Méthodes numériques basées sur l'apprentissage pour les EDP hyperboliques et cinétiques

Léo Bois

▶ **To cite this version:**

Léo Bois. Méthodes numériques basées sur l'apprentissage pour les EDP hyperboliques et cinétiques. Mathématiques [math]. Université de Strasbourg, 2023. Français. NNT : 2023STRAD060 . tel-04589397

HAL Id: tel-04589397
https://theses.hal.science/tel-04589397

Submitted on 27 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

INSTITUT DE RECHERCHE MATHÉMATIQUE AVANCÉE

UMR 7501

Strasbourg

présentée pour obtenir le grade de docteur de
l'Université de Strasbourg
Spécialité MATHÉMATIQUES APPLIQUÉES

Léo Bois

Méthode numériques basées sur l'apprentissage pour les EDP hyperboliques et cinétiques

Soutenue le 19 décembre 2023
devant la commission d'examen

Philippe Helluy, directeur de thèse
Stéphane Clain, rapporteur
Nicolas Crouseilles, rapporteur
Bruno Despres, examinateur
Virginie Grandgirard, examinatrice
Emmanuel Franck, examinateur
Laurent Navoret, examinateur
Vincent Vigon, examinateur

https://irma.math.unistra.fr

Université de Strasbourg

Remerciements

Je tiens à remercier en premier lieu Emmanuel Franck, Laurent Navoret et Vincent Vigon de m'avoir recruté pour ce projet et de m'avoir fait confiance tout au long de la thèse. Je remercie également Philippe Helluy d'avoir accepté d'être mon directeur et de sa participation au jury.

Je remercie Stéphane Clain et Nicolas Crouseilles du temps qu'ils ont consacré à rapporter cette thèse, ainsi que Bruno Despres et Virginie Grandgirard d'avoir accepté de participer à mon jury.

Enfin, je remercie mes parents et mes amis d'avoir été à mes côtés pendant ces trois années.

Table des matières

1 Introduction 9
1.1 Préambule . . . 9
1.2 Réseaux de neurones . . . 11
1.2.1 Principes généraux . . . 11
1.2.2 Multilayer perceptrons . . . 12
1.2.3 Réseaux de neurones convolutifs . . . 12
1.3 Du modèle cinétique au modèle fluide . . . 14
1.3.1 Modèle cinétique . . . 14
1.3.2 Modèle fluide . . . 14
1.3.3 Fermeture du modèle fluide . . . 15
1.4 Méthodes numériques pour les systèmes hyperboliques . . . 16
1.4.1 Méthode de Galerkin discontinu . . . 17
1.4.2 Méthode Lattice-Boltzmann . . . 18
1.5 Résumé des chapitres . . . 19

2 A neural network closure for the 1D Euler-Poisson system based on kinetic simulations 27
2.1 Introduction . . . 27
2.2 Fluid closure for Vlasov-Poisson . . . 29
2.2.1 Kinetic model . . . 29
2.2.2 Fluid model . . . 30
2.3 Closure with a neural network . . . 31
2.3.1 Interpolation of the heat flux with a neural network . . . 31
2.3.2 Detailed composition of the closure . . . 32
2.3.3 Architecture of the neural network . . . 34
2.3.4 Training of the neural network . . . 35
2.4 Data generation . . . 36
2.4.1 Global description . . . 37
2.4.2 Random initial conditions . . . 37
2.4.3 Random Knudsen numbers and recording times . . . 38
2.4.4 Computing of the fluid quantities . . . 38
2.5 Data processing . . . 39
2.5.1 Input standardization . . . 39
2.5.2 Output normalization . . . 40
2.5.3 Slicing and reconstructing . . . 40
2.5.4 Smoothing of the output . . . 41
2.6 Results . . . 42
2.6.1 Accuracy of the network . . . 42
2.6.2 Fluid model with the neural network . . . 45
2.6.3 Using the network with different configurations . . . 52
2.7 Conclusion . . . 57

Appendices **63**
2.A Numerical scheme for the kinetic model . . . 63
2.A.1 Time discretization . . . 63
2.A.2 Spatial discretization . . . 64
2.B Numerical scheme for the fluid models . . . 64
2.B.1 Explicit scheme for the neural network or the kinetic closure . . . 65
2.B.2 Semi-implicit scheme for the Navier-Stokes closure . . . 65

3 A neural network based closure for the 2D Boltzmann-BGK equation **67**
3.1 Introduction . . . 67
3.2 Models . . . 68
3.2.1 Kinetic model . . . 68
3.2.2 Fluid model . . . 69
3.3 Neural network closure . . . 69
3.3.1 Neural network architecture . . . 70
3.3.2 Training and testing dataset . . . 70
3.3.3 Data processing and augmentation . . . 71
3.3.4 Stabilization techniques . . . 71
3.4 Numerical results . . . 72
3.4.1 Accuracy on test dataset . . . 72
3.4.2 Stability of the fluid model . . . 72
3.4.3 Accuracy of the fluid model . . . 75
3.5 Conclusion . . . 75

Appendices **81**
3.A Numerical considerations . . . 81
3.A.1 Kinetic model . . . 81
3.A.2 Fluid model . . . 81
3.B Derivation of the fluid equations . . . 82
3.B.1 Conservation of momentum . . . 83
3.B.2 Conservation of energy . . . 83
3.C Data augmentation . . . 84
3.C.1 Density . . . 84
3.C.2 Mean velocity . . . 84
3.C.3 Pressure tensor . . . 85
3.C.4 Heat flux . . . 85
3.C.5 f^A is solution to the Boltzmann equation . . . 85
3.C.6 Application to data augmentation . . . 86
3.D Smoothing implementation . . . 87

4 An optimal control deep learning method to design artificial viscosities for Discontinuous Galerkin schemes **89**
4.1 Introduction . . . 89
4.2 An optimal control method for parameterized schemes . . . 91
4.2.1 Optimization problem . . . 91
4.2.2 Gradient descent and back-propagation . . . 91
4.2.3 Optimization on sub-trajectories using the reference solutions . . . 92
4.2.4 Algorithm . . . 93
4.3 Design of an artificial viscosity for discontinuous Galerkin schemes . . . 93
4.3.1 Discontinuous Galerkin method and artificial viscosity . . . 93
4.3.2 Definition of the cost function . . . 94
4.3.3 Neural network viscosity function . . . 96
4.3.4 Training data . . . 96
4.4 Numerical Results . . . 98
4.4.1 Advection equation . . . 98
4.4.2 Burgers equation . . . 101
4.4.3 Euler system . . . 104

4.5 Conclusion . . . 108

Appendices **111**
4.A Reference artificial viscosity models . . . 111

5 Neural network based relaxation matrix for vectorial Lattice-Boltzmann methods in 2D **113**
5.1 Lattice-Boltzmann equations . . . 113
5.2 Numerical scheme . . . 115
5.3 Data-driven relaxation matrix . . . 116
5.4 Numerical results . . . 120
5.4.1 Toy example . . . 120
5.4.2 Linear advection equation . . . 122
5.4.3 Burgers equation . . . 125
5.4.4 Wave equation . . . 129
5.5 Conclusion . . . 132

Appendices **135**
5.A Residual error in the limit $\Delta t \to 0$ . . . 135
5.A.1 Expansion of one iteration of the numerical scheme . . . 135
5.A.2 Expansion of the residual error . . . 136

6 Conclusion **139**

Chapitre 1

Introduction

1.1 Préambule

L'ensemble des travaux réalisés pendant la thèse s'intéresse à la simulation de fluides ou de plasmas.

À l'échelle microscopique, un fluide peut être décrit comme un ensemble de particules qui intéragissent les unes avec les autres, notamment à travers leurs collisions. Dans le cas d'un plasma, les particules en question sont des ions ou des électrons, qui ont donc une charge électrique et intéragissent également sous l'effet de forces électro-magnétiques. Le principe fondamental de la dynamique, qui relie l'accélération d'un corps à la somme des forces qui s'éxercent sur lui, peut être appliqué à chacune de ces particules pour en décrire son comportement en fonction de sa position et de sa vitesse par rapport aux autres particules. Il en résulte un système d'équations différentielles, avec autant d'équations qu'il y a de particules, couplées les unes avec les autres. Complété par la connaissance de la position et de la vitesse de chaque particule à un instant initial, la solution de ce système permet de décrire l'évolution dans le temps du fluide ou du plasma considéré. Ce modèle, extrêmement précis, souffre de deux défauts essentiels. Le premier, c'est qu'il requiert une connaissance bien trop fine de la condition initiale, qui échappe à nos instruments de mesure. Le deuxième, c'est la complexité qui résulte du couplage de milliards d'équations, et qui rend la résolution —analytique ou numérique— impossible avec nos outils actuels. En pratique, ce sont donc d'autres modèles qui doivent être utilisés pour simuler de tels systèmes de particules.

À l'échelle macroscopique, ce sont des propriétés comme la densité, la température ou la pression, qui sont mesurées et utilisées pour décrire localement l'état d'un fluide ou d'un plasma. Ces quantités donnent une description incomplète du système, car elles ne rendent pas compte de l'état de chaque particule individuellement, mais elles sont suffisantes pour les applications pratiques. Malheureusement, la connaissance de ces quantités à un instant initial ne suffit pas, en général, à prédire leur évolution future. Il est cependant possible de faire certaines approximations qui permettent de mettre en équations la dynamique des quantités macroscopiques, mais leur champ d'application se limite au domaine de validité de ces approximations. Dans ce domaine de validité, la description du fluide se fait alors par un système avec autant d'équations qu'il y a de quantités macroscopiques étudiées. Comme ces quantités sont fonctions à la fois de l'espace et du temps, il s'agit d'équations aux dérivées partielles, contrairement aux équations du modèle microscopique. Les équations classiques sont celles d'Euler ou de Navier-Stokes. Sauf cas particuliers, il n'est pas possible de les résoudre de manière exacte, et il faut donc recourir à des méthodes numériques pour en calculer une solution approchée. Ces méthodes passent par la discrétisation de l'espace, qui ramène chaque équation aux dérivées partielles à autant d'équations différentielles ordinaires qu'il y a de mailles dans cette discrétisation, chacune étant couplée aux équations qui portent sur les mailles voisines. Ainsi, la taille du problème macroscopique est bien moindre que celle du problème microscopique. En pratique, cela rend sa résolution numérique possible, pour un coût de calcul plus ou moins élevé selon la finesse du maillage requise pour capturer la dynamique du fluide étudié. Cela fait du modèle macroscopique le modèle de choix pour les applications qui s'y prêtent.

À l'interface entre le modèle microscopique, au coût excessif, et le modèle macroscopique, au champ d'application limité, se situe un modèle dit mésoscopique. Comme le modèle microscopique, le modèle mésoscopique regarde la position et la vitesse des particules qui composent le fluide ou le plasma

étudié. Mais plutôt que de considérer chaque particule individuellement, il s'intéresse à leur distribution statistique dans l'espace des phases, produit cartésien de l'espace physique et de l'espace des vitesses. En substance, cette distribution donne, en tout point de l'espace donné et pour toute vitesse donnée, le nombre de particules situées à cette position et qui vont à cette vitesse. Il s'agit d'une description du système plus riche que la seule donnée des quantités macroscopiques, ces dernières pouvant se déduire de la distribution des particules dans l'espace des phases. Connaissant cette distribution à un instant initial, son évolution dans le temps est solution de l'équation de Boltzmann pour un fluide, ou l'équation de Vlasov pour un plasma. Comme la distribution des particules dans l'espace des phases est une fonction de la position, de la vitesse et du temps, il s'agit comme pour le modèle macroscopique d'équations aux dérivées partielles. Là encore, sauf cas particulier, la solution exacte de ces équations n'est pas connue, et des méthodes numériques sont nécessaires pour en faire une résolution approchée. Ces méthodes numériques nécessitent non seulement la discrétisation de l'espace physique, comme pour le modèle macroscopique, mais aussi celle de l'espace des vitesses. Cette discrétisation supplémentaire rend la résolution numérique sensiblement plus coûteuse en mémoire et en ressources de calculs, au point qu'elle ne peut pas toujours être envisagée. C'est pourquoi la recherche d'approximations pour l'obtention d'un modèle macroscopique adapté à chaque application est une question qui fait preuve d'un grand intérêt.

Dans cette thèse, on s'intéresse à deux problèmes qui surviennent dans la simulation de fluides. Dans les régimes fortement collisionnels où les équations d'Euler peuvent être utilisées, leur résolution numérique n'est pas toujours aisée. Comme il s'agit de régimes très peu diffusifs, les solutions exhibent généralement des discontinuités, que les méthodes numériques standard ont tendance à lisser. Utiliser un maillage plus fin améliore le résultat, mais pour certaines applications le coût de calcul associé devient prohibitif. Une alternative consiste à utiliser des méthodes dites d'odre élevé, qui capturent mieux les discontinuités. Cependant, ces méthodes peuvent générer des oscillations numériques au voisinage de ces discontinuités. Le premier problème auquel on s'intéresse est la mise au point de méthodes numériques qui proposent le meilleur compromis possible entre diffusion, oscillations et temps de calcul. Pour les régimes moins collisionnels, les équations d'Euler et de Navier-Stokes ne permettent plus de décrire correctement la dynamique du fluide. Il peut alors être fait usage du modèle mésoscopique, mais comme indiqué précédemment sa simulation est très coûteuse. Le deuxième problème auquel on s'intéresse est la conception de fermetures pour fournir un modèle macroscopique adapté à ces régimes.

Les travaux décrits ici tentent d'adresser ces deux problèmes en exploitant les récents progrès en apprentissage machine, via l'utilisation de réseaux de neurones. Dans ce contexte, un réseau de neurones désigne une fonction mathématique avec une structure particulière, et dont les opérations sont configurables via des paramètres. Faire varier ces paramètres permet donc d'explorer un ensemble de fonctions, à la recherche de l'une d'entre elles qui permettrait d'effectuer la tâche souhaitée. Comme un réseau de neurones peut avoir de très nombreux paramètres —jusqu'à plusieurs dizaines de milliers dans cette thèse, des milliards dans d'autres applications—, il n'est pas envisageable de procéder par une recherche en grille, tant il y aurait de combinaisons à tester. Au lieu de cela, la méthode utilisée est celle de la descente de gradient, qui part d'un jeu de paramètres quelconque pour le modifier progressivement via un processus itératif visant à améliorer les performances du réseau de neurones à chaque étape. Comme son nom le laisse entendre, cette méthode s'appuie sur la différentiation du réseau de neurones par rapport à ses paramètres ; on utilise pour ce faire un algorithme appelé rétro-propagation. Cette optimisation des paramètres est malgré tout très coûteuse, ce qui explique que les réseaux de neurones aient vu leur essor avec le développement de machines de calcul de plus en plus performantes, qui font notamment usage de puissantes cartes graphiques. Depuis quelques années existent des bibliothèques de fonctions pour certains langages de programmation, notamment Python, qui implémentent cet algorithme et permettent d'exploiter facilement les cartes graphiques, rendant accessible le travail avec des réseaux de neurones à un large public. Tensorflow est utilisé dans cette thèse, mais on peut également citer Pytorch qui constitue une alternative très populaire.

Les trois parties qui suivent décrivent plus en détails les principales notions manipulées dans la thèse. D'abord les réseaux de neurones, car ils sont le principal outil utilisé dans les travaux effectués pendant le doctorat. Ensuite les modèles cinétique et fluide, et le passage du premier au second, pour introduire le problème de fermeture. Et enfin les méthodes numériques de Galerkin discontinu et de Lattice-Boltzmann pour la résolution approchée des équations de conservation, qui font l'objet des chapitres 4 et 5 respectivement.

1.2 Réseaux de neurones

1.2.1 Principes généraux

Un réseau de neurones peut se comprendre comme une fonction paramétrée, qui grâce à sa structure et ses nombreux paramètres peut implémenter un grand nombre de fonctions différentes. Un tel objet peut s'avérer particulièrement utile lorsqu'on suspecte l'existence d'une correspondance entre des données X et des données Y, mais que l'expression analytique de la fonction $X \mapsto Y$ est inconnue. Par exemple, l'une des premières applications des réseaux de neurones a été la reconnaissance automatique de chiffres manuscrits. Dans ce cas il s'agit d'effectuer une correspondance entre une image (numérisée) de taille donnée, et un chiffre compris entre 0 et 9. Le cerveau humain est généralement capable de faire cette correspondance sans problème, mais écrire « à la main » un programme qui ferait de même est extrêmement difficile. Il est en revanche possible d'utiliser un réseau de neurones en lieu et place de ce programme, et de modifier ses paramètres jusqu'à obtenir le résultat souhaité. On donne davantage de détails sur la méthode pour y parvenir ci-dessous. Dans les travaux décrits dans cette thèse, on se sert également de réseaux de neurones pour implémenter une certaine correspondance : X sera généralement la solution numérique à un instant donné, tandis que Y sera un élément du schéma numérique nécessaire pour calculer la solution à l'instant suivant.

Pour utiliser un réseau de neurones, la première étape consiste à écrire le problème sous la forme d'un problème d'optimisation, où on cherche une fonction f qui minimise une certaine fonctionnelle J. Une forme standard pour J est celle d'une somme d'erreurs commises par f, sur un jeu de données pour lequel on dispose du résultat souhaité :

$$J(f) = \sum_{(X_i, Y_i) \in \mathcal{D}} \|f_\theta(X_i) - Y_i\|,$$

où $\mathcal{D}$ est le jeu de données en question, Y_i le résultat souhaité pour la donnée X_i, et $\|.\|$ une fonction qui permet de quantifier l'erreur commise. On parle alors d'apprentissage supervisé, car cette approche se base sur des exemples de correspondances (X_i, Y_i), typiquement produits par des humains.

On décide ensuite d'une architecture de réseau de neurones, c'est-à-dire de la structure de la fonction paramétrée, avec laquelle on va tenter de minimiser J. Cela revient à chercher la fonction f dans un sous-espace particulier, qui correspond au sous-espace que décrit le réseau de neurones f_θ lorsqu'on fait varier ses paramètres θ. Le problème d'optimisation s'écrit alors

$$\min_{\theta \in \Theta} J(f_\theta),$$

où Θ est l'ensemble des jeux de paramètres admissibles, usuellement $\Theta = \mathbb{R}^p$ avec p le nombre de paramètres dans le modèle. À noter qu'en pratique, il est souvent nécessaire de tester plusieurs architectures avant de trouver celle qui fournit les meilleurs résultats. On présente deux types d'architectures différentes dans les parties suivantes : les *multilayer perceptrons* et les réseaux de neurones convolutifs.

Le processus d'optimisation des paramètres θ pour minimiser $J(\theta) = J(f_\theta)$ est appelé l'entrainement du réseau de neurones. Il consiste en une descente de gradient, qui part d'un jeu de paramètres θ quelconque (généralement tiré au hasard) et le modifie itérativement en appliquant la transformation

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta),$$

jusqu'à convergence vers un minimum local. En pratique, ce sont souvent des algorithmes de descente de gradient plus sophistiqués qui sont utilisés, qui permettent de converger plus rapidement ou vers un meilleur minimum local ; on peut par exemple citer l'ajout d'inertie ou encore l'algorithme ADAM. Dans tous les cas, l'évaluation du gradient $\nabla_\theta J(\theta)$ fait intervenir les dérivées partielles de la sortie du réseau de neurones par rapport à chacun de ses paramètres. Le calcul de ces dérivées en un point X est automatisé grâce à l'algorithme dit de rétro-propagation (*backpropagation* en anglais), dont le nom fait référence à la manière dont ces dérivées sont calculées : un réseau de neurones étant structuré comme une composée de fonctions —on parle de *couches* successives—, les dérivées se calculent bien en commençant par les paramètres de la dernière couche, puis en remontant les couches unes à unes en utilisant la règle de la chaine (*chain rule*).

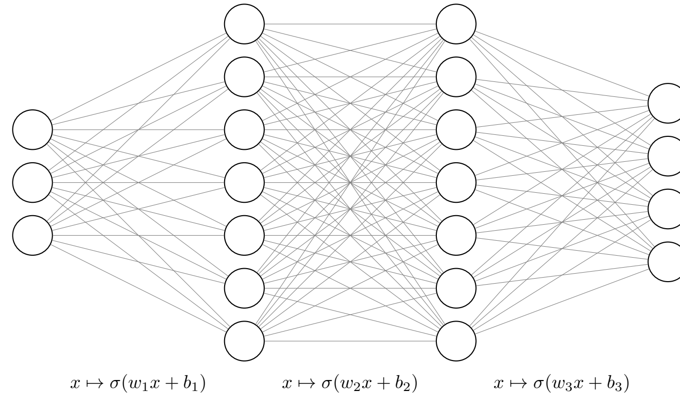


FIGURE 1.1 – Schéma illustrant un réseau de neurones de type *multilayer perceptron* avec une entrée de taille 3, une sortie de taille 4, et deux couches cachées de taille 7.

1.2.2 Multilayer perceptrons

Les *multilayer perceptrons* (MLP) sont les premiers modèles de réseaux de neurones qui ont été développés. Ce sont des modèles simples, inspirés des réseaux de neurones biologiques, et illustrés Figure 1.1. Dans cette figure, chaque cercle représente un *neurone*, qui combine les informations des neurones qui le précedent (à sa gauche), et communique son résultat aux neurones qui le suivent (à sa droite). Ainsi les neurones de la première colonne représentent les données en entrée du réseau de neurones, tandis que ceux de la dernière colonne représentent sa sortie. En particulier, le nombre de neurones dans ces deux colonnes est imposé par la tâche à effectuer par le réseau. Les autres colonnes sont appelées *couches cachées*, et peuvent elles être de taille arbitraire, ainsi qu'en nombre arbitraire.

Les calculs effectués par chaque neurone consistent en une simple combinaison linéaire de ses entrées, à laquelle est ajouté un scalaire, avant d'appliquer une fonction non linéaire. Ainsi, pour des variables $x_1, ..., x_n$ en entrée du neurone, sa sortie est donnée par

$$y = \sigma \left(\sum_{i=1}^{n} w_i x_i + b \right),$$

avec $\sigma : \mathbb{R} \to \mathbb{R}$ la fonction non linéaire appelée *fonction d'activation*, et $w_1, ..., w_n, b \in \mathbb{R}$ les paramètres du neurones, appelés poids et biais respectivement. Ainsi le vecteur $\mathbf{y}$ de toutes les sorties des neurones d'une même couche s'écrit simplement $\mathbf{y} = \sigma(\mathbf{wx}+\mathbf{b})$, où σ est appliqué terme à terme, et où $\mathbf{w}$ désigne la matrice des poids de chaque neurone, et $\mathbf{b}$ le vecteur des biais.

L'ensemble des paramètres du réseau de neurones est l'ensemble des poids et des biais de tous les neurones. Ainsi, plus le réseau contient de couches cachées, ou plus ces couches contiennent de neurones, plus le modèle contient de paramètres et peut explorer un vaste espace de fonctions. Il est important de noter que les fonctions d'activation sont essentielles pour que cette dernière assertion soit vraie : sans fonction d'activation, le réseau de neurones n'est qu'une composition de fonctions affines et se réduit donc à une fonction affine, quel que soit la taille du modèle. Des choix populaires pour la fonction d'activation σ sont la fonction sigmoïde $x \mapsto \frac{1}{1-e^{-x}}$, la tangente hyperbolique, ou encore la *rectified linear unit* (ReLU) $x \mapsto \max(0, x)$.

1.2.3 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs sont des réseaux de neurones avec une structure différente des MLP, conçue pour être mieux adaptée aux données de type images, sons, ou séries temporelles. En effet le traitement de telles données par des MLP soulève plusieurs difficultés, décrites notamment par LeCun et al. [20]. En particulier, un MLP ignore complètement la structure des données en entrées (aucune notion de distance entre les composantes de l'entrée), et aucune invariance par translation n'est imposée par la structure du MLP. Pour remédier à ces problèmes, les réseaux de neurones convolutifs s'appuient notamment sur les deux idées suivantes : l'utilisation de neurones avec un champ de perception limité d'une part, et le partage de paramètres par plusieurs neurones d'autre part. Ensembles, ces deux idées

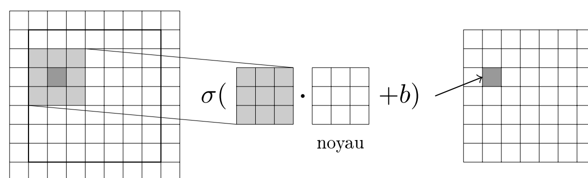


FIGURE 1.2 – Schéma de l'opération de convolution appliquée à une image en deux dimensions.

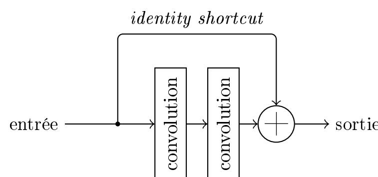


FIGURE 1.3 – Schéma d'un *identity shortcut*, comme introduit dans l'architecture ResNet.

aboutissent à l'opération de convolution représentée figure 1.2 pour une image 2D en entrée. Chaque pixel du résultat à droite s'obtient par combinaison de pixels voisins dans l'image de départ. Les poids utilisés pour cette combinaison sont toujours les mêmes, et rassemblés dans le noyau de convolution, ici de taille 3x3. Une fonction d'activation σ est appliquée au résultat comme pour les MLP.

Une « couche convolutive » est l'application de cette opération avec un ou plusieurs noyaux de convolutions en parallèle, résultant en autant d'images en sorties, concaténées selon une troisième dimension pour former une image en plusieurs « canaux ». Aussi, lorsque l'image en entrée est constituée de plusieurs canaux (ce qui est aussi le cas pour une image en rouge, vert, bleu par exemple), le noyau de convolution est constitué d'autant de canaux que l'image en entrée ; le reste de l'opération est inchangé. Plusieurs couches convolutives peuvent ainsi être enchainées successivement, pour obtenir un réseau de neurones convolutif.

Une autre opération que l'on retrouve souvent dans les réseaux de neurones convolutifs est celle de sous-échantillonnage (*subsampling* ou *pooling* en anglais), qui vise à réduire la taille de l'image non seulement pour réduire le coût des calculs en aval, mais aussi pour obtenir une certaine invariance aux petites perturbations locales. Une couche de sous-échantillonnage est typiquement insérée à intervalles réguliers entre des couches convolutives. En pratique, ce sous-échantillonnage est souvent implémenté en découpant l'image en carrés de taille 2x2, et en conservant la valeur la plus élevée dans chacun de ces carrés (*max pooling*), résultant en une image deux fois plus petite dans chaque direction. Une autre possibilité est l'usage de convolutions avec un paramètre de *strides* 2, qui consiste à faire glisser le noyau de convolution de deux pixels en deux pixels. À noter que cette deuxième possibilité, contrairement à la première, introduit des paramètres supplémentaires.

