ 2)		NN (scaling)	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	9.9474e−7	-	1.1133e−3	-	9.9474e−7	-	9.9572e−7	-
20	3.1481e−8	4.98	1.7106e−4	2.70	3.1481e−8	4.98	3.1487e−8	4.98
40	1.0073e−9	4.97	7.8974e−5	1.12	1.0073e−9	4.97	1.0075e−9	4.97
80	3.3036e−11	4.93	3.2356e−5	1.29	3.3036e−11	4.93	3.3040e−11	4.93
160	1.0925e−12	4.92	1.5723e−5	1.04	1.0925e−12	4.92	1.0927e−12	4.92

TABLE 5.8: L^2 convergence errors and estimated rate in the inviscid case and using the ANN-based method. Linear advection problem, $m = 4$.

one only asymptotically.
The convergence analysis motivates the choice to consider only the third NN version (i.e. with the improved scaling) for all the following examples, unless stated otherwise.

5.1.2 Burgers equation: a simple test

After validating the new technique, we need to test its ability to capture shocks, adding a larger amount of dissipation in spatial regions where discontinuities develop. We do it first with Burgers equation. The motivation lies in the fact that smooth initial conditions may lead to discontinuities (shock and rarefaction waves), due to a nonlinear flux function. Moreover, since the training set was generated using mainly this conservation law, this test represents a simple benchmark case to test the generalization properties of the ANN to initial conditions which were not included in the training set. Consider $\Omega = [0, 1]$, divided in $K = 160$ equal intervals and a final time $T = 0.4$. The initial condition is a compactly supported sine wave, that is set as

$$u_0(x) = -\sin(6\pi x)\mathbb{1}_{[1/6,5/6]}(x),$$

observing that its frequency is different from the ones considered in the dataset. The initial condition is shown in Figure 5.1, while the problem is completed with periodic (or Dirichlet) boundary conditions. By following the characteristic method, one observes that two shocks are formed at $x = 1/3$ and

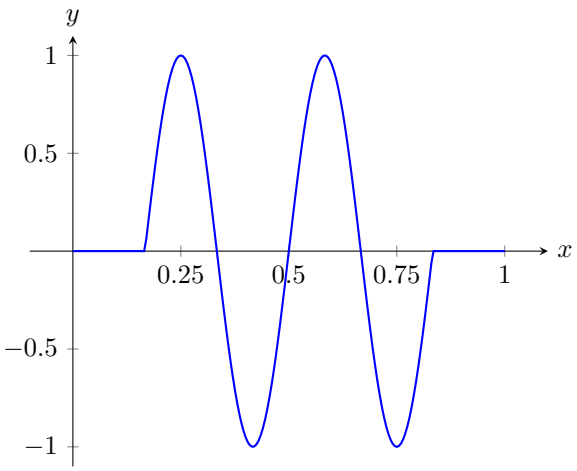


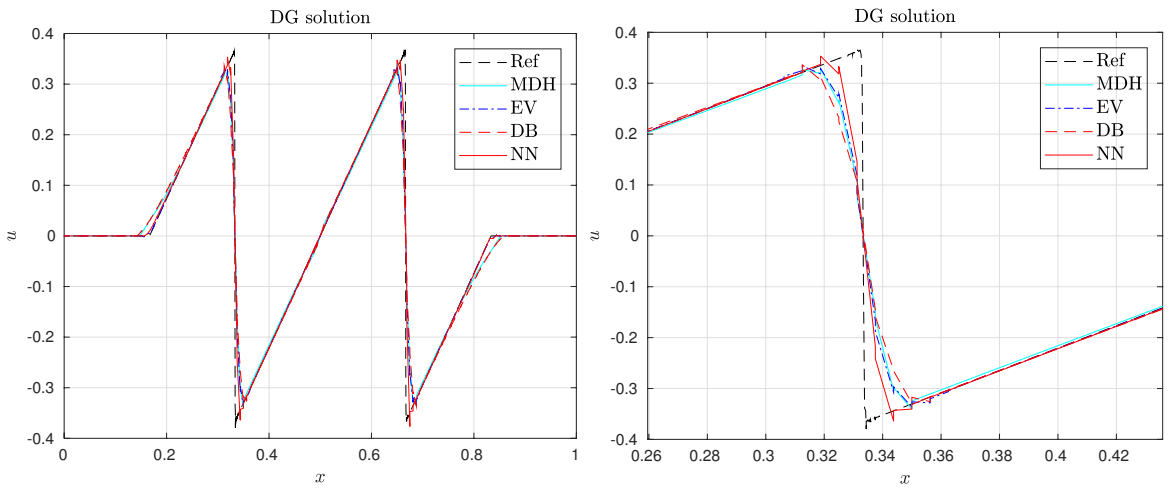
FIGURE 5.1: Initial condition for the second test case, Burgers equation.

$x = 2/3$, after a time $t = 1/(6\pi)$. Let us consider linear basis functions first, namely $m = 1$. The parameter values for the standard viscosity methods are reported in the corresponding column of Table 5.9, which have been tuned to be (pseudo-)optimal for this specific test case. The numerical results are reported in Figure 5.2.

Model	$m = 1$	$m = 4$
DB	$c_\beta = 3,\ c_{max} = 1.5$	$c_\beta = 2,\ c_{max} = 1$
EV	$c_E = 2,\ c_{max} = 1$	$c_E = 1,\ c_{max} = 0.5$
MDH	$c_A = 2.5,\ c_\kappa = 0.4,\ c_{max} = 1$	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.5$
MDA	-	$c_{max} = 1$

TABLE 5.9: Parameter values for the standard artificial viscosity models for the second test case.

By looking at the overall solution, no significant differences among the models are present. In particular, the numerical oscillations caused by the shocks are smoothed enough. Zooming close to the first shock, it appears that the network outperforms the other models, in the sense that the solution seems to be sharper. In other words, the added dissipation appears to be minimal with respect to the other cases. As expected, the drawback lies in the fact that overshoots and undershoots have a marginally larger amplitude. However, both appear to be controlled enough. Of course, where the solution is smooth,



(a) Overall result.

(b) Zoom close to the first shock.

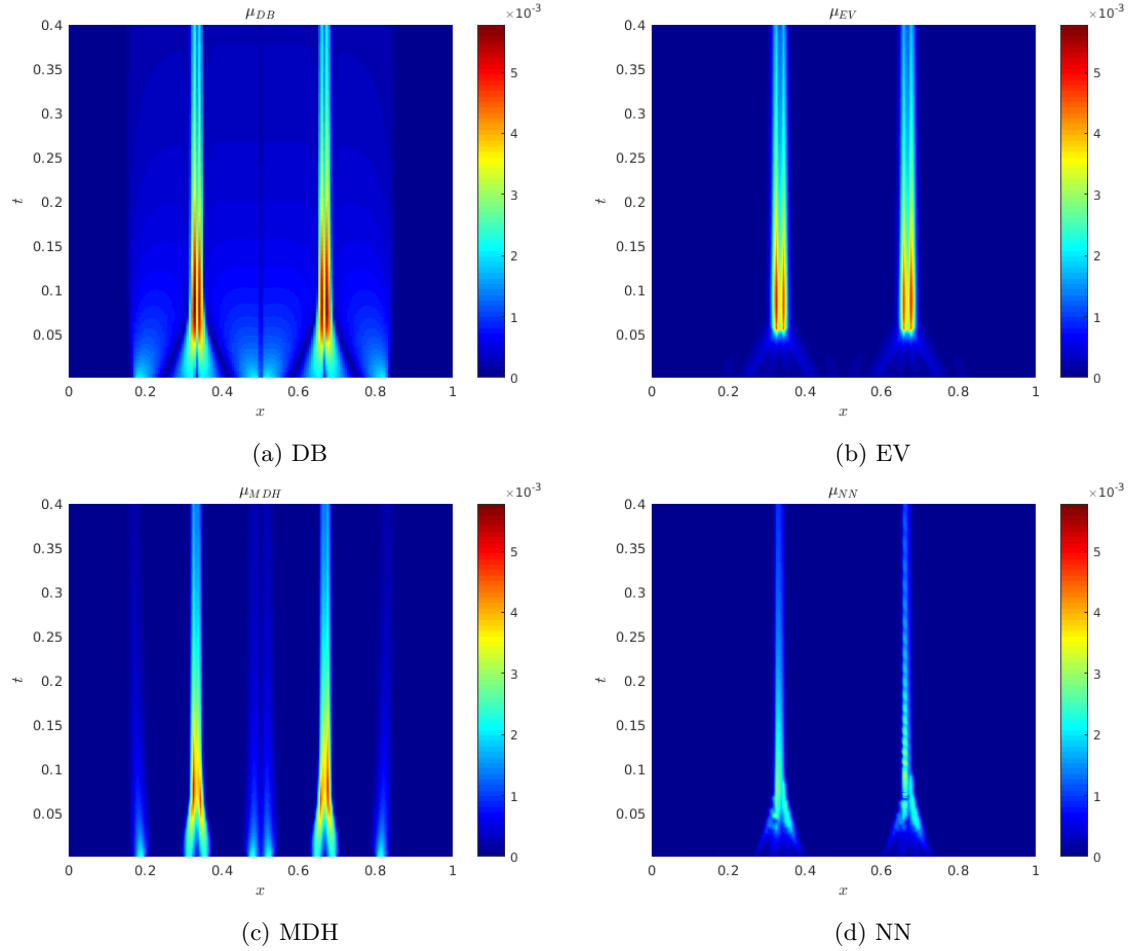
FIGURE 5.2: Numerical results for the second test case, Burgers equation, $m = 1$.

all the models follow the exact solution profile. To provide a more quantitative analysis, in Figure 5.3 we report the temporal history of artificial viscosity for all the models.

Theoretically, the viscosity models should not add dissipation at the initial times, where the solution is still smooth. After the shocks forms, viscosity has to be added in the spatial regions close to the discontinuities. Among the standard models, only the EV one achieves this goal. The DB model, due to its second-order nature, adds a viscosity which is less localized in space. On the other hand, the MDH should be capable of recognizing the solution at initial instants as smooth enough not to add dissipation. This failure might depend on the choice of the parameters, but mostly on the fact that a low discretization degree is used. This is probably not high enough to trigger the whole potential of the MDH model. The NN technique adds a localized viscosity in space, with the maximum value appearing to be lower compared to all the other models, at least from a global perspective.
A similar analysis can be carried out using higher degrees, like $m = 4$. The parameters for the standard models are reported in the corresponding column of Table 5.9. The solution, together with a zoom close to one of the discontinuities, is reported in Figure 5.4, while the corresponding viscosity profile is shown in Figure 5.5.

The qualitative behavior for the solution is similar to the linear case. In Figure 5.4(b) we observe that the profile obtained with the MDH and EV models is essentially the same. Since they are the two main models we relied on during the construction of the training set, the network is mimicking their behavior, so that NN-based solution is very close to both of them. Concerning the profile for the artificial viscosity, we note that all the models except the DB recognize the initial smoothness, and consequently they add essentially no dissipation before the shock appears. Again, the derivative-based technique continues adding dissipation in regions where it is not strictly necessary. The NN method seems to add an *oscillating* amount of dissipation, which is also observed (not reported here) for the some standard models (the MDH and the EV) for certain values of the parameters. This is also confirmed by looking at the maximum amount of dissipation in space, which is reported in Figure 5.6 as a function of time.

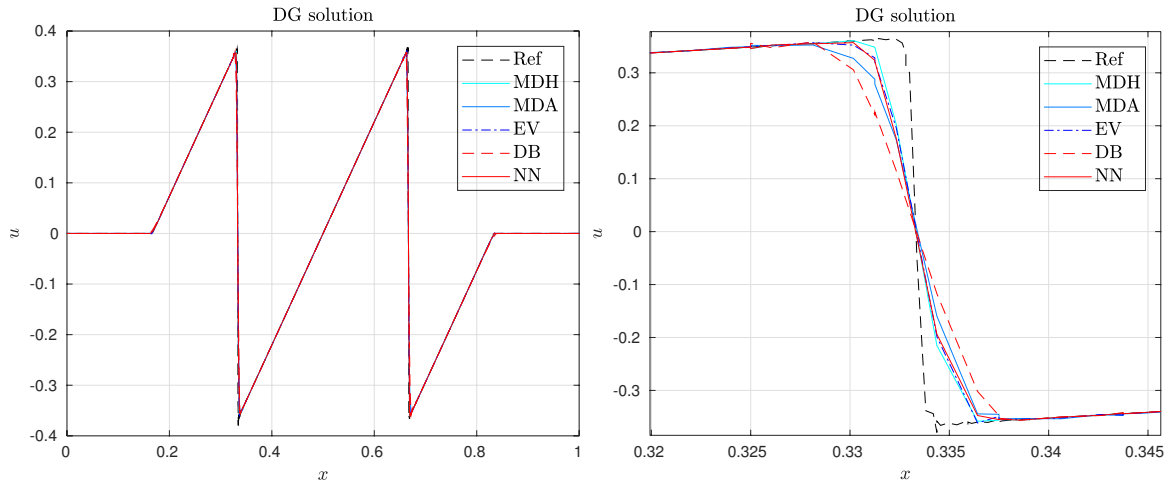
A possible explanation is found by noting that viscosity is added only when oscillations become too strong, while a lower amount is injected when their amplitude is not large enough. This oscillatory behavior is not a critical issue and it is perfectly acceptable. However, it might have a marginal effect on the time step, which is set proportionally to the inverse of the maximum viscosity.
In general, it is evident that the solution quality improves from $m = 1$ to $m = 4$ and the region where artificial viscosity is added is thinner. The absolute values of the dissipation values are lower, too. A final remark is related to the *symmetry breaking*, which is more evident as the discretization degree increases. In particular, one may expect a symmetry with respect to $x = 0.5$. However, since we use $K = 160$ elements and $m + 1 = 5$ nodal values per cell, one can verify that the two shocks fall in different locations within the cell. In particular $x = 1/3$ falls between the second and the third node in the 54^{th} cell, while $x = 2/3$ lies between the third and the fourth node of the 107^{th} element. Thus, it



(a) DB (b) EV

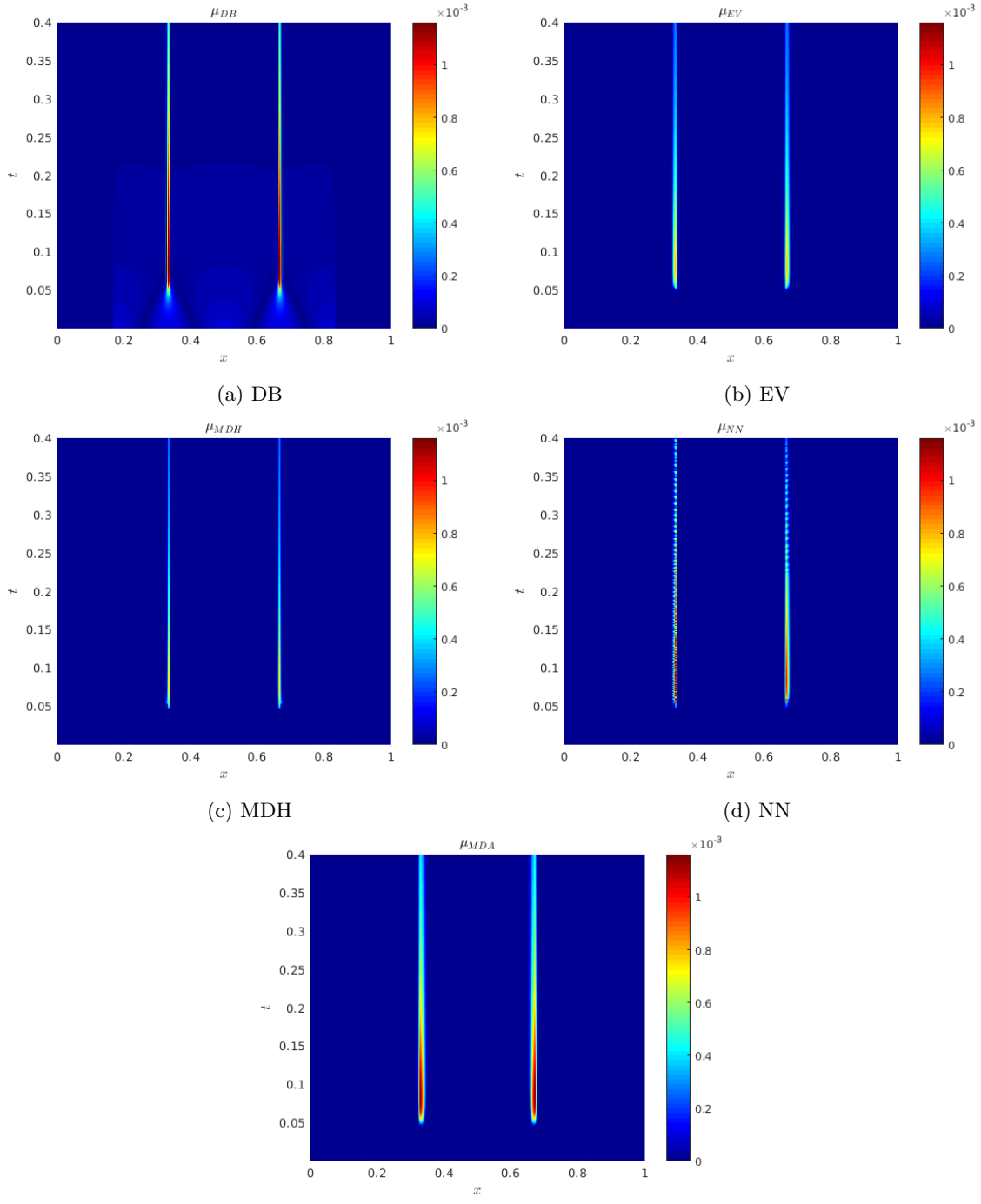
(c) MDH (d) NN

FIGURE 5.3: Temporal history of the artificial viscosity for the second test case, Burgers equation, $m = 1$.



(a) Overall result. (b) Zoom close to the first shock.

FIGURE 5.4: Numerical results for the second test case, Burgers equation, $m = 4$.



(a) DB

(b) EV

(c) MDH

(d) NN

(e) MDA

FIGURE 5.5: Temporal history of the artificial viscosity for the second test case, Burgers equation, $m = 4$.

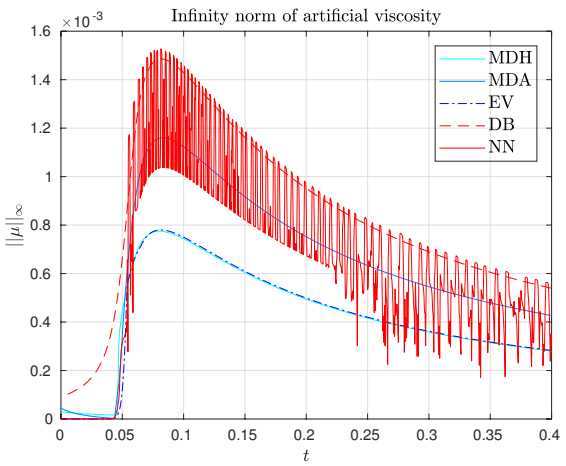


FIGURE 5.6: Infinity norm of the artificial viscosity with respect to time, second test case, Burgers equation, $m = 4$.

makes sense to expect a slightly different response from the network, since such symmetries have not been taken into account in the training set.

5.1.3 Burgers equation: a compound wave

The test case we proposed in Subsection 5.1.2 had an initial condition similar to those in the training set. Here, we show that the network is able to provide good results even for more general functions. Consider $\Omega = [-4, 4]$, divided in $K = 200$ equal intervals and a final time $T = 0.4$. We define the initial condition as a composition of smooth and discontinuous data [11] as

$$u_0(x) = \begin{cases} \sin(\pi x) & \text{if } 1 \leq |x| \leq 4, \\ 3 & \text{if } -1 < x \leq -0.5 \text{ or } 0 < x \leq 0.5, \\ 1 & \text{if } -0.5 < x < 0, \\ 2 & \text{if } 0.5 < x \leq 1, \end{cases}$$

while the problem is completed with periodic boundary conditions. The initial condition is plotted in Figure 5.7. Note that the domain, the number of elements, the grid spacing and the initial condition were not included in the training set.

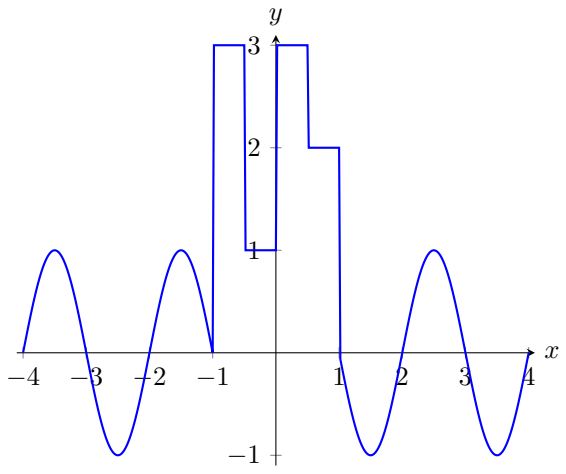


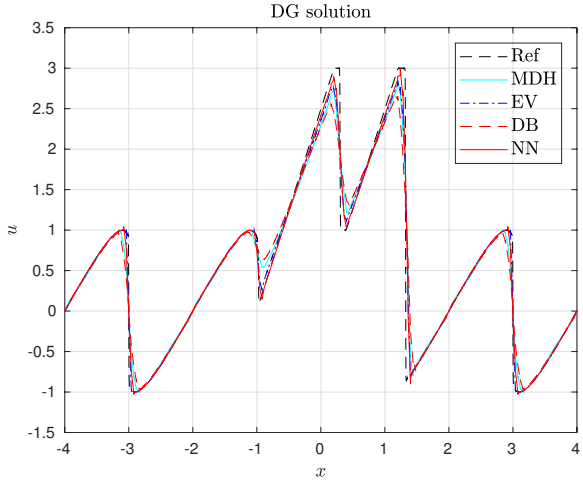
FIGURE 5.7: Initial condition for the third test case, Burgers equation.

Finally, the parameter we adopt for the standard model are reported in Table 5.10.

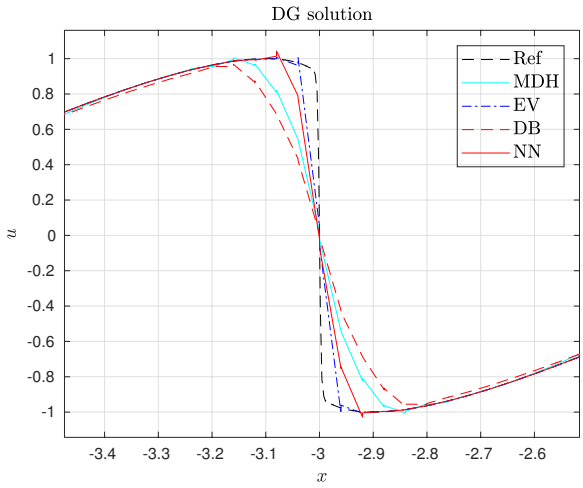
Model	$m = 1$	$m = 4$
DB	$c_\beta = 4,\ c_{max} = 2$	$c_\beta = 2,\ c_{max} = 1$
EV	$c_E = 2.5,\ c_{max} = 1.5$	$c_E = 2,\ c_{max} = 1$
MDH	$c_A = 2.6,\ c_\kappa = 0.4,\ c_{max} = 1$	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.5$
MDA	-	$c_{max} = 1$

TABLE 5.10: Parameter values for the standard artificial viscosity models for the third test case.

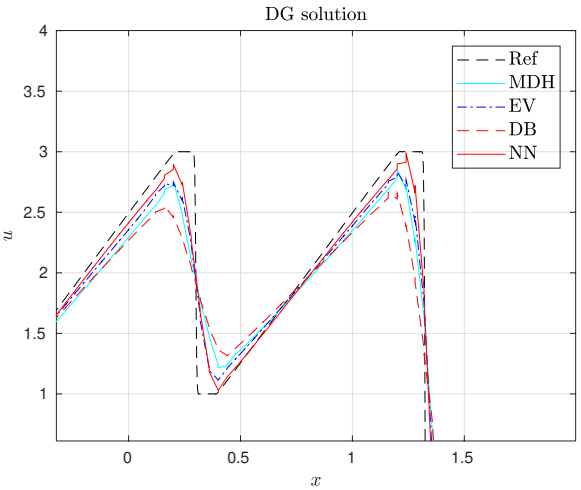
As the solution evolves, both shocks and rarefaction waves develop. We report the results obtained for low ($m = 1$) and high ($m = 4$) degrees in Figure 5.8 and Figure 5.9 respectively, while the history of the viscosity is shown, only for the latter case, in Figure 5.10.



(a) Overall result.



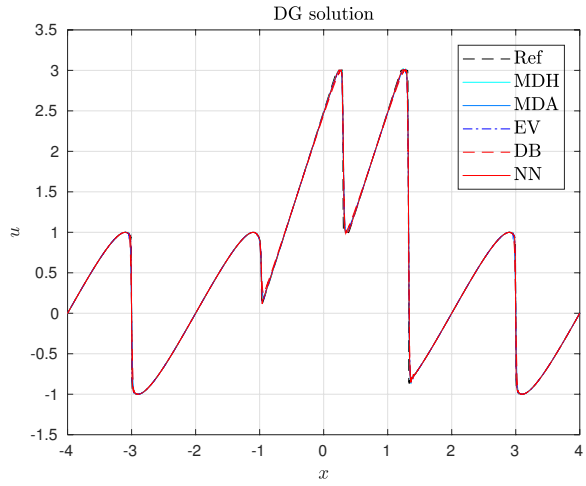
(b) Zoom close to the first shock.



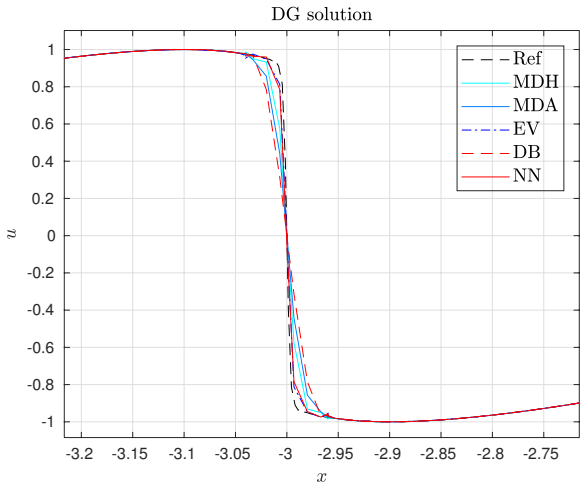
(c) Zoom close to the middle region.

FIGURE 5.8: Numerical results for the third test case, Burgers equation, $m = 1$.

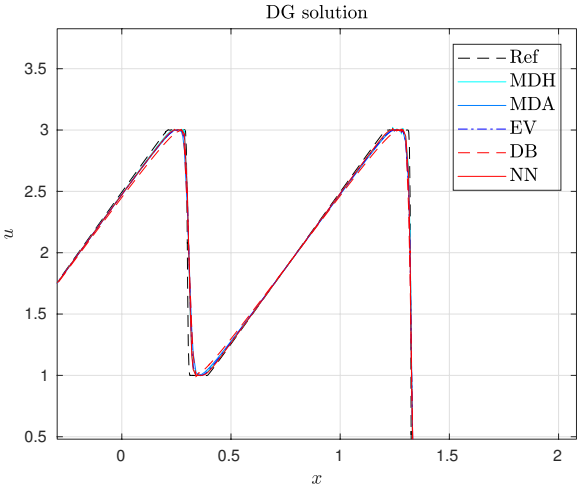
Again, we observe that the overall results obtained with all the considered models are similar. In the linear case, the EV and the NN models appear to provide better results. In particular, the former gives a better resolution of the shock originating from the sine wave, located at $x = -3$. Here, the NN model outperforms both the DB and the MDH ones, but it creates a low-amplitude overshoot. On the other hand, in the middle region of the solution profile, the NN model appears to resolve the solution features better than the others. Small oscillations are present, but the associated peaks never exceed the bounds provided by the reference solution. Increasing the polynomial order, the differences in all the models become less evident, even though the EV and the NN models are the best performers. Note that in this scenario, the NN resolves the first shock as accurately as the EV model.



(a) Overall result.

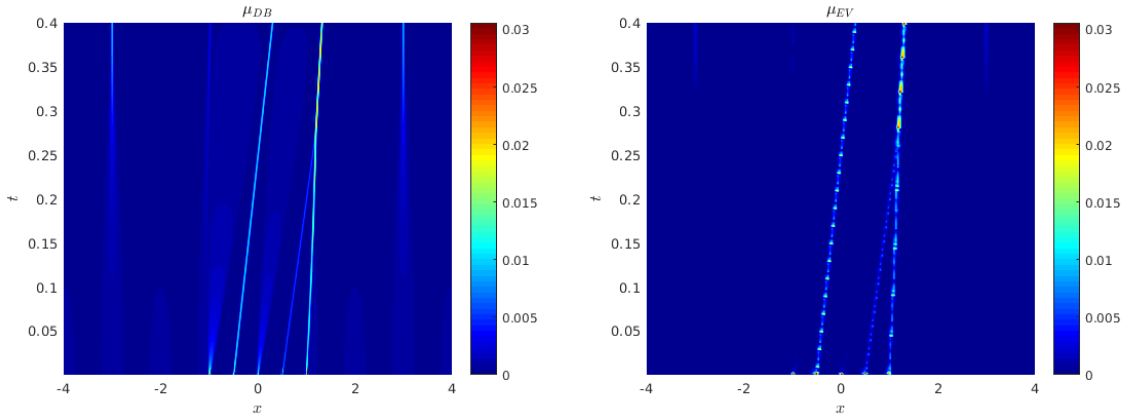


(b) Zoom close to the first shock.



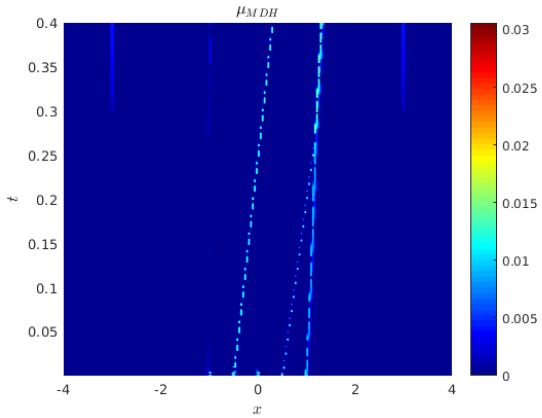
(c) Zoom close to the middle region.

FIGURE 5.9: Numerical results for the third test case, Burgers equation, $m = 4$.

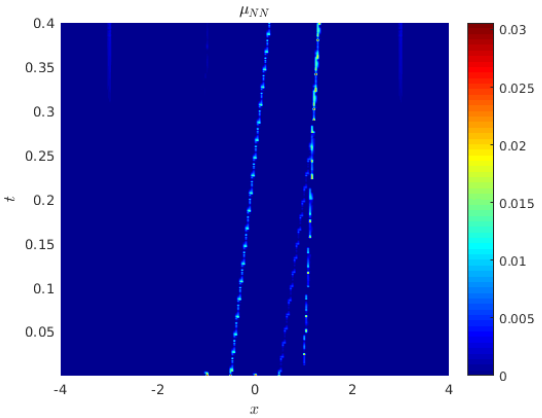


(a) DB

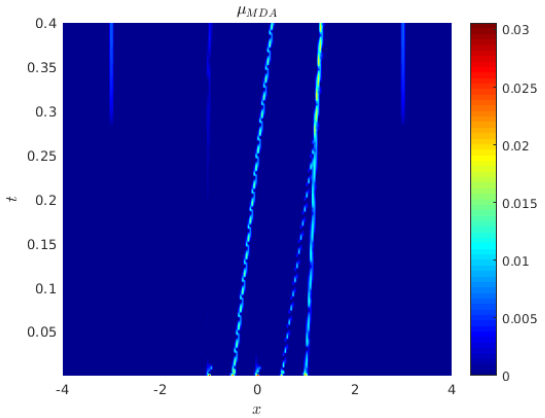
(b) EV



(c) MDH



(d) NN



(e) MDA

FIGURE 5.10: Temporal history of the artificial viscosity for the third test case, Burgers equation, $m = 4$.

By looking at the temporal history of the artificial viscosity, the oscillatory evolution of the dissipation values is evident for the MDH, EV and NN models. This can be perhaps explained by observing that the shocks have no fixed location in space, but they move at a certain speed. Thus, a different behavior is found when the discontinuity is located close to the extrema of the intervals or in the inner part. Again, we observe that the DB method is the most dissipative model.

5.1.4 A degree-4 flux function

The previous tests were mainly needed as a validation for our technique. In particular, we tested the convergence rate of the discretization error and the ability to capture shocks, as well as to resolve the solution features at different mesh resolutions. Now, we switch to conservation laws defined by flux functions which were not included in the training set. We start by choosing $f(u) = \frac{u^4}{4}$, in the domain $\Omega = [0, 1]$ with $K = 160$ elements. The initial condition is simply a `rect` function defined as

$$u_0(x) = \begin{cases} 1 & \text{if } 0 < x \leq 0.25 \text{ or } 0.75 < x \leq 1, \\ 3 & \text{otherwise in } [0, 1], \end{cases}$$

while the problem is completed using Dirichlet boundary conditions. Since the flux function is convex, its behavior is not far from Burgers equation. In particular, shocks and rarefaction waves develop. For small times, the analytical solution can be computed exactly using the characteristic method:

$$u(x,t) = \begin{cases} 1 & \text{if } 0 < x \leq 0.25 + 1t \text{ or } 0.75 + \dot{s}t < x \leq 1, \\ 3 & \text{if } 0.25 + 27t < x \leq 0.75 + \dot{s}t, \\ \mathcal{R}\left(\frac{x-0.25}{t}\right) & \text{if } 0.25 + 1t < x \leq 0.25 + 27t, \end{cases}$$

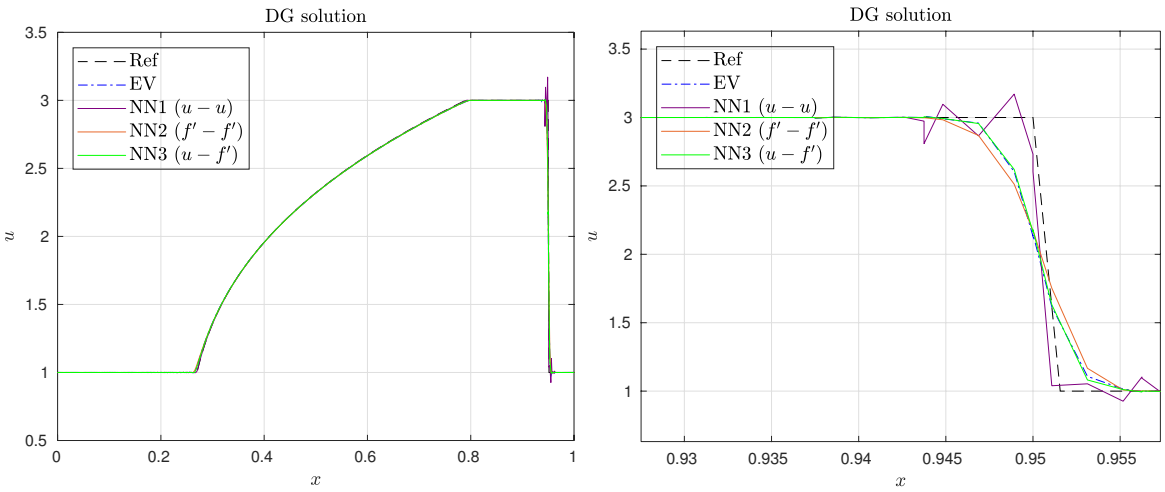
where

$$\dot{s} = \frac{f(1) - f(3)}{1 - 3} = 10$$

is the shock speed obtained through the Rankine-Hugoniot condition [54], and $\mathcal{R}(\cdot) = (f')^{-1}(\cdot) = (\cdot)^{1/3}$ is the inverse function of the first derivative of the flux function. Note that $\mathcal{R}$ exists due to the strict convexity of the flux function $f(u)$ within the considered solution range. The previous reasoning holds as long as the shock and rarefaction waves do not meet, namely for $t \leq 1/34 \simeq 0.029$. For successive instants, one could still rely on the characteristic method, but a simple analytical expression is no longer available, since a nonlinear ordinary differential equation has to be solved. However, for our purposes it is enough to consider $T = 0.02 < 1/34$.
Before showing the numerical results, it is worth spending a word on the choice of the three network strategies reported in Table 3.1. Indeed, for Burgers equation we have $f'(u) = u$, so that all the previous techniques collapse into a single one. Here we have $f'(u) = u^3$, so that significant differences start to appear among the proposed strategies, as we partially noted even for the linear advection problem analyzed in Subsection 5.1.1. In Figure 5.11 we show the results obtained with the three different approaches.

Clearly, the first strategy underestimates the required viscosity. This is expected, since the inverse scaling is done by multiplying by a scaling factor of $u \in [1, 3]$ instead of $u^3 \in [1, 27]$. In other words, the first strategy predicts a viscosity value which is approximately 9 times smaller compared to the others, at least in regions where u is close to its maximum value. Thus, oscillations are observed. On the other hand, the second and third strategies are similar, although the latter resolves the shocks slightly better. Moreover, as we already pointed out, the second approach is not well suited for linear advection problems and the Euler system. Therefore it makes sense to consider the third technique as the best one, together with the improved version with the $\mathcal{H}$-scaling. Our reasoning is confirmed by looking at the artificial viscosity added by the models. From Table 5.11, which reports the time-averaged infinity norm of μ, it is evident that the dissipation added with the first strategy is clearly not enough to stabilize the problem.

Finally, we carry out an analysis similar to the previous test cases, with parameters defined in Table 5.12. For simplicity, in Figure 5.12 and Figure 5.13 we show only the results for $m = 4$, in comparison with the EV model.



(a) Overall results. (b) Zoom close to the shock.

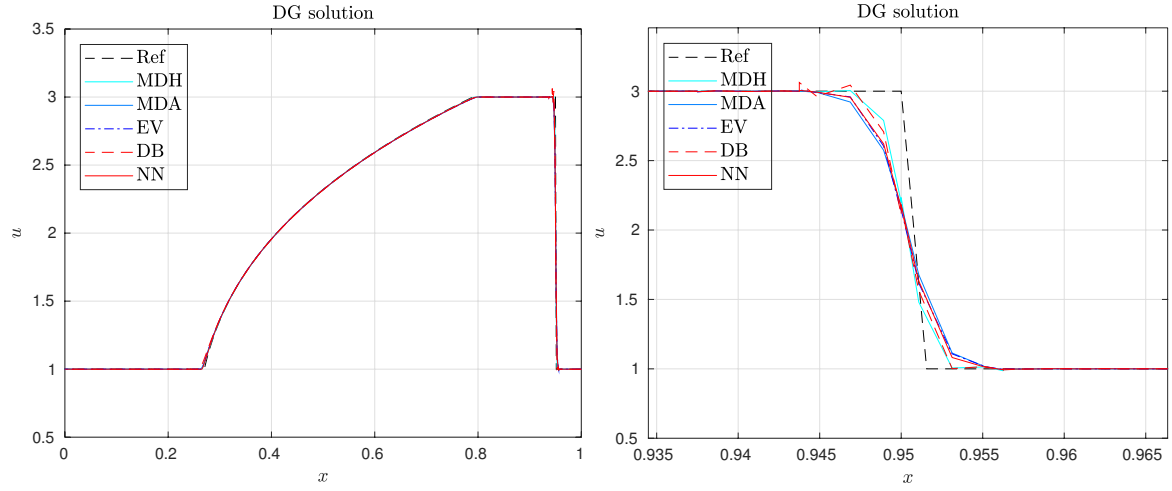
FIGURE 5.11: Comparison of the three different strategies for the NN technique, $f(u) = u^4/4$, $m = 4$.

Model	$\texttt{mean}(\|\|\mu\|\|_\infty)$
EV	1.6826e−2
NN1 (u-u)	1.2848e−3
NN2 (f'-f')	2.0254e−2
NN3 (u-f')	1.5518e−2

TABLE 5.11: Comparison of the time-averaged infinity norm of the artificial viscosity of the three different strategies for the NN technique, $m = 4$.

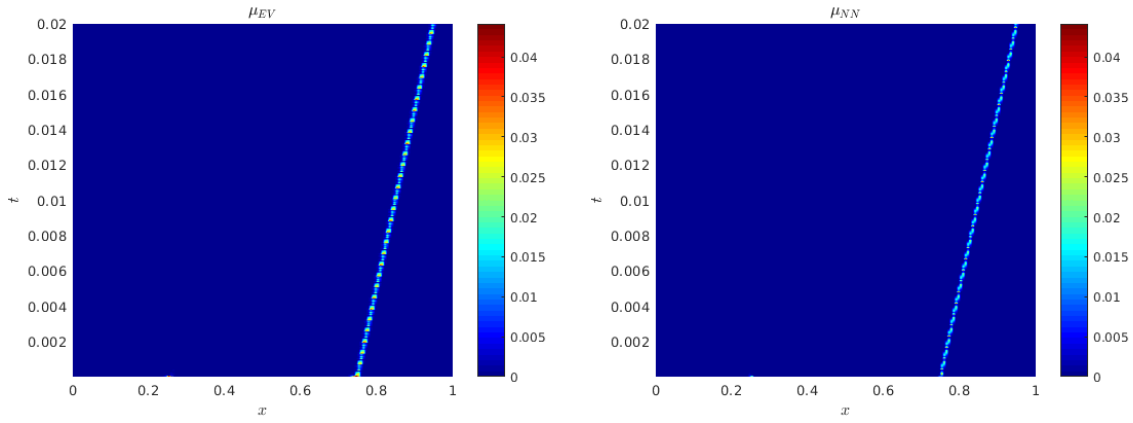
Model	$m = 4$
DB	$c_\beta = 8,\ c_{max} = 5$
EV	$c_E = 2,\ c_{max} = 1$
MDH	$c_A = 2.5,\ c_\kappa = 0.4,\ c_{max} = 0.8$
MDA	$c_{max} = 1$

TABLE 5.12: Parameter values for the standard artificial viscosity models for the fourth test case.