Des réseaux de neurones convolutifs sont utilisés dans plusieurs des chapitres qui suivent, pour tirer profit de leurs bonnes performances avec les images, dont se rapprochent les solutions d'équations aux dérivées partielles discrétisées sur des maillages uniformes. L'architecture de ces réseaux de neurones convolutifs s'inspirent de modèles qui se sont révélés performants pour l'analyse d'images, notamment le ResNet (*Residual Network*) [17] et le U-Net/V-Net [27, 25]. Le premier a introduit l'utilisation d'*identity shortcuts* qui consistent à additionner à la sortie d'une série de convolutions son entrée, comme représenté Figure 1.3, et permettent notamment de faciliter l'entrainement de réseaux de neurones convolutifs profonds. Le U-Net a été introduit pour traiter des problèmes de segmentation d'images, qui consistent en la classification de chacun des pixels de l'image parmi différentes catégories ; en particulier, la sortie d'un tel réseau de neurones est une image de la même taille que l'entrée. Afin de profiter des avantages et de l'analyse multi-échelles que permettent les opérations de sous-échantillonnage (*downsampling*), le U-Net introduit une opération de sur-échantillonnage (*upsampling*) qui permet de revenir à la résolution d'origine. Le V-Net est une variante du U-Net qui prend en entrée l'image entière (image volumétrique qui plus est) là où le U-Net la découpait en tuiles de taille donnée ; à noter que le V-Net ajoute également des *identity shortcuts* dans son architecture.

1.3 Du modèle cinétique au modèle fluide

1.3.1 Modèle cinétique

Le modèle cinétique s'intéresse à la distribution des particules dans l'espace des phases, notée $f(\mathbf{x}, \mathbf{v}, t)$, où $\mathbf{x} \in \mathbb{R}^d$ désigne la position, $\mathbf{v} \in \mathbb{R}^d$ la vitesse, et $t \in \mathbb{R}_+$ le temps. Dans cette thèse, on considère les dimensions $d = 1$ et $d = 2$. De cette distribution f peuvent se déduire les propriétés macroscopiques du fluide, qui sont des moments de la distribution en vitesse. Par exemple, la densité ρ, la vitesse moyenne $\mathbf{u}$ et la température T sont données par

$$\rho(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}, \qquad \mathbf{u}(\mathbf{x}, t) = \tfrac{1}{\rho} \int f(\mathbf{x}, \mathbf{v}, t)\mathbf{v}\, d\mathbf{v},$$
$$T(\mathbf{x}, t) = \tfrac{1}{\rho} \int f(\mathbf{x}, \mathbf{v}, t)\|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)\|^2\, d\mathbf{v}.$$

Dans le cas des gaz, l'évolution de la distribution f est donnée par l'équation de Boltzmann, constituée d'un terme d'advection et d'un terme de collisions $\mathcal{Q}$:

$$\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = \mathcal{Q}(f, f). \tag{1.1}$$

Le modèle considéré dans la suite utilise l'opérateur de collisions BGK (Bhatnagar-Gross-Krook), qui simplifie l'opérateur de collisions de Boltzmann par une relaxation vers un équilibre local $M(f)$:

$$\mathcal{Q}_{\mathrm{BGK}}(f, f) = \frac{1}{\varepsilon}(M(f) - f).$$

Cet équilibre est la maxwellienne de f, définie par

$$M(f) = \frac{\rho}{(2\pi T)^{d/2}} \exp\left(-\frac{\|\mathbf{v} - \mathbf{u}\|^2}{2T}\right),$$

où ρ, $\mathbf{u}$ et T désignent respectivement la densité, la vitesse moyenne et la température du gaz comme définis plus haut. Le paramètre $\varepsilon \in \mathbb{R}_+^*$ est le nombre de Knudsen, ratio entre le libre parcours entre deux collisions et une longueur caractéristique du système considéré. En pondérant la relaxation par $\frac{1}{\varepsilon}$, l'opérateur de collisions BGK est d'autant plus fort qu'il y a de collisions dans le fluide considéré.

Dans le chapitre 2, on s'intéresse à la simulation de plasmas en une dimension. Comme il s'agit d'un système de particules avec une charge électrique, l'évolution de la distribution f est donnée par l'équation de Vlasov, qui ajoute un terme d'intéraction électro-magnétiques à l'équation de Boltzmann. En une dimension, l'équation que l'on considère est la suivante :

$$\partial_t f + v\partial_x f - E\partial_v f = \frac{1}{\varepsilon}(M(f) - f),$$

où $E = E(x, t)$ désigne le champ électrique. Ce dernier est pris comme le champ électrique induit par les particules (pas de champ extérieur) et est donc donné par l'équation de Poisson

$$E = -\partial_x \phi, \quad \partial_{xx}\phi = \rho - \int \rho\, dx$$

1.3.2 Modèle fluide

Le modèle fluide décrit l'évolution des quantités macroscopiques, et peut se déduire du modèle cinétique décrit ci-dessus. La démarche consiste à se débarrasser de la variable $\mathbf{v}$ en remplaçant les distributions en vitesse $f(\mathbf{x}, ., t)$ par plusieurs de leurs moments (masse totale, moyenne, variance), afin d'obtenir une description du système par plusieurs fonctions, mais de $\mathbf{x}$ et t uniquement.

Ainsi, en intégrant l'équation de Boltzmann (1.1) par rapport à $\mathbf{v}$, on obtient l'équation de conservation de la masse du fluide :

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0,$$

où ρ et $\mathbf{u}$ désignent respectivement la densité et la vitesse moyenne du fluide, définis en fonction de f dans la partie précédente. Noter que le terme de droite s'annule car f possède la même densité que sa maxwellienne.

Si on multiplie l'équation de Boltzmann par $\mathbf{v}$ avant de l'intégrer en vitesse, on obtient l'équation de conservation de la quantité de mouvement

$$\partial_t(\rho\mathbf{u}) + \nabla \cdot \int f\mathbf{v} \otimes \mathbf{v}\, d\mathbf{v} = 0,$$

où $\mathbf{v} \otimes \mathbf{v}$ désigne le produit $\mathbf{v}\mathbf{v}^{\mathrm{T}}$. En notant $\mathbb{P}$ le tenseur de pression

$$\mathbb{P}(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)(\mathbf{v} - \mathbf{u}(\mathbf{x}, t)) \otimes (\mathbf{v} - \mathbf{u}(\mathbf{x}, t))\, d\mathbf{v},$$

l'intégrale dans le deuxième terme peut s'écrire

$$\int f\mathbf{v} \otimes \mathbf{v}\, d\mathbf{v} = \mathbb{P} + \rho\mathbf{u} \otimes \mathbf{u}.$$

En décomposant $\mathbb{P}$ en une partie scalaire pI et une partie de trace nulle Π, on a finalement

$$\partial_t(\rho\mathbf{u}) + \nabla \cdot (\rho\mathbf{u} \otimes \mathbf{u} + pI + \Pi) = 0,$$

où $p = \frac{1}{d}\int f\|\mathbf{v} - \mathbf{u}\|^2\, d\mathbf{v} = \rho T$ est la pression (scalaire), et $\Pi = \mathbb{P} - pI$ est le tenseur de stress.

De manière similaire, en multipliant l'équation de Boltzmann par $\frac{\|\mathbf{v}\|^2}{2}$ avant de l'intégrer en vitesse, on obtient l'équation de conservation de l'énergie

$$\partial_t \left(p + \tfrac{1}{2}\rho\|\mathbf{u}\|^2\right) + \nabla \cdot \left(2p\mathbf{u} + \tfrac{1}{2}\rho\|\mathbf{u}\|^2\mathbf{u} + \Pi\mathbf{u} + \mathbf{q}\right) = 0,$$

où $\mathbf{q}$ désigne le flux de chaleur :

$$\mathbf{q}(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)\|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)\|^2(\mathbf{v} - \mathbf{u}(\mathbf{x}, t))\, d\mathbf{v}.$$

On obtient ainsi un système de 3 équations sur les quantités ρ, $\mathbf{u}$ et p :

$$\begin{aligned}
&\partial_t\rho + \nabla \cdot (\rho\mathbf{u}) = 0 \\
&\partial_t(\rho\mathbf{u}) + \nabla \cdot (\rho\mathbf{u} \otimes \mathbf{u} + pI + \Pi) = 0 \\
&\partial_t \left(p + \tfrac{1}{2}\rho\|\mathbf{u}\|^2\right) + \nabla \cdot \left(2p\mathbf{u} + \tfrac{1}{2}\rho\|\mathbf{u}\|^2\mathbf{u} + \Pi\mathbf{u} + \mathbf{q}\right) = 0
\end{aligned}$$

Cependant ce système n'est pas fermé car il fait intervenir des moments de f supplémentaires, à savoir Π et $\mathbf{q}$. C'est pourquoi la simulation de fluides avec ce modèle requiert l'ajout d'une fermeture, qui donne les moments supplémentaires en fonction des moments simulés.

1.3.3 Fermeture du modèle fluide

Une première manière de fermer le modèle fluide est d'écrire la distribution f comme une perturbation de l'équilibre au voisinage de $\varepsilon = 0$, via un développement de Chapman-Enskog, par exemple

$$f = f_0 + \varepsilon f_1 + O(\varepsilon^2) \quad (\varepsilon \to 0).$$

En injectant ce développement dans les équations on obtient les équations d'Euler à l'ordre 1 et les équations de Navier-Stokes à l'ordre 2, qui correspondent respectivement aux fermetures

$$\Pi = 0, \quad \mathbf{q} = 0$$

et

$$\begin{aligned}
&\Pi = -\varepsilon p\sigma(\mathbf{u}), \quad \sigma(\mathbf{u}) = \nabla\mathbf{u} + \nabla\mathbf{u}^{\mathrm{T}} - \nabla \cdot \mathbf{u}I, \\
&\mathbf{q} = -2\varepsilon p\nabla T.
\end{aligned}$$

Des développements d'ordre supérieur permettent d'aboutir à d'autres équations comme celles de Burnett [4]. Cependant comme ces fermetures partent d'un développement de f au voisinage de $\varepsilon = 0$, elles sont d'autant moins précises que le fluide est dans un régime peu collisionnel. À noter qu'il est

possible de représenter le fluide avec davantage de moments que ρ, $\mathbf{u}$, p, et de déduire de l'équation de Boltzmann les équations correspondantes. C'est l'approche introduite par Grad [12] avec son modèle à 13 moments. C'est un modèle plus précis mais plus coûteux, et qui souffre notamment de problèmes de réalisabilité, c'est-à-dire qu'il est susceptible de générer des solutions qui ne correspondent à aucune distribution f positive. Du côté des plasmas, des fermetures peuvent être obtenues analytiquement pour intégrer certains phénomènes. La fermeture de Hammett-Perkins [15] en est un exemple qui intègre le *Landau damping*, et des développements plus récents peuvent être trouvés notamment dans [18].

Des approches faisant usage de réseaux de neurones sont proposées depuis quelques années. On peut citer les travaux de Han et al [16], qui reprennent l'introduction de moments additionnels, mais sous une forme plus générale et déterminés par un réseau de neurones de type auto-encodeur. Une des difficultés quand on use de réseaux de neurones pour la résolution d'équations aux dérivées partielles est de les contraindre à respecter les propriétés physiques du problème. Dans leurs récents travaux, Li et al [21] proposent une fermeture conçue justement de manière à posséder un certain nombre d'invariances qui découlent de l'équation de Boltzmann. Côté plasmas, la représentabilité par des réseaux de neurones de fermetures existantes comme celle de Hammett-Perkins a d'abord été étudiée dans [23] et [24], aboutissant à des résultats encourageants pour la recherche de nouvelles fermetures ainsi qu'à des recommandations sur les réseaux de neurones à utiliser pour y parvenir. On peut également citer [32] dont l'objectif est d'implémenter la fermeture *Landau fluid* (LF) à l'aide d'un réseau de neurones. Outre un travail réalisé dans cette thèse et qui fait l'objet du chapitre 2, on peut encore citer les travaux de Qin et al [26] qui utilisent l'approche des *physics-informed neural networks* (PINNs) pour construire un modèle fluide incorporant le *Landau damping*.

Un autre domaine qui fait intervenir un problème similaire de fermeture est celui de la modélisation de turbulences. En effet, il s'agit de systèmes particulièrement sujets aux phénomènes multi-échelles, où la dynamique d'intérêt, à grande échelle, est influencée par des dynamiques à des échelles plus petites mais dont la simulation est très coûteuse. On y retrouve donc naturellement des problèmes de fermeture du modèle macro pour y intégrer l'effet des phénomènes micro. On trouve deux approches pour y parvenir, d'une part les modèles RANS pour *Reynolds-averaged Navier-Stokes*, et d'autre part les LES pour *Large Eddy Simulations*. Des travaux impliquant l'apprentissage machine ont été effectués pour chacune de ces approches. Pour la première, on peut citer les travaux de Ling et al [22] qui introduisent une architecture de réseaux de neurones spécifique pour estimer le tenseur de stress de Reynolds de manière à en assurer l'invariance galiléenne. On peut également citer les travaux [31] et [33] qui utilisent la stratégie *physics-informed neural networks* pour prédire le tenseur de stress de Reynolds. En ce qui concerne l'approche LES, les premiers travaux impliquant l'usage de réseaux de neurones sont ceux de Beck et al [1], qui utilisent des simulations numériques directes (DNS) pour entrainer un réseau de neurones à faire la correspondance entre les variables macro et les termes de fermeture. Une approche similaire est utilisée dans les travaux de Lapeyre et al [19] et appliquée à la *premixed turbulent combustion*. De nombreux autres travaux impliquant les réseaux de neurones ont été effectués sur ces thématiques ; la revue de Vinuesa et al [28] en fait une liste bien plus riche, et qui s'étend à d'autres domaines de la mécanique des fluides numérique (*Computational Fluid Dynamics CFD*), à savoir l'accélération des simulations numériques directes (DNS) et la conception de modèles réduits.

1.4 Méthodes numériques pour les systèmes hyperboliques

Les équations d'Euler forment un système hyperbolique qui peut se mettre sous la forme

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0. \tag{1.2}$$

La méthode numérique naturelle pour résoudre de tels systèmes est celle des volumes finis car il s'agit d'une méthode simple et robuste qui satisfait naturellement les propriétés de conservation de ces équations. En revanche, il s'agit d'une méthode d'ordre 1 en espace, et qui peut donc nécessiter des maillages excessivement fins pour atteindre la précision souhaitée. On introduit dans cette partie les méthodes de Galerkin discontinu et de Lattice-Boltzmann, qui sont des alternatives potentiellement moins diffusives que les volumes finis, mais également moins robustes. En particulier, chacune de ces méthodes est susceptible de produire des oscillations lorsque la solution est discontinue, comme illustré Figure 1.4, et qui peuvent engendrer des problèmes de stabilité. Cet aspect est particulièrement problématique pour les systèmes hyperboliques, car ces derniers produisent facilement des discontinuités, quand bien

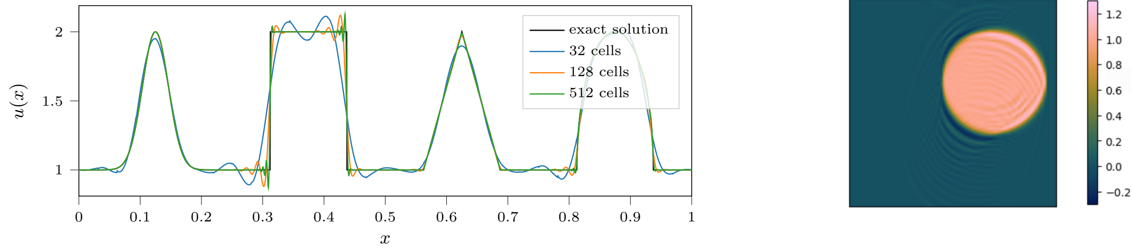


FIGURE 1.4 – Gauche : Oscillations produites par une méthode de Galerkin discontinu 1D d'ordre 4, lors de l'advection d'une fonction irrégulière (conditions aux bords périodiques). Droite : Oscillations produites par une méthode de Lattice-Boltzmann 2D d'ordre 2, lors de l'advection de la fonction indicatrice d'un disque.

même la condition initiale est régulière. Des moyens de trouver un bon compromis entre diffusion et oscillations pour chacune de ces deux méthodes font l'objet des chapitres 4 et 5 respectivement.

1.4.1 Méthode de Galerkin discontinu

La méthode de Galerkin discontinu permet d'obtenir des schémas numériques d'ordre arbitrairement élevé. Cette méthode consiste à approcher la solution du système par une fonction polynomiale sur chaque maille, sans contrainte de continuité à l'interface entre les mailles. L'ordre de convergence de la méthode va dépendre du degré maximal des polynômes en question.

On détaille la méthode ci-après en dimension 1, dans le cas d'un système réduit à une équation, la généralisation à plusieurs équations se faisant sans difficulté particulière. On considère une maille de la discrétisation, et une base des polynômes de degré inférieur ou égal à p définis sur cette maille. En dimension 1, une telle base est constituée de $p+1$ fonctions $\phi_0, ..., \phi_p$, ce qui permet d'écrire la solution approchée sur cette maille de la manière suivante :

$$u(x,t) = \sum_{i=0}^{p} u_i(t)\phi_i(x).$$

On approxime également $f(u)$ sur cette maille par un polynôme de degré au plus p pour pouvoir écrire

$$f(u(x,t)) = \sum_{i=0}^{p} f_i(t)\phi_i(x).$$

Ainsi, en multipliant l'équation (1.2) par un ϕ_j et en intégrant par rapport à x, il vient après intégration par parties :

$$\partial_t \sum_{i=0}^{p} u_i(t) \int \phi_i(x)\phi_j(x)\,dx - \sum_{i=0}^{p} f_i(t) \int \phi_i'(x)\phi_j(x)\,dx + [f^*(x)\phi_j(x)]_{x_L}^{x_R} = 0,$$

x_L et x_R désignant les bornes gauche et droite de la maille respectivement. Comme ces bornes sont à l'interface entre deux mailles, f est remplacé par f^* dans le terme entre crochets pour signifier que cette quantité doit tenir compte de la valeur de f dans les deux mailles avoisinantes. L'égalité ci-dessus pour tout $j \in \{0, ..., p\}$ peut s'écrire simplement sous la forme matricielle

$$\tfrac{d}{dt}U(t)M - F(t)S + F^*(t) = 0,$$

où on a noté $U = (u_0 \cdots u_p)$, $F = (f_0 \cdots f_p)$, $M = \left(\int \phi_i(x)\phi_j(x)\,dx\right)_{0\leqslant i,j\leqslant p}$, $S = \left(\int \phi_i'(x)\phi_j(x)\,dx\right)_{0\leqslant i,j\leqslant p}$, et $F^* = (f^*(x_R)\phi_j(x_R) - f^*(x_L)\phi_j(x_L))_{0\leqslant j\leqslant p}$. En considérant un maillage uniforme, cette égalité est rendue vraie pour toute maille en translatant la base $(\phi_0, ..., \phi_p)$ de manière adéquate, de sorte que les matrices M et S restent inchangées. Ainsi, l'égalité ci-dessus peut être utilisée pour décrire l'évolution de la solution sur l'ensemble du maillage.

Plusieurs bases de polynômes $(\phi_i)_i$ peuvent être utilisées pour la méthode Galerkin discontinu. Dans cette thèse on utilise une base nodale, c'est-à-dire une base de polynômes de Lagrange associée à des points $x_0, ..., x_p$:

$$\phi_i(x) = \frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)}.$$

Ainsi les coordonnées $u_i(t)$ sont simplement les valeurs $u(x_i, t)$, et de même $f_i(t) = f(u(x_i, t)) = f(u_i(t))$. Aussi, en utilisant les points de quadrature de Gauss-Lobatto, qui ont la particularité d'inclure les bornes de l'intervalle, le dernier terme se simplifie de la manière suivante :

$$[f^*(x)\phi_j(x)]_{x_L}^{x_R} = f^*(x_R)\delta_{0,j} - f^*(x_L)\delta_{p,j},$$

où $\delta_{i,j}$ vaut 1 si $i = j$ et 0 sinon, si bien qu'on a

$$F^* = \begin{pmatrix} f^*(x_R) & 0 & \cdots & 0 & -f^*(x_L) \end{pmatrix}.$$

En pratique, f^* est un flux numérique de type *local Lax-Friedrich* : en notant maintenant u_L et u_R la valeur de la solution à l'interface entre deux maille, la valeur de f^* à cette interface est donnée par

$$f^* = \tfrac{1}{2}(f(u_L) + f(u_R)) - \tfrac{s}{2}(u_L - u_R),$$

où s désigne la vitesse d'onde maximale à l'interface.

La méthode Galerkin discontinu décrite ci-dessus est d'ordre $p+1$ (p étant le degré des polynômes) en espace. Pour tirer profit de cet ordre élevé, la discrétisation de $\frac{d}{dt}U$ peut se faire par une méthode de Runge-Kutta adaptée. Un défaut important de cette méthode d'ordre élevé est qu'elle se comporte mal avec les solutions discontinues, qui sont communes dans les problèmes hyperboliques. En effet, l'approximation polynômiale a tendance à osciller au voisinage des discontinuités, ce qui pose des problèmes de précision mais aussi de stabilité, par exemple quand ces oscillations produisent une pression négative. Une manière de lutter contre ces oscillations est de résoudre une version modifiée de l'équation de départ, en lui ajoutant un terme de viscosité artificielle :

$$\partial_t u + \partial_x f(u) = \partial_x(\mu \partial_x u).$$

L'objectif de ce terme de viscosité artificielle est de rendre la solution de l'équation plus lisse, faisant ainsi disparaître les chocs à l'origine des oscillations. Le coefficient de viscosité $\mu = \mu(x, t)$ permet de pondérer la viscosité de manière à ne lisser la solution que lorsque cela est nécessaire à la prévention des oscillations, pour ne pas dégrader la solution inutilement. En cela, ce coefficient se rapproche d'un détecteur de chocs. Aussi, ce coefficient a vocation à tendre vers 0 lorsque la taille h des mailles tend vers 0, afin que la solution de l'équation modifiée converge vers la solution de l'équation de départ. Plusieurs méthodes ont été proposées pour définir le coefficient de viscosité μ. Ces méthodes s'appuient sur les dérivées de la solution, ou bien sur l'analyse de sa décomposition modale, ou encore sur sa production d'entropie. Une étude comparative de certaines de ces méthodes a été réalisée par Yu et al [34]. Depuis quelques années, le design du coefficient de viscosité par réseaux de neurones a également commencé à être exploré, notamment par Discacciati et al [10]. Dans sa thèse, il génère une base de données à partir de solutions particulières, pour lesquelles il détermine au cas par cas la meilleure viscosité parmi celles mentionnées plus haut. Il entraine ensuite un réseau de neurones à renvoyer cette meilleure viscosité, implémentant ainsi une sorte d'interpolation entre plusieurs modèles connus, pour obtenir un nouveau modèle plus performant. Dans le chapitre 4, on explore une autre manière de générer le coefficient de viscosité par un réseau de neurones, qui ne s'appuie pas sur les modèles pré-existant.

1.4.2 Méthode Lattice-Boltzmann

La méthode Lattice-Boltzmann (LBM) est une autre méthode qui peut être utilisée pour résoudre les équations de la forme (1.2). Cette méthode trouve son inspiration dans le modèle cinétique, et introduit une distribution de particules F (ici notée avec une majuscule pour la distinguer du flux f de l'équation de conservation) . Cependant, contrairement au modèle cinétique où la distribution des particules $F(x, v, t)$ est définie pour des vitesses v dans $\mathbb{R}^d$, dans cette méthode on considère des particules fictives qui ne se déplacent qu'à certaines vitesses particulières $v \in \{v_1, ..., v_q\}$; aussi écrit-on $F_i(x, t) = F(x, v_i, t)$. On suppose ensuite que la distribution F satisfait une équation de Boltzmann

avec un opérateur de collisions de type BGK, qui consiste en une relaxation vers un équilibre local F^{eq} :

$$\forall i \in \{1, ..., q\}, \quad \partial_t F_i + v_i \cdot \nabla F_i = \frac{1}{\varepsilon}(F_i^{\text{eq}} - F_i),$$

avec F_i^{eq} qui dépend des variables macroscopiques.

Dans sa formulation originelle, la méthode Lattice-Boltzmann consiste à identifier les variables du système (1.2) à des moments de la distribution F, et à trouver un équilibre local F^{eq} de sorte que les équations ci-dessus soient consistantes avec le système (1.2) lorsque le paramètre ε tend vers 0. Cette méthode peut aboutir à différentes formulations selon la dimension d de l'espace et le nombre q de vitesses considérées, aussi on désigne par $DdQq$ (par exemple $D2Q9$) le schéma correspondant. Les équations de Navier-Stokes sont les premières à avoir été traitées avec cette approche ; la démarche est détaillée par exemple dans l'introduction de Wagner [29]. D'autres équations ont ensuite fait l'objet de la méthode Lattice-Boltzmann, comme par exemple les équations de la magnéto-hydrodynamique de Braginskii [9], mais chaque nouveau système considéré nécessite une étude dédiée pour trouver une formulation qui convient.

Une variante plus générique, qui peut s'appliquer directement à n'importe quel système sous la forme (1.2), consiste à introduire autant de distributions de particules qu'il y a d'équations dans le système à résoudre [13]. Plutôt qu'identifier les différentes variables à différents moments d'une distribution unique, on identifie alors chaque variable à la densité (moment d'ordre 0) de sa distribution respective. C'est cette méthode, dite de Lattice-Boltzmann *vectoriel* qui est utilisée dans le chapitre 5.

Dans tous les cas, ce sont les équations sur F qui sont résolues numériquement. Pour ce faire, on opère en alternant étape d'advection et étape de relaxation à chaque itération en temps. L'un des avantages de la méthode Lattice-Boltzmann est qu'elle permet de résoudre la partie advection de manière exacte, en discrétisant la solution sur une grille compatible avec les q vitesses des particules et le pas de temps Δt : si pour point x de la grille le point $x + \Delta t\, v_i$ est aussi sur la grille, alors l'advection de F_i sur un pas de temps Δt se traduit par une simple translation de la solution dans la direction correspondante (difficultés aux bords mises à part).

La méthode est intéressante de par sa simplicité de programmation. De plus, il est possible d'utiliser des méthodes d'ordre deux en temps en utilisant une méthodes dite de surrelaxation [7, 5]. Le point délicat est d'assurer une certaine stabilité à ces méthodes. En effet, différents critères de stabilité ont été exhibées pour les méthodes Lattice-Boltzmann vectorielle [3, 2, 14]. Une condition suffisante de stabilité est la condition dite sous-caractéristique, qui contraint la norme des vitesses cinétiques discrètes, notée λ, à être plus grande que les vitesses caractéristiques du système hyperbolique (pour une définition plus précise voir le chapitre 5). Cependant, il est nécessaire également de prendre ce paramètre λ le plus faible possible afin de réduire la diffusion numérique. Des problèmes de stabilité peuvent ainsi survenir en cours de simulation du fait d'oscillations parasites, la contrainte sous-caractéristique n'étant ainsi plus vérifiée localement.

Différentes variantes de ces schémas Lattice-Boltzmann vectoriel ont été proposés afin de réduire la diffusion numérique et contrer les problèmes de stabilité. Certains travaux étudient des choix particuliers de vitesses cinétiques discrètes [11, 6]. D'autres méthodes considèrent plusieurs temps de relaxation [8, 30]. Il s'agit dans ce cas de combiner les termes de relaxation avec des poids (des temps de relaxation) $\tau_{i,j}$ à déterminer :

$$\forall i \in \{1, ..., q\}, \quad \partial_t F_i + v_i \cdot \nabla F_i = \sum_j \tau_{i,j}(F_j^{\text{eq}} - F_j).$$

Ce type de relaxation peut également profiter à la méthode de Lattice-Boltzmann vectorielle, et c'est la piste qui est explorée dans le chapitre 5, avec un réseau de neurones pour déterminer les coefficients $\tau_{i,j}$.