(a) Overall result. (b) Zoom close to the shock.

FIGURE 5.12: Numerical results for the fourth test case, $f(u) = u^4/4$, $m = 4$.



(a) EV (b) NN

FIGURE 5.13: Temporal history of the artificial viscosity for the fourth test case, $f(u) = u^4/4$, $m = 4$.

We observe that the DB model is not able to smoothen the solution as expected, resulting in a quite oscillatory profile. Despite the values for c_β and c_{max} are very high compared to the previous tests, we found no better result in this scenario, at least in the parameter range we considered. Once more, this demonstrates that the DB technique does not provide good results. The results with the other models are comparable.

5.1.5 Buckley-Leverett problem

The goal of this final test is to demonstrate the performances of the NN technique for non-convex flux functions. We recall that the problem is set by defining

$$f(u) = \frac{u^2}{u^2 + 0.5(1-u)^2}.$$

Consider the test case presented in [11]. In particular, we select $\Omega = [0, 1.5]$ and a final time $T = 0.4$. Again, we pick $K = 160$ elements. The initial condition is a step function, defined as

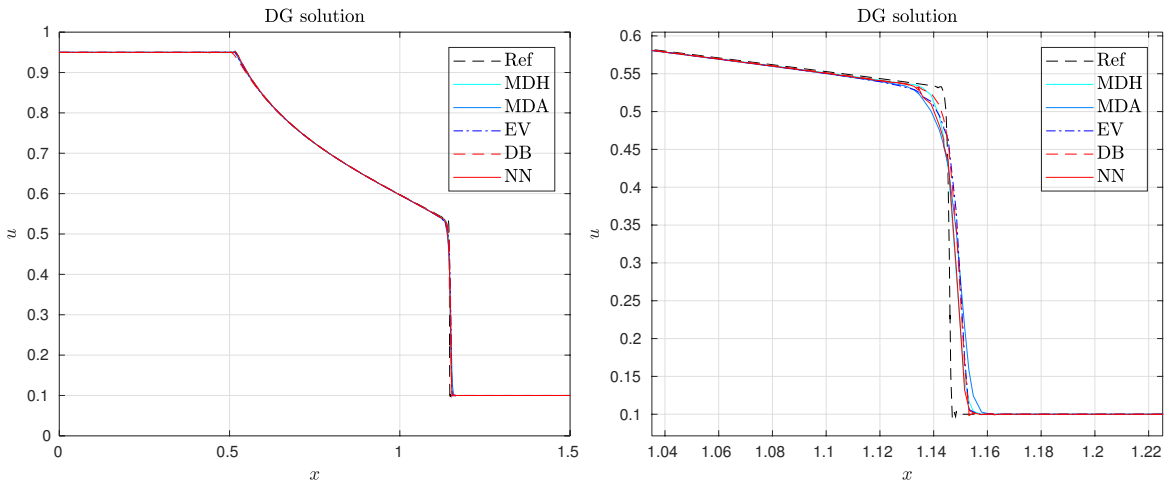
$$u_0(x) = \begin{cases} 0.95 & \text{if } 0 \leq x < 0.5, \\ 0.1 & \text{if } 0.5 \leq x \leq 1.5, \end{cases}$$

and finally the problem is completed with Dirichlet boundary conditions. It evolves into a compound wave consisting of a shock and a rarefaction. We recall that this phenomenon is present only with non-convex flux functions, where the slope of the characteristics $f'(u)$ is not monotone. In particular, for Riemann problems, a number of waves greater than the number of equation might develop starting from two different left and right values [31], as in this scenario.
Once more, the network provides good results, as shown for $m = 4$ in Figure 5.14.

Model	$m = 4$
DB	$c_\beta = 4,\ c_{max} = 2$
EV	$c_E = 2,\ c_{max} = 1$
MDH	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.6$
MDA	$c_{max} = 1$

TABLE 5.13: Parameter values for the standard artificial viscosity models for the fifth test case.

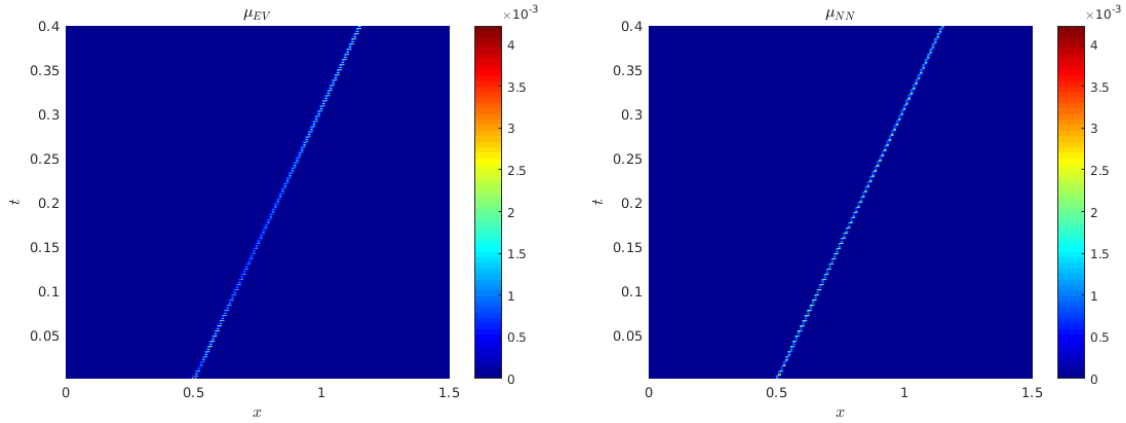
Compared to Burgers equation, the NN does not clearly outperform the other methods, even though significant differences are not present among all the models. In particular, the upper part of the shock



(a) Overall result. (b) Zoom close to the shock.

FIGURE 5.14: Numerical results for the fifth test case, Buckley-Leverett problem, $m = 4$.

is not well resolved by the network. This behavior could be related to the specific problem we chose or, more generally, by the network trained only with Burgers equation. We believe the former to be the more probable reason. The temporal history of the artificial viscosity is once more coherent with the previous analysis.



(a) EV (b) NN

FIGURE 5.15: Temporal history of the artificial viscosity for the fifth test case, Buckley-Leverett problem, $m = 4$.

5.1.6 Remarks

In this Section we numerically tested the performances of the network for one-dimensional scalar conservation laws. Both qualitative and quantitative analyses validate the model. In particular, the network is able to perform well for problems which were not in the training set, obtained by varying the mesh resolution and/or the flux function.
Due to the test case in Subsection 5.1.4, we verified that the best strategy is to use the solution as input variable, scaled in the reference interval $[-1, 1]$. The inverse scaling is performed by multiplying by a local wave speed and a scaling factor. We recall that all the results we showed, except in the linear advection case, were obtained using the improved scaling based on the solution jump. However, considering the original factor h, the qualitative results are rather similar, excluding again the linear advection problem.
We also stress on the fact that, in all the problems, an optimized set of parameters for the classical

models was chosen. In other words, a parameter tuning was performed. Thus, even though it may happen that one standard model provides better results than the ANN for a specific problem, for a generic set of parameters it is not the case. Thus, if no criterion is given to choose them, we claim that the ANN is the best technique, since it is parameter-free.
We finally report that all the simulations were run using a value of 0.1 for the CFL constant in equation (2.18). We recall that its dependence on m is weak, in the sense that it is already taken into account by the terms m^2 and m^4 in its definition. Thus, picking an m-independent value does not create instabilities as the discretization degree is varied.

5.2 One-dimensional Euler system

Beyond different scalar conservation laws, the first natural extension of the previous framework goes in the direction of systems of equations, with a particular focus on Euler system. As described in Section 3.4, the adopted strategy is rather simple. Motivated by a reasoning similar to the scalar case, we consider only the third network strategy presented in Table 3.1, using the density ρ as predicting variable and scaling with the local wave speed $\max(|v| + c)$. On the other hand, the h-scaling still suffers from the accuracy issue. We can again rely on the ANN as troubled-cell indicator or we modify the factor by taking into account the solution jump. Here, we adopt the latter approach, unless stated otherwise.

5.2.1 A smooth problem

As in the scalar case, we first need to validate the standard models, as well as the ANN technique. Choosing $\Omega = [0, 1]$, the initial condition [55] is set as

$$(\rho, v, p)_0 = (\rho_\infty + A \sin(2\pi x), v_\infty, p_\infty)\,,$$

which is converted in the conserved variables as using the ideal gas law (2.22). Periodic boundary conditions are applied. Here, we consider $\rho_\infty = 1,\ A = 0.5,\ v_\infty = 1,\ p_\infty = 1$. The problem turns out to be a simple advection equation for the density variable, leading to the following exact solution:

$$(\rho, v, p) = (\rho_\infty + A \sin(2\pi(x - v_\infty t)), v_\infty, p_\infty)\,.$$

Again, in order to measure the discretization error, we rely on the $L^2(\Omega)$-norm at a fixed time T, which we choose to be equal to $T = 0.2$. Dealing with multiple variables, we define

$$\begin{aligned}\epsilon^2 = \|\boldsymbol{u}_h(\cdot, T) - \boldsymbol{u}(\cdot, T)\|^2_{L^2(\Omega)} &= \sum_{i=1}^{n} \|u_{h,i}(\cdot, T) - u_i(\cdot, T)\|^2_{L^2(D^k)} \\ = \sum_{i=1}^{n}\sum_{k=1}^{K} (\boldsymbol{u}_{h,i} - \boldsymbol{u}_i)^T \boldsymbol{M} (\boldsymbol{u}_{h,i} - \boldsymbol{u}_i) &= \sum_{k=1}^{K} (\boldsymbol{u}_h - \boldsymbol{u})^T \tilde{\boldsymbol{M}} (\boldsymbol{u}_h - \boldsymbol{u}),\end{aligned}$$

where we set $\tilde{\boldsymbol{M}} = \texttt{blkdiag}(\{\boldsymbol{M}\}_{i=1}^n)$. For the one-dimensional Euler system we have $n = 3$ equations. Essentially, the scalar $L^2(\Omega)$-norm is computed separately for each conserved variable, and all the contributions are added together. Since all the equations are discretized with the same polynomial degree, for the inviscid problem we expect the optimal scaling $\epsilon \sim h^{m+1}$, provided that the solution is smooth enough. We can now show the convergence results for the viscous equation, focusing only on $m = 1$ and $m = 4$ in Tables 5.14 and 5.15 respectively. The behavior for $m = 2, 3$ is similar and it is not reported here.

Here, high accuracy is obtained even with the DB model. Indeed, recalling its definition for Euler system (2.27), the dissipation is proportional to the divergence of the velocity field. Since for this test case $v = v_\infty = const$, no viscosity is added and the inviscid scheme is retained. This is a general behavior for Euler equations, where the DB model may outperform the others, especially when weak compressibility is present. As shown in the following Subsection, issues may arise when contact discontinuities are present. Concerning the other methods, the comments made for the scalar case are still valid. In particular, the EV is always able to get a good convergence rate, while the MDH has a

K	Inv		DB		EV		MDH		NN	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	4.1094e−3	-	4.1094e−3	-	1.9135e−1	-	3.1188e−1	-	1.5711e−2	-
20	9.6045e−4	2.10	9.6045e−4	2.10	3.0750e−2	2.64	1.4719e−1	1.08	2.7337e−3	2.52
40	2.3664e−4	2.02	2.3664e−4	2.02	4.6967e−3	2.71	1.5970e−2	3.20	6.9356e−4	1.98
80	5.8947e−5	2.01	5.8947e−5	2.01	6.9346e−4	2.76	5.8947e−5	8.08	1.2840e−4	2.43
160	1.4723e−5	2.00	1.4723e−5	2.00	9.4459e−5	2.88	1.4723e−5	2.00	1.4767e−5	3.12
320	3.6800e−6	2.00	3.6800e−6	2.00	1.2614e−5	2.90	3.6800e−6	2.00	3.6813e−6	2.00

TABLE 5.14: L^2 convergence errors and estimated rate in the inviscid case, using standard artificial viscosity models and with the ANN model using the improved scaling. Smooth problem, $m = 1$.

K	Inv		DB		EV		MDH		MDA		NN	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.2192e−6	-	1.2192e−6	-	2.7605e−5	-	1.2192e−6	-	1.2192e−6	-	1.2194e−6	-
20	4.6135e−8	4.72	4.6135e−8	4.72	6.0809e−7	5.50	4.6135e−8	4.72	4.6135e−8	4.72	4.6135e−8	4.72
40	1.5494e−9	4.90	1.5494e−9	4.90	1.2528e−8	5.60	1.5494e−9	4.90	1.5494e−9	4.90	1.5494e−9	4.90
80	5.0248e−11	4.94	5.0248e−11	4.94	2.7371e−10	5.52	5.0248e−11	4.94	5.0248e−11	4.94	5.0248e−11	4.94
160	1.5892e−12	4.98	1.5892e−12	4.98	8.9280e−12	4.94	1.5892e−12	4.98	1.5892e−12	4.98	1.5894e−12	4.98

TABLE 5.15: L^2 convergence errors and estimated rate in the inviscid case, using standard artificial viscosity models and with the ANN model using the improved scaling. Smooth problem, $m = 4$.

linear scaling for low degrees and resolutions and goes back to the inviscid scheme for high orders or sufficiently fine meshes. With the improved scaling, the NN is able to achieve the correct convergence rate. Note that the magnitude of the discretization error for the inviscid case and the NN model is essentially the same.

5.2.2 Single waves

We consider now a couple of test cases with discontinuous initial conditions. Both are defined by assigning a left and a right state which can be connected by a single wave [56]. For both examples, we consider $\Omega = [0, 1]$, $K = 200$ elements and $T = 0.25$.
Let us set

$$(\rho, v, p)_0 = \begin{cases} (1, 1, 1) & \text{if } 0 \leq x \leq 0.2, \\ (0.4, 1, 1) & \text{if } 0.2 < x \leq 1, \end{cases}$$

and complete the problem with Dirichlet boundary conditions. Note that both velocity and pressure are constant and continuous. It can be verified that left and right states can be connected by a single contact wave, moving with a speed $\dot{s} = v = 1$. The exact solution is

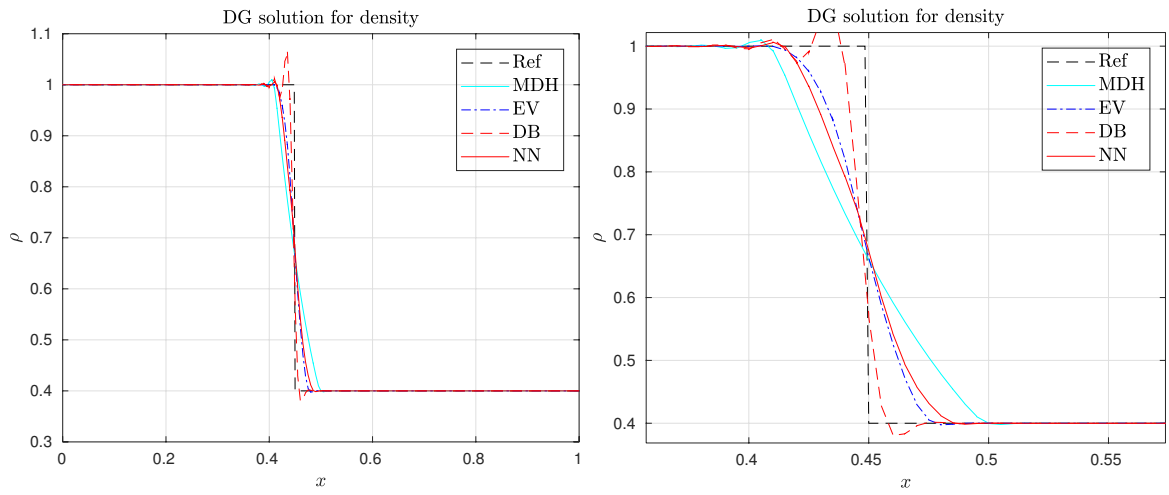
$$(\rho, v, p) = \begin{cases} (1, 1, 1) & \text{if } 0 \leq x \leq 0.2 + \dot{s}t, \\ (0.4, 1, 1) & \text{if } 0.2 + \dot{s}t < x \leq 1. \end{cases}$$

This problem resembles the scalar linear advection equation with a discontinuous initial condition. Here, to simplify the analysis we focus on the case $m = 1$. The parameters used for the standard models are reported in Table 5.16, while the final density profile is plotted in Figure 5.16.

Model	Contact	Shock
DB	$c_\beta = 2,\ c_{max} = 1$	$c_\beta = 4,\ c_{max} = 2$
EV	$c_E = 2,\ c_{max} = 0.5$	$c_E = 5,\ c_{max} = 2.5$
MDH	$c_A = 3.2,\ c_\kappa = 0.5,\ c_{max} = 1.2$	$c_A = 3,\ c_\kappa = 0.5,\ c_{max} = 2.5$

TABLE 5.16: Parameter values for the standard artificial viscosity models for the single wave case.

Here, the DB model adds no dissipation. As in the smooth problem 5.2.1, the reason is that the velocity variable is constant in space and time. In particular, it is continuous across the contact wave.



(a) Overall result.

(b) Zoom close to the contact wave.

FIGURE 5.16: Numerical results for the single contact wave case, $m = 1$.

Thus, the scheme is equivalent to the inviscid problem, so that Gibbs phenomenon is triggered and numerical oscillations appear. We note that they are not *destructive*, in the sense that they do not force the solution to become unstable. This is a typical behavior of contact waves. Ignoring the DB method, which fails to smoothen the solution, the other models work similarly among them. The MDH is the most dissipative model and the contact wave is not well resolved. The EV and NN techniques behave similarly, with the former being slightly better.
On the other hand, setting

$$(\rho, v, p)_0 = \begin{cases} (1, 0.8276, 1) & \text{if } 0 \leq x \leq 0.2, \\ (0.5313, 0.1, 0.4) & \text{if } 0.2 < x \leq 1, \end{cases}$$

together with Dirichlet boundary conditions, a single shock moves at a speed

$$\dot{s} = \frac{\rho_l v_l - \rho_r v_r}{\rho_l - \rho_r} = \frac{0.8276 - 0.05313}{1 - 0.5313} \simeq 1.65,$$

leading to an exact solution equal to

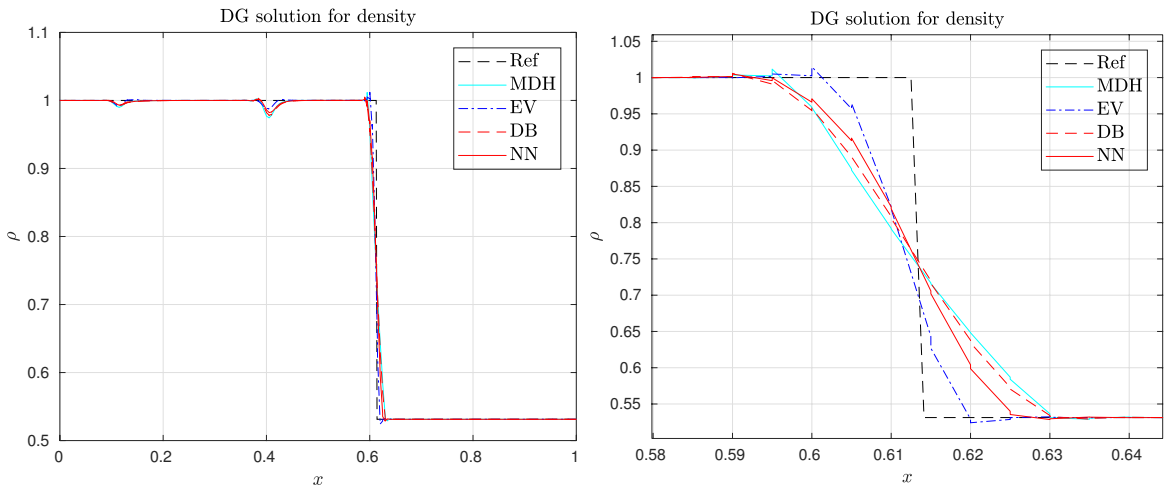
$$(\rho, v, p) = \begin{cases} (1, 0.8276, 1) & \text{if } 0 \leq x \leq 0.2 + \dot{s}t, \\ (0.5313, 0.1, 0.4) & \text{if } 0.2 + \dot{s}t < x \leq 1. \end{cases}$$

The parameters are again reported in Table 5.16, leading to the final density profile plotted in Figure 5.17.

This single shock problem can be interpreted in a way similar to Burgers equation, thus analogous comments hold. Here, the DB model performs quite well, since the velocity is discontinuous across the shock. The EV seems again to be the best one, even if a low-amplitude undershoot is present. The others perform similarly, with the NN resolving the donwstream profile slightly better than the MDH and DB. The bumps observed at $x \simeq 0.1$ and $x \simeq 0.4$ appear to be numerical artefacts that exist for this problem, and they have also been observed in [56].

5.2.3 Sod problem

Obviously, systems of conservation laws exhibit a higher degree of complexity with respect to scalar problems. In particular, two different states can be connected by multiple waves, generating intermediate



(a) Overall result.

(b) Zoom close to the shock wave.

FIGURE 5.17: Numerical results for the single shock wave case, $m = 1$.

states [54]. A popular test case is the Sod (shock-tube) problem [46]. The initial state is given by

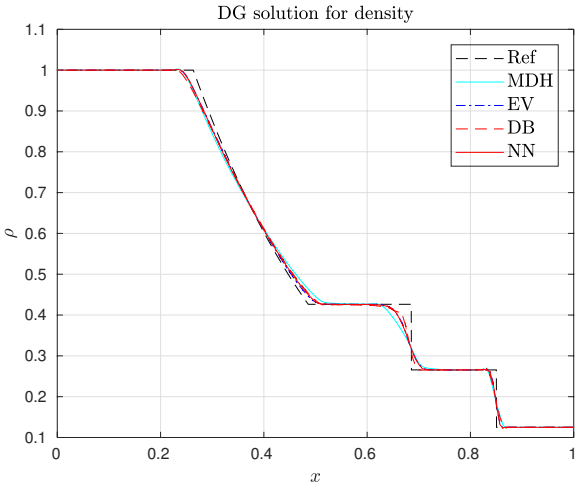
$$(\rho, v, p)_0 = \begin{cases} (1, 0, 1) & \text{if } 0 \leq x \leq 0.5, \\ (0.125, 1, 0.1) & \text{if } 0.5 < x \leq 1, \end{cases}$$

in the domain $\Omega = [0, 1]$, completed with Dirichlet boundary conditions. Three different waves are generated. A left-moving rarefaction fan, where no dissipation should be added, and right-moving contact and shock waves. In principle, the contact wave should be smoothed only initially not to generate oscillations, while for large times no dissipation has to be added. On the other hand, viscosity should be continuously injected in the region close to the shock. A pseudo-analytical solution is available, obtained by (numerically) solving a single nonlinear equation [57]. To generate the results, we consider $K = 160$ elements, a discretization degree $m = 1$ and $T = 0.2$, and the parameters listed in Table 5.17. The final solution profile for all the conserved variables is reported in Figure 5.18. The corresponding viscosity profile is shown in Figure 5.19.

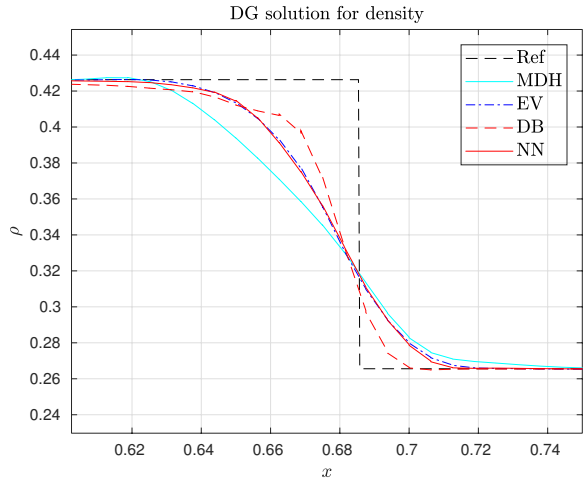
Model	$m = 1$
DB	$c_\beta = 2,\ c_{max} = 1$
EV	$c_E = 5,\ c_{max} = 1.5$
MDH	$c_A = 2.5,\ c_\kappa = 0.5,\ c_{max} = 1.5$

TABLE 5.17: Parameter values for the standard artificial viscosity models for the Sod problem.

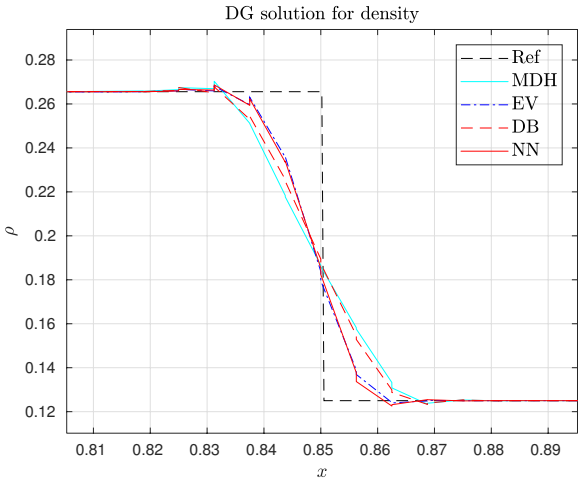
Again, all the models perform similarly. A small overshoot in the profile of the momentum is present close to the shock, with a comparable amplitude present in all the models. Looking at the temporal history of the artifical dissipation, we observe that all the models keep track of the shock by adding high dissipation in that region. At the same time, the viscosity amount close to the contact and the rarefaction wave is less compared to the one injected where the shock is present. Here, we note that the DB model adds dissipation in the rarefaction area, which is not highly desirable, since the solution is smooth there. Moreover, since all the waves start from the same point, it is enough to track the position of the shock. This is somehow similar to [11], where most of the troubled cells are in the initial instants. Qualitatively equivalent results are obtained for higher discretization degrees and are not reported here.



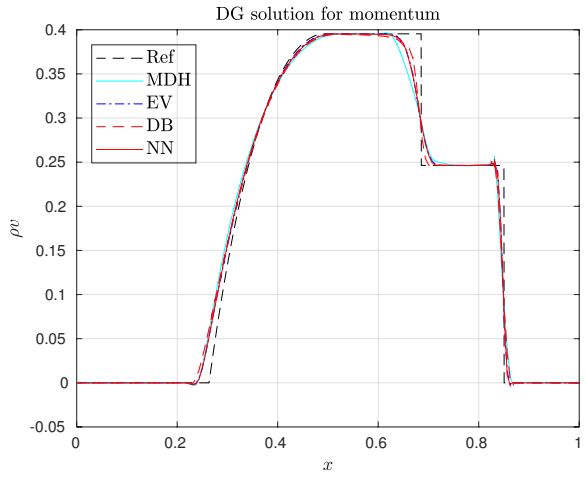
(a) Overall result for density.



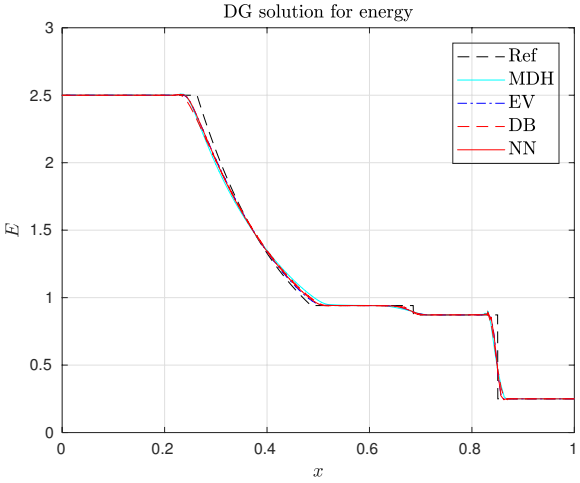
(b) Zoom close to the contact wave.



(c) Zoom close to the shock wave.

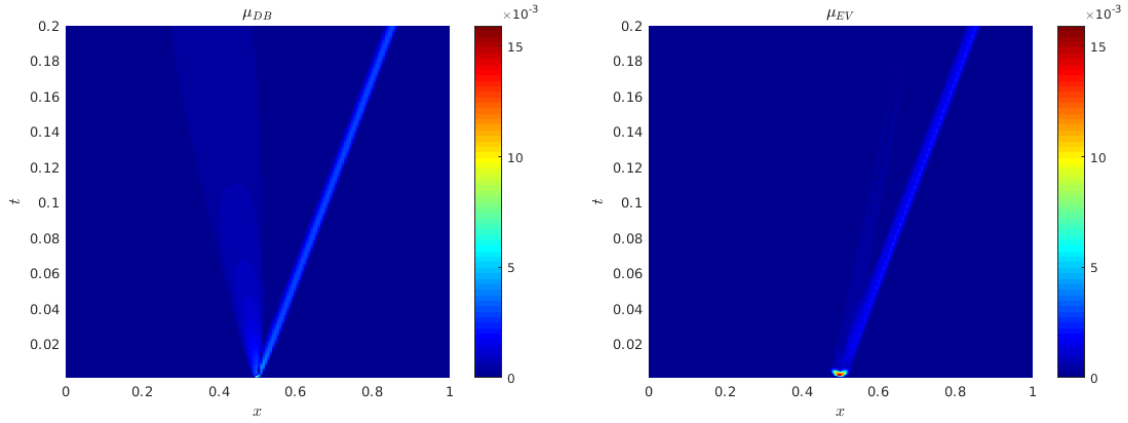


(d) Overall result for momentum.



(e) Overall result for energy.

FIGURE 5.18: Numerical results for the Sod problem, $m = 1$.



(a) DB (b) EV

(c) MDH (d) NN

FIGURE 5.19: Temporal history of the artificial viscosity for the Sod problem, $m = 1$.

5.2.4 Shu-Osher problem

The next problem we tackle is the Shu-Osher test case [58]. Consider $\Omega = [-5, 5]$, $K = 200$ and $T = 1.8$. The initial condition is defined as

$$(\rho, v, p)_0 = \begin{cases} (3.857143, 2.629369, 10.333333) & \text{if } -5 \leq x \leq -4, \\ (1 + 0.5\sin(5x), 0, 1) & \text{if } -4 < x \leq 5, \end{cases}$$

with Dirichlet-Neumann boundary conditions. Combination of smooth and discontinuous data are present, making this case well suited to test the capabilities to both capture shocks, i.e. numerical oscillations, and physical oscillations. The results for $m = 1, 2, 3, 4$ are shown in Figures 5.20, 5.21, 5.22, 5.23 respectively and are obtained with the parameters listed in Table 5.18.

Model	$m = 1$	$m = 2$	$m = 3$	$m = 4$
DB	$c_\beta = 0.8,\ c_{max} = 0.2$	$c_\beta = 0.8,\ c_{max} = 0.2$	$c_\beta = 0.8,\ c_{max} = 0.2$	$c_\beta = 0.8,\ c_{max} = 0.2$
EV	$c_E = 0.5,\ c_{max} = 0.2$	$c_E = 0.5,\ c_{max} = 0.2$	$c_E = 1.5,\ c_{max} = 0.5$	$c_E = 2,\ c_{max} = 0.5$
MDH	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 1$	$c_A = 3,\ c_\kappa = 0.5,\ c_{max} = 1.5$	$c_A = 3,\ c_\kappa = 0.5,\ c_{max} = 1.5$	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.5$
MDA	-	-	$c_{max} = 0.25$	$c_{max} = 0.2$

TABLE 5.18: Parameter values for the standard artificial viscosity models for the Shu-Osher problem.

The NN model shows better solution profiles compared to the MDH and EV ones, while the DB technique appears to outperform the other methods for all the discretization degrees. Zooming close to the (physical) oscillations, we note that for $m = 1$ the resolution is not high enough to capture them, while, increasing the degree, all the models do not suppress them. Focusing on the region where the solution deviates from the constant value of 3.857143, some numerical oscillations are present, even though most of the models are able to reduce their amplitude. Even considering high discretization degrees, the NN technique is the one with more significant bumps. This can be explained by recalling that the simulations using standard models have been run using optimized values for the parameters, designed in a way that such oscillations are fully suppressed. Finally, note that the overall solution quality improves when m is increased, so that the added dissipation does not destroy the benefits of choosing higher discretization degrees.
The artificial viscosity profile is shown in Figure 5.24. For simplicity, only the case $m = 1$ is reported.

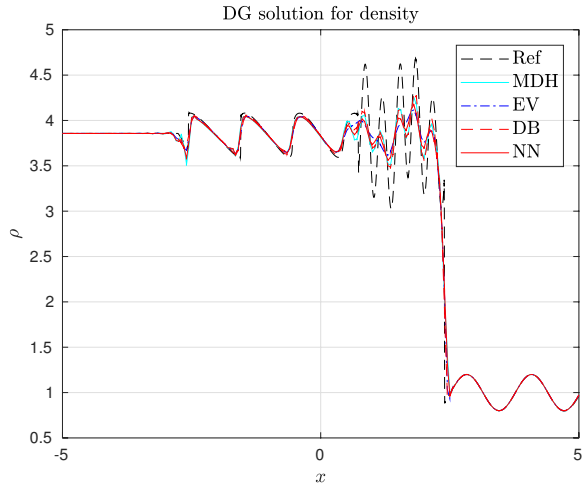
Qualitatively, no significant differences among the models are present. Dissipation is added in the same spatial regions, with slightly different values.

5.2.5 Remarks

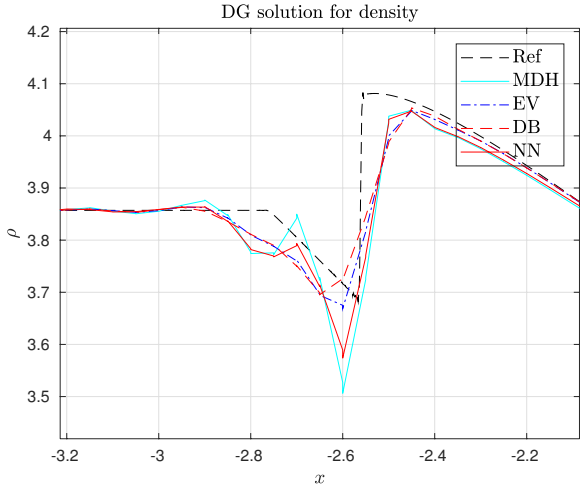
In this Section we observed how the network technique generalizes well to capture discontinuities even for the Euler system. Compared to scalar problems, it seems that the ANN does not clearly outperform the standard models, even though the overall solution profile is not significantly different. Recalling that the results obtained with the classical models are generated with optimized sets of parameters, using generic values for them, their performances may drop. In this scenario, due to the presence of problem-dependent parameters, it might happen that the positivity property of density and/or pressure is lost, leading to non-physical results. We never experienced this issue using the ANN technique, at least in the reported test cases.
Finally, note that the CFL constant in equation (2.18) has a value of $C = 0.2$ independently of the discretization degree.

5.3 Two-dimensional scalar problems

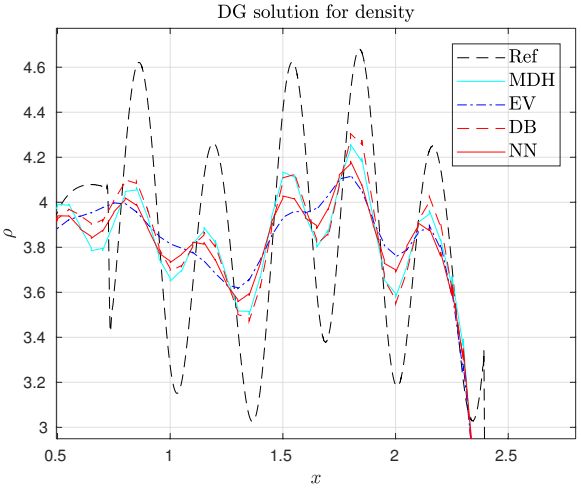
The second extension relates to two-dimensional problems (scalar equations and systems). Since we already validated the classical artificial viscosity methods and the network capabilities in the one-dimensional scenario, we can skip part of the details, since most of the comments are still valid. It is worth recalling that in this context two different strategies are available. The first one relies on



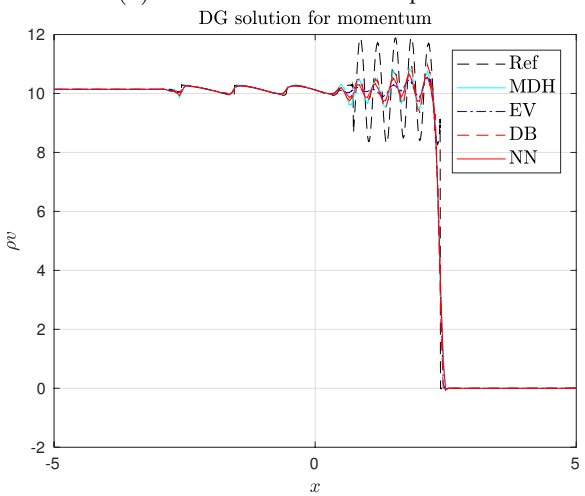
(a) Overall result for density.



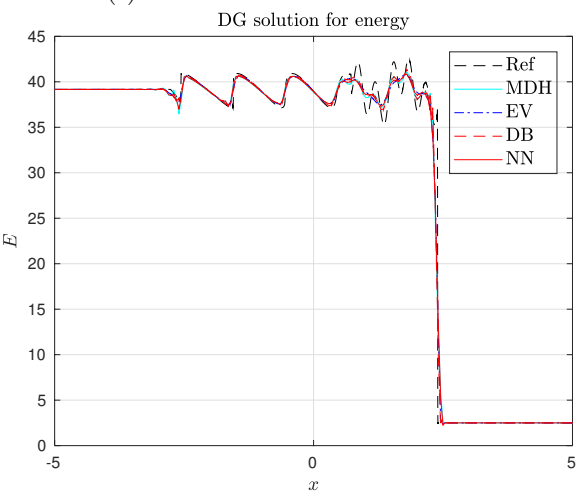
(b) Zoom close to the first peak.



(c) Zoom close to the oscillations.

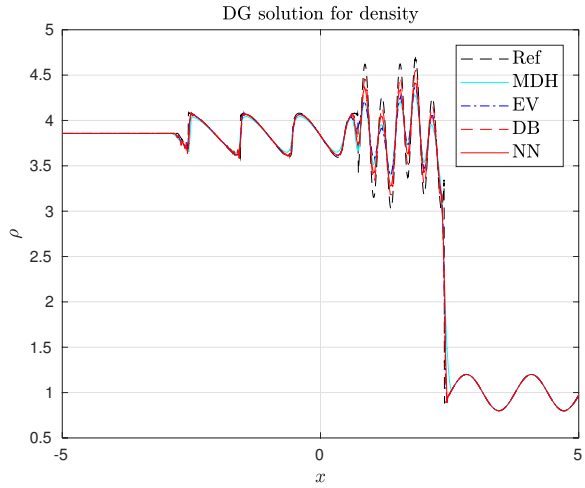


(d) Overall result for momentum.

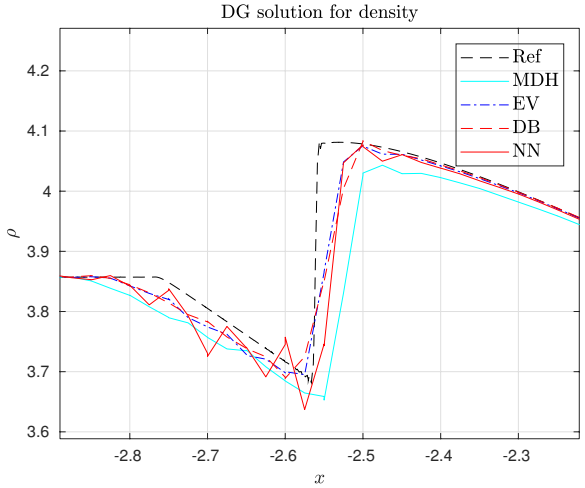


(e) Overall result for energy.

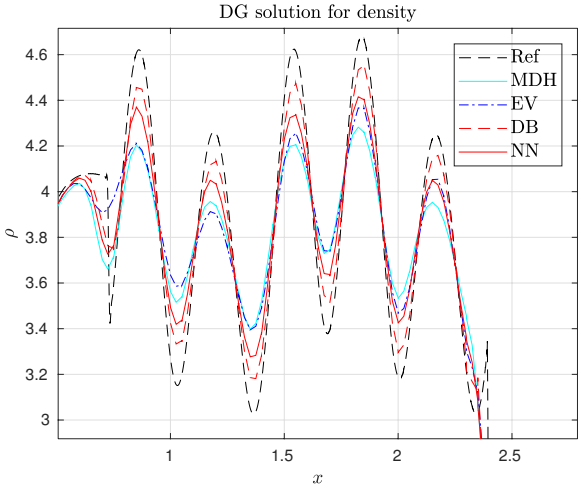
FIGURE 5.20: Numerical results for the Shu-Osher problem, $m = 1$.