1.5 Résumé des chapitres

Les quatre chapitres qui suivent cette introduction sont au format d'articles scientifiques, et peuvent être lus indépendamment les uns des autres. Chaque chapitre porte sur un projet différent qui mêle réseaux de neurones et méthodes numériques, et est brièvement décrit ci-dessous.

Le chapitre 2 s'intéresse à la fermeture du modèle fluide pour les plasmas. On y considère l'équation de Vlasov à une dimension d'espace et une dimension de vitesse, avec opérateur de collisions BGK

(Bhatnagar-Gross-Krook) et un champ électrique donné par l'équation de Poisson. Pour le modèle fluide, on considère le modèle standard qui porte sur la densité, la vitesse moyenne et la température. Dans ce contexte, la fermeture du modèle fluide se fait par la donnée du flux de chaleur uniquement, en fonction des trois quantités précédemment citées. Les régimes de collisions que l'on regarde sont des régimes intermédiaires avec un nombre de Knudsen compris entre $0,01$ et 1, régimes où la fermeture Navier-Stokes montre ses limites. L'objectif est donc d'entrainer un réseau de neurones à fournir une bonne estimation du flux de chaleur, connaissant le nombre de Knudsen, la densité, la vitesse macroscopique et la température. Pour ce faire, on utilise un réseau de neurones convolutif avec une architecture de *V-Net*, et dont l'enchainement des convolutions aboutit à un modèle non-local. Son entrainement est réalisé sur des données générées par des simulations du modèle cinétique. Les résultats numériques montrent la capacité du réseau à estimer le flux de chaleur, dont il résulte des simulations fluides avec une erreur uniforme en le nombre de Knudsen.

Le chapitre 3 reprend la méthode du premier, mais pour l'appliquer à l'équation de Boltzmann en deux dimensions d'espace et deux dimensions de vitesse, toujours avec l'opérateur de collisions BGK. Le passage à la dimension supérieure rend la fermeture plus riche, puisqu'elle consiste alors en la donnée du flux de chaleur mais aussi du tenseur de stress. Le premier étant un vecteur et le second une matrice symétrique de trace nulle, ce sont donc quatre quantités scalaires qu'il faut estimer pour fermer le modèle fluide. On considère ici encore les régimes de collisions avec un nombre de Knudsen compris entre $0,01$ et 1. On entraine un réseau de neurones par quantité à estimer, à partir de données issues de simulations du modèle cinétique. Les réseaux de neurones montrent une bonne capacité à estimer la fermeture, mais leur intégration au modèle fluide pose d'importants problèmes de stabilités. On propose plusieurs moyens de limiter cette instabilité, mais ces dernières déteriorent la précision du modèle. Surtout, ces méthodes sont sensibles à la résolution du maillage, et doivent donc être adaptées dès lors que cette dernière est modifiée.

Le chapitre 4 porte sur la méthode Galerkin discontinu pour les équations hyperboliques. Cette méthode permet d'obtenir des schémas d'ordre élevé, plus précis que les schémas volumes finis utilisés classiquement pour les lois de conservation. Mais la présence de chocs dans les solutions fait produire à ces schéma d'ordre élevé des oscillations numériques indésirables. Une manière de contrôler ces oscillations est l'ajout d'un terme de viscosité artificiel à l'équation, qui va lisser les chocs et ainsi retirer la cause des oscillations. Afin que cette viscosité artificielle ne détériore pas toute la solution, elle doit s'appliquer localement pour n'intervenir que lorsqu'elle est nécessaire. Pour ce faire on a recours à un coefficient de viscosité, qu'il faut concevoir dans cet objectif. C'est la tâche qui est réalisée par un réseau de neurones dans ce travail. Il prend en entrée la solution à un instant donné, et renvoie le coefficient de viscosité en tout point du maillage. Un élément important de la méthode utilisée réside dans l'algorithme d'apprentissage. Afin de se passer d'un coefficient de viscosité de référence, le critère à optimiser est calculé non pas sur la sortie du réseau de neurones directement, mais à partir de la solution numérique obtenue en faisant usage du coefficient de viscosité donné par le réseau de neurones. Le schéma numérique est ainsi intégré à la fonction qu'il faut différencier pour procéder à la descente de gradient, et est donc implémenté de manière à permettre la différenciation automatique. Les résultats numériques montrent des performances similaires voire supérieures aux coefficients de viscosité pré-existant.

Le chapitre 5 s'intéresse à une autre méthode numérique pour résoudre les équations hyperboliques, celle de Lattice-Boltzmann. Dans cette méthode, chaque variable est décomposée en plusieurs termes se déplaçant chacun à une vitesse donnée, et soumis à un terme de relaxation. Cette décomposition augmente la taille du problème, mais a l'avantage de venir avec un terme d'advection linéaire, là où l'équation d'origine pouvait avoir un flux non linéaire. La méthode numérique standard pondère le terme de relaxation par un coefficient compris entre 1 et 2. Égal à 2, le schéma numérique obtenu est d'ordre 2 mais produit des oscillations non physiques aux abords des discontinuité ; plus il est pris proche de 1, moins ces oscillations sont présentes mais plus le schéma est diffusif. Dans ce travail, on propose de pondérer le terme de relaxation par un coefficient matriciel fourni par un réseau de neurones, qui peut non seulement adapter localement la relaxation de chaque équation, mais aussi les coupler pour obtenir la meilleure solution possible. Le réseau de neurones utilisé est un *multi-layer perceptron* (MLP) qui prend en entrée la solution sur un certain *stencil*, et fournit la matrice de relaxation pour la maille au centre du *stencil* en sortie. N'ayant pas de matrice de relaxation de référence, l'algorithme d'apprentissage utilisé est le même que dans le chapitre précédent. La méthode est testée sur des cas tests en 2 dimensions d'espace. Les résultats numériques montrent des performances très

encourageantes pour l'advection de fonctions indicatrices ainsi que pour l'équation de Burgers appliquée à des conditions initiales gaussiennes, mais l'équation d'Euler pose des problèmes de stabilité qui n'ont pas encore été résolus.

Bibliographie

[1] Andrea Beck, David Flad et Claus-Dieter Munz. "Deep neural networks for data-driven LES closure models". en. In : *Journal of Computational Physics* 398 (déc. 2019), p. 108910. ISSN : 0021-9991. DOI : 10.1016/j.jcp.2019.108910. URL : https://www.sciencedirect.com/science/article/pii/S0021999119306151.

[2] François Bouchut. "Construction of BGK models with a family of kinetic entropies for a given system of conservation laws". In : *Journal of statistical physics* 95 (1999), p. 113-170.

[3] François Bouchut. "Entropy satisfying flux vector splittings and kinetic BGK models". In : *Numerische Mathematik* 94 (2003), p. 623-672.

[4] D. Burnett. "The Distribution of Velocities in a Slightly Non-Uniform Gas". en. In : *Proceedings of the London Mathematical Society* s2-39.1 (1935), p. 385-430. ISSN : 1460-244X. DOI : 10.1112/plms/s2-39.1.385. URL : https://onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-39.1.385.

[5] David Coulette et al. "High-order implicit palindromic discontinuous Galerkin method for kinetic-relaxation approximation". In : *Computers & Fluids* 190 (2019), p. 485-502.

[6] Clémentine Courtès et al. "Vectorial kinetic relaxation model with central velocity. Application to implicit relaxations schemes". In : *Communications in Computational Physics* 27.4 (2020).

[7] Paul J Dellar. "An interpretation and derivation of the lattice Boltzmann method using Strang splitting". In : *Computers & Mathematics with Applications* 65.2 (2013), p. 129-141.

[8] Paul J. Dellar. "Incompressible limits of lattice Boltzmann equations using multiple relaxation times". In : *Journal of Computational Physics* 190.2 (sept. 2003), p. 351-370. ISSN : 0021-9991. DOI : 10.1016/S0021-9991(03)00279-1. URL : https://www.sciencedirect.com/science/article/pii/S0021999103002791.

[9] Paul J. Dellar. "Lattice Boltzmann formulation for Braginskii magnetohydrodynamics". In : *Computers & Fluids*. 10th ICFD Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010) 46.1 (juill. 2011), p. 201-205. ISSN : 0045-7930. DOI : 10.1016/j.compfluid.2010.12.004. URL : https://www.sciencedirect.com/science/article/pii/S0045793010003518.

[10] Niccolò Discacciati. "Controlling oscillations in high-order Discontinuous Galerkin schemes using artificial viscosity tuned by neural networks". en. In : *Journal of Computational Physics* (2020), p. 30. DOI : 10.1016/j.jcp.2020.109304.

[11] François Dubois, Tony Février et Benjamin Graille. "On the stability of a relative velocity lattice Boltzmann scheme for compressible Navier–Stokes equations". In : *Comptes Rendus Mécanique* 343.10-11 (2015), p. 599-610.

[12] Harold Grad. "On the kinetic theory of rarefied gases". en. In : *Communications on Pure and Applied Mathematics* 2.4 (1949), p. 331-407. ISSN : 1097-0312. DOI : 10.1002/cpa.3160020403. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160020403.

[13] B. Graille. "Approximation of mono-dimensional hyperbolic systems : A lattice Boltzmann scheme as a relaxation method". In : *Journal of Computational Physics* 266 (juin 2014), p. 74-88. ISSN : 0021-9991. DOI : 10.1016/j.jcp.2014.02.017. URL : https://www.sciencedirect.com/science/article/pii/S0021999114001375.

[14] Kévin Guillon, Romane Hélie et Philippe Helluy. "Stability analysis of the vectorial Lattice-Boltzmann Method". In : *preprint* (2023).

[15] Gregory W. HAMMETT et Francis W. PERKINS. “Fluid moment models for Landau damping with application to the ion-temperature-gradient instability”. In : *Physical Review Letters* 64.25 (juin 1990). Publisher : American Physical Society, p. 3019-3022. DOI : 10.1103/physrevlett.64.3019. URL : https://link.aps.org/doi/10.1103/PhysRevLett.64.3019.

[16] Jiequn HAN et al. “Uniformly Accurate Machine Learning Based Hydrodynamic Models for Kinetic Equations”. In : *Proceedings of the National Academy of Sciences* 116.44 (oct. 2019), p. 21983-21991. ISSN : 0027-8424, 1091-6490. DOI : 10/gjt9ch. URL : http://arxiv.org/abs/1907.03937.

[17] Kaiming HE et al. “Deep Residual Learning for Image Recognition”. In : *arXiv :1512.03385 [cs]* (déc. 2015). URL : http://arxiv.org/abs/1512.03385.

[18] P. HUNANA et al. “New Closures for More Precise Modeling of Landau Damping in the Fluid Framework”. In : *Physical Review Letters* 121.13 (sept. 2018). Publisher : American Physical Society, p. 135101. DOI : 10.1103/PhysRevLett.121.135101. URL : https://link.aps.org/doi/10.1103/PhysRevLett.121.135101.

[19] Corentin J. LAPEYRE et al. “Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates”. In : *Combustion and Flame* 203 (mai 2019), p. 255-264. ISSN : 0010-2180. DOI : 10.1016/j.combustflame.2019.02.019. URL : https://www.sciencedirect.com/science/article/pii/S0010218019300835.

[20] Yann LECUN, Yoshua BENGIO et T Bell LABORATORIES. “Convolutional Networks for Images, Speech, and Time-Series”. en. In : ().

[21] Zhengyi LI, Bin DONG et Yanli WANG. “Learning Invariance Preserving Moment Closure Model for Boltzmann–BGK Equation”. en. In : *Communications in Mathematics and Statistics* 11.1 (mars 2023), p. 59-101. ISSN : 2194-671X. DOI : 10.1007/s40304-022-00331-5. URL : https://doi.org/10.1007/s40304-022-00331-5.

[22] Julia LING, Andrew KURZAWSKI et Jeremy TEMPLETON. “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance”. en. In : *Journal of Fluid Mechanics* 807 (nov. 2016). Publisher : Cambridge University Press, p. 155-166. ISSN : 0022-1120, 1469-7645. DOI : 10.1017/jfm.2016.615. URL : https://www.cambridge.org/core/journals/journal-of-fluid-mechanics/article/abs/reynolds-averaged-turbulence-modelling-using-deep-neural-networks-with-embedded-invariance/0B280EEE89C74A7BF651C422F8FBD1EB.

[23] Chenhao MA et al. “Machine Learning Surrogate Models for Landau Fluid Closure”. In : *Physics of Plasmas* 27.4 (avr. 2020), p. 042502. ISSN : 1070-664X, 1089-7674. DOI : 10/gjt9ck. URL : http://arxiv.org/abs/1909.11509.

[24] Romit MAULIK et al. “Neural network representability of fully ionized plasma fluid model closures”. In : *Physics of Plasmas* 27.7 (juill. 2020), p. 072106. ISSN : 1070-664X, 1089-7674. DOI : 10/gjt9cj. URL : http://arxiv.org/abs/2002.04106.

[25] Fausto MILLETARI, Nassir NAVAB et Seyed-Ahmad AHMADI. “V-Net : Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation”. In : *arXiv :1606.04797 [cs]* (juin 2016). URL : http://arxiv.org/abs/1606.04797.

[26] Yilan QIN et al. “Data-driven modeling of Landau damping by physics-informed neural networks”. In : *Physical Review Research* 5.3 (août 2023). Publisher : American Physical Society, p. 033079. DOI : 10.1103/PhysRevResearch.5.033079. URL : https://link.aps.org/doi/10.1103/PhysRevResearch.5.033079.

[27] Olaf RONNEBERGER, Philipp FISCHER et Thomas BROX. “U-Net : Convolutional Networks for Biomedical Image Segmentation”. In : *arXiv :1505.04597 [cs]* (mai 2015). URL : http://arxiv.org/abs/1505.04597.

[28] Ricardo VINUESA et Steven L. BRUNTON. “Enhancing Computational Fluid Dynamics with Machine Learning”. In : *Nature Computational Science* 2.6 (juin 2022), p. 358-366. ISSN : 2662-8457. DOI : 10.1038/s43588-022-00264-7. URL : http://arxiv.org/abs/2110.02085.

[29] Alexander J WAGNER. “A practical introduction to the lattice Boltzmann method”. In : *Adt. notes for Statistical Mechanics* 463 (2008), p. 663.

[30] Jia Wang et al. "Lattice Boltzmann simulations of thermal convective flows in two dimensions". In : *Computers & Mathematics with Applications*. Special Issue on Mesoscopic Methods in Engineering and Science (ICMMES-2010, Edmonton, Canada) 65.2 (jan. 2013), p. 262-286. ISSN : 0898-1221. DOI : 10.1016/j.camwa.2012.07.001. URL : https://www.sciencedirect.com/science/article/pii/S0898122112004671.

[31] Jian-Xun Wang, Jin-Long Wu et Heng Xiao. "A Physics Informed Machine Learning Approach for Reconstructing Reynolds Stress Modeling Discrepancies Based on DNS Data". In : *Physical Review Fluids* 2.3 (mars 2017). arXiv : 1606.07987, p. 034603. ISSN : 2469-990X. DOI : 10.1103/physrevfluids.2.034603. URL : http://arxiv.org/abs/1606.07987.

[32] Libo Wang et al. "Deep learning surrogate model for kinetic Landau-fluid closure with collision". In : *AIP Advances* 10.7 (juill. 2020), p. 075108. ISSN : 2158-3226. DOI : 10.1063/5.0010917. URL : https://doi.org/10.1063/5.0010917.

[33] Jin-Long Wu, Heng Xiao et Eric Paterson. "Physics-informed machine learning approach for augmenting turbulence models : A comprehensive framework". In : *Physical Review Fluids* 3.7 (juill. 2018). Publisher : American Physical Society, p. 074602. DOI : 10.1103/PhysRevFluids.3.074602. URL : https://link.aps.org/doi/10.1103/PhysRevFluids.3.074602.

[34] Jian Yu et Jan S. Hesthaven. "A comparative study of shock capturing models for the discontinuous Galerkin method". fr. In : *Journal of Computational Physics* (2017). ISSN : 0021-9991. URL : https://infoscience.epfl.ch/record/231188.

Chapter 2

A neural network closure for the 1D Euler-Poisson system based on kinetic simulations

This chapter was published as an article in *Kinetic & Related Models*:
Léo Bois, Emmanuel Franck, Laurent Navoret, and Vincent Vigon. "A neural network closure for the Euler-Poisson system based on kinetic simulations". en. In: *Kinetic & Related Models* 15.1 (2022) DOI: https://doi.org/10.3934/krm.2021044.

Abstract

This work deals with the modeling of plasmas, which are ionized gases. Thanks to machine leaning, we construct a closure for the one-dimensional Euler-Poisson system valid for a wide range of collision regimes. This closure, based on a fully convolutional neural network called V-net, takes as input the whole spatial density, mean velocity and temperature and predicts as output the whole heat flux. It is learned from data coming from kinetic simulations of the Vlasov-Poisson equations. Data generation and preprocessings are designed to ensure an almost uniform accuracy over the chosen range of Knudsen numbers (which parametrize collision regimes). Finally, several numerical tests are carried out to assess validity and flexibility of the whole pipeline.

2.1 Introduction

Plasmas are gases composed of charged particles, i.e. ions and electrons, which are the subject of many studies because they are a very common state of matter in the universe (e.g. in stars, ionosphere). They are also present in industrial devices, like in fusion reactors. Plasma dynamics is complex since particles have both long range interactions, through the self-consistent electromagnetic fields, and short-range collisions. The most complete model to describe plasmas is given by the Vlasov equation, which is a kinetic equation satisfied by the distribution function of particles in the position-velocity (x, v)-phase space and which is coupled with the electromagnetic equations. To reduce the dimention, it is possible to use *fluid models* which are derived from the kinetic Vlasov equation with a collision operator by taking the velocity moments corresponding to the monomials 1, v, v^2. This raises up three equations linking the four following quantities: the *density* (the zeroth order moment), the *mean velocity* (involving the two first moments), the *temperature* (involving the three first order moments) and the *heat flow* (involving the four first moments). Fluid models are very attractive as they are much computationally cheaper as they involve only spatial quantities. However, they require an additional equation called a *closure* to link the heat flow to the other three moments.

The first two fluid models obtained in the collisional regime are the Euler and the Navier-Stokes system. The closure is obtained from an asymptotic Chapman-Enskog procedure, assuming that the distribution function is close to be at thermodynamical equilibrium due to collisions. These equations are thus valid for small Knudsen number $\varepsilon > 0$, which is the mean free path between two collisions

divided by the characteristic length studied. The Navier-Stokes system provides $O(\varepsilon)$ corrections of the pressure tensor and heat flux. Higher order corrections have been considered but lead to ill-posed systems (Burnett equations). We refer to [4, 35] for specific derivations in the case of plasmas and to [11] for a review.

To extend the validity of fluid models for larger Knudsen numbers, larger moment systems have been proposed for the Boltzmann kinetic equation, with neutral particles: the Grad 13 order moment model using perturbative theory [20], the Levermore 14 order moment model using entropy maximisation closure [29, 30]. They both suffer from intrinsic mathematical imperfections: a lack of hyperbolicity for the former, so-called realizibility issues for the latter [27, 40]. However, several developments propose corrections for the Grad model [44, 46, 6, 47] to extend its hyperbolic region and the entropy methodology has been very successful with compact velocity domain in the context of radiative transfer applications [15, 19, 16].

Instead of increasing the number of involved moments and thus the size of the system, closures have been developed for plasma fluid models in order to incorporate specific kinetic effects. The main example in this direction is the Hammett-Perkins closure [22, 21]. This closure is designed to recover the Landau damping effect, which results from the transfer spatial modes to velocity modes due to phase-mixing and leads to the damping of the electrostatic energy. This closure is devised such that the dispersion relation of the linearized fluid equations leads to the same damping rate as the original kinetic equation. As a consequence, the heat flux has to depend on the temperature through a non-local integral dissipative operator, namely the heat flux equals the Hilbert transform of the temperature. In this context, non-locality is mandatory to recover the right physical damping rate. This methodology have been extended to numerous physical plasma regimes. For magnetized plasmas, various fluid equations have been derived with specific closures [8] and non-local closures have been specially introduced for recovering Landau-damping effects [21, 7, 42]. We refer also to [36, 26, 5, 32, 37, 45] for some recent developments.

Other strategies exist for adding kinetic effects. Let us mention model reduction technic to obtain fluid models like the water-bag method [3, 2, 13], where the kinetic distribution function is approximated with piece-wise constant functions in velocity. Kinetic effects can also be numerically introduced through micro-macro decomposition methods [10, 12, 9].

A more recent approach consists in using supervised machine learning to find a data-driven closure. Artificial neural networks (ANN) have proven very efficient to interpolate data and detect underlying structures. In particular, convolutional neural networks are very efficient to analyse images and thus the outputs of numerical simulations. There are numerous works for designing physics-based models using neural networks (see for instance [1, 48, 18]).

Such tools have already been used in several works regarding moment closure. For instance, in the context of neutral gases, in [23], the authors propose to learn the appropriate higher moments required in the model. For the Reynolds stress transport models for turbulent flows, in [49], the authors propose a neural network-based closure. With regard to plasma models, two works have explored the representability of pre-existing analytic closures by neural networks. In [31], the authors compare three types of non-local neural networks on their ability to represent the non-local Hamett-Perkins closure. In [33], the authors approach three different analytic closures, both local and non-local, and propose a neural network for each. In particular, it is pointed out that the Hamett-Perkins closure and the underlying kinetic phase-mixing are only satisfactorily achieved with a non-local fully connected neural network. Both of these recent works give encouraging results for the use of neural network-based closures in plasma models, that we further explore in this work.

Description of our work. Here, we introduce a data-driven closure, based on a fully-convolutionnal neural network, which is valid for a large spectrum of collisional regimes (i.e. a large range of ε). Data are obtained from numerical simulations of the Vlasov-Poisson system satisfied by the distribution function and the electric potential. This method has already been mentioned in [23] for neutral gases, but the authors preferred to develop another approach. Here we further investigate the design of the closure to obtain accurate predictions.

Collisions between charged particles would normally be modeled with the Fokker-Planck-Landau operator. However, as it is usually done, we replace it by a BGK (Bhatnagar-Gross-Krook) operator, that models the relaxation of the distribution function towards the equilibrium distributions, the Maxwellians. The inverse of the Knudsen number ε is in prefactor of this operator.

Knudsen numbers ε range is chosen equal to $[0.01, 1]$. Indeed, for $\varepsilon < 0.01$, the Navier-Stokes closure already gives good results. An upper bound of the Knudsen interval has to be prescribed. Here we choose $\varepsilon = 1$ to embrace mildly collisional regimes, but larger values could be considered.

The closure takes as input three one-dimensional vectors corresponding to the spatial discretized density, mean velocity and temperature, and one vector corresponding to the parameter ε. The closure returns an estimation of the spatial discretized heat flux.

The core of the closure is a fully convolutional neural network which has a *V-Net* architecture. It was first described and developed by Milletari et al. [34] to treat medical images (3D signals). It is an evolution of the U-Net, developed for biomedical 2D images [39]. It consists in a succession of several convolution kernels, down-samplings and up-samplings that perform multi-scale analysis of the signal. No fully connected layer is used, so the whole network has relatively few parameters. The non-locality of the neural network is actually linked to the number of down-samplings and up-samplings. The architecture of the network can mostly be described by three hyperparameters: the number of levels of the "V", the depth (which rules the number of channels at each level), and the size of the kernels of convolution. The influence of these parameters on the performance is investigated.

As usual in neural networks construction, some processing of the data is required to predict the heat flux in the widest possible range. Therefore multiple steps of processing are included in the closure, on top of the central neural network. One of these is the normalization of the heat flux with the Navier-Stokes approximation, that helps lowering the otherwise large relative error on predictions of low heat fluxes. Another weakens the dependency of the closure on the underlying mesh size, by performing a data slicing before applying the neural network part. We also use a resampling strategy for discretizations with a resolution different from the one used to train the neural network.

The insertion of this closure in a fluid solver raises mathematical issues. Indeed, it is not guaranteed that the neural network closure is dissipative and thus it could lead to instabilities. To prevent such scenario, a smoothing of the prediction is added into the closure.

The paper is organised as follows. In Section 2.2, we introduce the Vlasov-Poisson system and the moment closure issue. In particular, we give the Euler and Navier-Stokes closures. Then the neural network closure strategy is explained in Section 2.3: prediction strategy, architecture of the network and training methodology are presented. Then Section 2.4 details the data generation and Section 2.5 their processing. Finally, in Section 2.6, we carry out several numerical tests to quantify and analyze the accuracy of the closure.

2.2 Fluid closure for Vlasov-Poisson

In this section we present the kinetic description of the plasma dynamics in one space dimension and its fluid approximations.

2.2.1 Kinetic model

A kinetic model is a model interested in the function $f : (x, v, t) \mapsto f(x, v, t)$ that describes the evolution of the distribution of the particles in the (x, v)-phase space, where $x \in [0, L]$ denotes the space variable, $v \in \mathbb{R}$ the velocity variable and $t \in \mathbb{R}$ the time. We will consider periodic boundary conditions in space. From this distribution function can be computed some macroscopic physical quantities. The first three moments give the particle density $\rho(x, t)$, the mean velocity $u(x, t)$ and the total energy $w(x, t)$ defined as:

$$\rho(x,t) = \int_{\mathbb{R}} f(x,v,t)dv, \quad \rho(x,t)u(x,t) = \int_{\mathbb{R}} f(x,v,t)vdv, \tag{2.1}$$

$$w(x,t) = \frac{1}{2}\int_{\mathbb{R}} f(x,v,t)v^2dv. \tag{2.2}$$

We can also define the pressure $p(x, t)$, the temperature $T(x, t)$ and the heat flux $q(x, t)$:

$$p(x,t) = \int_{\mathbb{R}} f(x,v,t)(v-u(x,t))^2dv, \quad \rho(x,t)T(x,t) = p(x,t), \tag{2.3}$$

$$q(x,t) = \int_{\mathbb{R}} \frac{1}{2} f(x,v,t)(v-u(x,t))^3dv. \tag{2.4}$$

Note that we have the following relation: $w = \rho u^2/2 + \rho T/2 = \rho u^2/2 + p/2$.
The evolution of the distribution is described by the Vlasov equation:

$$\partial_t f + v\partial_x f - E\partial_v f = Q(f), \tag{2.5}$$

where $E(x,t)$ is the self-induced electric field, which satisfies the Poisson equation:

$$E = -\partial_x \phi \quad , \quad \partial_{xx}\phi = \rho - \int_{[0,L]} \rho \, dx. \tag{2.6}$$

Here $\phi(x,t)$ denotes the electric potential.

The source term Q is called a collision operator and allows the model to take into account the collisions between particles. Different collision operators can be considered to deal with different situations. In this work we use the BGK operator (Bhatnagar, Gross and Krook), built to conserve the mass, momentum and kinetic energy of the system, and model the relaxation induced by the collisions toward a local equilibrium distribution $M(f)$. This operator simply reads

$$Q(f) = \frac{1}{\varepsilon}(M(f) - f),$$

where $M(f)$ is called the Maxwellian of f and is given by

$$M(f)(x,v,t) = \frac{\rho(x,t)}{\sqrt{2\pi T(x,t)}}\, e^{-\frac{(v-u(x,t))^2}{2T(x,t)}}, \tag{2.7}$$

with ρ, u and T the density, mean velocity and temperature associated to f and defined in (2.1)-(2.3). The parameter $\varepsilon > 0$ is called the Knudsen number and represents the mean free path between two collisions divided by the characteristic length studied. In the limit $\varepsilon \to 0$, the distribution function is expected to be closed to local equilibria. In this regime, the phase-space dynamics could be infered from the spatial dynamics of its macroscopic moments ρ, u, T.

The kinetic equation (2.5) can be easily solved numerically in this one dimensional case. Several numerical methods have been proposed in the literature [43, 14]. In this work, we use a Finite Difference/Finite Volume method similar to the one introduced in [38, 25] and described in appendix 2.A. Although it is possible to extend such computations in dimension 2 or 3, the computational cost becomes very prohibitive. We are therefore led to consider fluid models.

2.2.2 Fluid model

A fluid model of a plasma describes the evolution of its density $\rho(x,t)$, mean velocity $u(x,t)$ and kinetic energy $w(x,t)$, instead of the distribution of its particles $f(x,v,t)$ in a kinetic model. Such models are far less expensive to simulate numerically than the original kinetic model. Indeed, this requires the computation of only three spatial quantities ($3 \times N_x$ unknowns), instead of the full kinetic distribution ($N_x \times N_v$ unknowns), where N_x and N_v stands for the number of discretization points in space and velocity.

Fluid models can be obtained by using the moment method [11]. Formally, it consists in computing the first three moments in velocity of the Vlasov equation, i.e. multiplying Equation (2.5) by 1, v and $v^2/2$ and then integrating in velocity:

$$\text{for } p = 0,\ 1,\ 2, \quad \int_{\mathbb{R}} v^p \left(\partial_t f + v\partial_x f - E\partial_v f\right) dv = \int_{\mathbb{R}} v^p Q(f)\, dv.$$

Since the collision operator conserves mass, momentum and energy, the right-hand sides vanish. Therefore, it results in the following system:

$$\begin{cases} \partial_t \rho + \partial_x(\rho u) = 0, \\ \partial_t(\rho u) + \partial_x(\rho u^2 + p) = -E\rho, \\ \partial_t w + \partial_x(wu + pu + q) = -E\rho u, \end{cases} \tag{2.8}$$

where p is the pressure and q the heat flux defined in (2.3)-(2.4). Note that the pressure p is actually a function of ρ, u, w since we have the following relation:

$$p = 2w - \rho u^2.$$

The electric field E is still given by the Poisson equation (2.6).

We thus get a system of three equations on the four variables ρ, u, w and q. For it to be closed, we need a fourth equation connecting these four unknowns, called a closure. Usually this closure consists in replacing the true heat flux q with a simplified one $\hat{q}$ given as a function of the other quantities , and that can be written

$$\hat{q} = \mathcal{C}(\varepsilon, \rho, u, T).$$

Note that this closure depend on the physical regime we consider through the parameter ε. Let us also mention here that in higher dimension, the pressure stress tensor is not completely determined by ρ, u and w and an additionnal closure relation is necessary.

In fluid regimes, where most of the kinetic effects can be neglected, two closures are classically considered: the Euler or the Navier-Stokes closures. The Euler closure is valid in regimes where the distribution function is closed to be Maxwellian: $f = M(f) + O(\varepsilon)$. Using this ansatz, we obtain the Euler closure:

$$\hat{q} = 0.$$

When we are interested in regime with $O(\varepsilon)$ deviation from the Maxwellians, $f = M(f) + \varepsilon g + O(\varepsilon^2)$, we get the Navier-Stokes closure:

$$\hat{q} = -\frac{3}{2}\varepsilon\, p\, \partial_x T. \tag{2.9}$$

Note that it is a $O(\varepsilon)$ correction of the Euler closure. We refer to [11] for more details.

As mentioned in the introduction, more complex closures have been developed to capture more kinetic effects, in regimes where the disbribution function is more distant from the Maxwellians. See Ref. [22, 21]. Such closures are chosen non local, meaning that the heat flux $q(x)$ at location x does not rely on the values of ρ, u, T and their derivatives at location x only, but on their values on all the domain.

The goal of our work is to provide a method to provide a new closure, where the function $\mathcal{C}$ is implemented using a neural network. The next section describes this approach in more details.

2.3 Closure with a neural network

In this section we introduce the overall principle of our method to build the fluid closure presented in Section 2.2.2 with a neural network. First we describe the basic functioning of the neural network, before introducing the other operations included in the closure, and that surround the neural network. We end this section with the detailed description of our neural network and of its training.

2.3.1 Interpolation of the heat flux with a neural network

In this work we are interested in the ability of machine learning to find a function C that maps the Knudsen number, the density, the mean velocity and the temperature of the plasma to its heat flux. Such a mapping cannot be exact as the heat flux is not a function of these four quantities but a function of the particles' distribution in the phase space. Nonetheless, the ability of neural networks to find non obvious patterns and correlations can be used to provide a good approximation of the heat flux, that can then be used as closure for the fluid model described in Section 2.2.2.

Neural networks enable to build functions relying on many parameters. The obtained closure can be formally written as:

$$\hat{q} = C_{\hat{\theta}}(\varepsilon, \rho, u, T),$$

where $\hat{\theta}$ denotes the set of parameters of the obtained network. Here the closure is chosen to be *non-local*: ρ, u, T as well as q are all spatial discretized quantities. The closure takes as input four vectors or, in the neural network terminology, a 1D signal with four channels corresponding respectively to the Knudsen number (turned into a constant vector), the density, the mean velocity and the temperature at spatial discretization points:

$$X = (\varepsilon, \rho, u, T) \in (\mathbb{R}^{N_x})^4,$$

where N_x is the number of spatial discretization points. The output of the closure is one vector of size N_x

$$Y = C_\theta(\varepsilon, \rho, u, T) \in \mathbb{R}^{N_x},$$

that will correspond to the estimated heat flux $\hat{q} \in \mathbb{R}^{N_x}$.

To find such a function, there are two main steps:

1. First, by setting the *architecture* of the neural network, we define a family of functions C_θ parametrized by $\theta \in \Theta$, where Θ represents the set of all the possible parameters of the neural network.
2. Then, by *training* the neural network, we find a set of parameters $\hat{\theta}$ so as to minimize the error of the neural network predictions on a dataset, for which the true heat flux is known.

The neural network is the core component of the closure C_θ, but it is not its only one. Indeed, for the data to be usable by the neural network and to provide satisfying results, it needs to go through a certain amount of processing. Section 2.3.2 introduces these other components and their articulation in the closure. Sections 2.3.3 and 2.3.4 then describe respectively the architecture and the training of the neural network.

2.3.2 Detailed composition of the closure

In practice, the closure is used at each iteration in time to compute the heat flux q on all the mesh from the input $(\varepsilon, \rho, u, T)$ on all the mesh. Such a use of a neural network based closure into a numerical scheme requires some work to transform the data handled by the solver into data usable by the network, and vice-versa. This section briefly introduces the different steps involved in the closure and their role in relation to the network.

There are three main processing operations designed to integrate the closure into the numerical scheme. The first one deals with the difference in resolution between the numerical scheme and the data used to train the network, by resampling the input of the network to its training resolution, and its output back to the original resolution. The second one deals with the difference in length between the resampled data of the scheme and the inputs handled by the network, by slicing the data into several pieces to give to the network, and then aggregating the outputs to reconstruct the full heat flux. The third one deals with the stability of the resulting numerical scheme, by smoothing the data from the network to prevent any oscillation to propagate into the numerical scheme. These three steps are part of a whole process, that can be broken downs as follows:

$$C_\theta: \; X \overset{\text{(Re)+(P)}}{\longmapsto} X^{(P)} \overset{\text{(Sl)}}{\longmapsto} (X_j^{(P)})_j \overset{(\text{NN}_\theta)}{\longmapsto} (\hat{Y}_j^{(P)})_j \overset{\text{(R)}}{\longmapsto} \hat{Y}^{(P)} \overset{\text{(P')+(Sm)+(Re)}}{\longmapsto} \hat{Y}.$$

This process is illustrated by figure 2.1, and the different steps are described below.

1. **Resampling (Re) and pre-processing (P)** The input X of size N_x is resampled with a Fourier method (that relies on the periodicity of the signal) and pre-processed into a signal $X^{(P)}$ of size N'_x. As explained in section 2.6.3, the resampling changes the resolution of the input to match the one used for training and get better results. The pre-processing that follows is a standardization step, very common when using neural networks, that allows to improve the accuracy. It is detailed in section 2.5.1.
2. **Slicing (Sl)** The inputs are sliced into what we call *windows*, that are overlapping pieces $(X_j^{(P)})_j$ of size N, the size of the inputs the neural network works with.
3. **Neural network (NN$_\theta$)** For each window $X_j^{(P)}$, the network predicts a piece of heat flux $\hat{Y}_j^{(P)}$, also of size N.
4. **Reconstructing (R)** The windows $\hat{Y}_j^{(P)}$ are aggregated in order to reconstruct an entire heat flux $\hat{Y}^{(P)}$. An added benefit of this slicing and reconstructing process is that it allows to prevent some edge effect introduced by the network: by using overlapping windows, the ends of each piece can be ignored when reconstructing the whole signal. As described in section 2.5.3, the amount of overlapping is a parameter that can be tweaked.
5. **Post-processing (P'), smoothing (Sm) and resampling (Re')**
   The post-processing is an operation of normalization designed to improve the accuracy of the network and detailed in section 2.5.2. The smoothing of the signal is here to avoid oscillations coming from the predictions to propagate and amplify, thus making the numerical scheme unstable. The intensity of this smoothing can be tweaked, and is discussed in section 2.6.2. Finally it is followed by a resampling to recover the original resolution, for the output $\hat{Y}$ to be used in the rest of the computations.

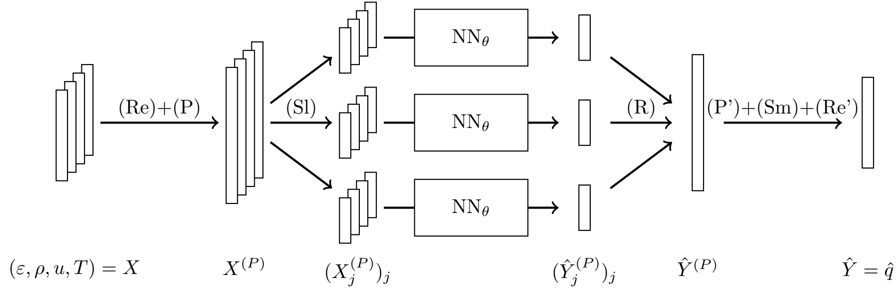


Figure 2.1 – Graph of the composition of the closure. The different operations are (Re)-(Re') resampling, (P) pre-processing, (Sl) slicing, (NN_θ) neural network, (R) reconstruction, (P') post-processing and (Sm) smoothing.

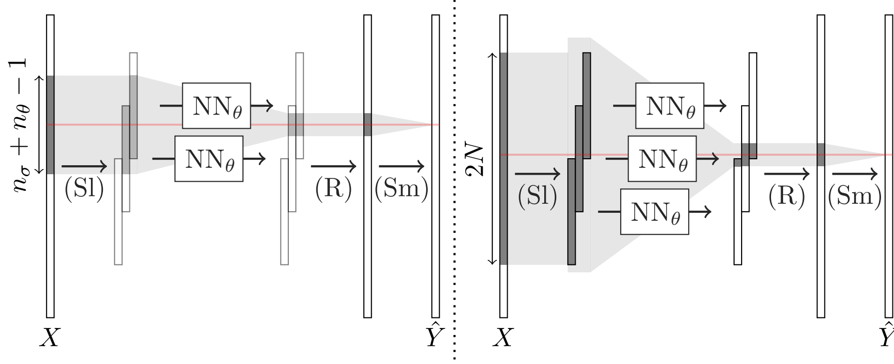


Figure 2.2 – Visual representation of the three operations involved in the non-locality: the (Sm) smoothing, the (R) reconstruction and (Sl) slicing mechanism, and the (NN_θ) neural network. Highlighted in gray is the part of the data used to compute the output value whose position is indicated by the red line. For better readability, only one of the four vectors of the input X, and only the three windows involved, are represented. These examples use the redundancy parameter $r = 2$. Two scenarios are illustrated: on the left, each output value of the neural network depends on relatively few input values, while on the right it depends on relatively many. Also, the position of the output value considered is not the same.

These different operations will be fully exposed in Section 2.5.

The resulting closure is non-local, meaning that it does not perform a pointwise mapping, but instead each point in the output depends on a large region of the input. Two elements of the closure contribute to increase this non-locality: the final smoothing, which consists in averaging n_σ neighbouring values, and most importantly the convolutional neural network, that maps a great number n_θ of neighbouring input values to each output value. However, since the neural network is applied to overlapping windows of the input, and not to the whole input itself, the slicing mechanism can limit the span of the non-locality. Putting aside the margins described in section 2.5.3, it involves two parameters: the size N of the windows, and the redundancy r specifying the number of windows in which each point must be present (more details are given in section 2.5.3).

In order to give a better idea of how these operations interact with regard to the non-locality of the closure, let us provide a quick overview of the possible scenarios when we put aside the margin mechanism (described in 2.5.3) and assume that $n_\sigma \leqslant \frac{N}{r}$. Figure 2.2 illustrates two of them. The left side of the figure illustrates what happens when $n_\theta \leqslant 2\frac{N}{r} - 1$. In this case, n_θ is small enough that the size of non-locality is not affected by the slicing mechanism, and is always given by $n_\sigma + n_\theta - 1$. The right side of the figure illustrates what happens when $n_\theta \geqslant 2N - 1$. In this case, n_θ is big enough that the slicing mechanism systematically constrains the non-locality. Specifically, as shown in figure 2.2, the n_σ reconstructed values can come from either r windows (on the left) or $r + 1$ windows (on the right), depending on the position of the output value considered; as a result, the size of the non-locality is either $(2 - \frac{1}{r})N$ or $2N$, respectively. Intermediate values of n_θ with respect to N and r lead

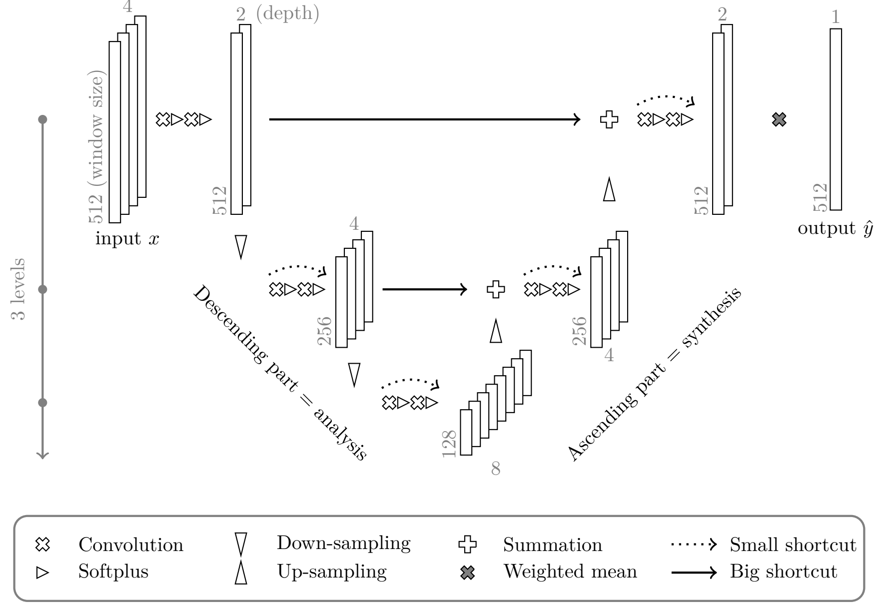


Figure 2.3 – Graph of a 1D V-Net with a window of size $N = 512$, a depth $k = 2$ and $\ell = 3$ levels.

to intermediate results, for which the non-locality is limited by the span of the windows on one side of the interval or the other, or none, or both, depending on the position of the output value considered and the number of windows it involves.

In this work, we end up with $N = 512$, $r = 2$, $n_\sigma = 31$ and an architecture that leads to $n_\theta = 939$. It is an intermediate case, where the size of the non-locality can vary from 768 to 1024. For the sake of completeness, let us mention that taking the margin mechanism (see section 2.5.3) into account decreases the size of the non-locality. In reality, our parameters make us fall into the second scenario described above, and the size of the non-locality is either 716 (for most output values) or 922 (for the rarer output values involving 3 windows instead of 2). Note that this choice of parameters is the result of our attempt to find out what works best for approximating a heat flux to use in the fluid model (as measured in section 2.6); the extent of the non-locality was not the motivation for this choice, but rather a side effect.

Finally, it is worth mentioning that reducing the size of the non-locality does not necessarily come with a decrease of the computational cost. Indeed, if this reduction is achieved by decreasing the size N of the windows, it only increases their number and does not change the overall complexity (as opposed to decreasing the redundancy parameter r). On the other hand, some of the *hyper-parameters* of the neural network can be modified to decrease both its non-locality and the computational cost, but also its ability to approximate heat fluxes. In order to determine the best compromise, we rely on results shown in section 2.6.1. In the next section, we introduce these hyper-parameters, along with the architecture we use for the neural network.

2.3.3 Architecture of the neural network

The closure C_θ depends on the architecture of the network. In our work we use a 1D version of the V-Net [34, 39] implemented with the Tensorflow 2 library through its python interface. In this section, we describe this architecture in details. Since the neural network is intended to be trained on data from several simulations with no specific information about the links between the $X_j^{(P)}$ from a given simulation, we denote a generic input to the neural network by x, and the corresponding output by $\hat{y}$.

The V-net is a fully convolutional neural network meaning that it is based only on a successive sequence of convolutions (no fully connected layers). It consists of two parts. First a descending part that acts as an "encoder", and decomposes the input x into multiple features, performing a multiscale analysis. Then an ascending part that acts as a "decoder", and synthesizes the created features to

predict the output $\hat{y}$. As we will see below, it can be characterized by three hyperparameters: the number of levels ℓ, the depth d of the first convolution and the kernel size p of all convolutions. The composition of the V-net illustrated Figure 2.3 is the following:

Input As described previously, it is a 1D signal x made of 4 channels $(\varepsilon, \rho, u, T)$, and with a given length called the *window size* that we set to N.

Initialization We first perform successively two 1D convolutions [1] with a kernel size of p (we tried $p = 5, 7, 9, 11$), both followed by the softplus activation function $s(t) = \ln(1+e^t)$, and change the number of channels from 4 to a depth d (we tried $d = 4, 5, 6, 8$). All convolutions preserve the size of the signal with the help of a constant padding that extends the signal by continuity. The original V-Net uses the ReLu activation function, but the regularity of the softplus activation function seems to give us better results.

Descending part The V-net is made of ℓ levels (we tried $\ell = 3, 4, 5$), so $\ell - 1$ descents and as many ascents. Let us describe one descent.

- The input is down-sampled by a convolution of kernel size 2, stride 2, and that doubles the depth of its input. The stride allows to halve the length of the input, incidentally blurring the precise locations of the highlighted features.
- Then are applied two successive convolutions of kernel size p and that conserve the depth of their input, both followed by the softplus function.
- The output of this double convolution is added to its input. This "shortcut" is represented by the dotted arrows in Figure 2.3. This way the convolutions produce additive (or residual) modifications. This is the principle of the famous Resnet. It has been shown that it accelerates the training process and limits the problems of vanishing or exploding gradient [24].

The output of this descent is a signal with half the length and double the depth of the input signal.

Ascending part It works as a mirror of the descending part. Each ascent simultaneously increases the length of the signal and decreases its depth by a factor of 2. Technically, we up-sample the data using a transposed-convolution [17] of size 2 followed by two convolutions of kernel size p with softplus activations.

Long shortcuts At each level, the features obtained in the ascents are combined with the features created by the descents, using a summation. This allows to retrieve some fine resolution details, which are lost by the blurring produced by down and up samplings.

Weighted mean The resulting signal of depth d is turned into a one dimensional output $\hat{y}$ by simply taking a weighted mean of all d channels. This operation is implemented with a convolution of kernel size 1. No activation function is added, as we deal with a regression problem.

All the coefficients of the convolutions constitute the set of parameters θ of the neural network.

Table 2.1 summarizes the hyper-parameters we choose in practice as a reference. This choice was mostly motivated by the results given in Section 2.6.1. In order to avoid the oscillations that can be produced by the upsampling operation of the V-Net architecture, we initialize the transposed convolutions with constant kernels.

2.3.4 Training of the neural network

The training consists in finding an optimal set of parameters $\hat{\theta}$ for the neural network. It is defined as the minimizer of a loss function that measures the error of the network:

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \ \mathrm{Loss}(\theta).$$

1. For the sake of completeness, we recall the formula for a one dimensional convolution: applied on an input x of shape (N, d) to get an output y of shape (N, d'), a 1D convolution with an odd kernel size p uses a kernel k of shape (p, d, d') and we have

$$y_{i,k} = \sum_{j=1}^{d} \sum_{di=1}^{p} \tilde{x}_{i+di,j} k_{di,j,k},$$

where $\tilde{x}$ is the input x padded on both ends so that $y_{i,k}$ is well defined for i up to N.

Hyper-parameter	Value
size of the input window (N)	512
number of levels (ℓ)	5
depth (d)	4
size of the kernels (p)	11
activation function	softplus

Table 2.1 – Hyper-parameters of the reference neural network

This error is defined as the average distance between the outputs $\hat{y}$ of the network and the expected outputs y, over a given *training dataset* $\mathcal{D}$. For this distance, we use the mean absolute error, or MAE, which gives us

$$\text{Loss}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x;y)\in\mathcal{D}} \text{MAE}(\text{NN}_\theta(x), y), \quad \text{MAE}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|,$$

where $|\mathcal{D}|$ is the number of entries in the training dataset and N the window size of the network (described in section 2.3.3), which is also the length of the $\hat{y}$ and y vectors. In practice, the minimization of this loss is carried out with a gradient descent algorithm. This whole process falls under the category of supervised learning, as it requires a *labelled* dataset, including both the inputs and the corresponding expected outputs.

In our case, we first generate a raw training dataset with entries $(X; Y) = (\varepsilon, \rho, u, T; q)$ using the kinetic model, as described in section 2.4. Then we turn these into entries (x, y) which can be used to train the network, by applying them the processing outlined previously in section 2.3.2 and described in details in section 2.5.

The data generation process results in a raw training dataset with 10 000 entries $(X; Y)$ of vectors of size $N_x = 1\,024$. The processing of these includes a slicing of the vectors into 8 overlapping windows of size $N = 512$, which results in 80 000 entries $(x; y)$ of vectors of size $N = 512$. 4% of these are isolated as a small test dataset and 10% of what remains are used for validation. The rest is used for the actual training.

This training uses the mini-batch gradient descent algorithm, with the Adam optimizer and mini-batches of size 1 024, to minimize the mean absolute error (MAE). It consists in 5 series of 120 epochs with a decaying learning rate, reset to its initial value of 0.005 between each series.

As a conclusion for this section, let us point out that the making of the raw training dataset requires crucial choices in the initial conditions, the recording times and the range of the Knudsen numbers, that have a direct impact on the range of validity of the closure obtained. The process of data generation and the choices we made regarding these parameters are presented in the next section.

2.4 Data generation

In order to train a neural network to predict the heat flux q knowing the Knudsen number ε, the density ρ, the mean velocity u and the temperature T, we need to generate a *training dataset*. This dataset must meet two criteria. First, it must be a *labelled* dataset, *i.e.* a dataset with both the input $X = (\varepsilon, \rho, u, T)$ and the expected output $Y = q$. The expected output is the "correct" output of the given input, from which we want the network to interpolate, or *generalize*, for new inputs. Then, this dataset must contain as much diversity as possible, for the interpolation to be usable in many different situations.

Consequently, the generation of the training dataset relies on two key ingredients. First, a solver for the kinetic model that can reliably estimate the distribution of particles in the phase space, from which can be derived all the fluid quantities ρ, u, T and q. Second, a mechanism to produce a variety of distributions that can be given to the solver as initial conditions. This whole process of data generation is illustrated in Figure 2.4 and described in more details in this section ; except for the solver for the kinetic model that is not specific to our work and is described in appendix 2.A.

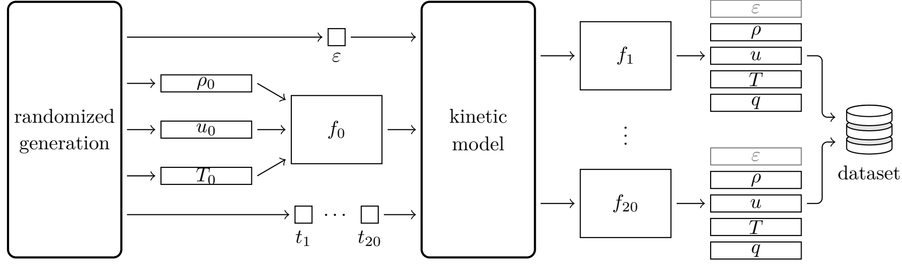


Figure 2.4 – Scheme of the data generation process for one simulation with the kinetic model. With one Knudsen number and one initial condition, we produce 20 entries in the dataset.

2.4.1 Global description

Let us first get a glance at the global process illustrated by Figure 2.4. To begin with, a randomized process allows us to produce a Knudsen number ε, an initial condition f_0, and a *recording time* t. These three elements are given to the kinetic model that computes the evolution of the distribution up to t. The resulting distribution f is then used to compute the density ρ, the mean velocity u, the temperature T and the heat flux q of the system. On top of these four quantities, we add the Knudsen number ε as a constant vector of the same size as the other four, so that it can be used as input by the network. This could also allow us to generalize our method to a problem with a Knudsen number that varies in space. In the end, these five vectors $(\varepsilon, \rho, u, T; q)$ form an *entry* in the dataset.

Actually, in order to capitalize on the simulations that can be computationally expensive —especially in higher dimensions—, we decide to produce several entries with each simulation. For a given Knudsen number and a given initial condition, we generate not one but 20 different recording times $t_1 < \cdots < t_{20}$, and compute the distribution at each one of these times, respectively f_1, ..., f_{20}. From each one of these distributions can be derived ρ, u, T, q, to which we prepend ε to form 20 different entries to be stored in the dataset. Thus, for the cost of one simulation up to t_{20}, we produce 20 different entries corresponding to different intermediate steps in the simulation.

To build a whole dataset, we repeat this process with 100 different values of ε, and with 5 different randomly selected initial conditions for each one of these ε. Thus, we get a dataset with 10 000 entries, based on 500 different initial conditions. We produce two such datasets: a training dataset used to train the neural networks, and a test dataset used to measure and compare the performance of the trained networks on new data.