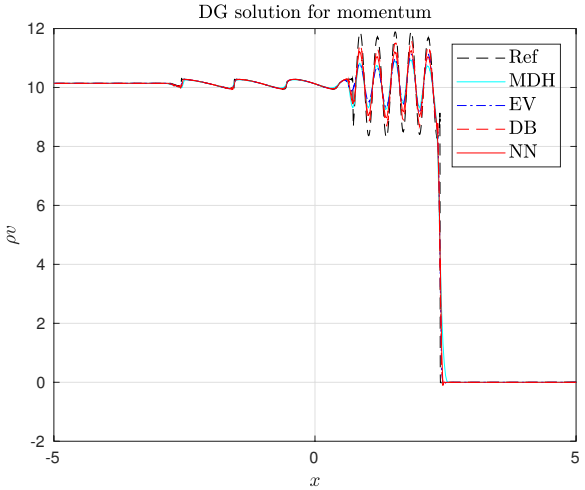
(a) Overall result for density.



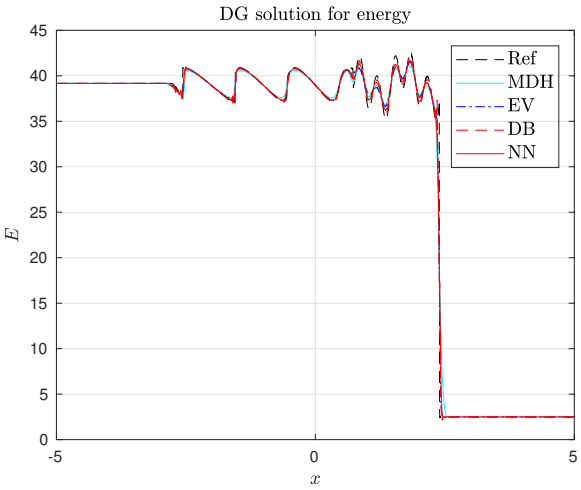
(b) Zoom close to the first peak.



(c) Zoom close to oscillations.

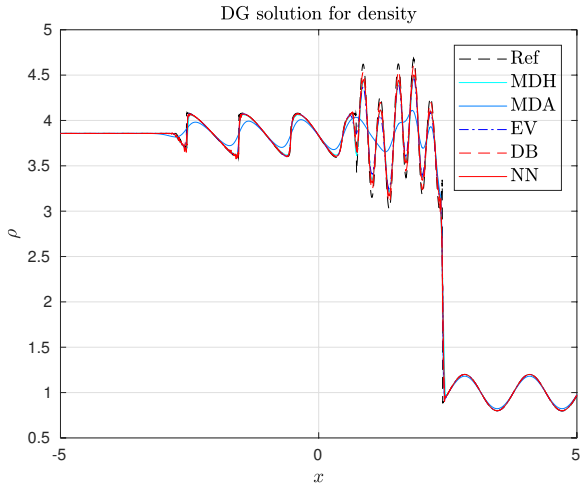


(d) Overall result for momentum.

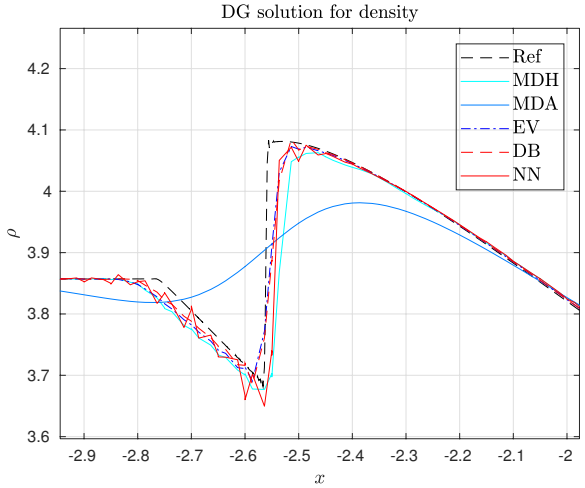


(e) Overall result for energy.

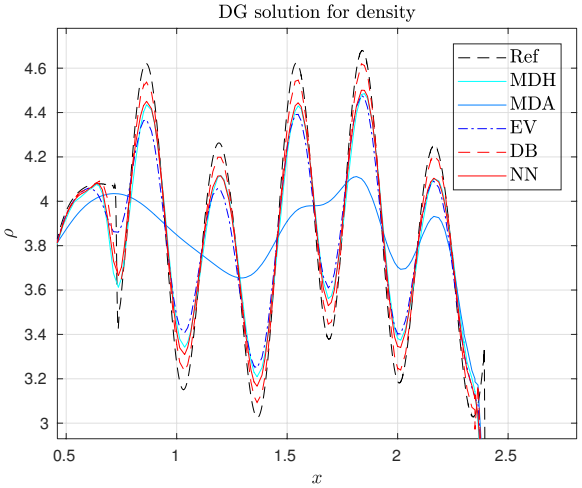
FIGURE 5.21: Numerical results for the Shu-Osher problem, $m = 2$.



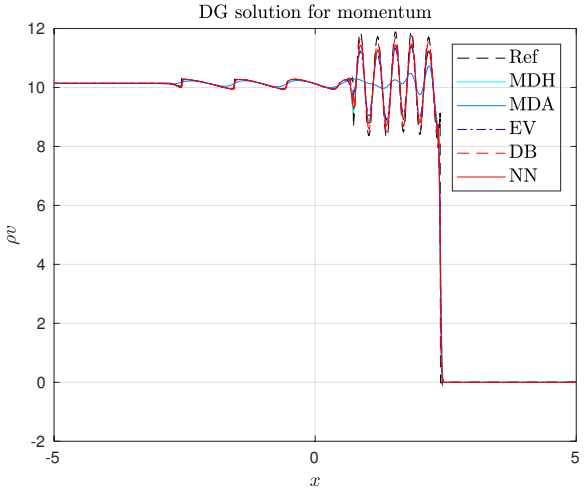
(a) Overall result for density.



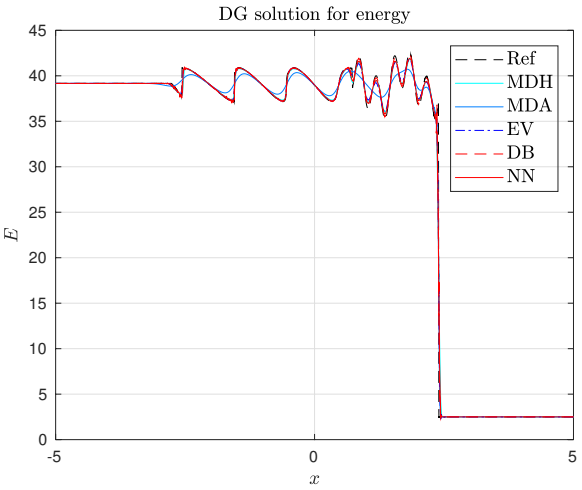
(b) Zoom close to the first peak.



(c) Zoom close to oscillations.

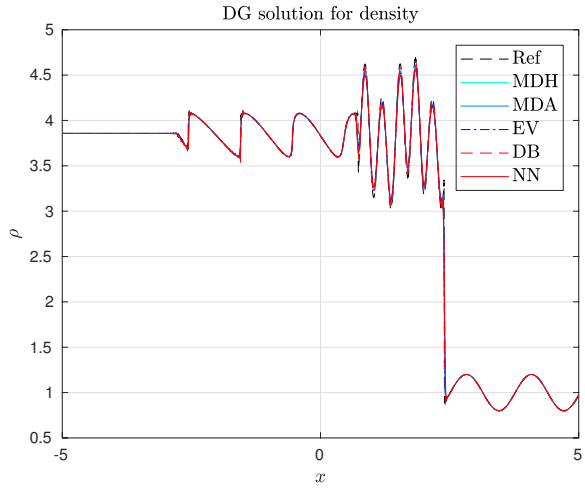


(d) Overall result for momentum.

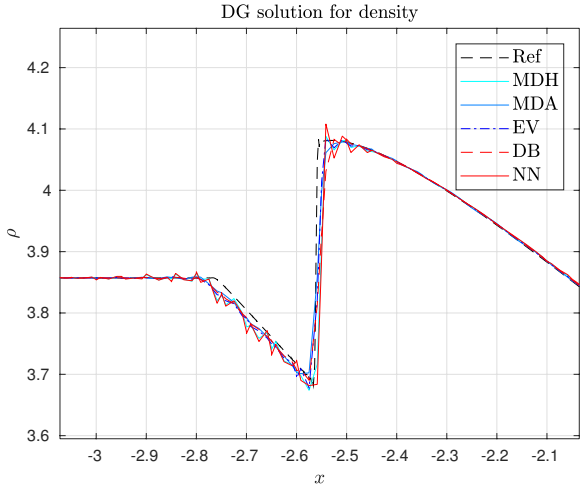


(e) Overall result for energy.

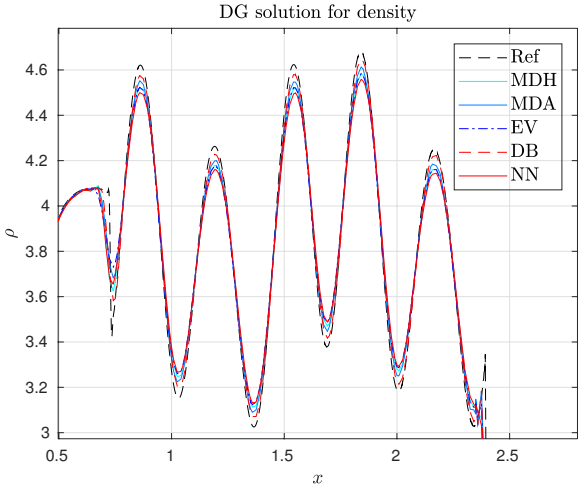
FIGURE 5.22: Numerical results for the Shu-Osher problem, $m = 3$.



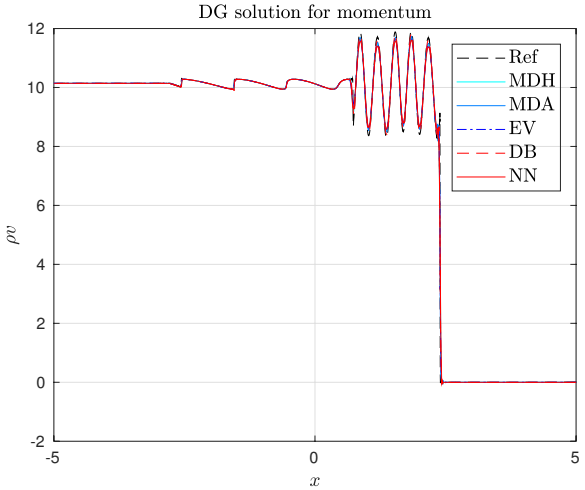
(a) Overall result for density.



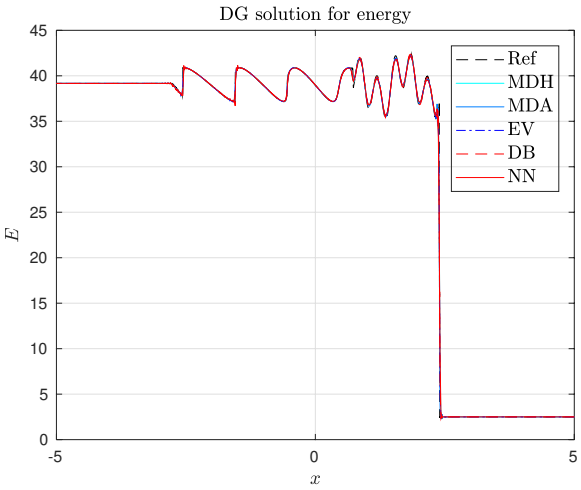
(b) Zoom close to the first peak.



(c) Zoom close to oscillations.

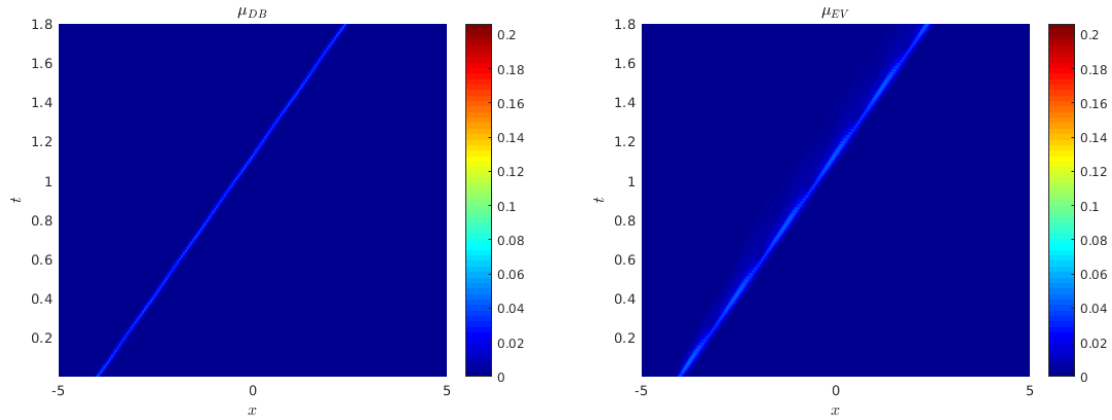


(d) Overall result for momentum.



(e) Overall result for energy.

FIGURE 5.23: Numerical results for the Shu-Osher problem, $m = 4$.



(a) DB (b) EV

(c) MDH (d) NN

FIGURE 5.24: Temporal history of the artificial viscosity for the Shu-Osher problem, $m = 1$.

the one-dimensional ANN, estimating a different coefficient along each edge of a given triangle. The second one is based on a two-dimensional network. Again, let us start with scalar equations.

5.3.1 A smooth problem

The smooth problem we consider here is the two-dimensional extension of the linear advection equation. Let us pick a constant transport field, say $\beta = (\beta^x, \beta^y) = (1, 1)$. The computational domain is $\Omega = [0, 1]^2$, whereas the problem is completed by assigning the initial condition

$$u_0(x, y) = 1 + \sin(2\pi x)\sin(2\pi y),$$

periodic boundary conditions and a final time $T = 0.2$. The exact solution is simply given by

$$u(x, y, t) = u_0(x - \beta^x t, y - \beta^y t) = 1 + \sin(2\pi(x - t))\sin(2\pi(y - t)),$$

and the discretization error ϵ is computed as in the one-dimensional case, since the definition of the mass matrix is not (formally) altered. The convergence results are reported in Tables 5.19 and 5.20 for degrees $m = 1$ and $m = 4$ respectively, skipping the details for $m = 2$ and $m = 3$. They are generated using a structured mesh built starting from a mesh of quadrilaterals (in particular squares), and separating each element into two triangles. Supposing that each domain edge is divided into N elements, we have $K = 2N^2$ total triangles, as reported in Figure 5.25.

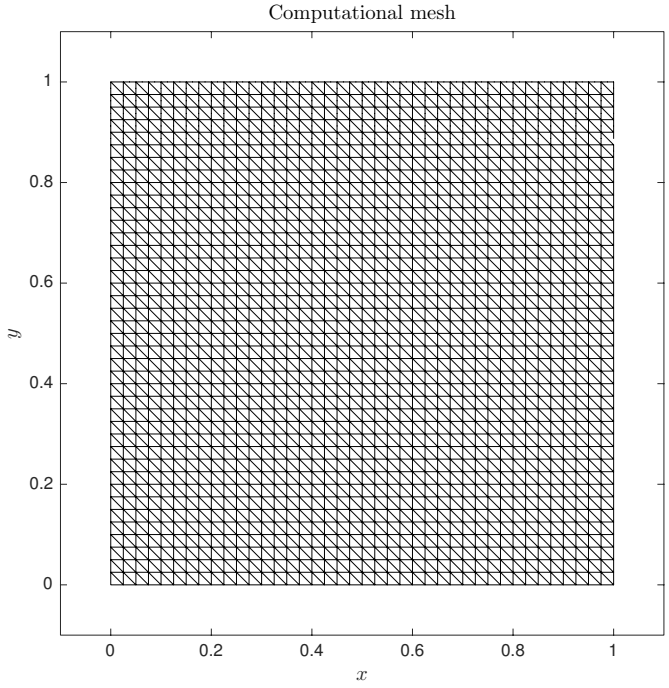


FIGURE 5.25: A graphical representation of the structured mesh we employ in this work.

N	Inviscid		DB		EV		MDH		NN1D		NN2D	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.8672e−2	-	3.4792e−1	-	3.0321e−1	-	4.1350e−1	-	1.6452e−1	-	1.1511e−1	-
20	4.5400e−3	2.04	2.1094e−1	0.72	1.1735e−1	1.37	2.9138e−1	0.51	4.8747e−2	1.75	2.9692e−2	1.95
40	1.1194e−3	2.02	8.0485e−2	1.39	1.3438e−2	3.13	1.5561e−2	0.91	1.1152e−2	2.13	6.1154e−3	2.28
80	2.7864e−4	2.01	2.3854e−2	1.75	1.7744e−3	2.92	8.1373e−2	0.94	2.8707e−3	1.96	1.0747e−3	2.51
160	6.9578e−5	2.00	6.3178e−3	1.92	2.3654e−4	2.91	4.1920e−2	0.96	5.3736e−4	2.42	1.6979e−4	2.66

TABLE 5.19: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models and with both the 1D and 2D ANNs. Linear advection problem, $m = 1$.

N	Inviscid		DB		EV		MDH		MDA		NN1D		NN2D	
	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p	ϵ	p
10	1.1961e−5	-	8.0957e−2	-	4.2204e−3	-	1.3695e−2	-	1.1961e−5	-	2.5591e−5	-	1.2135e−5	-
20	3.5969e−7	5.06	2.3887e−2	1.76	1.2057e−5	8.45	3.5969e−7	15.22	3.5969e−7	5.06	4.3844e−7	5.37	3.6035e−7	5.07
40	1.0980e−8	5.03	6.3190e−3	1.92	2.1698e−7	5.80	1.0980e−8	5.03	1.0980e−8	5.03	1.8223e−8	5.29	1.0989e−8	5.04
80	3.4094e−10	5.01	1.6049e−3	1.98	3.9194e−9	5.79	3.4094e−10	5.01	3.4094e−8	5.01	3.4149e−10	5.04	3.4107e−10	5.00

TABLE 5.20: L^2 convergence errors and estimated rate in the inviscid case and using standard artificial viscosity models and with both the 1D and 2D ANNs. Linear advection problem, $m = 4$.

Essentially, all the results are coherent with their one-dimensional counterpart and meet our expectations. We refer to Subsection 5.1.1 for a more detailed comment. We just observe that both the 1D and the 2D ANNs (using the versions with the $\mathcal{H}$-scaling) give low errors and guarantee optimal accuracy rates.