2.4.2 Random initial conditions

The initial condition f_0 is a Maxwellian as in (2.7), with density ρ, mean velocity u and temperature T randomly generated as partial Fourier series under the form

$$\alpha \times \left(\frac{a_0}{2} + 0.5 \sum_{n=1}^{N} (a_n \cos(nx) + b_n \sin(nx)) \right), \quad x \in [0, 2\pi].$$

We set N to 20 and we randomly generate the a_n and b_n coefficients for $n \geqslant 1$ according to a uniform distribution on $\left[-\frac{1}{n}, \frac{1}{n}\right]$. This choice for the a_n and b_n coefficients set the high frequencies to low amplitudes, resulting in more regular functions. Then, the choice for α and a_0 depends on the function to be generated. For the density, $\frac{a_0}{2} = 1$ and $\alpha = 1$. For the temperature, $\frac{a_0}{2} = 1$ and α is chosen uniformly in $[0.1, 1]$. For the mean velocity, $\frac{a_0}{2}$ is chosen uniformly in $[-1, 1]$, and α is chosen so that the maximum Mach number

$$\max_{x \in [0, 2\pi]} \frac{|u(x)|}{\sqrt{2T(x)}},$$

falls uniformly in $[10^{-4}, 5 \cdot 10^{-1}]$ in logarithmic scale. We also make sure that the density and temperature generated are positive functions. Possible functions generated with this process are given Figure 2.5. For their discrete form, these functions are sampled with $N_x = 1\,024$ points to get vectors, that are turned into a discrete Maxwellian our solver can work with, with $N_v = 141$ points is velocity.

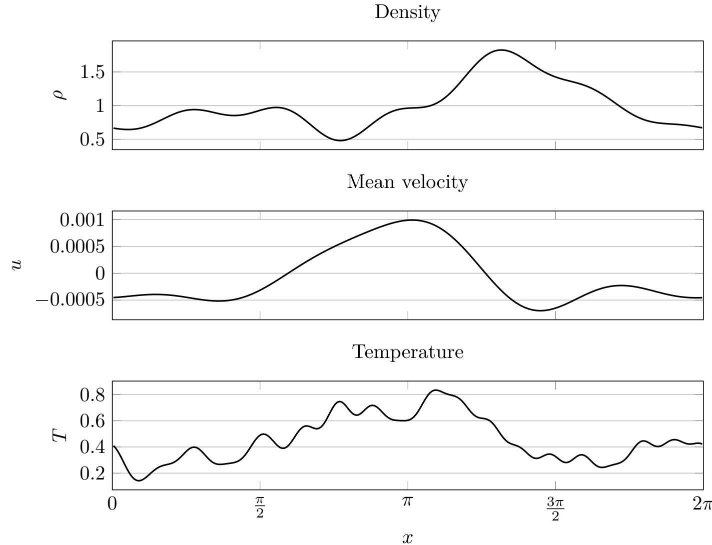


Figure 2.5 – Possible functions ρ, u and T generated with the process described in Section 2.4.2.

2.4.3 Random Knudsen numbers and recording times

In this work we focus on Knudsen numbers in the range $[\varepsilon_{\min}, \varepsilon_{\max}] = [0.01, 1]$. First because below 0.01 the fluid model with the Navier-Stokes estimation is already very accurate, and also because we do not want a range too wide. Regarding the distribution of the values in this interval, we want a decent amount of entries with ε really close to 0.01, but we do not want too few entries for ε close to 1. As a result, a uniform or logarithmic distribution over the interval $[0.01, 1]$ is not satisfying. Instead we opt for a uniform distribution for $\sqrt{\varepsilon}$ in the interval $[\sqrt{\varepsilon_{\min}}, \sqrt{\varepsilon_{\max}}]$. This allows us to have about one fourth of the Knudsen numbers between 0.01 and 0.1, and three fourths between 0.1 and 1. Also we use a deterministic distribution for the train dataset in order to have a nice coverage of the interval, while we use a random distribution for the test dataset in order to test the ability of the trained networks to generalize to new Knudsen numbers.

For the recording times $t_1 < \cdots < t_{20}$, we do not want t_1 to be too close to zero as the initial maxwellian state always has a heat flux of zero and does not carry much physical information, and we do not want t_{20} to be too big so that the simulations do not last too long. For these reasons we choose 20 times uniformly in the interval $[0.1, 2]$, that are then sorted and given successively to the solver for the kinetic model.

2.4.4 Computing of the fluid quantities

Once the Knudsen number, the initial condition and the times are generated, a simulation is computed with the numerical method described in appendix 2.A. It relies on a discretization of the phase space

$$(x_i, v_j) \quad , \quad i \in \{1, ..., N_x\},\ j \in \{1, ..., N_v\},$$

and results in an approximation $(f_{i,j})$ of the real distribution f:

$$f_{i,j} \simeq f(x_i, v_j).$$

From this distribution can be derived the density ρ, mean velocity u, temperature T and heat flux q needed for the dataset, as mentioned in Section 2.2.1. In their discrete form, these quantities are vectors computed as follows: for any $1 \leqslant i \leqslant N_x$,

$$\rho_i = \frac{1}{N_v} \sum_{j=1}^{N_v} f_{i,j}, \quad \rho_i u_i = \frac{1}{N_v} \sum_{j=1}^{N_v} v_j f_{i,j}, \quad \rho_i T_i = \frac{1}{N_v} \sum_{j=1}^{N_v} (v_j - u_i)^2 f_{i,j}$$

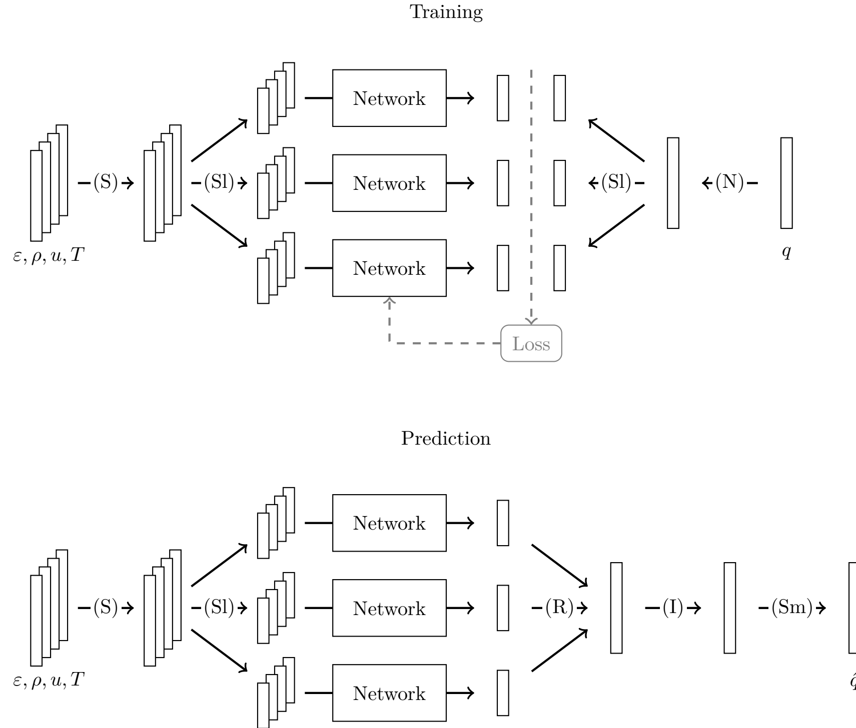


Figure 2.6 – Scheme of the whole data processing for the training and the prediction processes respectivily. The different operations are (S) standardization, (Sl) slicing, (N) normalization, (I) inverse normalization, (R) reconstruction and (Sm) smoothing.

and

$$q_i = \frac{1}{2N_v} \sum_{j=1}^{N_v} (v_j - u_i)^3 f_{i,j}.$$

For our datasets, we use a discretization with $N_x = 1\,024$ points in space, and $N_v = 141$ points in velocity.

2.5 Data processing

In this section we describe how the data is processed on both sides of the neural network, both for the training process and the prediction process. At this point we have generated some data with the kinetic model, and want to process it for a neural network that takes as input a 1D signal with four *channels* —the Knudsen number, the density, the mean velocity and the temperature—, and outputs a 1D signal with one channel —the heat flux. This data processing consists in the transformations described below and summarized in Figure 2.6. In the remainder of this section, we use the word *standardized* to describe a quantity with mean zero and unit variance, and *normalized* a quantity with norm one or that has been applied the transformation described in Section 2.5.2.

2.5.1 Input standardization

Neural networks are known to be easier to train with standardized inputs, where all channels have the same order of magnitude. Consequently, we standardize the inputs channel by channel, by removing the mean and scaling to unit variance the whole training dataset. For example for the density ρ, if (ρ^k) is the family of densities of the whole training dataset, this standardization applied on a given entry ρ^{k_0} can be written

$$\rho^{k_0}_{\text{standard}} = \frac{\rho^{k_0} - \operatorname*{mean}_{k,i} \rho^k_i}{\operatorname*{std}_{k,i} \rho^k_i},$$

where *mean* is the empirical mean and *std* the empirical standard deviation.

Since the neural network is trained with this standardized data, it is designed to work only with inputs that received that specific standardization: after the training, any new input given to the neural network must be applied that very same standardization. In particular, the means and standard deviation computed in the training dataset have to be stored with the neural network, in order to be available later to make predictions for the fluid model.

2.5.2 Output normalization

To prevent the outputs in our datasets from being smaller than the typical error of our neural network, we choose to train the network to predict normalized output. Otherwise, the network would produce pure noise when trying to predict them, which turns to be problematic when used by the fluid model, especially regarding its stability. Another solution would be to use a cost fonction measuring a relative error instead of an absolute one but it turns out to be less efficient. Let us describe how we proceed.

The main issue when training the network with normalized output is the reversibility of the normalization. Since in the end we want to predict a heat flux, and not a normalized heat flux, we need to be able to apply an inverse normalization at the output of the neural network. As a consequence, this transformation cannot use any knowledge on the heat flux to be predicted, that would be available in the labelled training dataset, but would not in a real case scenario. For this reason, we choose to normalize the heat flux with its estimation given by the Navier-Stokes approximation (2.9), and that can be computed from the Knudsen number, the density and the temperature, all available in a real case scenario.

But using an estimation of the heat flux brings another concern: when this estimation is way off, the normalization (or inverse normalization) would distort the information a lot. And this is expected to happen, as the Navier-Stokes approximation tends to greatly overestimate big heat fluxes with Knudsen numbers in the range we consider. To prevent this from happening, we choose to normalize heat fluxes only when the corresponding Navier-Stokes estimation is below a given threshold θ, that we set to 0.1. This threshold is also a way of only addressing the heat fluxes that were problematic in the first place.

Thus, for a given input $(\varepsilon^{k_0}, \rho^{k_0}, u^{k_0}, T^{k_0})$ in the training dataset with expected output q^{k_0}, the normalization we use can be written

$$q^{k_0}_{\text{norm}} = \begin{cases} q^{k_0} \times \dfrac{\theta}{q^{k_0}_{NS}}, & \text{if } 0 < q^{k_0}_{NS} \leqslant \theta, \\ \\ q^{k_0}, & \text{otherwise,} \end{cases}$$

with

$$q^{k_0}_{NS} = \max_{i=1,\dots,N} \left| \frac{3}{2} \varepsilon^{k_0} \rho^{k_0}_i (\partial_x T)^{k_0}_i \right|,$$

where i refers to the index of the vectors and the spatial derivative is approximated with a centered finite difference formula. Note that we choose not to normalize at all heat fluxes such that $q_{NS} = 0$. Once trained to predict these normalized heat fluxes, the neural network must be followed by an inverse normalization when used to make predictions of non normalized heat fluxes. This inverse normalization simply reads

$$q^{k_0} = \begin{cases} q^{k_0}_{\text{norm}} \times \dfrac{q^{k_0}_{NS}}{\theta}, & \text{if } 0 < q^{k_0}_{NS} \leqslant \theta, \\ \\ q^{k_0}_{\text{norm}}, & \text{otherwise.} \end{cases} .$$

2.5.3 Slicing and reconstructing

In order to be usable with meshes of different sizes, the neural network is designed to work with only one portion of the signal at a time as input, and to return the corresponding portion as output. As a consequence, each signal given to the network has first to be sliced into several *windows* of the right size. For the training process, both the input and the expected output have to be applied the

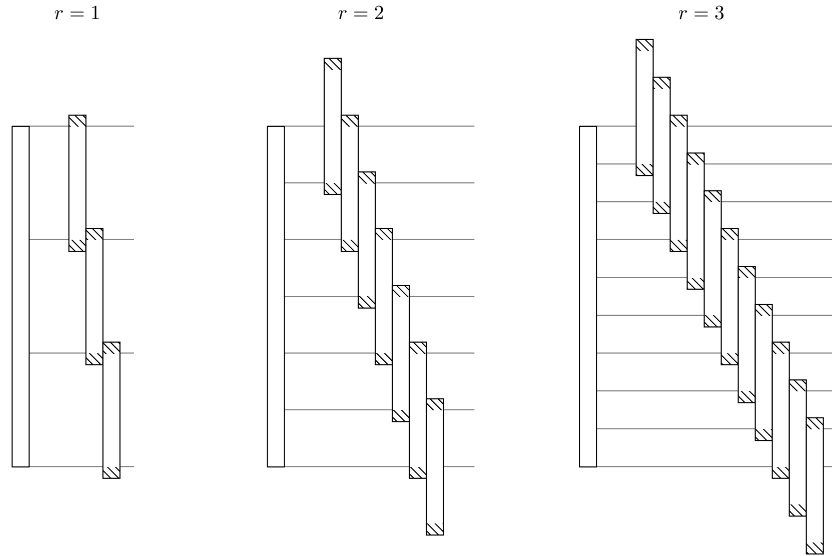


Figure 2.7 – Slicing of a 1D signal into overlapping windows with different redundancy parameters r. The original signal is represented vertically on the left, and the windows are on the right. The hatched parts of each window are the margins that are ignored when reconstructing the signal. The windows that exceed the span of the original signal are completed by periodicity.

same slicing. For the prediction process, the input has to be sliced into windows, and the resulting predictions have then to be aggregated to reconstruct a complete signal.

In one case or the other, we slice the signal into overlapping windows. The overlapping serves two purposes. For the training process, it allows us to artificially increase the amount of data we can give to the network from the training dataset. Though it introduces redundancy, such data augmentation is known to help prevent overfitting and improve the accuracy of the network [41]. For the prediction process, we use this redundancy introduced by the overlapping when reconstructing the output signal, to provide a first smoothing process and to dismiss the defects on the borders of the predictions.

These defects come from the necessary padding in our V-Net architecture, that can cause oscillations of big amplitudes at the ends of the predictions. As a consequence, each window must have a *margin* that will be ignored when reconstructing the signal. We set this margin to 10% of the output on each side, leaving 80% actually useful for the reconstruction, later referred to as the *useful part*. This observation on its own is enough to require some overlapping between the windows to reconstruct the whole signal, but we introduce even more overlapping. We slice the original signal such that each one of its points is found in a fixed number r of windows, margins aside. We call this number the *redundancy parameter*. The bigger it is, the bigger the overlapping, and the more windows are produced. This slicing is illustrated Figure 2.7.

For the reconstruction of an entire signal from the predictions of the network, we get rid of the margins and multiply the useful parts by the kernel

$$\frac{1}{r}\left(\cos\left(\frac{2\pi}{L_u}x - \pi\right) + 1\right), \quad x \in [0, L_u]$$

where r is the redundancy parameter and L_u the length of the useful part in the physical space. We then sum up the resulting windows. This kernel has the property to sum up to one when duplicated every $\frac{L_u}{r}$ (see Figure 2.8). As a consequence, each value in the reconstructed signal is a weighted mean of the r values predicted by the neural network, which results in a somewhat smooth estimation of the heat flux.

2.5.4 Smoothing of the output

At this point we have a process that allows us to estimate the heat flux q with the neural network, and that can be used in a fluid model. But as it is, even with the smoothing provided by the aggregation

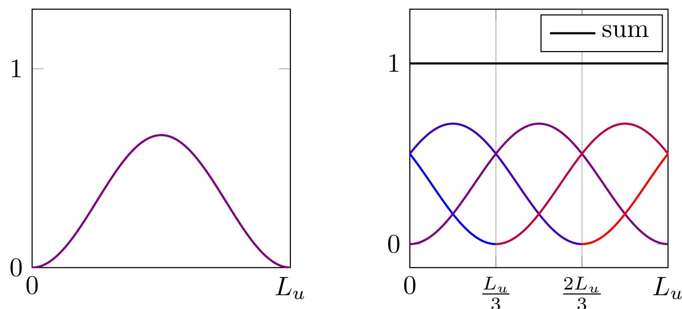


Figure 2.8 – Kernel used for the reconstruction of an entire signal from the windows predicted by the neural network, with the redundancy parameter $r = 3$.

of different predictions as described in the previous section, this estimation can show some small irregularities. In the fluid model, these irregularities on q are amplified by the computation of $\partial_x q$, and can cause the instability of the hyperbolic system. This instability cannot be prevented by refining the time discretization, we have to control the irregularities in the estimated heat flux. To do so, we smooth the reconstruted heat flux thanks to a convolution with a gaussian kernel. This kernel with standard deviation σ reads

$$w : t \in [-3\sigma, 3\sigma] \mapsto \frac{e^{-\frac{1}{2}\frac{t^2}{\sigma^2}}}{\int_{-3\sigma}^{3\sigma} e^{-\frac{1}{2}\frac{t'^2}{\sigma^2}}\, dt'},$$

and the smoothed heat flux is

$$\tilde{q}(x) = \int_{-3\sigma}^{3\sigma} q(x+t)w(t)\, dt.$$

We show in Section 2.6.2 that σ can be chosen relatively large ($\sigma \simeq 0.06$) without impacting the accuracy of the estimation, and has to be for the method to be stable.

Note that the parameter n_σ introduced in section 2.3.2 is the size of the discretized kernel, and is related to σ by

$$\frac{n_\sigma}{N_x} = \frac{6\sigma}{L},$$

where $L = 2\pi$ is the length of the x-domain $[0, 2\pi]$.

2.6 Results

This section is divided in three parts: first we introduce the results regarding the neural network only, then when it is used in the fluid model, and finally how the whole model performs in configurations that differ from the one used for the training. Unless otherwise stated, the hyperparameters of the neural network and the training are those presented in section 2.3.3.

2.6.1 Accuracy of the network

In this section we show numerical results of the neural network, independently of its use with the fluid model. To this end we use the test dataset with entries $(\varepsilon, \rho, u, T; q)$ computed with the kinetic model, or directly compare the predictions with a simulation of the kinetic model. To measure the accuracy of the network on a given entry, we use the relative error in norm L^2

$$\frac{||q - \hat{q}||_2}{||q||_2},$$

where $\hat{q}$ is the prediction of the neural network with inputs $(\varepsilon, \rho, u, T)$. We also observed the relative error in norm L^∞, but it was essentially the same, that is why we focus on the L^2 norm in what follows.

Unless stated otherwise, the results given use a redundancy parameter of 2 and a smoothing of $\sigma \simeq 0.06$. This last choice is motivated by the results of Section 2.6.2.

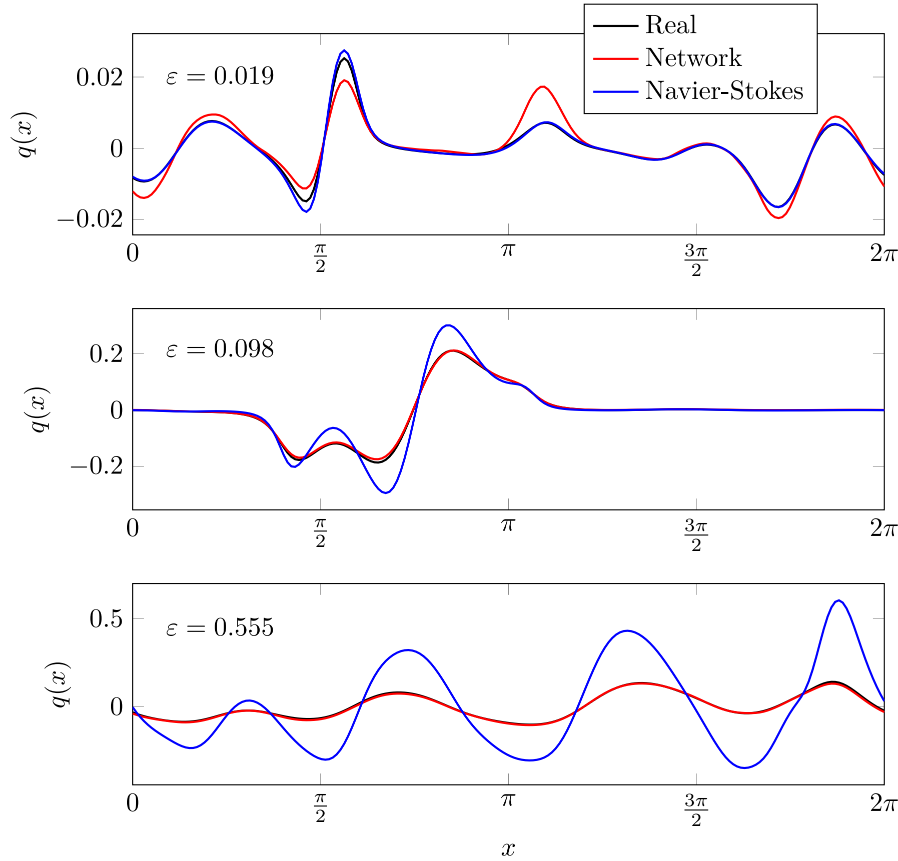


Figure 2.9 – Three examples of predictions. Note the different scales of the y-axis: the amplitude of the heat flux tends to increase with the Knudsen number.

Comparison with the Navier-Stokes estimation

Figure 2.9 shows some examples of predictions, compared to the real heat flux and the estimation of Navier-Stokes (2.9). The first example illustrates how the Navier-Stokes estimation tends to perform better on data with small Knudsen numbers. The second and third example show how on the contrary it very often overestimate the heat flux when the Knudsen number increases.

The relative errors of both the network and the Navier-Stokes estimation over the whole test dataset are summarized in Figure 2.10. On this dataset, the overall error of the neural network is about an order of magnitude below that of the Navier-Stokes estimation.

Figure 2.11 gives a closer look at these errors and highlights two important factors in the performance of the network over the Navier-Stokes estimation: the Knudsen number and the norm of the real flux. The first scatter plot (Fig. 2.11, left) shows that the error of the Navier-Stokes estimation heavily depends on the Knudsen number ε, which is not the case for the neural network predictions, except for small values. This is even better illustrated by Figure 2.12, that uses the same data but where the error is not set in logarithmic scale. The second scatter plot (Fig. 2.11, right) shows that there is a strong correlation between the norm of the real heat flux and the relative error of the network: the smaller the heat flux, the bigger the error of the network. This could be explained by a lack of small heat fluxes in the training dataset or by the normalization we chose for small heat fluxes. In any case, it should not be that big of a problem when used with the fluid model, as smaller heat fluxes have a smaller impact on the result, at least as long as they are quite smooth.

It can also be interesting to look at the evolution of the error during a simulation, to get a better idea of what will happen when used with the fluid model. To do so, we choose 50 random initial conditions and 50 random Knudsen numbers in the same way we did to build the test dataset, and run the kinetic model with these data. At regular times, the heat flux of the kinetic model is compared to the heat flux the network would have predicted and to the Navier-Stokes estimation. Figure 2.13 shows

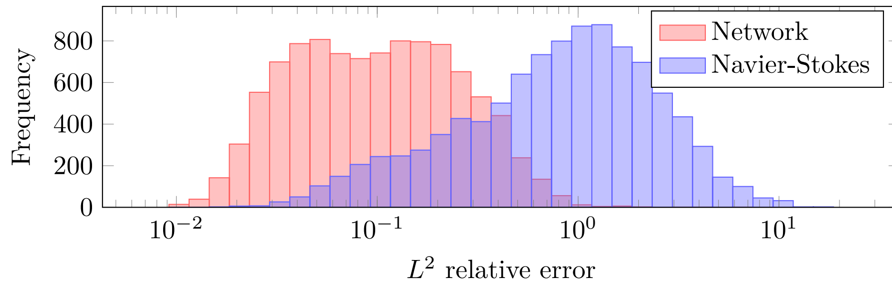


Figure 2.10 – Distribution of the relative errors of the neural network and the Navier-Stokes estimation over the test dataset (10,000 predictions).

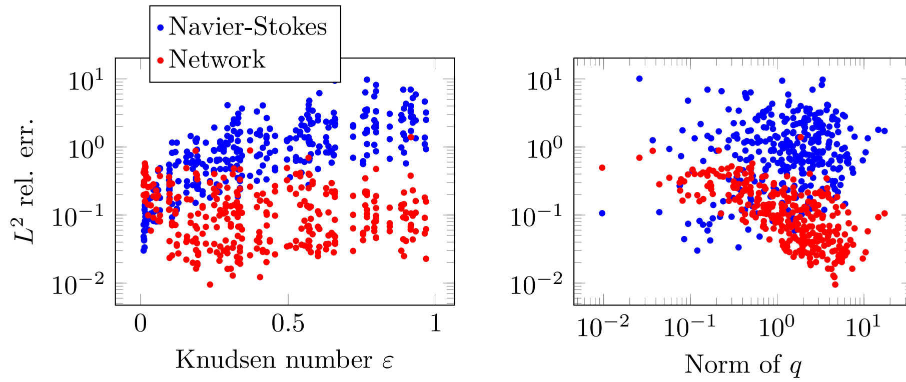


Figure 2.11 – Error of the Navier-Stokes estimations and of the network on the test dataset, depending on the Knudsen number and the L^2 norm of the real heat flux. Each dot corresponds to one entry in the test dataset. For better clarity, only one 31th of the data is shown here.

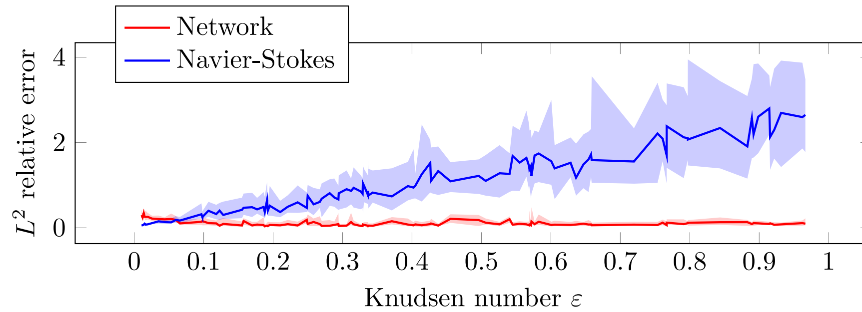


Figure 2.12 – Relative error of the Navier-Stokes estimations and of the network on the test dataset, depending on the Knudsen number. The line represents the median of the error over the 100 entries of the test dataset for each epsilon, and the coloured area the interquartile interval.

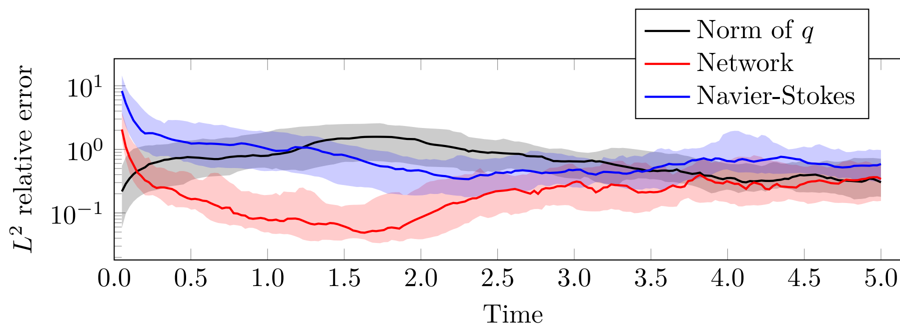


Figure 2.13 – Norm of the real heat flux and relative errors of the predicted heat flux and the Navier-Stokes estimation throughout simulations. Median and interquartile interval over 50 simulations.

the statistical results of this experiment. We observe that the network has a significant advantage over the Navier-Stokes estimation during the first 2.5 time units, before decreasing to a smaller advantage. This result can be explained by the fact that the heat flux is bigger during the first three time units (the dissipation seems to decrease the size of q in time), and by the strong correlation between the norm of the heat flux and the error of the network, also visible in this figure.