5.3.2 2D Burgers equation: a Riemann problem

We can now switch to the two-dimensional extension of Burgers equation, choosing $f^x = f^y = u^2/2$. The physical domain we consider is $\Omega = [0, 1]^2$, whereas the initial condition is defined as

$$u_0(x, y) = \begin{cases} -1 & \text{if } 0.5 < x < 1,\ 0.5 < y < 1, \\ -0.2 & \text{if } 0 < x < 0.5,\ 0.5 < y < 1, \\ 0.5 & \text{if } 0 < x < 0.5,\ 0 < y < 0.5, \\ 0.8 & \text{if } 0.5 < x < 1,\ 0 < y < 0.5. \end{cases}$$

Compared to the one-dimensional case, here discontinuities are present even at the physical boundaries. To avoid spurious effects from the boundary itself, we extend the domain to $\Omega = [-1, 2]^2$, we apply periodic boundary conditions and we analyze the solution in the physical domain only. A pictorial representation is provided in Figure 5.26. Only shocks and rarefaction waves are present in the problem,

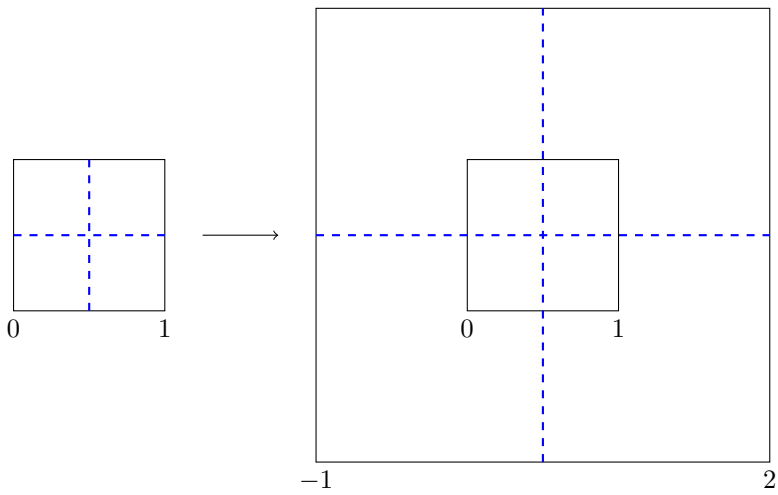


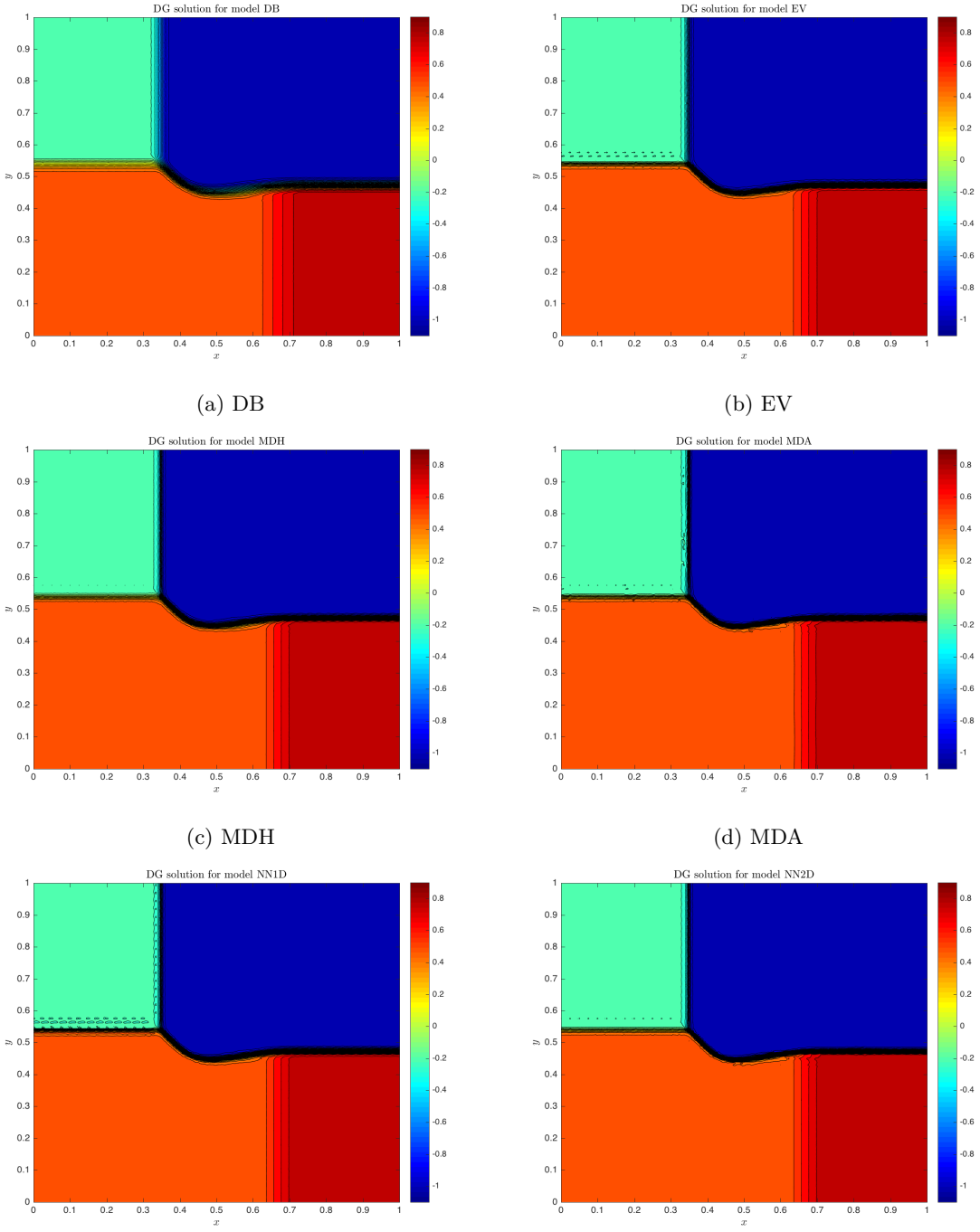
FIGURE 5.26: A graphical representation of the domain extension. The blue lines separate the four sectors in which the domain is divided into.

as in its one-dimensional counterpart. An exact solution is also available [18]. For our simulations, we divide each edge into $N = 40 \cdot 3$ elements, corresponding to $K = 3200$ triangles in the physical domain. The numerical results at $T = 0.25$, generated with the set of parameters listed in Table 5.21, are reported in Figure 5.27 with degree $m = 4$. For lower degrees, similar comments hold.

Here, the worst model is the DB, which is clearly too dissipative close to the shocks. The MDA, on the other hand, appears to be the least dissipative, since some wiggles are present close to discontinuities. All the other models are similar. The one-dimensional neural network seems to be slightly more

Model	$m = 4$
DB	$c_\beta = 2,\ c_{max} = 1$
EV	$c_E = 1,\ c_{max} = 0.25$
MDH	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.5$
MDA	$c_{max} = 0.8$

TABLE 5.21: Parameter values for the standard artificial viscosity models for 2D Riemann problem.



(a) DB (b) EV

(c) MDH (d) MDA

(e) NN1D (f) NN2D

FIGURE 5.27: Numerical results for the 2D Riemann problem, $m = 4$.

dissipative than its two-dimensional counterpart, and it looks similar to the EV. Since the 1D network does not take into account the internal degrees of freedom and the final viscosity coefficient is computed as a weighted average of the edge values, this smoother result is not unexpected. To finalize the analysis, in Figure 5.28 we report the spatial infinity norm of the artificial viscosity. Except for the highly-dissipative DB model, the maximum viscosity added by the other models is comparable.

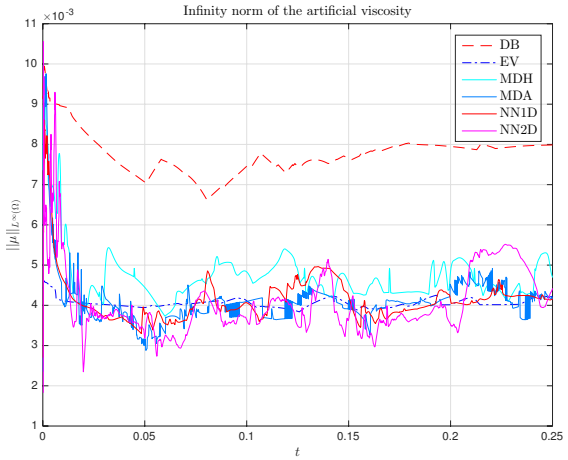


FIGURE 5.28: Infinity norm of the artificial viscosity with respect to time, 2D Riemann problem, $m = 4$.

In all the cases, the CFL constant was set to $C = 0.8$.

5.3.3 KPP rotating wave problem

We now switch to a conservation law which has non-convex x and y components of the flux function. More precisely they are set equal to $f^x = \sin(u)$ and $f^y = \cos(u)$ respectively. Here $\Omega = [-2, 2]$, $K = 2 \cdot 120^2$ and $T = 1$. The initial condition is

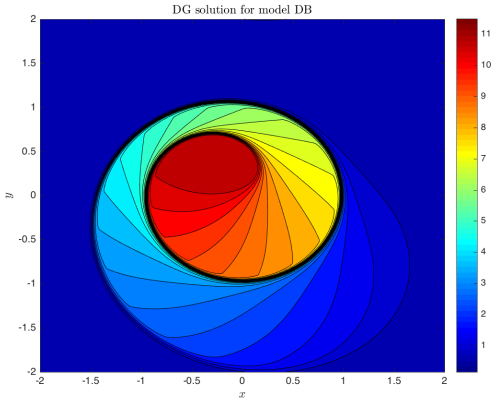
$$u_0(x, y) = \begin{cases} 3.5\pi & \text{if } x^2 + y^2 < 1, \\ 0.25\pi & \text{otherwise,} \end{cases}$$

and periodic boundary conditions are considered. This is a rather challenging test case, since a two-dimensional composite wave structure is present [18]. In Figure 5.29 we report the results, generated with the parameters listed in Table 5.22. Finally, the maximum artificial viscosity as a function of time is shown in Figure 5.30.

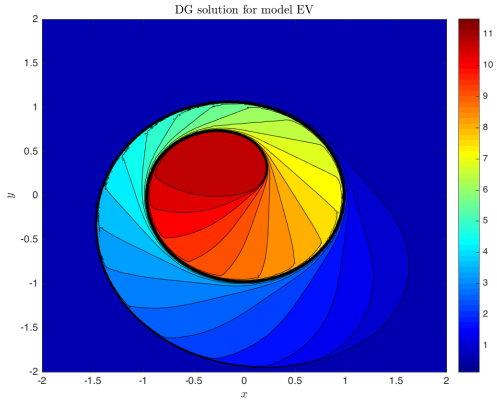
Model	$m = 4$
DB	$c_\beta = 2,\ c_{max} = 1$
EV	$c_E = 1,\ c_{max} = 0.25$
MDH	$c_A = 2,\ c_\kappa = 0.4,\ c_{max} = 0.5$
MDA	$c_{max} = 0.8$

TABLE 5.22: Parameter values for the standard artificial viscosity models for the KPP problem.

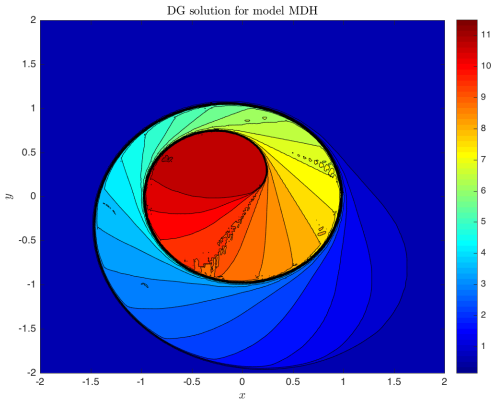
It appears that that the best models are the DB, EV and the 2D ANN ones. The decay-based techniques are not well suited for such a problem, since a lot of wiggles are present. This phenomenon is present independently of the chosen parameter values. Even the 1D network exhibits some spurious oscillations, yet they are less evident compared to the MDH and MDA models. The artificial viscosity profile confirms our reasoning. The 2D NN adds more dissipation in comparison with the 1D version, which results in a smoother solution profile. However, the time evolution for all the models looks rather similar. The CFL constant was set to $C = 0.8$.



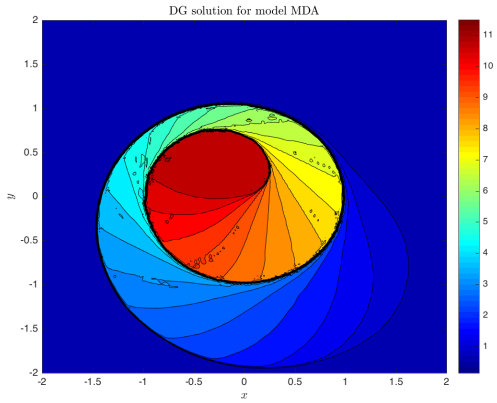
(a) DB



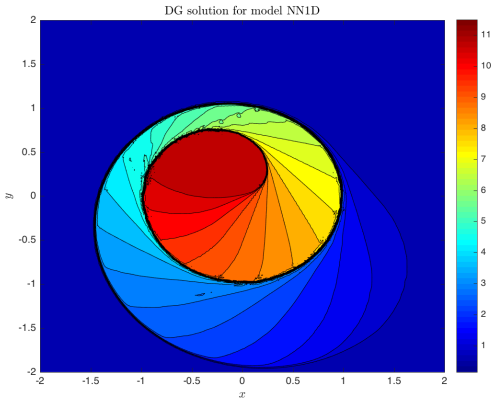
(b) EV



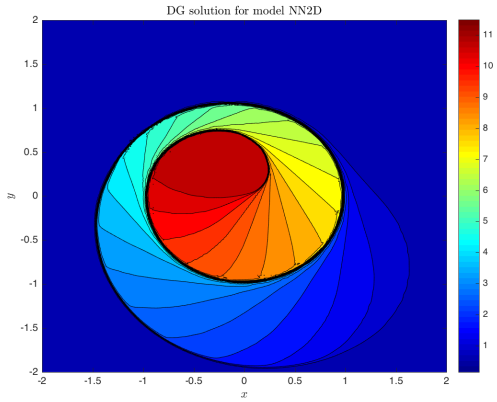
(c) MDH



(d) MDA



(e) NN1D



(f) NN2D

FIGURE 5.29: Numerical results for the KPP problem, $m = 4$.

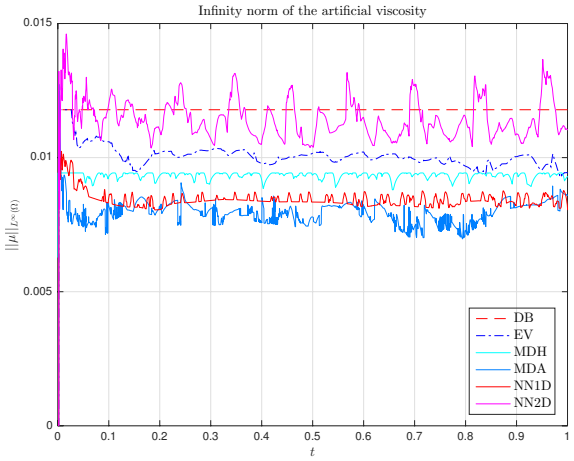


FIGURE 5.30: Infinity norm of the artificial viscosity with respect to time, KPP problem, $m = 4$.

5.4 Two-dimensional Euler system

In this Section we focus on the two-dimensional Euler system. For practical reasons, we skip the convergence analysis, which turned out to be very similar to the one-dimensional one. In particular, both the 1D and the 2D networks are able to achieve optimal convergence rates. Instead, we prefer to focus on two-dimensional Riemann problems, whose detailed description is provided in [47, 48]. Different combinations of rarefaction, shock and contact waves lead to at least 15 different admissible configurations.
The physical domain is again $\Omega = [0, 1]^2$, which is enlarged to $\Omega = [-1, 2]^2$ to avoid any effect from the boundaries (see Subsection 5.3.2). Periodic boundary conditions are applied. Then, we divide each edge into $N = 40 \cdot 3$ intervals. Each configuration is determined by cutting the domain in four equal parts, in which a different initial value for $\boldsymbol{u} = (\rho, \rho v_x, \rho v_y, E)$ is assigned. As it was done in the one-dimensional scenario, we instead assign conditions for (ρ, v_x, v_y, p), that are again converted in the conserved variables using the ideal gas law (2.24).
The results presented in this Section are obtained using different CFL constants as m is varied. In particular, $C = 0.2, 0.6, 0.6, 1.5$ are set for $m = 1, 2, 3, 4$ respectively. For the sake of simplicity, we report the contour lines for the density variable only. Their number is always set to 30.

5.4.1 Riemann problem case 4

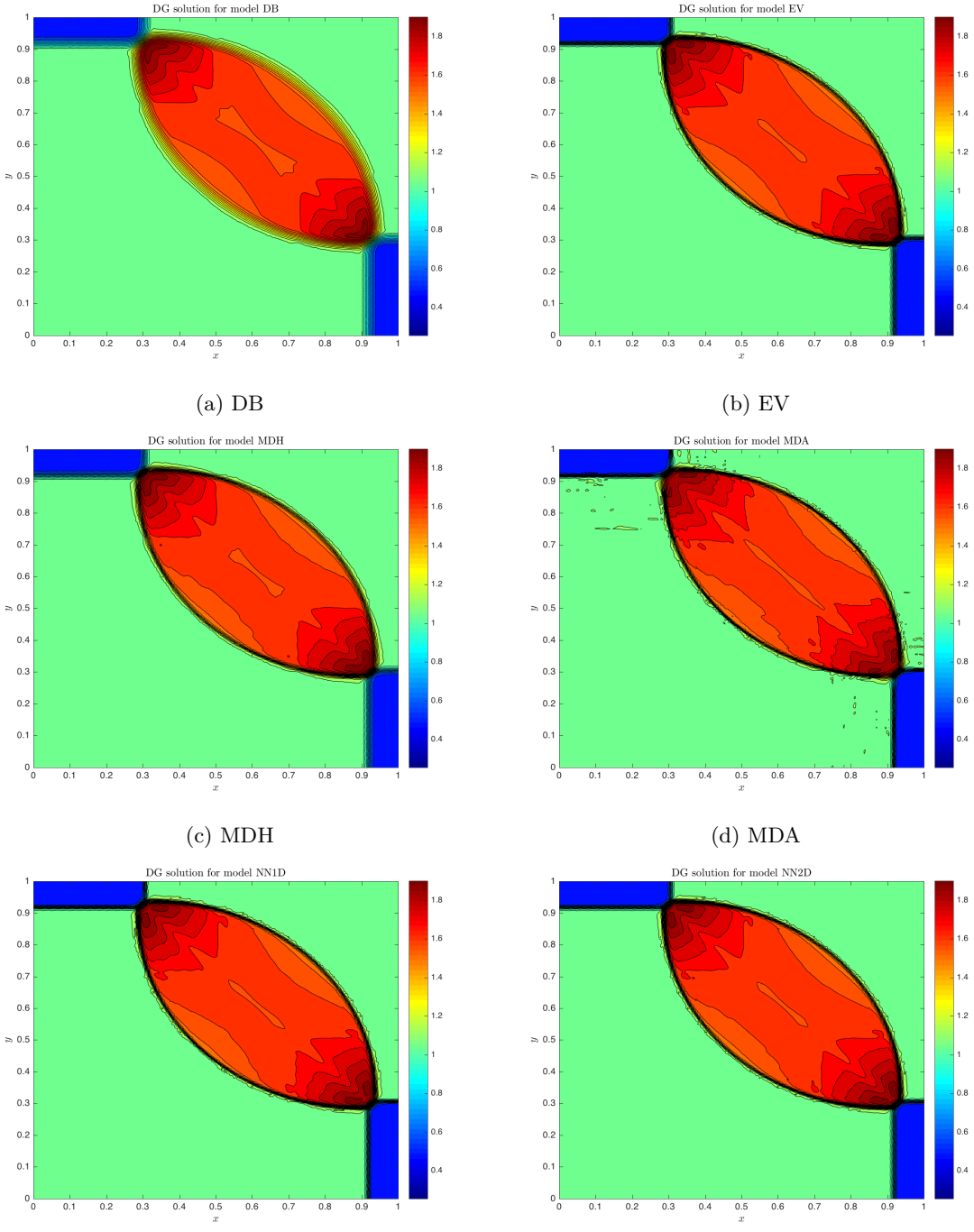
In this scenario, only shock waves are present. We set $T = 0.25$ and

$$(\rho, v_x, v_y, p)_0 = \begin{cases} (1.1, 0, 0, 1.1) & \text{if } 0.5 < x < 1,\ 0.5 < y < 1, \\ (0.5065, 0.8939, 0, 0.35) & \text{if } 0 < x < 0.5,\ 0.5 < y < 1, \\ (1.1, 0.8939, 0.8939, 1.1) & \text{if } 0 < x < 0.5,\ 0 < y < 0.5, \\ (0.5065, 0, 0.8939, 0.35) & \text{if } 0.5 < x < 1,\ 0 < y < 0.5. \end{cases}$$

The results, generated with the set of parameters listed in Table 5.23, are reported in Figure 5.31 for a degree $m = 4$ only.

Model	$m = 4$
DB	$c_\beta = 2,\ c_{max} = 0.5$
EV	$c_E = 1,\ c_{max} = 0.25$
MDH	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 0.5$
MDA	$c_{max} = 0.5$

TABLE 5.23: Parameter values for the standard artificial viscosity models for Riemann problem (case 4), Euler system.



(a) DB (b) EV

(c) MDH (d) MDA

(e) NN1D (f) NN2D

FIGURE 5.31: Numerical results for the 2D Riemann problem, configuration 4, $m = 4$.

Again, the DB is the most dissipative model and the MDA the least dissipative one. Both of them are not acceptable models in this framework. The best technique seems to be the EV, since the contour lines appear more smoothed and less wiggles are present. Both the NN-based profiles look rather similar, with the 1D version exhibiting a slightly more oscillatory behavior.