Hyper-parameters of the neural network

In this section we show the results that motivated the choice for some hyper-parameters of the V-Net. We focus on the number ℓ of levels, the depth d, and the size p of the kernels of the convolutions. Figure 2.14 compares the median relative error of V-Nets with different sets of hyper-parameters. For these results the window size was set to 256, but we later decided to set it to 512 as it gave slightly better results.

We observe that all hyper-parameters do not affect the performance of the V-Net in the same way. For instance, decreasing the size of the kernels significantly decreases the accuracy of the trained network, while not decreasing the number of parameters as much. On the other hand, decreasing the depth allows to decrease the number of parameters with a very small effect on the accuracy. In the remainder of this paper we work with the "l5,d4,p11" architecture, as it seems to be a good compromise between the number of parameters and the accuracy of the network.

2.6.2 Fluid model with the neural network

In this section we look at how the neural network performs when used in the fluid model. We denote by "Fluid+Network" this method. The fluid model is solved using an explicit finite volume method as presented in appendix 2.B.1. We compare it to three others:

Kinetic : the kinetic model. It is the most accurate model and serves as a reference for the real target. The numerical method is described in appendix 2.A.

Fluid+Kinetic : the fluid model with the heat flux from the kinetic model. It is the result we would get with a perfect neural network that makes no error, and helps to distinguish the error of the fluid model from the error of the neural network on the heat flux. The numerical method is identical to the one of the Fluid+Network method (see appendix 2.B.1). Theoretically this model should give the same result as the kinetic model, but since the numerical schemes and viscosities are different it is not always true in practice.

Navier-Stokes : the fluid model with the Navier-Stokes estimation. It does not use the same numerical method as our model, since the formula for the heat flux requires an implicit scheme to avoid too stringent stability condition (see appendix 2.B.2).

Our criteria to compare the fluid models is the L^2 relative error on the logarithm of the electric energy $\mathcal{E}$

$$\mathcal{E}(t) = \int_{[0,2\pi]} E(x,t)^2 \, dx,$$

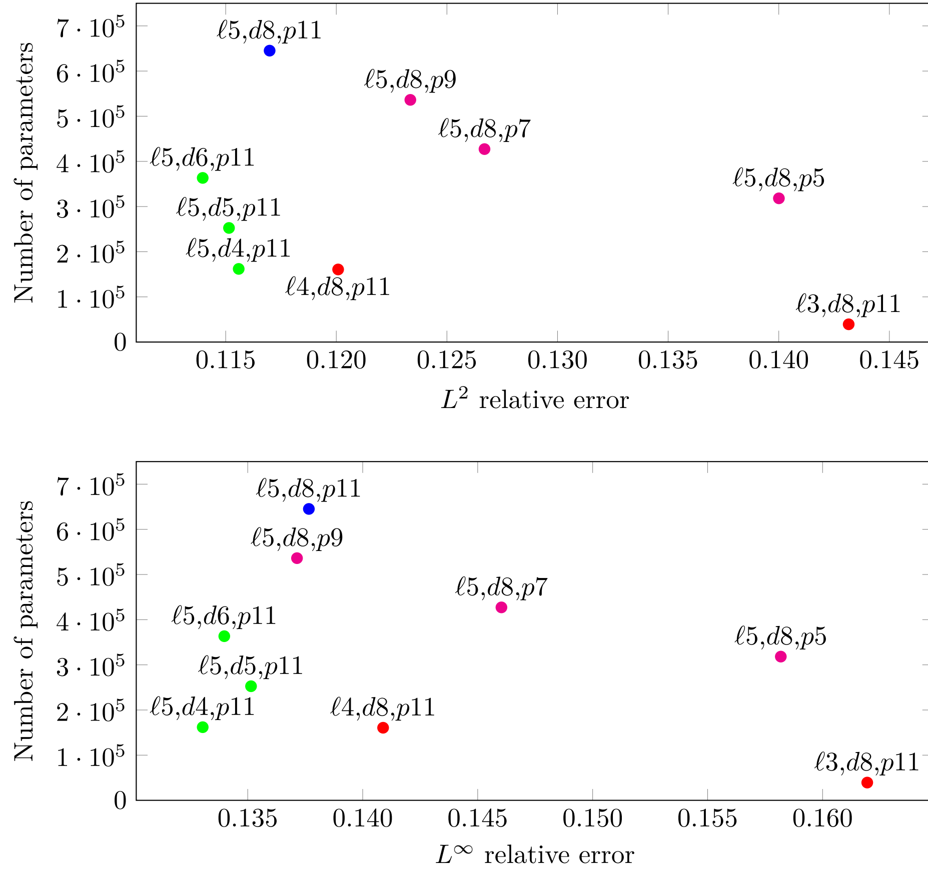


Figure 2.14 – Median relative error and number of learnable parameters for V-Nets with different hyper-parameters : (ℓ) number of levels, (d) depth, and (p) size of the kernels. Points sharing the same color represent networks that differ by only one hyper-parameter.

compared to the kinetic model. For all the following tests we use a discretization of $N_x = 512$ points on $[0, 2\pi]$ in physical space, and $N_v = 101$ points on $[-7, 7]$ in velocity space (for the kinetic model). In particular, the data is resampled on 1 024 points at each iteration for the neural network as explained in Section 2.3.2. For the neural network to be called on inputs similar to these used for its training, the initial conditions are generated with the same process than for the data generation, described in 2.4.2. The simulations are computed up to $t = 8$.

Examples

In this section we give a few examples of simulations to give the reader an idea on how the quantities involved in the simulations look. Figure 2.15 shows the evolution of the fluid quantities up to $t = 1$ in a kinetic simulation with a randomly generated initial condition and with $\varepsilon = 0.1$. Figure 2.16 compares the density and the heat flux of the four models at time $t = 1$ of this simulation.

Figure 2.17 shows examples of electric energies obtained with the four models described above. The first one uses a very small Knudsen number, and with no surprise the Navier-Stokes model performs better than the fluid model with the network. On the second example on the other hand, the Knudsen number is bigger and we observe that the oscillations are slightly shifted: Navier-Stokes shows some dispersion that the fluid model with the network does not. This dispersion often increases as the Knudsen number gets bigger as it shows on the third example. Finally the fourth one illustrates a case where the Navier-Stokes model really struggles while the fluid model with the network does pretty good.

Global performances

The L^2 relative errors of the three fluid models over 200 simulations with different initial conditions and different Knudsen numbers are summarized in Figure 2.18. We observe that our closure with the neural network does not reach relative errors of 0.01 or below, contrary to the closure with the real heat flux or even with the Navier-Stokes estimation. But it successfully maintains the relative error below 0.2, which is close to what the closure with the real heat flux achieves, and much better than the Navier-Stokes estimation that often exceeds 0.2. As already noted, the Fluid+Kinetic relative error should be zero but is not, due to numerical diffusion errors.

Influence of the Knudsen number

In Figure 2.19 we look at the influence of the Knudsen number on the result of the fluid models. All three seem to decrease in accuracy as the Knudsen number increases, but it is much more significant for the Navier-Stokes model. For Knudsen numbers above 0.1, not only the fluid model with the network seems to work systematically better than the Navier-Stokes model, but it is also quite close to the fluid model with the kinetic heat flux. This would seem to indicate that the network appropriately plays its role and that its error on the heat flux does not impact the whole model too much.

Smoothing of the prediction and stability

In this section we show the impact of the smoothing on the accuracy of the predictions, as well as its impact on the stability of the fluid model using these predictions. Figure 2.20 gives a visual example of how the smoothing modifies a prediction of the neural network.

To plot Figure 2.21, the network was used to predict the heat flux of each entry in the test dataset, and different quantities of smoothing were applied before computing the relative error. We observe that the accuracy of the resulting heat flux does not deteriorate for standard deviations lower than $\sigma \simeq 0.04$, and then slightly decreases as σ increases.

This smoothing was introduced to cope with the instability of the fluid model relying on the predictions of the network. To measure how this stability is improved by smoothing the output of the network, we choose 30 random initial conditions as we did to build the test dataset and 30 Knudsen numbers uniformly distributed between 0.01 and 1. Then we choose a set of smoothing parameters σ, and for each one of these we run the 30 simulations with the fluid model, using the network with this quantity of smoothing. Finally, for each σ we look at the proportion of simulations that reached $t = 3$, from which we assume the model will remain stable. The results are shown Figure 2.22. It appears that all 30 simulations reach $t = 3$ for σ above approximately 0.05. We use $\sigma \simeq 0.06$ for our tests with

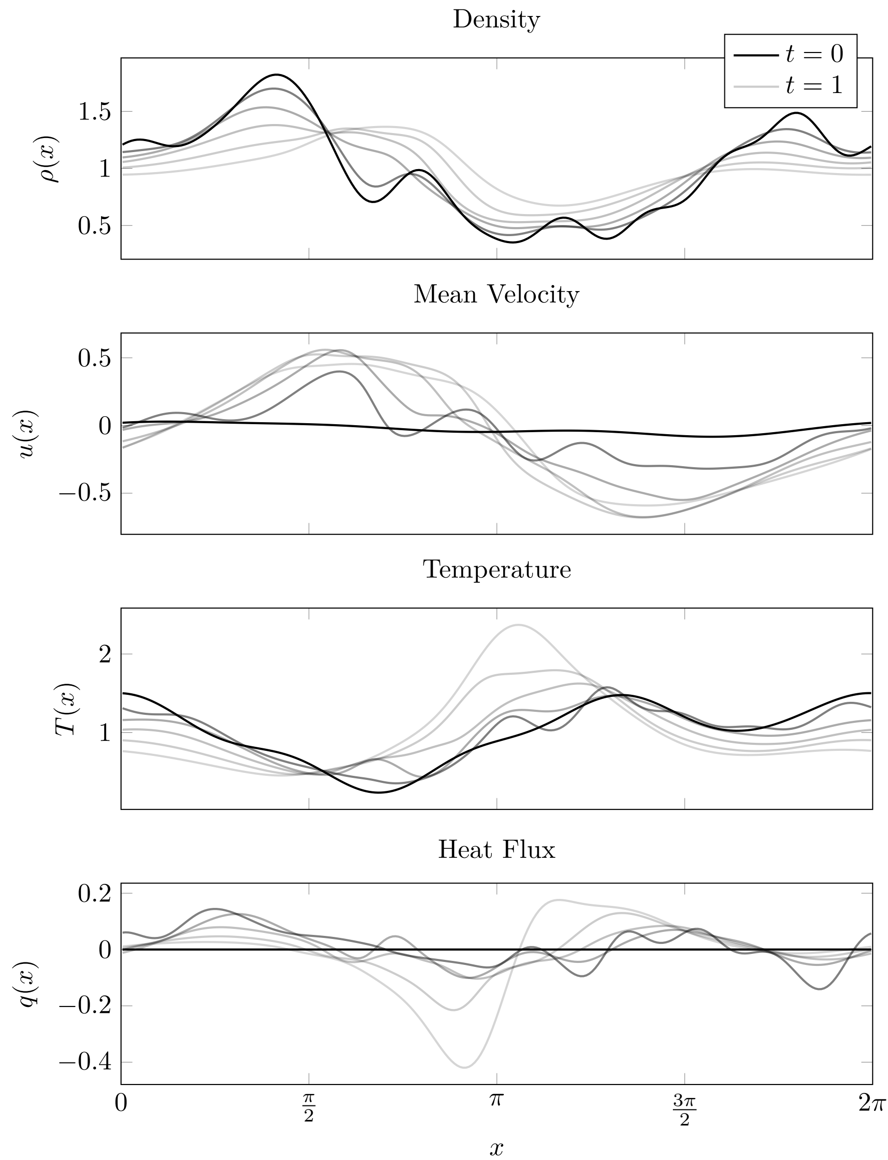


Figure 2.15 – Example of the evolution of the density, mean velocity, temperature and heat flux during the first time unit with a randomly generated initial condition and $\varepsilon = 0.1$, obtained with the kinetic model.

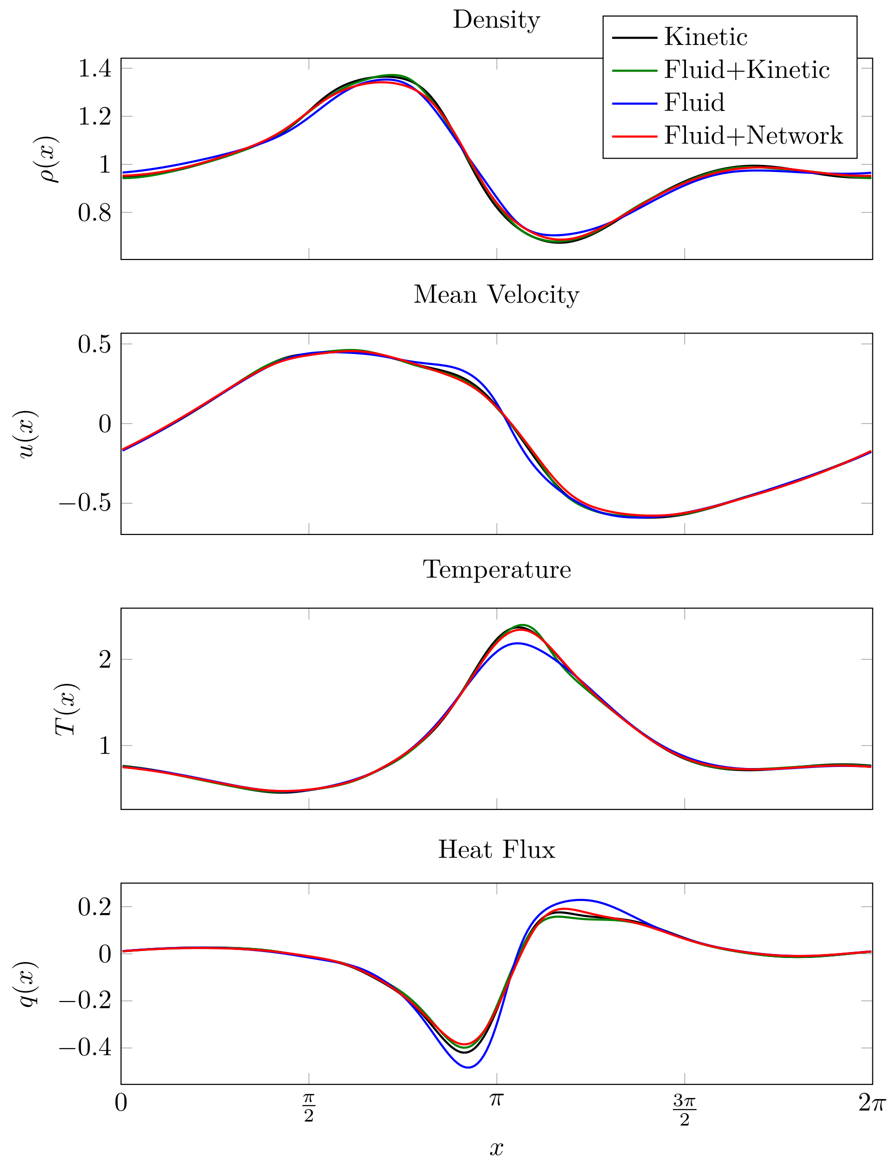


Figure 2.16 – Comparison of the four models on the density and heat flux at $t = 1$ with a randomly generated initial condition and $\varepsilon = 0.1$.

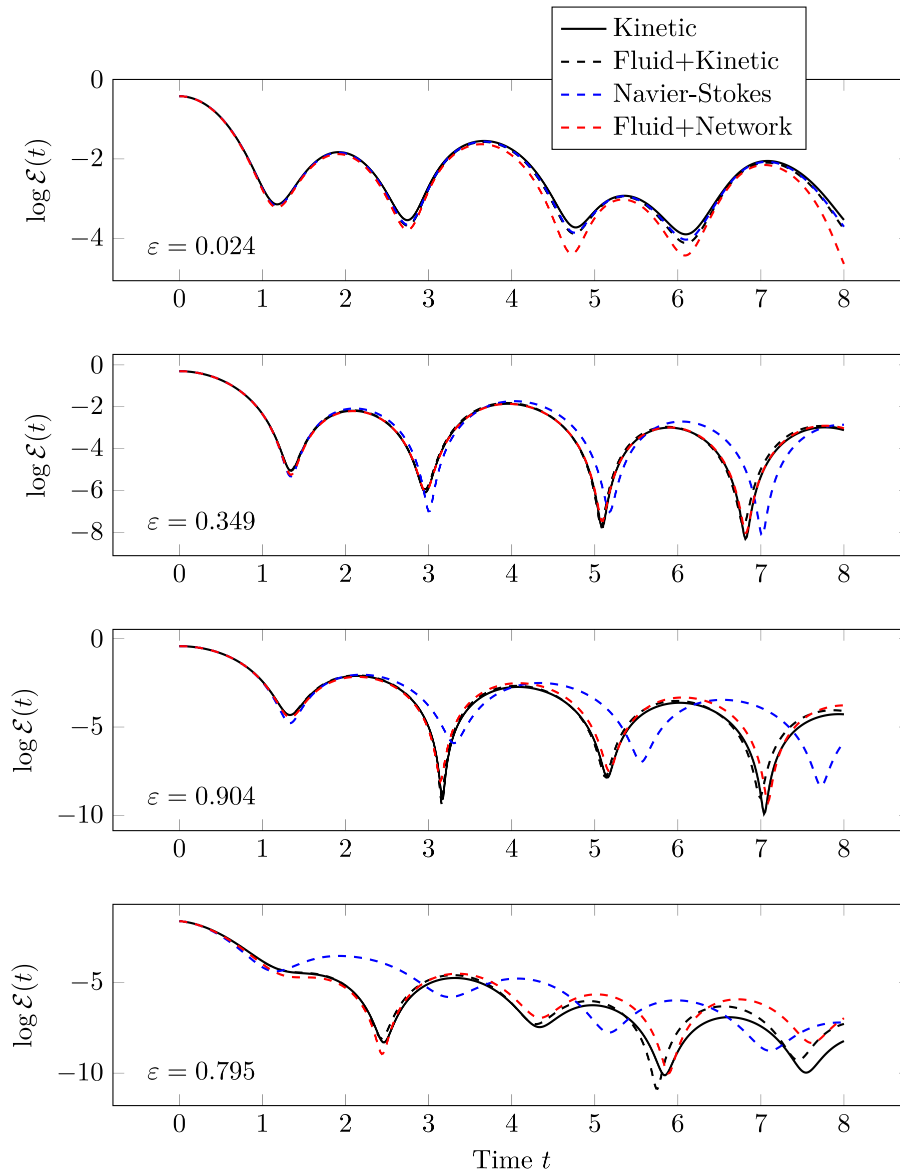


Figure 2.17 – Examples of the evolution of the electric energy with different initial conditions and different Knudsen numbers.

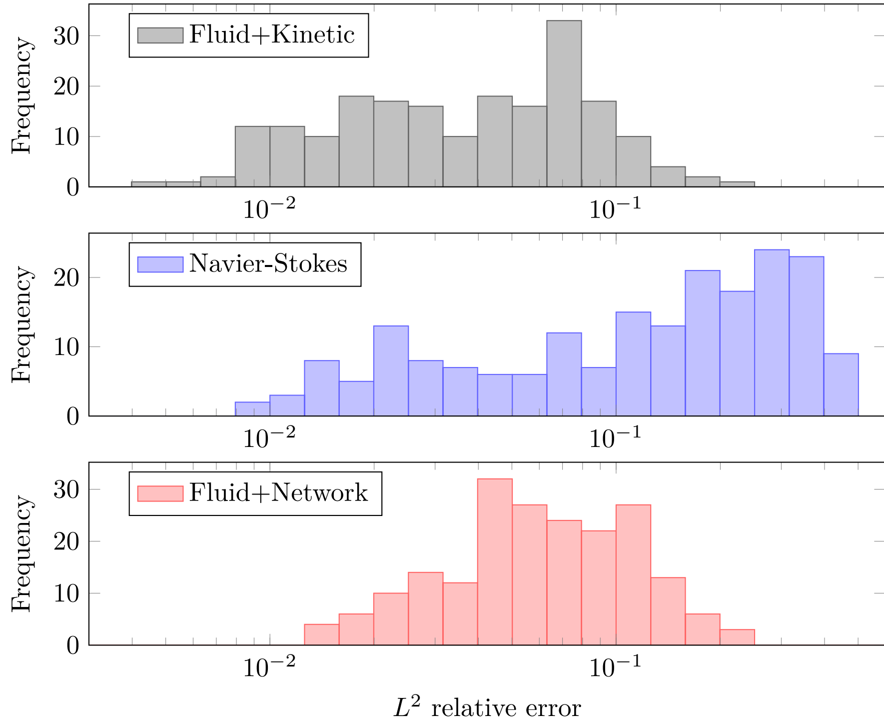


Figure 2.18 – Distributions over 200 simulations of the relative errors of the three fluid models compared to the kinetic model, measured on the logarithm of the electric energy from $t = 0$ to $t = 8$. For better readability, the x-axis in in logarithmic scale.

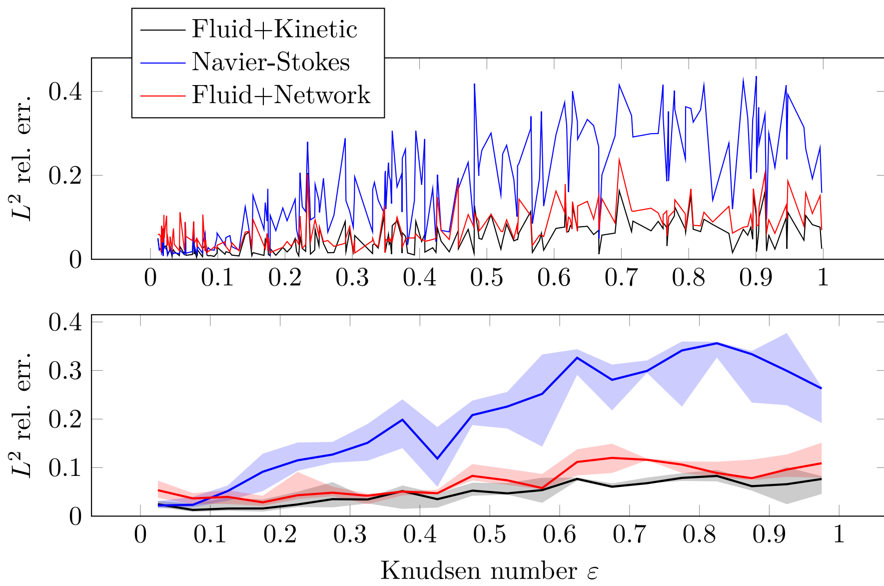


Figure 2.19 – L^2 relative errors of the fluid models over the kinetic model depending on the Knudsen number. The first plot shows the raw data and the second shows the median and interquartile interval for 20 uniform classes of Knudsen numbers between 0.01 and 1.

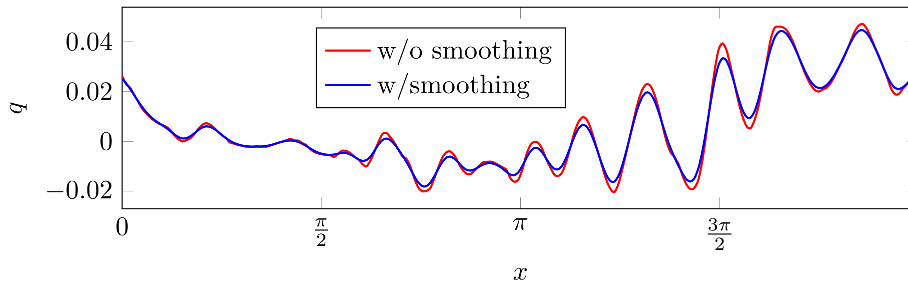


Figure 2.20 – Example of a heat flux predicted by the neural network, with and without smoothing. The smoothing here uses $\sigma \simeq 0.05$.

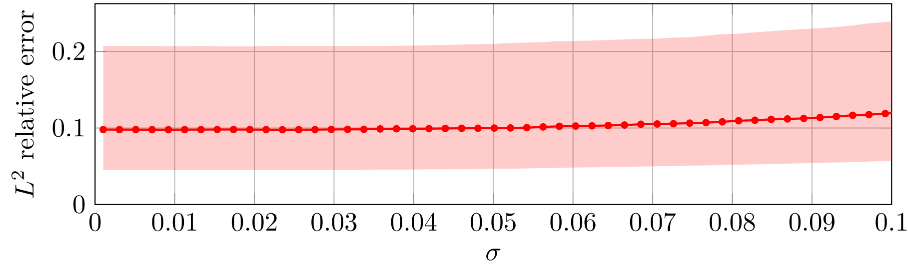


Figure 2.21 – Relative errors of the predictions over the test dataset depending on the quantity σ of smoothing.

the fluid model shown in the next section, and it appears to be enough since no simulation failed so far with this quantity of smoothing.

2.6.3 Using the network with different configurations

In this section we are interested in the flexibility of our approach to different configurations. First we take a look at different discretizations of space to measure the ability of the network to generalize to other resolutions than the one it has been trained with. Then we introduce discontinuities in the initial conditions to see how the fluid model with the network reacts in terms of stability and accuracy.

Different resolutions and convergence discussion

It is interesting to know how the fluid+network model behaves when dealing with different resolution, wether it is a lower resolution in order to gain computation time, or a higher resolution to gain accuracy. In this section we show that there is a way to make the fluid+network model work with different resolutions (called option 2 below), and discuss its potential for convergence.

When using the network closure, there are at least two options to deal with a change of resolution:

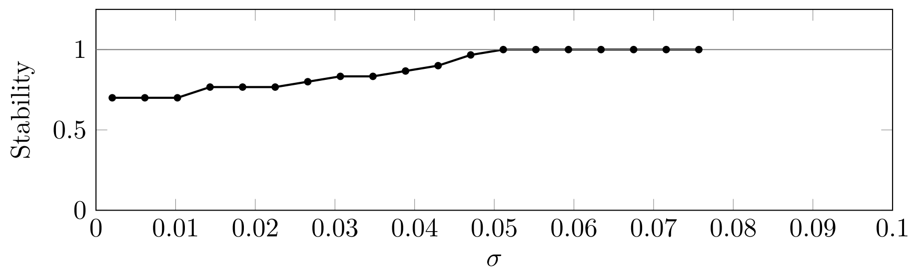


Figure 2.22 – Proportion of simulations of the fluid model reaching $t = 3$ out of 30 simulations, depending on the quantity σ of smoothing.

Option 1 Taking advantage of the slicing mechanism described in 2.5.3, made to adapt to inputs of different sizes.

Option 2 Resampling the input of the network to match the training resolution, and resampling its output to recover the original resolution.

The detailed composition of the closure given in section 2.3.2 includes the resampling of option 2. As suggested by the presence of an option 1, it is not strictly needed, but as we show in this section, it is highly recommended.

Let us consider an example. Say we want to use our network with a discretization of 2 048 points on $[0, 2\pi]$ instead of the 1 024 points used to generate the dataset the network was trained with. Then:

Option 1 consists in slicing the inputs of size 2 048 in about twice as many windows of size 512, giving them to the network, and reconstructing a heat flux of 2 048 points by aggregating its outputs;

Option 2 consists in down-sampling the inputs to 1 024 points, slicing it into windows of size 512, giving them to the network, reconstructing a heat flux on 1 024 points by aggregating the outputs, and finally up-sampling it back to 2 048 points.

In this example, option 1 comes with more windows which requires more uses of the neural network, while option 2 comes with additionnal steps of down and up-sampling. However, the predictions on different windows being made in parallel and the down and up-sampling being relatively cheap, we observe no significant difference in computational time between the two options in our simulations. It is still the case when decreasing the number of points instead of increasing it.

In order to determine the better option, we consider the following experiment. We randomly generate 10 initial conditions, and arbitrarily set the Knudsen number to $\varepsilon = 0.1$. To use as the target, we run a kinetic simulation with each one of the initial conditions with a high resolution ($N_x = 2\,048$, $N_v = 141$), up to $t = 1$. Then, for both options, simulations of the fluid+network model are run with the same 10 initial conditions but sampled with different resolutions, ranging from 512 points to 2 048 points. Finally, L^2 relative errors on the density, mean velocity, temperature, and heat flux at $t = 1$ are computed for each simulation, as well as the L^2 relative error on the electric energy from $t = 0$ to $t = 1$. These errors are averaged over the 10 initial conditions, in order to get a mean error for each resolution and each option. The results are shown figure 2.23.

We observe that option 2 almost always performs better than option 1. Furthermore, while both options allow to increase the accuracy by increasing the resolution up to the training resolution (1 024 points), only option 2 allows to consistently increase the accuracy with higher resolutions, and only option 2 seems to be converging.

Let us propose an explanation for these results. Considering option 1, the change in resolution modifies the way the data is interpreted by the neural network. For instance an initial condition sampled on 2 048 points instead of 1 024 has its signal frequencies as if divided by two in the eyes of the neural network. Thus, past $N_x = 1\,024$, as the resolution increases, the distance between the model's data and the training data also increases, which results in heat fluxes less and less accurate. At some point, this prevents the improvement of the model and even leads to its degradation. In order to use the neural network as is with different resolutions, a solution could be to give the resolution as input to the network and to use different resolutions for the training.

Considering option 2, the resampling ensures that the model's data stays close to the training data. In fact, one could argue that as the resolution increases, the fluid model gets more and more accurate, and so the simulations gets closer and closer to the kinetic simulations, ie to the training data. As a result, the error on the predicted heat fluxes decreases, allowing the fluid model to perform better and better. However, the error on the predicted heat fluxes should still be bounded by the intrinsic ability of the network to approximate the heat flux, so we do not expect the fluid+network model (with a given network) to converge toward the fluid+kinetic model, and even less toward the kinetic model or the truth. Figure 2.24 gives an example of such convergence.

Also, these results highlight the importance for the data given to the network to be close to the training data. It is a natural limit to the data-driven approach, but one that should be kept in mind. For instance, using initial conditions different from the ones used to generate the training dataset should be expected to yield to less accurate results.

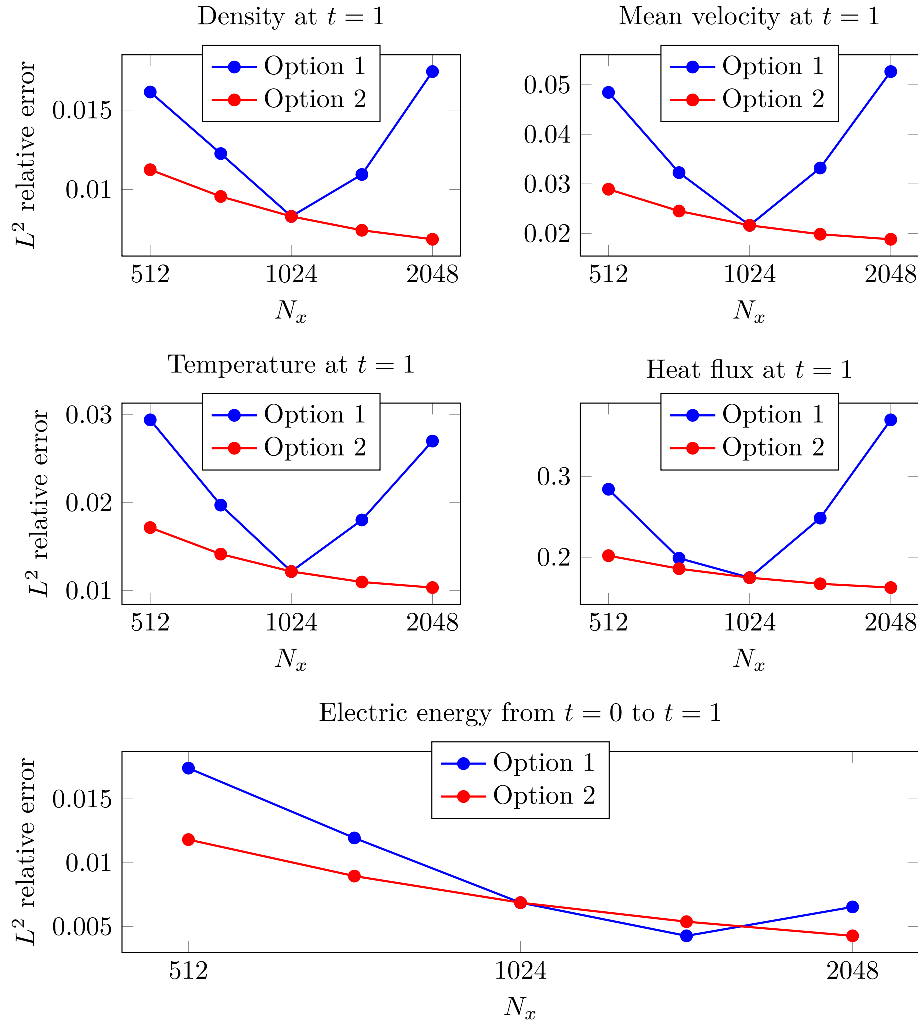


Figure 2.23 – Relative errors of the fluid+network model on five different quantities depending on the resolution used. Each point is an average of 10 simulations with always the same 10 different initial conditions and $\varepsilon = 0.1$. The network used was trained with data generated with $N_x = 1\,024$. To adapt to different resolutions, option 1 relies on the slicing mechanism to adapt to the different input sizes, while option 2 uses the resampling of the input to $N_x = 1\,024$ and the resampling of the output back to the original resolution. For all resolutions, the errors are computed relative to kinetic simulations with a high resolution ($N_x = 2\,048$, $N_v = 141$).

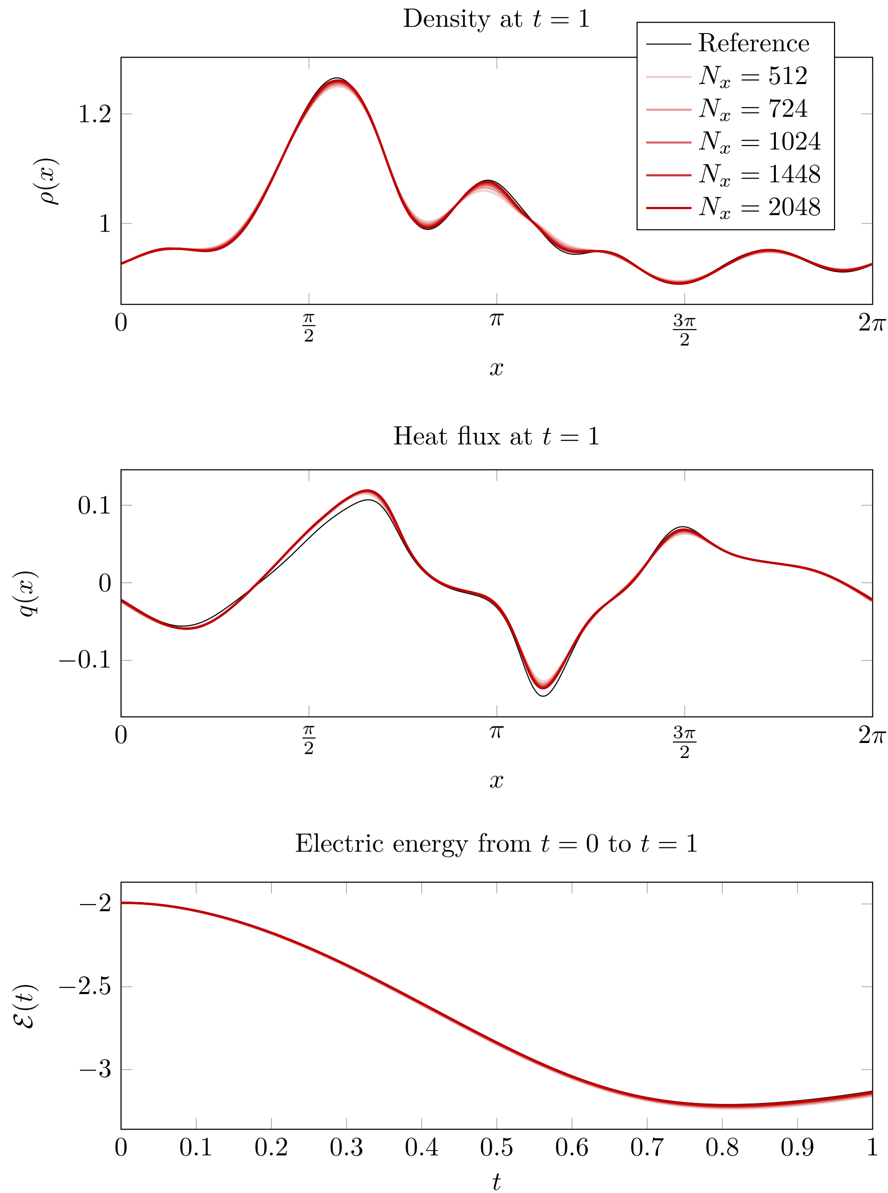


Figure 2.24 – Example of convergence of the fluid+network model with option 2, towards what seems to be a slightly different solution than the kinetic model.

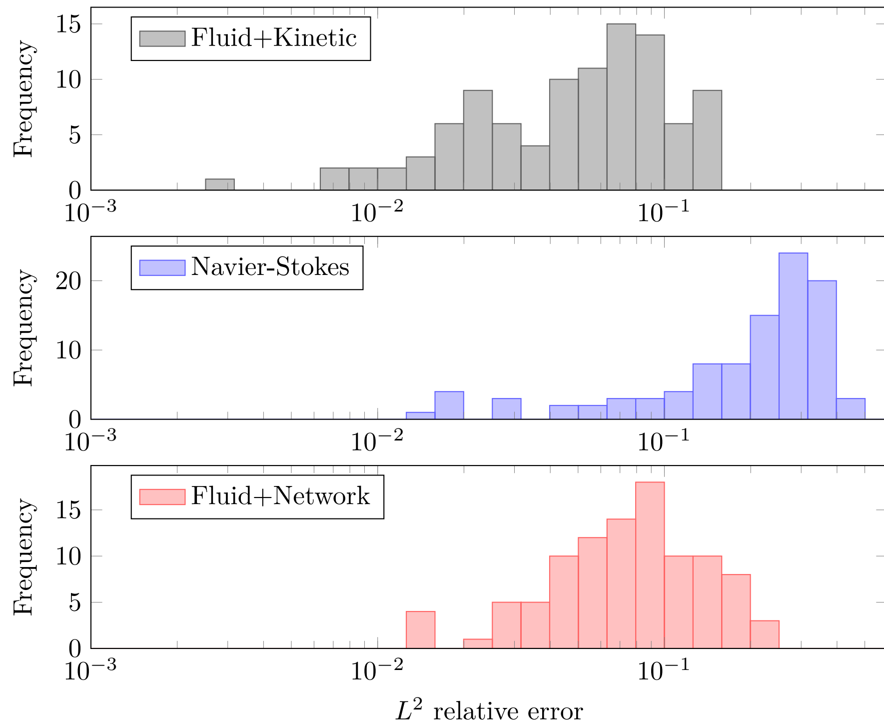


Figure 2.25 – Distributions over 100 simulations of the relative errors of the three fluid models compared to the kinetic model, measured on the logarithm of the electric energy from $t = 0$ to $t = 8$, *with discontinuous initial conditions*. For better readability, the x-axis in in logarithmic scale. It looks a lot like figure 2.18, which seems to indicate that the fluid+network model is not affected much by the initial discontinuities.

Discontinuities

The datasets were built from simulations starting with a continuous initial condition, but using the network in the fluid model with discontinuous initial conditions could be an option. In practice, the initial discontinuities quickly fade away in this physical system, but it might be enough for the predictions of the network to cause some oscillations and the instability that goes with it.

To see if that is the case, we run multiple simulations with different Knudsen numbers and different initial conditions, similar to those used to build the dataset, except that each function (density, velocity and temperature) can be multiplied by a function

$$x \in [0, 2\pi] \mapsto \begin{cases} \frac{c}{\pi}(x - x_d) + (1 + c), & \text{if } x < x_d, \\ \frac{c}{\pi}(x - x_d) + (1 - c), & \text{if } x \geqslant x_d, \end{cases}$$

adding a discontinuity at a random location $x_d \in [0, 2\pi]$ with a random amplitude $c \in [-1, 1]$. With these initial conditions, no simulation failed to reach $t = 8$, and as can be seen Figure 2.25, the results are comparable to those presented in Section 2.6.2.

This seems to indicate that the discontinuities do not prevent the fluid models —and in particular the fluid+network model— from working correctly with these discontinuous initial conditions. A possible explanation is that the discontinuities vanish quickly enough to make the initial loss of accuracy due to the network closure negligeable, thus not affecting the overall accuracy of the fluid+network model. Figure 2.26 illustrates the smoothing of the discontinuities by the dynamic of the system on an example with a randomly generated initial condition and with $\varepsilon = 0.1$, simulated with the kinetic model.

Our conclusion is that a neural network trained with continuous initial conditions could still be used with discontinuous initial conditions with little or no loss of accuracy. However, this may not remain true with a great number of discontinuities or with a dynamic that makes these discontinuities persistant in time. To put it differently, although one should always aim to make the training data as

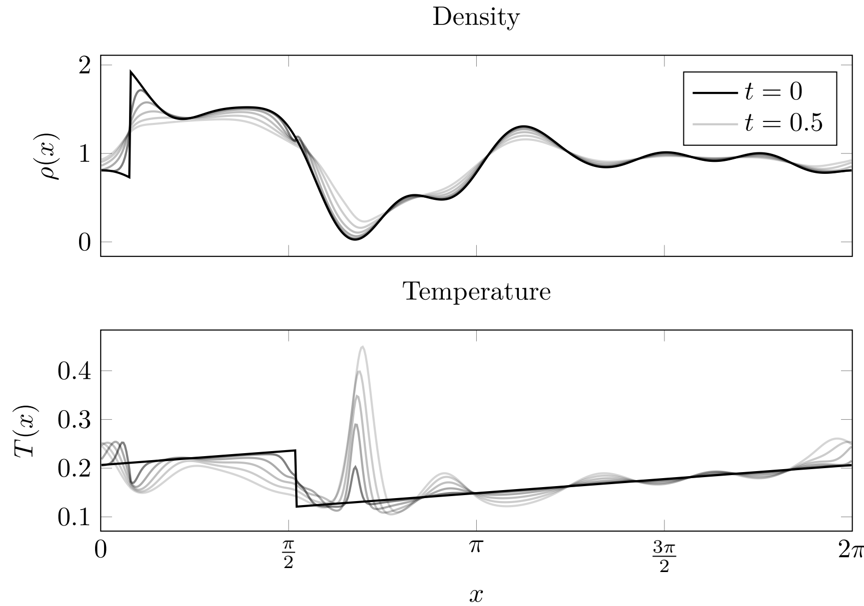


Figure 2.26 – Example of the evolution of the density and temperature up to $t = 0.5$ with a randomly generated *discontinuous* initial condition and $\varepsilon = 0.1$, obtained with the kinetic model. The initial discontinuities are quickly smoothed out by the dynamic of the system.

close as possible to the data the closure will be used with, this experiment shows that some flexibility does exist.

2.7 Conclusion

We construct a fluid closure for the Vlasov-Poisson dynamics based on V-net neural network and supervised learning from kinetic simulations. Slicing process of the data is introduced in order to manage meshes of different sizes. Several data processing have been also designed to improve the quality and regularity of the heat flux estimation. The numerical results show that the closure predicts the heat flux with a uniform relative error on the Knudsen interval $[0.01, 1]$, while the Navier-Stokes closure does not as expected. We also observe that the prediction is better at the beginning of the numerical simulations where the distribution function farthest from the equilibria set and so the real heat flux is larger. Surprisingly, we numerically observe that the neural network closure does not introduce instabilities when inserted in the fluid simulations, provided, however, that the outputs are regularized. Finally, the closure is quasi-optimal as the relative error between a full kinetic and a Fluid+Network closure behaves like the one between a full kinetic and a Fluid+Kinetic closure.

This work raises several issues from the numerical point of view. The first question is its efficiency in terms of computing time or its energy cost. Presently, the Fluid+Network model requires about the same computing time as the kinetic model. However, we can hope that it will be more efficient in higher dimension, as the number of computations for a V-Net with ℓ levels, depth d and kernel size p in dimension 1 is about $O(2^\ell d^2 pN)$, while it is about $O(\ell d^2 p^2 N^2)$ in dimension 2 and $O(d^2 p^3 N^3)$ in dimension 3. Compared with the $O(N^m N_v^m)$ computations required for the kinetic model where m denotes the space dimension, it increases much more slowly since $p << N_v$. Moreover, the neural network approach can greatly benefit from GPU parallelism.

The stability of the Fluid+Network closure is also an important issue. Despite the good numerical results obtained, a mathematical guaranty is lacking. Constructing neural networks closure ensuring such stability remains to be done.

Finally, we plan to apply this method to both the Vlasov-Poisson system in higher dimension and also to other closure problems arising in the MHD or gyrofluid design.

Bibliography

[1] A. Beck, D. Flad, and C. D. Munz. "Deep neural networks for data-driven LES closure models." In: *J. Comput. Phys.* 398 (2019), p. 108910.

[2] N. Besse and P. Bertrand. "Gyro-water-bag approach in nonlinear gyrokinetic turbulence." In: *J. Comput. Phys.* 228.11 (2009), pp. 3973–3995.

[3] N. Besse et al. "The multi-water-bag equations for collisionless kinetic modeling". In: *Kinet. Relat. Models* 2.1 (2009), p. 39.

[4] S.I. Braginskii. "Transport phenomena in plasma". In: *Rev. plasma phys.* 1 (1963), p. 205.

[5] A. Brizard. "Nonlinear gyrofluid description of turbulent magnetized plasmas". In: *Phys. Fluids B* 4.5 (1992), pp. 1213–1228.

[6] Z. Cai, Y. Fan, and R. Li. "Globally hyperbolic regularization of Grad's moment system". In: *Communications on pure and applied mathematics* 67.3 (2014), pp. 464–518.

[7] Z. Chang and J.D. Callen. "Unified fluid/kinetic description of plasma microinstabilities. Part I: Basic equations in a sheared slab geometry". In: *Physics of Fluids B: Plasma Physics* 4.5 (1992), pp. 1167–1181.

[8] G.F. Chew et al. "Application of dispersion relations to low-energy meson-nucleon scattering". In: *Phys. Rev.* 106.6 (1957).

[9] A. Crestetto, N. Crouseilles, and M. Lemou. "Kinetic/fluid micro-macro numerical schemes for Vlasov-Poisson-BGK equation using particles". In: *Kinet. Relat. Models* 5.4 (2012), p. 787.

[10] N. Crouseilles, P. Degond, and M. Lemou. "A hybrid kinetic/fluid model for solving the gas dynamics Boltzmann–BGK equation". In: *J. Comput. Phys.* 199.2 (2004), pp. 776–808.

[11] P. Degond. "Macroscopic limits of the Boltzmann equation: a review". In: *Modeling and Computational Methods for Kinetic Equations*. Birkhäuser, Boston, MA, 2004.

[12] P. Degond, G. Dimarco, and L. Mieussens. "A multiscale kinetic–fluid solver with dynamic localization of kinetic effects". In: *J. Comput. Phys.* 229.13 (2010), pp. 4907–4933.

[13] O. Desjardins, R. O. Fox, and P. Villedieu. "A quadrature-based moment method for dilute fluid-particle flows". In: *J. Comput. Phys.* 227.4 (2008), pp. 2514–2539.

[14] G. Dimarco and L. Pareschi. "Numerical methods for kinetic equations". In: *Acta Numer.* 23 (2014), p. 369.

[15] B. Dubroca and J. L. Feugeas. "Etude théorique et numérique d'une hiérarchie de modèles aux moments pour le transfert radiatif". In: *C. R. Acad. Sci. Paris Sér. I Math.* 329.10 (1999), pp. 915–920.

[16] B. Dubroca, J. L. Feugeas, and M. Frank. "Angular moment model for the Fokker-Planck equation". In: *The European Physical Journal D* 60.2 (2010), pp. 301–207.

[17] D. Dumoulin and F. Visin. *A guide to convolution arithmetic for deep learning*. 2016. arXiv: 1603.07285 [stat.ML].

[18] K. Duraisamy, G. Iaccarino, and H. Xiao. "Turbulence modeling in the age of data". In: *Annu. Rev. Fluid Mech.* 51 (2019), pp. 357–377.

[19] C. K. Garrett and C. D. Hauck. "A comparison of moment closures for linear kinetic transport equations: the line source benchmark". In: *Transport Theor. Stat.* 42.6-7 (2013), pp. 203–235.

[20] H. Grad. "On the kinetic theory of rarefied gases". In: *Commun. Pure Appl. Math.* 2 (1949), pp. 331–407.

[21] G. W. Hammett, W. Dorland, and F. W. Perkins. "Fluid models of phase mixing, Landau damping, and nonlinear gyrokinetic dynamics". In: *Phys. Fluids B* 4.7 (1992), pp. 2052–2061.

[22] G. W. Hammett and F. W. Perkins. "Fluid moment models for Landau damping with application to the ion-temperature-gradient instability". In: *Phys. Rev. Lett.* 64 (25 June 1990), pp. 3019–3022. DOI: 10.1103/PhysRevLett.64.3019. URL: https://link.aps.org/doi/10.1103/PhysRevLett.64.3019.

[23] J. Han et al. "Uniformly accurate machine learning-based hydrodynamic models for kinetic equations". In: *PNAS* 116.44 (Oct. 2019), pp. 21983–21991. ISSN: 1091-6490. DOI: 10.1073/pnas.1909854116. URL: http://dx.doi.org/10.1073/pnas.1909854116.

[24] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[25] P. Helluy et al. "Reduced Vlasov-Maxwell simulations". In: *C. R. Mécanique* 342.10-11 (2014), pp. 619–635.

[26] P Hunana et al. "New closures for more precise modeling of Landau damping in the fluid framework". In: *Physical review letters* 121.13 (2018), p. 135101.

[27] M. Junk. "Domain of definition of Levermore's five-moment system". In: *J. Stat. Phys.* 93.5-6 (1998), pp. 1143–1167.

[28] R.J. LeVeque. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge University Press, 2002.

[29] C. D. Levermore. "Moment closure hierarchies for kinetic theories". In: *J. Stat. Phys.* 83.5-6 (1996), pp. 1021–1065.

[30] C. D. Levermore and W. J. Morokoff. "The Gaussian moment closure for gas dynamics". In: *SIAM J. Appl. Math.* 59.1 (1998), pp. 72–96.

[31] Chenhao Ma et al. "Machine Learning Surrogate Models for Landau Fluid Closure". In: *Physics of Plasmas* 27.4 (Apr. 2020), p. 042502. ISSN: 1070-664X, 1089-7674. DOI: 10/gjt9ck. URL: http://arxiv.org/abs/1909.11509.

[32] G. Manfredi. "Density functional theory for collisionless plasmas–equivalence of fluid and kinetic approaches." In: *J. Plasma Phys.* 86.2 (2020).

[33] R. Maulik et al. "Neural network representability of fully ionized plasma fluid model closures". In: *Phys. Plasmas* 27.072106 (2020).

[34] F. Milletari, N. Navab, and S. Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *2016 Fourth International Conference on 3D Vision (3DV)*. 2016, pp. 565–571.

[35] C. Negulescu and S. Possanner. "Closure of the strongly magnetized electron fluid equations in the adiabatic regime". In: *Multiscale Model. Sim.* 14.2 (2016), pp. 839–873.

[36] T. Passot, P.L. Sulem, and P. Hunana. "Extending magnetohydrodynamics to the slow dynamics of collisionless plasmas". In: *Physics of Plasmas* 19.8 (2012), p. 082113.

[37] M. Perin et al. "Hamiltonian closures for fluid models with four moments by dimensional analysis". In: *J. Phys. A Math. Theor.* 48.27 (2015), p. 275501.

[38] N. Pham, P. Helluy, and A. Crestetto. "Space-only hyperbolic approximation of the Vlasov equation". In: *ESAIM: Proc.* 43 (2013), pp. 17–36.

[39] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241. arXiv: 1505.04597 [cs.CV].

[40] J. Schneider. "Entropic approximation in kinetic theory". In: *Esaim Math Model Numer Anal.* 38.3 (2004), pp. 541–561.

[41] C. Shorten and T.M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (2019), p. 60. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[42] P.B. Snyder, G.W. Hammett, and W. Dorland. "Landau fluid models of collisionless magnetohydrodynamics". In: *Phys. Plasmas* 4.11 (1997), pp. 3974–3985.

[43] E. Sonnendrücker. *Numerical Methods for the Vlasov-Maxwell equations*. 2015.

[44] H. Struchtrup and M. Torrilhon. "Regularization of Grad's 13 moment equations: Derivation and linear analysis". In: *Physics of Fluids* 15.9 (2003), pp. 2668–2680.

[45] E. Tassi. "Hamiltonian closures in fluid models for plasmas". In: *Eur. Phys. J. D* 71.11 (2017), p. 269.

[46] M. Torrilhon. "Hyperbolic moment equations in kinetic gas theory based on multi-variate Pearson-IV-distributions." In: *Commun. in Comput. Phys.* 7.4 (2010), p. 639.

[47] M. Torrilhon. "Modeling nonequilibrium gas flow based on moment equations." In: *Annual review of fluid mechanics* 48 (2016), pp. 429–458.

[48] J.-X. Wang, J.-L. Wu, and H. Xiao. "Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data". In: *Physic. Rev. Fluids* 2.3 (2017), p. 034603.

[49] X.H. Zhou, J. Han, and H. Xiao. "Learning nonlocal constitutive models with neural networks". In: *arXiv:2010.10491 [physics]* (Oct. 2020). arXiv: 2010.10491. URL: http://arxiv.org/abs/2010.10491 (visited on 03/25/2021).