5.4.2 Riemann problem case 12

This problem is characterized by the presence of both contact waves and shocks. We set $T = 0.25$ and

$$(\rho, v_x, v_y, p)_0 = \begin{cases} (0.5313, 0, 0, 0.4) & \text{if } 0.5 < x < 1,\ 0.5 < y < 1, \\ (1, 0.7276, 0, 1) & \text{if } 0 < x < 0.5,\ 0.5 < y < 1, \\ (0.8, 0, 0, 1) & \text{if } 0 < x < 0.5,\ 0 < y < 0.5, \\ (1, 0, 0.7276, 1) & \text{if } 0.5 < x < 1,\ 0 < y < 0.5. \end{cases}$$

Since it is a rather challenging problem, we choose to provide the results for all the discretization degrees. They are obtained with parameters listed in Table 5.24, and are reported in Figures 5.32, 5.33, 5.34 and 5.35 for degrees $m = 1, 2, 3, 4$ respectively.

Model	$m = 1$	$m = 2$	$m = 3$	$m = 4$
DB	$c_\beta = 2,\ c_{max} = 0.8$	$c_\beta = 2,\ c_{max} = 0.8$	$c_\beta = 2,\ c_{max} = 0.8$	$c_\beta = 2,\ c_{max} = 0.5$
EV	$c_E = 1.5,\ c_{max} = 1$	$c_E = 1,\ c_{max} = 1$	$c_E = 1,\ c_{max} = 1$	$c_E = 1,\ c_{max} = 0.5$
MDH	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 1.2$	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 1.2$	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 1.2$	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 1.2$
MDA	-	-	$c_{max} = 0.5$	$c_{max} = 0.5$

TABLE 5.24: Parameter values for the standard artificial viscosity models for Riemann problem (case 12), Euler system.

A simple yet effective way to show the robustness of a model is to evaluate how the fine structure close to $(x, y) = (0.5, 0.5)$ is captured. For low degrees, the resolution is too low to fully resolve it, with all the models smoothing the solution too much around the center of the domain. Increasing the discretization degrees, the results improve a lot. The MDA model is very dissipative for $m = 3$, while the contact waves seem to show some non-physical oscillations for $m = 4$. The DB model is rather dissipative, as expected. The MDH and EV models provide good results, as well as both the proposed NN-based techniques. In particular, the solutions obtained with the 2D networks seems to be the best among all the models, with this trend holding for all the considered degrees.

5.4.3 Riemann problem case 6

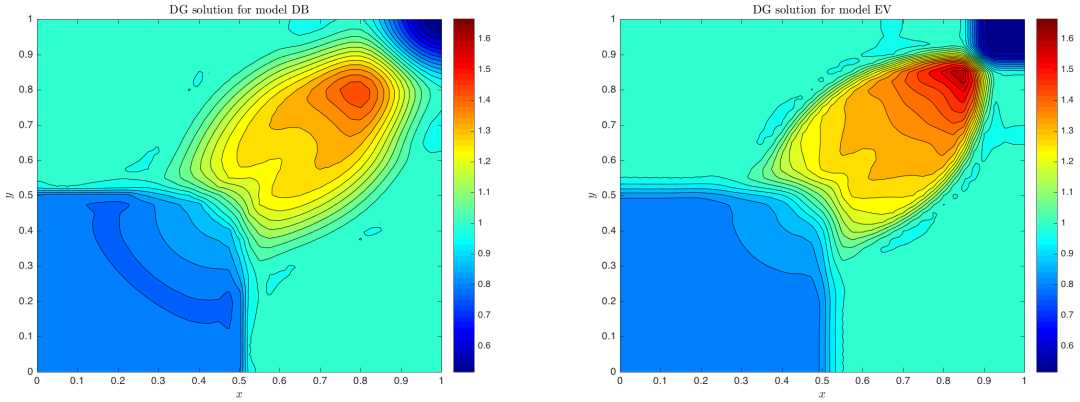
The final test exhibits four contact waves. We set $T = 0.3$ and

$$(\rho, v_x, v_y, p)_0 = \begin{cases} (1, 0.75, -0.5, 1) & \text{if } 0.5 < x < 1,\ 0.5 < y < 1, \\ (2, 0.75, 0.5, 1) & \text{if } 0 < x < 0.5,\ 0.5 < y < 1, \\ (1, -0.75, 0.5, 1) & \text{if } 0 < x < 0.5,\ 0 < y < 0.5, \\ (3, -0.75, -0.5, 1) & \text{if } 0.5 < x < 1,\ 0 < y < 0.5. \end{cases}$$

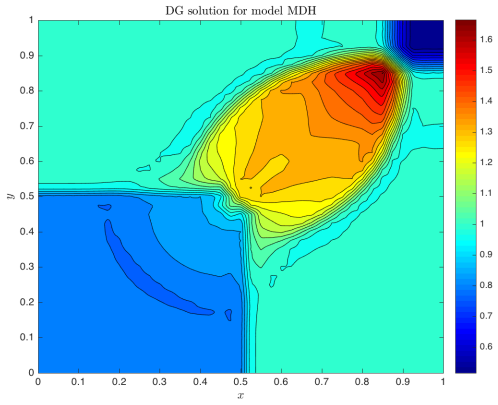
The results, generated with the set of parameters listed in Table 5.25, are reported in Figure 5.36 for a degree $m = 4$ only.

Model	$m = 4$
DB	$c_\beta = 2,\ c_{max} = 0.5$
EV	$c_E = 1,\ c_{max} = 0.25$
MDH	$c_A = 2.5,\ c_\kappa = 0.2,\ c_{max} = 0.5$
MDA	$c_{max} = 0.5$

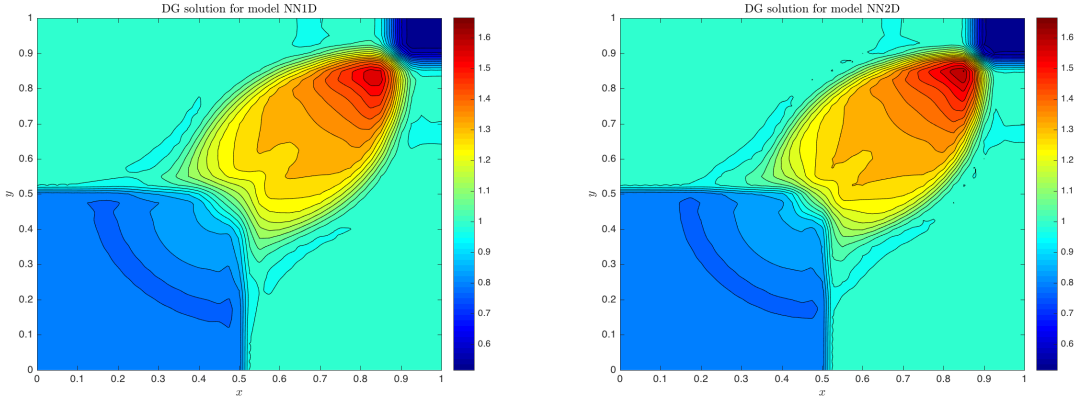
TABLE 5.25: Parameter values for the standard artificial viscosity models for Riemann problem (case 6), Euler system.



(a) DB (b) EV

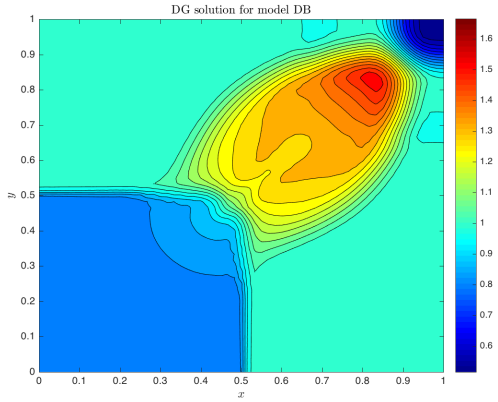


(c) MDH

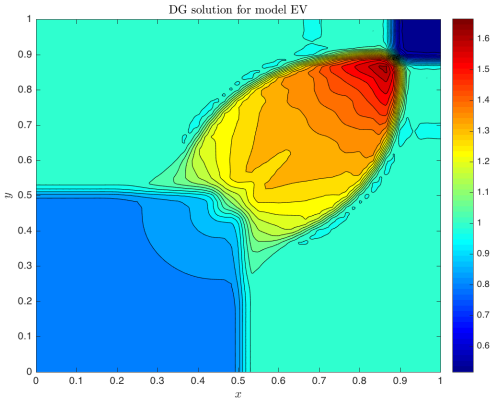


(d) NN1D (e) NN2D

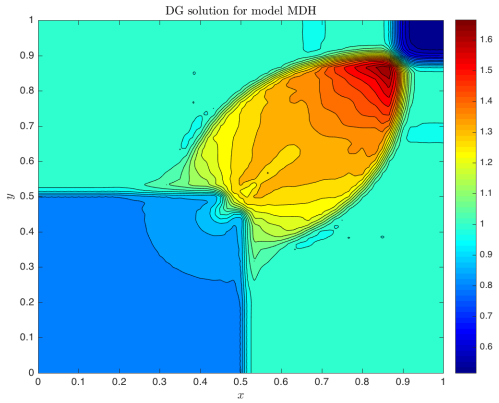
FIGURE 5.32: Numerical results for the 2D Riemann problem, configuration 12, $m = 1$.



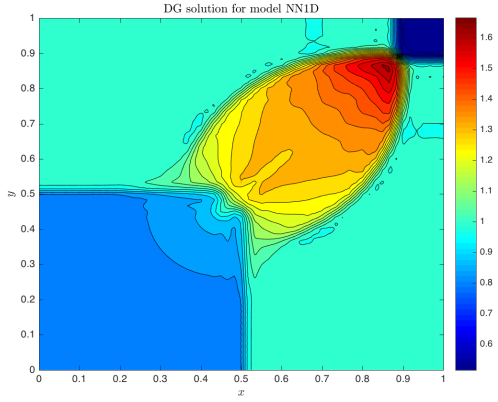
(a) DB



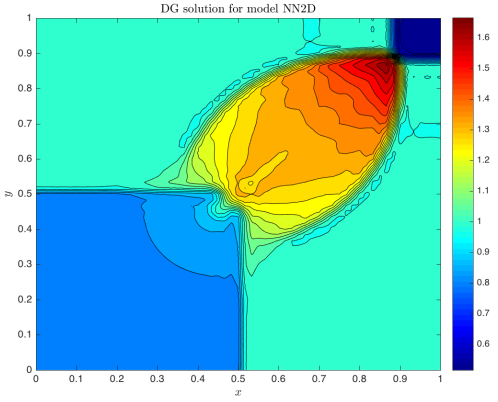
(b) EV



(c) MDH

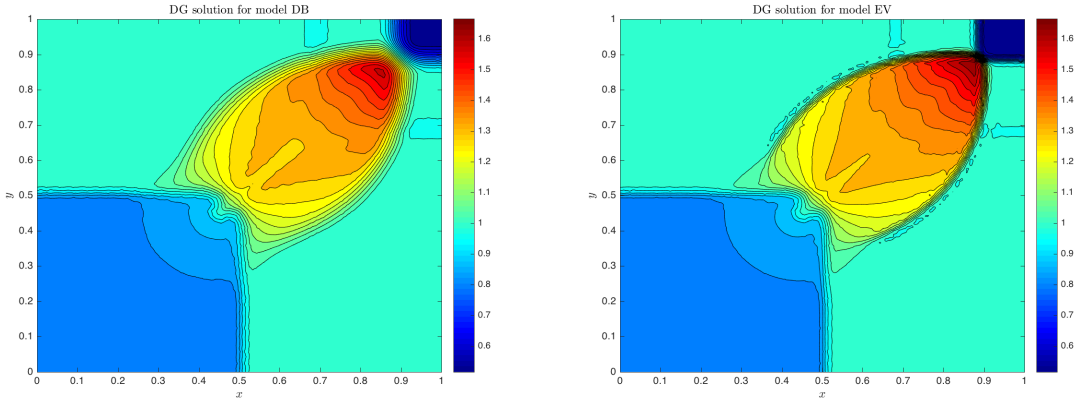


(d) NN1D



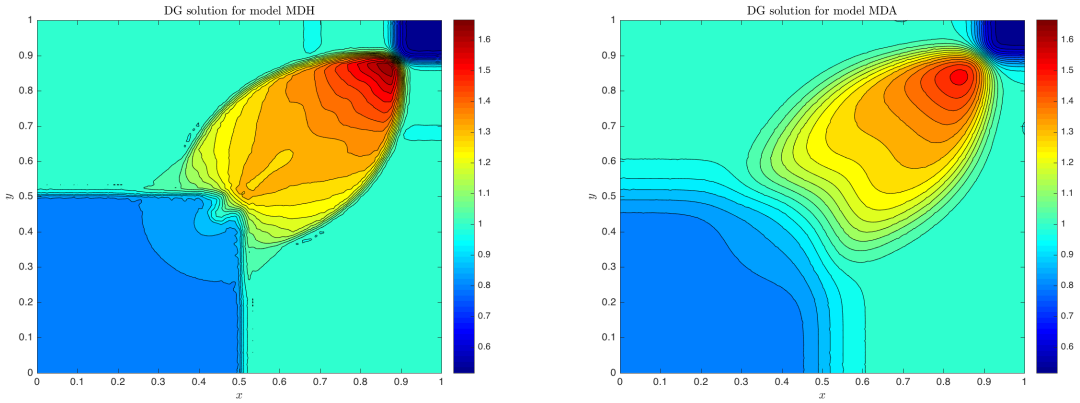
(e) NN2D

FIGURE 5.33: Numerical results for the 2D Riemann problem, configuration 12, $m = 2$.



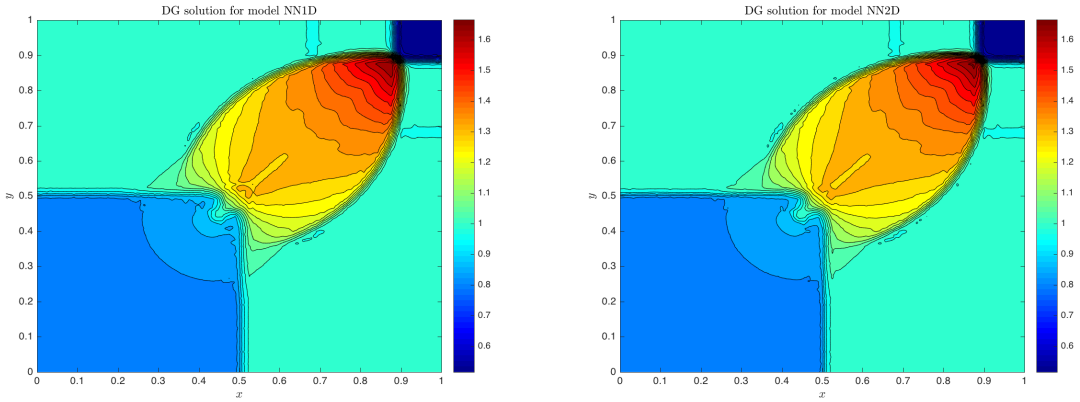
(a) DB

(b) EV



(c) MDH

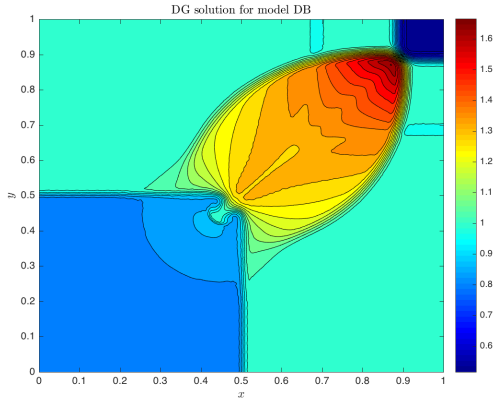
(d) MDA



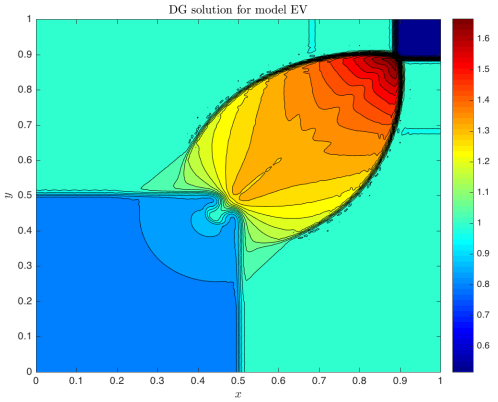
(e) NN1D

(f) NN2D

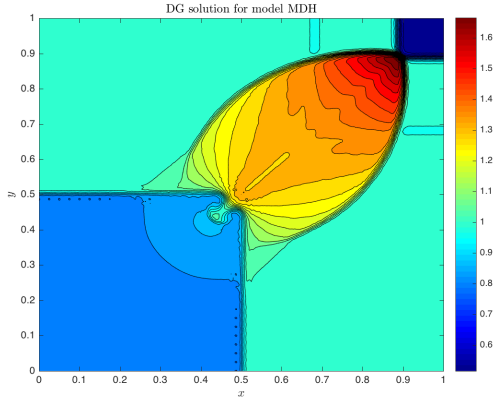
FIGURE 5.34: Numerical results for the 2D Riemann problem, configuration 12, $m = 3$.



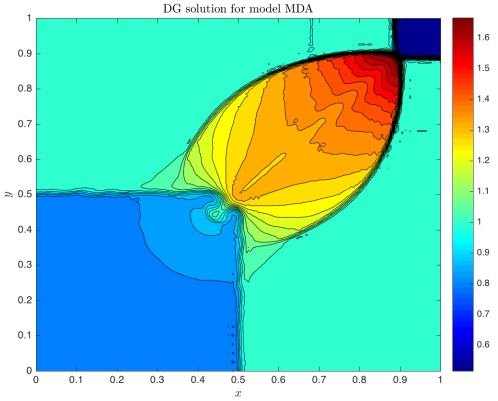
(a) DB



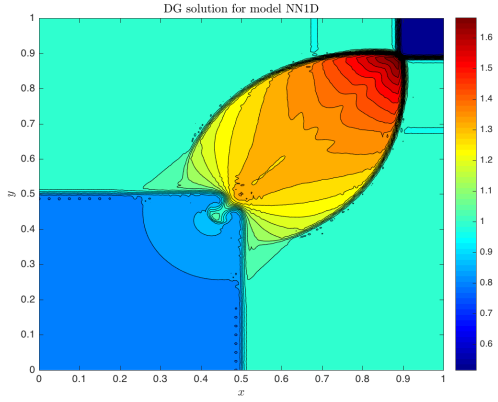
(b) EV



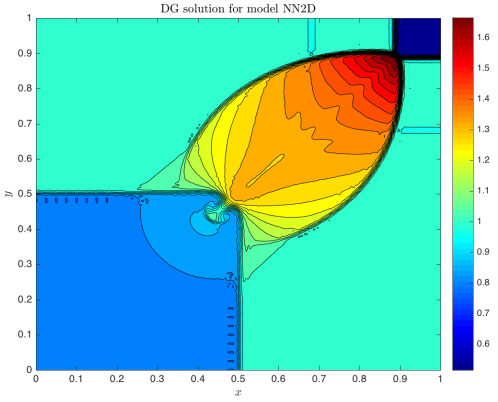
(c) MDH



(d) MDA

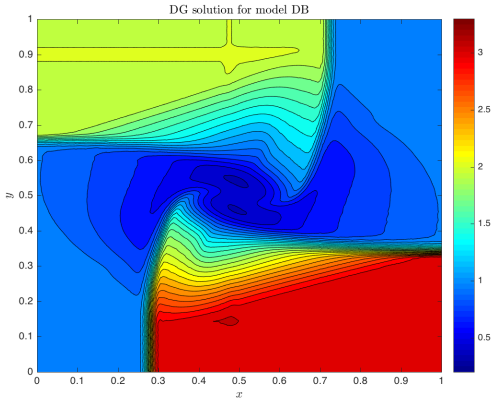


(e) NN1D

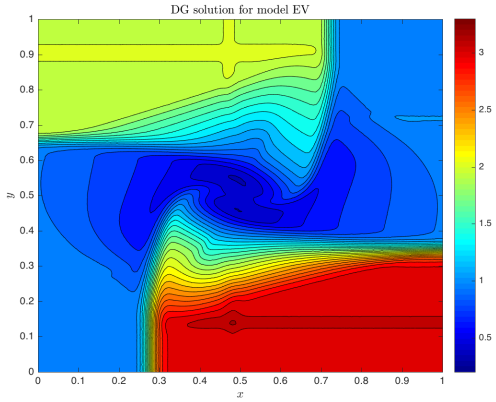


(f) NN2D

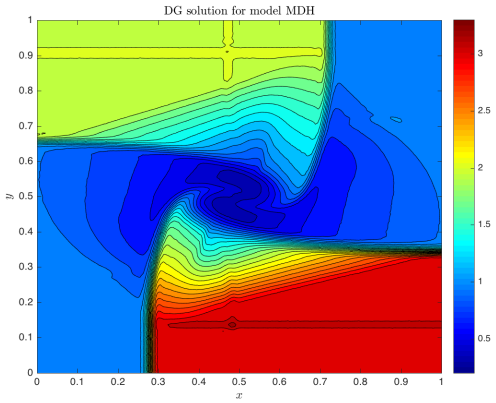
FIGURE 5.35: Numerical results for the 2D Riemann problem, configuration 12, $m = 4$.



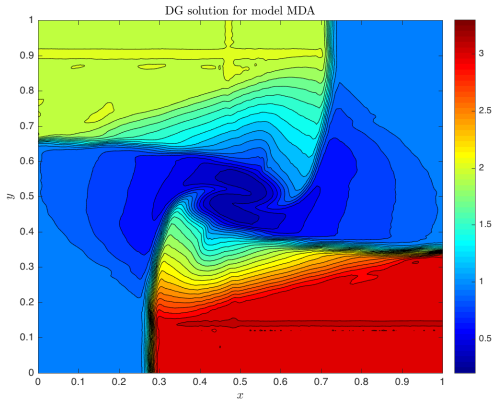
(a) DB



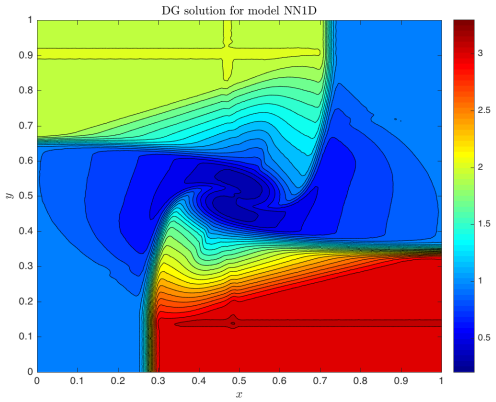
(b) EV



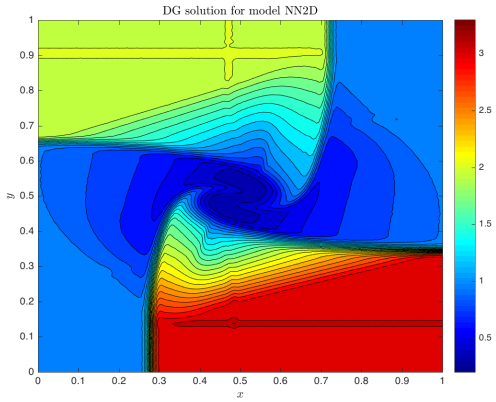
(c) MDH



(d) MDA



(e) NN1D



(f) NN2D

FIGURE 5.36: Numerical results for the 2D Riemann problem, configuration 6, $m = 4$.

Unlike its one-dimensional counterpart consisting of a single contact wave, the DB adds dissipation in the problem. It happens because the velocity variable does not stay constant in time, so that dissipation is injected, at least after the initial states. The EV appears to provide the smoothest solution. This could be a parameter issue, in the sense that there might be a better set of parameters in a range we did not explore, but it is coherent with [14], where the EV model turned out to be quite dissipative for complex flow structures. All the other models seems to provide similar results, with the networks performing well even with contact waves only. The 2D NN appears to be better than the 1D one, and it turns out to be the best model for this test case.

5.5 Performance analysis

In this Section, we provide more detailed comments on the performances of the proposed methods, highlighting strengths and weaknesses of the techniques. We choose to keep the analysis separated from the numerical results in order not to deviate from the main goal of the previous Sections, i.e. showing the capabilities of the schemes in terms of accuracy and shock-capturing. Moreover, this is not meant to be a fully rigorous analysis, which is left as a future work. Firstly, we make a few general comments, while a couple of quantitative results is shown afterwards.
The computational cost per temporal iteration is rather similar among all the considered models. This fact can be explained due to a couple of arguments. Firstly, the viscosity estimation procedure is similar among most of the models, aiming to compute a local dissipation value. The differences among the techniques do not play a significant role, and cache effects might contribute to reduce such discrepancies. Secondly, the computational cost required by the assembly of the right-hand-side $\mathcal{F}$ in (2.16) is much larger than the one required by the viscosity estimation procedure. When dealing with systems of conservation laws, this phenomenon is enhanced. Moreover, the computation of the artificial viscosity is carried out once per time iteration, while the right-hand-side has to be assembled five times, since a five-stage time-advancing scheme is employed (see Algorithm 4.1).
On the other hand, the total execution time can exhibit significant variations. Indeed, we recall that the time step is chosen adaptively according to (2.18). Thus, different viscosity values have an impact on Δt, consequently altering the number of iterations and the computational time. Qualitatively, large viscosities result in a large number of iterations, even though a trade-off between μ and the maximum wave speed is present. The variations in the elapsed time become more evident for two-dimensional problems, where the solution exhibits more complex features.
Now, we provide more a quantitative analysis. For one-dimensional problems, the selected test case is the Shu-Osher problem presented in Subsection 5.2.4, to which we refer for the numerical setup. Table 5.26 reports the computational times, number of iterations and the time per iteration for different models and discretization degrees.

Model	$m=1$			$m=2$			$m=3$			$m=4$		
	Time	Iter	TpI	Time	Iter	TpI	Time	Iter	TpI	Time	Iter	TpI
DB	3.6271	850	4.27e−3	13.0587	3439	3.80e−3	28.4422	7763	3.66e−3	55.3077	13825	4.00e−3
EV	3.1480	839	3.75e−3	14.4412	3382	4.27e−3	31.5348	7676	4.11e−3	60.0911	13688	4.39e−3
MDH	3.3275	843	3.95e−3	12.4094	3393	3.65e−3	28.5726	7696	3.71e−3	55.0017	13722	4.01e−3
MDA	-	-	-	-	-	-	30.1544	7593	3.97e−3	58.1663	13763	4.23e−3
NN	4.1515	843	4.92e−3	14.3815	3408	4.22e−3	30.9583	7703	4.02e−3	60.1432	13735	4.38e−3

TABLE 5.26: Computational times, iterations and time per iteration for the Shu-Osher problem. Both the total time and the time per iteration are expressed in seconds.

Essentially, the time per iteration can be considered constant for all the models and degrees. From one side, the slight differences among the models are explained due to the different techniques employed to estimate the viscosity. On the other hand, increasing the degree m one may expect the time per iteration to increase. This is not the case, possibly due to memory optimizations. The total computational time and iterations are essentially independent of the chosen model. In most of the cases, the MDH model appears to be the fastest.
A prototype problem for two-dimensional cases is the Riemann problem for Euler system (configuration 12), described in Subsection 5.4.2. The results of the performance analysis are reported in Table 5.27.

Model	$m = 1$			$m = 2$			$m = 3$			$m = 4$		
	Time	Iter	TpI	Time	Iter	TpI	Time	Iter	TpI	Time	Iter	TpI
DB	393.49	258	1.52e0	927.33	416	2.23e0	3811.26	1231	3.10e0	6384.41	1482	4.31e0
EV	267.87	174	1.54e0	516.42	228	2.27e0	1885.91	597	3.16e0	2937.97	696	4.23e0
MDH	280.82	186	1.51e0	721.20	331	2.18e0	3542.61	1162	3.05e0	6482.58	1516	4.27e0
MDA	-	-	-	-	-	-	3102.84	1000	3.10e0	6262.62	1470	4.26e0
NN1D	260.27	165	1.58e0	496.40	222	2.24e0	2094.54	672	3.12e0	3179.92	744	4.27e0
NN2D	236.26	152	1.55e0	476.21	213	2.24e0	2013.19	648	3.11e0	3522.84	822	4.29e0

TABLE 5.27: Computational times, iterations and time per iteration for the 2D Riemann problem (configuration 12). Both the total time and the time per iteration are expressed in seconds.

The time per iteration is constant among the models, while it clearly increases with m. Since more degrees of freedom are present, such a behavior is expected. The total computational time is very different. The DB appears to be the slowest one, possibly due to the impact of high dissipation values on the time step. The MDH and MDA seem to become less efficient for high orders. Both the 1D and the 2D NN-based techniques are computationally fast, with performances comparable to the EV model.
To sum up, we claim that the new technique does not negatively impact on the computational cost. The performances per iteration are comparable with most of the models, with no significant overhead caused by the application of the neural networks. Moreover, the new model is a parameter-free method, so that a possibly larger computational cost is compensated by eliminating the tuning of the parameters. A more extensive performance evalutation would consists of choosing a constant, i.e. non-adaptive, time step, fixed a priori in compliance with condition (2.18) and independent of the viscosity model. Based on our results, in this scenario we expect the elapsed time to be similar among the models.

Chapter 6

Conclusion

In this work we addressed the problem of estimating an optimal artificial viscosity amount in high-order numerical solvers for conservation laws. The focus was given to the Runge-Kutta Discontinuous Galerkin method. An overview of some classical artificial viscosity models was first provided. Despite showing great potential, they suffer from the dependence on empirical parameters. Since no optimal rule is available to obtain the best values, their tuning has to be done in a problem-dependent fashion, creating additional computational cost.
Thus, we proposed a new approach based on artificial neural networks. The multilayer perceptron model was the chosen architecture. The networks have been trained in an offline time-consuming process by means of a robust dataset, constructed by collecting data from simulations run using the classical models with optimal parameters. The online evaluation was then performed inside the Runge-Kutta time-advancing loop. A few versions of the MLP, which mainly differ in the choice of input and output variables, have been proposed, highlighting their potential and drawbacks. Coherently with our expectations, we found that the best strategy is to estimate a scaled viscosity coefficient using a suitable scaled scalar quantity as predictor. An inverse scaling is carried out by multiplying with a local wave speed and a factor which takes into account the solution jump and the grid spacing. With this choice, we are able to predict the viscosity based on the solution features, keeping a consistent behavior in terms of scaling and guaranteeing high-order accuracy. Moreover, such a technique is parameter-free and acts as a black box, i.e. a very general tool to estimate the required amount of dissipation.
A significant part of this work was devoted to demonstrate the capabilities of the new model. We showed that, despite training the NN using rather simple problems, it can be applied to more general contexts (different mesh resolutions, non-convex flux functions, ...). We benchmarked the models using smooth problems, observing that the DB and the MDH models might fail to get high-order accuracy, while the NN often guarantees an optimal convergence rate. However, more emphasis was given to non-smooth cases, in order to present the shock-capturing properties of the model. The NN technique guarantees results comparable to (or better than) the optimal model among the classical ones, and does not suffer from issues that might be present with standard models. For instance, the DB might fail in problems with a constant velocity field, and the EV model can be too dissipative when complex structures are present. A further advantage of the new method lies in the fact that, unlike the parameter-dependent techniques, positivity constraints in Euler system were always satisfied.
A performance analysis was also carried out, showing that no computational overheads are present when applying the neural networks. The cost appears to comparable with other models.
Thus, we conclude that the trade-off among accuracy in the results, universality of the method and computational cost clearly goes in the direction of the NN-based models.
Among the possible extensions to this work, we highlight three possible directions. The first relates to port it to three-dimensional problems. No significant issues should arise switching from 2D to 3D cases, and we believe that the proposed technique could be easily extended. The second involves a generalization to polynomial orders greater than four. Again, the implementation should not substantially vary, although the network hyperparameters might have to be tuned again. Finally, a more extensive performance analysis can be carried out, starting from the presented results.

Bibliography

[1] Z. J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. T. Huynh, et al. *High-order CFD methods: Current status and perspective*. 2013. arXiv: fld.1 [DOI: 10.1002].

[2] J. S. Hesthaven, T. Warburton. *Nodal discontinuous Galerkin methods*. Vol. 54 TS - C. 2008.

[3] B. L. Rozhdestvenskii. "Discontinuous solutions of hyperbolic systems of quasilinear equations". *Russ. Math. Surv.* 15.53 (1960).

[4] D. Gottlieb, C.-W. Shu. "On the Gibbs Phenomenon and Its Resolution". *SIAM Review* 39.4 (1997), pp. 644–668.

[5] J. S. Hesthaven. *Numerical Methods for Conservation Laws: From Analysis to Algorithms*. SIAM Publishing, 2018, p. 555.

[6] B. Cockburn, C.-W. Shu. "The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V". *Journal of Computational Physics* 141.2 (1998), pp. 199–224. arXiv: arXiv:1011.1669v3.

[7] C.-W. Shu. "Discontinuous Galerkin Methods: General Approach and Stability". *Numerical solution of partial differential equations* (2009), pp. 149–202.

[8] B. Cockburn, S. Y. Lin, C. W. Shu. "TVB runge-kutta local projection discontinuous galerkin finite element method for conservation laws III: One-dimensional systems". *Journal of Computational Physics* 84.1 (1989), pp. 90–113.

[9] J. Qiu, C.-W. Shu. "Runge-Kutta Discontinuous Galerkin Method Using WENO Limiters". *SIAM Journal on Scientific Computing* 26.3 (2005), pp. 907–929.

[10] J. Zhu, J. Qiu, C. W. Shu, M. Dumbser. "Runge-Kutta discontinuous Galerkin method using WENO limiters II: Unstructured meshes". *Journal of Computational Physics* 227.9 (2008), pp. 4330–4353.

[11] D. Ray, J. S. Hesthaven. "An artificial neural network as a troubled-cell indicator". *Journal of Computational Physics* 367 (2018), pp. 166–191.

[12] A. W. Cook, W. H. Cabot. "Hyperviscosity for shock-turbulence interactions". *Journal of Computational Physics* 203.2 (2005), pp. 379–385.

[13] A. Mani, J. Larsson, P. Moin. "Suitability of artificial bulk viscosity for large-eddy simulation of turbulent flows with shocks". *Journal of Computational Physics* 228.19 (2009), pp. 7368–7374.

[14] J. Yu, J. S. Hesthaven. "A comparative study of shock capturing models for the discontinuous Galerkin method". *EPFL-Article* 231188 (2017), pp. 1–42.

[15] D. Moro, N. C. Nguyen, J. Peraire. "Dilation-based shock capturing for high-order methods". *International Journal for Numerical Methods in Fluids* 82.7 (2016), pp. 398–416.

[16] P.-O. Persson, J. Peraire. "Sub-Cell Shock Capturing for Discontinuous Galerkin Methods". *44th AIAA Aerospace Sciences Meeting and Exhibit*. 2006.

[17] A. Klöckner, T. Warburton, J. S. Hesthaven. "Viscous Shock Capturing in a Time-Explicit Discontinuous Galerkin Method". *Mathematical Modeling of Natural Phenomena* 6.3 (2011), pp. 57–83. arXiv: 1102.3190.

[18] J. L. Guermond, R. Pasquetti, B. Popov. "Entropy viscosity method for nonlinear conservation laws". *Journal of Computational Physics* 230.11 (2011), pp. 4248–4267.

[19] V. Zingan, J. L. Guermond, J. Morel, B. Popov. "Implementation of the entropy viscosity method with the discontinuous Galerkin method". *Computer Methods in Applied Mechanics and Engineering* 253 (2013), pp. 479–490.

[20] R. Hartmann. "Adaptive discontinuous Galerkin methods with shock-capturing for the compressible NavierStokes equations". *International Journal for Numerical Methods in Fluids* 51.9-10 (2006), pp. 1131–1156.

[21] F. Bassi, A. Crivellini, A. Ghidoni, S. Rebay. "High-order discontinuous Galerkin discretization of transonic turbulent flows". *47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* January (2009).
[22] G. E. Barter, D. L. Darmofal. "Shock capturing with PDE-based artificial viscosity for DGFEM: Part I. Formulation". *Journal of Computational Physics* 229.5 (2010), pp. 1810–1827.
[23] S. Haykin. "A comprehensive foundation". *Neural Networks* 2.2004 (2004), p. 41. arXiv: arXiv:1312.6199v4.
[24] K. Rudd, S. Ferrari. "A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks". *Neurocomputing* 155 (2015), pp. 277–285.
[25] J. Hesthaven, S. Ubbiali. "Non-intrusive reduced order modeling of nonlinear problems using neural networks". *Journal of Computational Physics* 363 (2018), pp. 55–78.
[26] G. Cybenko. *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient*. Tech. rep. Department of Computer Science, Tufts University, 1988.
[27] G. Cybenko. "Approximations by superpositions of sigmoidal functions". *Approximation Theory and its Applications* 9.3 (1989), pp. 17–28.
[28] A. Quarteroni, R. Sacco, F. Saleri. *Numerical Mathematics*. 2007, p. 655.
[29] M. H. Carpenter, a. Kennedy. "Fourth-Order Kutta Schemes". *Nasa Technical Memorandum* 109112 (1994), pp. 1–26.
[30] J. S. Hesthaven. "From Electrostatics to Almost Optimal Nodal Sets for Polynomial Interpolation in a Simplex". *SIAM Journal on Numerical Analysis* 35.2 (1998), pp. 655–676.
[31] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. 2002, pp. 129–138. arXiv: 9809069v1 [arXiv:gr-qc].
[32] http://www.cse.psu.edu/~rtc12/CSE486/lecture05_6pp.pdf. [Online; accessed 29-Jun-2018].
[33] http://www2.lawrence.edu/fast/GREGGJ/Math400/Section_12_5.pdf. [Online; accessed 22-May-2018].
[34] C. M. Dafermos. *Hyperbolic Conservation Laws in Continuum Physics*. Springer-Verlag Berlin Heidelberg, 2010.
[35] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007. arXiv: arXiv:1011.1669v3.
[36] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016.
[37] https://www.robertoreif.com/blog/2017/12/16/importance-of-feature-scaling-in-data-modeling-part-1-h8nla. [Online; accessed 03-Jul-2018].
[38] A. Krogh, J. A. Hertz. "A Simple Weight Decay Can Improve Generalization". *Advances in Neural Information Processing Systems* 4 (1992), pp. 950–957.
[39] A. L. Maas, A. Y. Hannun, A. Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". *Proceedings of the 30 th International Conference on Machine Learning* 28 (2013), p. 6.
[40] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, R. Garcia. "Incorporating second-order functional knowledge for better option pricing". *Advances in Neural Information Processing Systems* January 2002 (2001), pp. 472–478.
[41] https://en.wikipedia.org/wiki/Rectifier_(neural_networks). [Online; accessed 30-May-2018].
[42] https://www.quora.com/What-are-hyperparameters-in-machine-learning. [Online; accessed 04-Jul-2018].
[43] D. Kingma, J. Ba. "Adam: a method for stochastic optimization". *arXiv:1412.6980* (2014), pp. 1–13. arXiv: 1412.6980.
[44] B. Cockburn, C.-W. Shu. "TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework". *Mathematics of Computation* 52.186 (1989), p. 411.
[45] J. Qiu, C.-W. Shu. "A Comparison of Troubled-Cell Indicators for Runge-Kutta Discontinuous Galerkin Methods Using Weighted Essentially Nonoscillatory Limiters". *SIAM Journal on Scientific Computing* 27.3 (2005), pp. 995–1013.
[46] G. A. Sod. "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws". *Journal of Computational Physics* 27 (1978), pp. 1–31.
[47] A. Kurganov, E. Tadmor. "Solution of Two-dimensional Riemann Problems for Gas Dynamics without Riemann Problem Solvers". *Numerical Methods for Partial Differential Equations* 18.5 (2002), pp. 584–608.
[48] C. W. Schulz-Rinne. "Classification of the Riemann problem for two-dimensional gas dynamics". *SIAM J. Math. Anal.* 24.1 (1993), pp. 76–88.

[49] A. Quarteroni. *Numerical Models for Differential Problems*. Springer, 2014.
[50] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". *None* 1.212 (2015), p. 19. arXiv: 1603.04467.
[51] M. Bartholomew-Biggs, S. Brown, B. Christianson, L. Dixon. "Automatic differentiation of algorithms". *Journal of Computational and Applied Mathematics* 124.1-2 (2000), pp. 171–190. arXiv: arXiv:1011.1669v3.
[52] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". *Neural Networks* 61 (2015), pp. 85–117.
[53] https://github.com/tminka/lightspeed. [Online; accessed 09-Jul-2018].
[54] S. Salsa. *Partial Differential Equations in Action*. 2008, p. 568. arXiv: arXiv:1011.1669v3.
[55] D. Ghosh, J. D. Baeder. "Compact Reconstruction Schemes with Weighted ENO Limiting for Hyperbolic Conservation Laws". *SIAM Journal on Scientific Computing* 34.3 (2012), A1678–A1706.
[56] G. Puppo, M. Semplice. "Numerical Entropy and Adaptivity for Finite Volume Schemes". *Communications in Computational Physics* 10.05 (2011), pp. 1132–1160.
[57] http://www.ita.uni-heidelberg.de/~dullemond/lectures/num_fluid_2012/Chapter_6.pdf. [Online; accessed 17-Jul-2018].
[58] C. W. Shu, S. Osher. "Efficient implementation of essentially non-oscillatory shock-capturing schemes, II". *Journal of Computational Physics* 89 (1989), pp. 32–78.