Appendix

2.A Numerical scheme for the kinetic model

In this appendix we describe the numerical method used to solve the Vlasov-Poisson equations (2.5)-(2.6) resulting from the kinetic model. This numerical method is used to produce data that can in turn be used by the neural network to interpolate the heat flux. It also serves as a reference to compute the error of the other methods, allowing us to compare them. For better readability, let us remind the one dimensional Vlasov-Poisson equations:

$$\partial_t f + \ v\,\partial_x f - \ E\,\partial_v f = \ \frac{1}{\varepsilon}(M(f) - f),$$
$$E = -\partial_x \phi, \quad \partial_{xx}\phi = \rho - \int_0^L \rho\,dx.$$

The spatial domain is given by $[0, L]$, where $L > 0$ is the spatial length. For numerical purpose, the velocity domain is restricted to the bounded interval $[-v_{\max}, v_{\max}]$. Thus we complement the equation with the following boundary conditions:

$$(\pm E)^- \, f = 0, \quad \text{at } v = \pm v_{\max}.$$

We consider a time discretization $(t^n)_n$ with variable time step Δt and a discretized phase space $(x_i, v_j)_{i,j}$ with constant steps Δx and Δv respectively. The number of discretization points in space (resp. in velocity) is denoted N_x (resp. N_v). We denote by $f^n_{i,j}$ the approximation

$$f^n_{i,j} \simeq f(x_i, v_j, t^n).$$

We also use the notations $\mathbf{f}^n$ for the matrix $(f^n_{i,j})_{i,j}$, $\mathbf{f}^n_i$ for the vector $(f^n_{i,j})_j$ and $\mathbf{f}^n_j$ for the vector $(f^n_{i,j})_i$.

2.A.1 Time discretization

To solve the Vlasov-Poisson equations over the time interval $[t^n, t^n + 1]$ we use a splitting between three stages:

1. Compute the electric field at time t^n by solving the Poisson system:

$$E = -\partial_x \phi \quad , -\partial_{xx}\phi = \frac{1}{L}\int_0^L \rho dx,$$

2. Transport the distribution function by solving the Vlasov equation over the time inverval $[t^n, t^n + 1]$:

$$\partial_t f + v\partial_x f - E\partial_v f = 0,$$

3. Update the distribution function by taking into account the collision operator:

$$\partial_t f = \frac{1}{\varepsilon}(M(f) - f).$$

The first two stages rely on the space discretization and are discussed in the next section. For the third operator with a stiff source term, we use an implicit scheme:

$$\frac{f^{n+1}-f^n}{\Delta t} = \frac{1}{\varepsilon}(M(f^{n+1}) - f^{n+1}).$$

Knowing that the fluid quantities ρ, u, T are preserved by this operator, we have $M(f^{n+1}) = M(f^n)$, so the scheme can be rewritten as:

$$f^{n+1} = f^n + \omega\,(M(f^n) - f^n), \quad \text{with } \omega = \frac{\Delta t}{\Delta t + \varepsilon}. \tag{2.10}$$

2.A.2 Spatial discretization

For the spatial discretization, we propose a method introduced in [38, 25]. First we discretize in velocity with a centered finite difference scheme. After discretization, we get the following hyperbolic system:

$$\frac{\mathbf{f}^{n+1}-\mathbf{f}^n}{\Delta t} + \Lambda\partial_x \mathbf{f}^n + EB(\mathbf{f}^n) = 0 \tag{2.11}$$

with Λ the diagonal matrix of velocities and $B(f^n)$ a vector given by

$$B(f^n)_j = -\frac{f^n_{j+1} - f^n_{j-1}}{2\Delta v}, \quad j \in \{2, \ldots, N_v - 1\},$$
$$B_1 = -\frac{1}{2}\max(E,0)Ef^n_1, \quad B_{N_v} = \frac{1}{2}\min(E,0)Ef^n_{N_v}$$

The choice of the boundary terms allows to obtain a dissipative hyperbolic system and to ensure L^2 stability. Then we discretize in space the hyperbolic system (2.11). We use a finite volume scheme with an upwind flux:

$$\frac{\mathbf{f}^{n+1}_i - \mathbf{f}^n_i}{\Delta t} + \frac{\mathbf{f}^n_{i+\frac{1}{2}} - \mathbf{f}^n_{i-\frac{1}{2}}}{\Delta x} - EB(\mathbf{f}^n) = 0, \tag{2.12}$$

with $\mathbf{f}^n_{i+\frac{1}{2}} = \frac{1}{2}\Lambda(\mathbf{f}^n_{i+1} + \mathbf{f}^n_i) - \frac{1}{2}\,|\Lambda|\,(\mathbf{f}^n_{i+1} - \mathbf{f}^n_i)$.

The Poisson equation is solved using a classical finite difference scheme:

$$E^n_i = -\frac{\phi^n_{i+1} - \phi^n_{i-1}}{2\Delta x}, \quad -\frac{\phi^n_{i+1} - 2\phi^n_i - \phi^n_{i-1}}{\Delta x^2} = \rho_j - \frac{1}{N_x}\sum_i^N \rho_i. \tag{2.13}$$

where the density ρ_j is computed as follows: $\rho_i = \sum_{j=1}^{N_v} f_{i,j}$.

The total scheme is given by (2.13)-(2.12)-(2.10). The time step is chosen such as to satisfy the following stability condition:

$$\Delta t \leqslant \min\left\{\frac{\Delta x}{v_{\max}}, \frac{\Delta v}{\max_i |E_i|}\right\}.$$

Note that it is only first order accurate in order to avoid dispersive oscillations in the numerical solutions.

2.B Numerical scheme for the fluid models

In this section we introduce the numerical methods for solving fluid models (2.8). The time discretization depends on the considered closure. For the neural network based (Fluid+Network) or the kinetic one (Fluid+Network), we use an explicit scheme. For the Navier-Stokes closure (Navier-Stokes), an implicit scheme is required to avoid a too stringent stability condition. In one case or the other, the Poisson equation is solved at the beginning of each iteration in time to compute the electric field, exactly as in Eq. (2.13). This section therefore focuses on the fluid equations. The following schemes are classical. We briefly present them for the sake of completeness.

2.B.1 Explicit scheme for the neural network or the kinetic closure

Fluid equations (2.8) can be written

$$\partial_t \mathbf{U} + \partial_x \mathbf{F}(\mathbf{U}) = -E\mathbf{H}(\mathbf{U}), \tag{2.14}$$

with $\mathbf{U} = (\rho, \rho u, w)$, $\mathbf{F}(\mathbf{U}) = (\rho u, \rho u^2 + p, wu + pu + q)$, $\mathbf{H}(\mathbf{U}) = (0, \rho, \rho u)$ and where the heat flux q is given by the neural network based closure (Fluid+Network) or the one obtained from full kinetic simulations (Fluid+Kinetic). We solve it on the spatial domain $[0, L]$.

When $q = 0$ (Euler closure), equation 2.14 is an hyperbolic system that can be solved using a finite volume method with a local Lax-Friedrichs numerical flux and an explicit scheme in time. The hyperbolic system has characteristic speeds equal to $u, u + c, u - c$ where $c = \sqrt{3p/\rho}$ is the sound speed. When considering an additional heat flux, we propose to use the same scheme and just add a centered approximation of the heat flux in the numerical flux, as explained below.

Like for the kinetic model, we consider a time discretization $(t^n)_n$ with variable time step Δt and a discretized phase space $(x_i)_i$ with constant step Δx. We denote by U_i^n the approximation

$$\mathbf{U}_i^n \simeq \mathbf{U}(x_i, t^n).$$

The finite volume scheme results in the following formula:

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t} + \frac{\mathbf{F}(\mathbf{U})_{i+\frac{1}{2}}^n - \mathbf{F}(\mathbf{U})_{i-\frac{1}{2}}^n}{\Delta x} = -\mathbf{H}(\mathbf{U})_i^n E_i^n,$$

with

$$\mathbf{F}(\mathbf{U})_{i+\frac{1}{2}}^n = \frac{1}{2}(\mathbf{F}(\mathbf{U})_{i+1}^n + \mathbf{F}(\mathbf{U})_i^n) - \frac{S_{i+1/2}^n}{2}(\mathbf{U}_{i+1}^n - \mathbf{U}_i^n),$$

where

$$S_{i+\frac{1}{2}}^n = \max(|u_i^n| + c_i^n, |u_{i+1}^n| + c_{i+1}^n) \quad \text{and} \quad c_i^n = \sqrt{\frac{3p_i^n}{\rho_i^n}}.$$

Quantities $S_{i+\frac{1}{2}}^n$ are chosen to be larger than the maximum characteristic speed of the hyperbolic system, as u_i^n is the local speed of particles and c_i^n the local sound speed. Comparing with the classical scheme for Euler system ($q = 0$), the numerical flux for the momentum have an additional term equal to $\frac{1}{2}(q_{i+1}^n + q_i^n)$. The time step is chosen such as to satisfy the CFL stability condition:

$$\left(\max_i S_{i+\frac{1}{2}}^n\right) \Delta t = \frac{1}{2}\Delta x. \tag{2.15}$$

We refer to [28] for more details on finite volume methods.

For the Fluid+Network method, the heat flux q involved in the flux term is computed from ε, ρ, u and T at each iteration. For the Fluid+Kinetic method, it is obtained from an underlying kinetic simulation, using the same initial condition at $t = 0$. Note that the stability condition (2.15) does not take into account the non-zero heat flux. However, as observed in Section 2.6.2, this term does not result in an additional stability condition for the Fluid+Kinetic and the Fluid+Network method, provided for the latter that the heat flux is sufficiently smoothed out.

2.B.2 Semi-implicit scheme for the Navier-Stokes closure

With the Navier-Stokes closure $q = -\frac{3}{2}\varepsilon p \partial_x T$, the first two equations remain the same as above and are solved using the same explicit finite volume method, while the third one reads

$$\partial_t w + \partial_x(wu + pu) - \frac{3}{2}\varepsilon \partial_x(p\partial_x T) = -E\rho u.$$

To solve this equation we use the relation $w = \frac{1}{2}\rho u^2 + \frac{1}{2}\rho T$ to turn it into an equation on T, as ρ and u are known after solving the first two equations. We use a finite difference approximation for the term

$\partial_x(p\partial_x T)$, and T^{n+1} can then be computed by solving the following linear system:

$$\begin{aligned}
&\frac{\frac{1}{2}\rho_i^{n+1}(u_i^{n+1})^2 + \frac{1}{2}\rho_i^{n+1}T_i^{n+1} - \frac{1}{2}\rho_i^n(u_i^n)^2 - \frac{1}{2}\rho_i^n T_i^n}{\Delta t} \\
&+ \frac{\mathbf{F}(\mathbf{U})[2]_{i+\frac{1}{2}}^n - \mathbf{F}(\mathbf{U})[2]_{i-\frac{1}{2}}^n}{\Delta x} \\
&- \frac{3}{2}\varepsilon \frac{p_{i+1/2}^n T_{i+1}^{n+1} - (p_{i+1/2}^n + p_{i-1/2}^n)T_i^{n+1} + p_{i-1/2}^n T_{i-1}^{n+1}}{\Delta x^2} \\
&= -E_i^n \rho_i^n u_i^n.
\end{aligned}$$

Here the only unknown is T^{n+1}, and $\mathbf{F}(\mathbf{U})[2]$ is the third coordinate of $\mathbf{F}(\mathbf{U})$. Finally, $(w)^{n+1}$ can be computed using again the relation $w = \frac{1}{2}\rho u^2 + \frac{1}{2}\rho T$. Here the time step is chosen such as to satisfy the same CFL condition (2.15) as the implicit Navier-Stokes term does not introduce additional stability constraint.

Chapter 3

A neural network based closure for the 2D Boltzmann-BGK equation

Abstract

A gas can be described by the distribution of its particles in the phase space, whose evolution follows the Boltzmann equation. Simulations of this kinetic model are expensive because they require the discretization of the phase space. The first three moments in velocity of the Boltzmann equation lead to a system of equations that is cheaper to solve, but that requires a closure. When the mean free path between collisions tends to zero, closures can be found analytically using the Chapman-Enskog expansion. This process leads to the Euler closure at first order and the Navier-Stokes closure at order two.

In this work we propose a framework to produce an empirical closure to the fluid system that fits the dynamic of a gas further away from this asymptotic behaviour. In order to achieve this goal we use convolutional neural networks, trained on data generated with the kinetic model. We explore different architectures and different ways to stabilize the resulting numerical scheme despite the use of neural networks. We give numerical results to evaluate the performance of the data-driven closure.

3.1 Introduction

The modeling of gases for numerical simulations has to strike the right balance between physical accuracy and computational cost. Kinetic models, which describe gases by the distribution of their particles in both positional and velocity space, are generally prohibitively expensive due to the high-dimensionalality of the phase space. A convenient way of reducing the dimension of the problem is to get rid of the velocity dependence by replacing the velocity distributions with a number of their moments. The latter relate to macroscopic quantities such as density, mean velocity, pressure, etc. From the kinetic equation can thus be derived systems of equations, that typically require a closure to make up for the loss of information. A Chapman-Enskog expansion in the Knudsen number ε, ratio between the mean free path between two collisions and a characteristic length of the system, allows to get such a closure and leads to the Euler equations at order 1, and the Navier-Stokes equations at order 2. By construction these fluid models are well suited to highly collisional regimes when the kinetic solution is close to a maxwellian equilibrium, but they prove inaccurate in less collisional regimes (say $\varepsilon > 0.1$).

Many attempts have been made to extend the validity of fluid models. Higher order expansions in ε lead to Burnett equations [3], but these suffer from ill-posedness. Models involving more moments have been proposed, first by Grad [6] with a 13 moments model relying on an expansion of the velocity distributions in Hermite polynomials, then by Levermore [9] with a 14 moments model relying on the minimization of an entropy. However these models come with their own disadvantages, including higher complexity due to the number of moments involved, and realisability issues.

A new approach that takes advantage of the recent advances in machine learning was recently introduced in order to find data-driven closures, better suited to a given set of problems, and affordable

to compute. The method generally involves the use of neural networks, *ie* functions relying on many parameters, and their optimization by means of a gradient descent algorithm. In [15][14], the authors first study the ability of such neural networks to learn known analytical closures in the context of plasmas. Closure with neural networks is performed in [1][8] for large eddy simulations (LES), and in [2] for plasmas. In [19], the authors propose two approaches to train a neural network based closure that preserves structural properties of the Boltzmann equation. Other ways of using neural networks for simulating the Boltzmann equation have been studied. In [20] and [16], the authors use neural networks as surrogates for the Boltzmann collision operator, to get rid of its expensive calculation. Neural networks can also be used to replace the numerical scheme entirely and perform the mapping between the variables and the solution directly. This is the popular approach of so called PINNs (physics-informed neural networks), that has been carried out for the Boltzmann equation by [13] to solve Kovasznay flow, Taylor-Green flow, cavity flow and Couette flow, by [5][4] to solve thermal creep flow and Poiseuille problems respectively, by [10][11] for phonon Boltzmann transport equation, alone and coupled with the electron Boltzmann transport equation respectively. The field of machine learning applied to PDE solving is quickly evolving, and the list above is by no means exhaustive.

In this work, we aim to extend our previous results [2] from dimension one to two. The goal is to provide a straightforward process to train a neural network based closure with data from kinetic simulations. We consider the 2D Boltzmann equation with BGK collision operator, and look for a closure to the equations on density, mean velocity and pressure, for Knudsen numbers between 0.01 and 1; the idea being to perform better than the Navier-Stokes closure that looses its validity somewhere in this range. To do so we use neural networks whose role is to compute an estimation of the heat flux and stress tensor, knowing the density, mean velocity, pressure and the Knudsen number. We use convolutional neural networks to allow our closure to be non-local, and train them using simple supervised learning, with inputs and expected outputs extracted from kinetic simulations carried out for this purpose. Despite these neural networks seemingly performing well, their use in numerical schemes raises stability issues. We propose a few techniques to mitigate these issues, and evaluate the accuracy of the stabilized model.

The paper is structured as follows. In section 3.2 we briefly describe the kinetic model we start from, and the fluid model that we derive from it. Section 3.3 introduces the neural network approach for the closure of the fluid model, and gives details on its implementation. Finally numerical results are given and discussed in section 3.4.

3.2 Models

3.2.1 Kinetic model

In this paper we work with the 2D Boltzmann equation with BGK collision operator, for its simplicity:

$$\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = \frac{1}{\varepsilon}(M(f) - f) \tag{3.1}$$

The unknown f is a function of $\mathbf{x}$, $\mathbf{v}$ and t that describes the distribution of the particles in the phase space. The Knudsen number $\varepsilon > 0$ is the ratio between the mean free path between two collisions and a characteristic length of the system, and gives more or less strength to the BGK collision operator that simply consists in a relaxation towards a Maxwellian equilibrium state

$$M(f)(\mathbf{x}, \mathbf{v}, t) = \frac{\rho(\mathbf{x}, t)}{2\pi T(\mathbf{x}, t)} \exp\left(-\frac{|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)|^2}{2T(\mathbf{x}, t)}\right),$$

where ρ, $\mathbf{u}$, and T are respectively the density, mean velocity and temperature of the gas:

$$\begin{aligned}
\rho(\mathbf{x}, t) &= \int f(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v} \\
\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) &= \int f(\mathbf{x}, \mathbf{v}, t)\mathbf{v}\, d\mathbf{v} \\
\rho(\mathbf{x}, t)T(\mathbf{x}, t) &= \int f(\mathbf{x}, \mathbf{v}, t)|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)|^2\, d\mathbf{v}
\end{aligned}$$

For simplicity, we use periodic boundary conditions in all this work.

3.2.2 Fluid model

From the kinetic model can be derived a fluid model, by multiplying the Boltzmann equation (3.1) by the vector $\begin{pmatrix} 1 & \mathbf{v} & \|\mathbf{v}\|^2 \end{pmatrix}^T$ and integrating with respect to $\mathbf{v}$. Doing so results in the following system:

$$\begin{aligned} &\partial_t \rho + \nabla_{\mathbf{x}} \cdot (\rho \mathbf{u}) = 0 \\ &\partial_t (\rho \mathbf{u}) + \nabla_{\mathbf{x}} \cdot (\rho \mathbf{u} \otimes \mathbf{u} + pI + \Pi) = 0 \\ &\partial_t \left(p + \tfrac{1}{2}\rho|\mathbf{u}|^2\right) + \nabla_{\mathbf{x}} \cdot \left(2p\mathbf{u} + \tfrac{1}{2}\rho|\mathbf{u}|^2\mathbf{u} + \Pi\mathbf{u} + \mathbf{q}\right) = 0 \end{aligned} \tag{3.2}$$

with unknowns ρ, $\mathbf{u}$ and p, and where expressions for Π and $\mathbf{q}$ are required for this system to be closed. All these quantities are defined as follows:

Density ρ	$\rho(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}$
Mean velocity $\mathbf{u}$	$\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)\mathbf{v}\, d\mathbf{v}$
Scalar pressure p	$p(\mathbf{x}, t) = \frac{1}{2} \int f(\mathbf{x}, \mathbf{v}, t)|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)|^2\, d\mathbf{v}$
Pressure tensor $\mathbb{P}$	$\mathbb{P}(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t)(\mathbf{v} - \mathbf{u}(\mathbf{x}, t)) \otimes (\mathbf{v} - \mathbf{u}(\mathbf{x}, t))\, d\mathbf{v}$
Stress tensor Π	$\Pi(\mathbf{x}, t) = \mathbb{P}(\mathbf{x}, t) - p(\mathbf{x}, t)I$
Heat flux $\mathbf{q}$	$\mathbf{q}(\mathbf{x}, t) = \frac{1}{2} \int f(\mathbf{x}, \mathbf{v}, t)|\mathbf{v} - \mathbf{u}(\mathbf{x}, t)|^2(\mathbf{v} - \mathbf{u}(\mathbf{x}, t))\, d\mathbf{v}$

A Chapman-Enskog expansion of f in the Knudsen number ε allows to recover an expression of Π and $\mathbf{q}$ in function of the unknowns asymptotically when $\varepsilon \to 0$. At first order we thus obtain the Euler closure

$$\begin{aligned} \Pi &= 0 \\ \mathbf{q} &= 0, \end{aligned}$$

while the second order leads to the Navier-Stokes closure

$$\begin{aligned} &\Pi = -\varepsilon p \sigma(\mathbf{u}), \quad \sigma(\mathbf{u}) = \nabla_{\mathbf{x}}\mathbf{u} + (\nabla_{\mathbf{x}}\mathbf{u})^T - (\nabla_{\mathbf{x}} \cdot \mathbf{u})I \\ &\mathbf{q} = -2\varepsilon p \nabla T. \end{aligned}$$

3.3 Neural network closure

In this work, our goal is to obtain an empirical closure, suited to a given set of initial conditions and to a given set of Knudsen numbers ε, further away from the asymptotic behavior $\varepsilon \to 0$. To do so, we use neural networks that learn a mapping from the input quantities $(\varepsilon, \rho, \mathbf{u}, p)$ to the output quantities $(\Pi, \mathbf{q})$, trained with supervised learning using data from kinetic simulations. These neural networks can then be used to compute Π and $\mathbf{q}$ at each timestep of a numerical scheme to solve the fluid system (3.2).

This section describes the main elements of our approach: the architecture of the neural networks, the data generation process for their training, and stabilization techniques to better integrate the networks into the numerical scheme. In the remainder of this paper we use the following notations (note that Π is traceless and symmetrical):

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \quad \Pi = \begin{bmatrix} \pi_{xx} & \pi_{xy} \\ \pi_{xy} & -\pi_{xx} \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} q_x \\ q_y \end{bmatrix}.$$

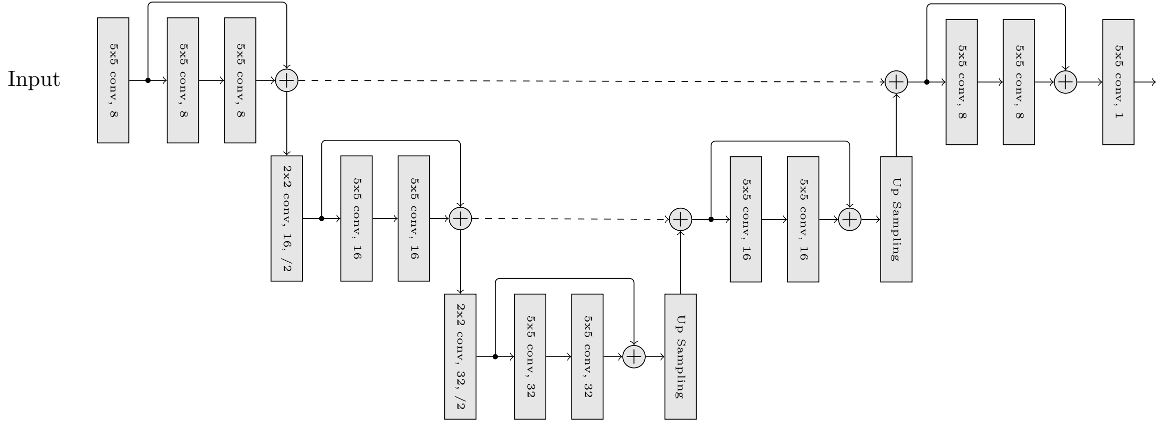


Figure 3.3.1 – Architecture of the neural network used for each output feature. At each level downward, data size is multiplied by 2 in depth, and divided by 2 in x and y, using a convolution with strides 2 (denoted by $/2$) and kernel size 2×2. The upsampling is done by duplicating each cell and smoothing the result using a convolution with kernel size 2×2.

3.3.1 Neural network architecture

The goal of the neural network is to compute an estimation of the four quantities π_{xx}, π_{xy}, q_x and q_y, from the five inputs ε, ρ, u_x, u_y, p. In our numerical scheme for the fluid system (see appendix 3.A.2), we use a finite volume method on a uniform discretization of the 2D space, so that all these quantities at a given time are represented by a matrix (except for ε), with local properties similarly to images. We therefore decide to use convolutional neural networks for their great performance with such kind of data.

We experimented with the use of a single neural network to output all four quantities π_{xx}, π_{xy}, q_x and q_y, but with no gain compared to using four completely independant neural networks. Since the latter comes with benefits in flexibility and memory requirements, we end up using four distinct neural networks, one for each output feature.

We try out three types of standard architecture for convolutional networks: Resnet [7], V-Net [18][17], and Fourier networks (FNO) [12]. For each of these we train multiple networks with different hyperparameters: size of the convolution kernel, width of the convolutions (number of filters), and depth of the network (number of convolutions). Some results are shown in section 3.4.1, which lead us to use the following architecture for each of the output features: a V-Net with 3 levels, of width 8 and kernel size 5, with 92769 trainable parameters, shown Figure 3.3.1.

3.3.2 Training and testing dataset

Simulations are computed using the kinetic model in order to retrieve all the data for training and testing: the density, mean velocity and pressure for the input of the neural networks, and the stress tensor and heat flux for the expected outputs.

Each simulation is carried out with a different initial condition and a different Knudsen number. The initial condition f is taken as a maxwellian with density, mean velocity and temperature randomly generated. To do so, we use a random generator of 2D functions based on Fourier series:

$$g(x, y; \omega) = \Re \left(\sum_{j,k \in \{-10,\dots,10\}} \frac{A_{jk}(\omega) + iB_{jk}(\omega)}{1 + j^2 + k^2} e^{ij\pi x} e^{ik\pi y} \right), \quad x, y \in [-1, 1],$$

where A_{jk} and B_{jk} are uniform random variable on $[-1, 1]$, and ω represents a random seed. The scaling factor $\frac{1}{1+j^2+k^2}$ ensures lower amplitudes for higher frequencies, for more regularity. The variables (ρ,

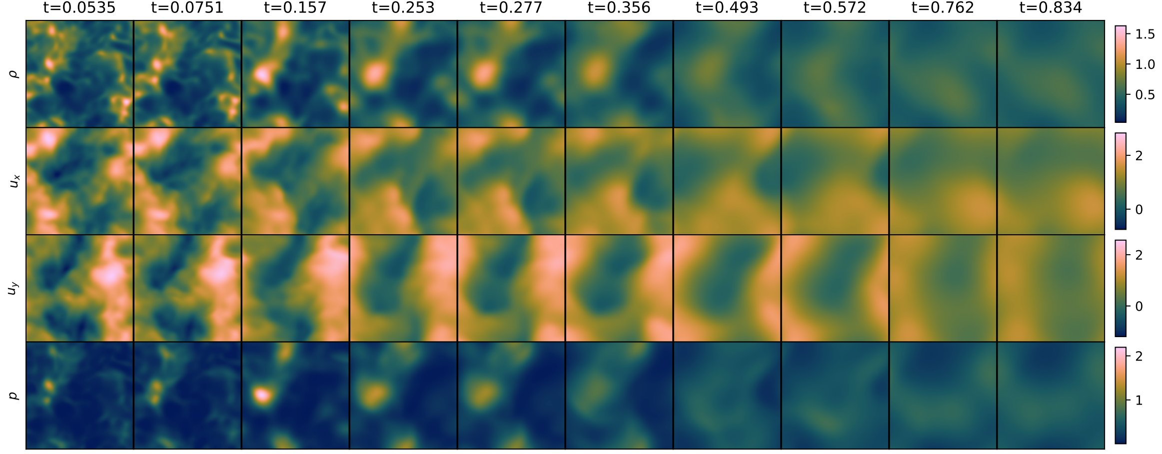


Figure 3.3.2 – Example of input data (ρ, u_x, u_y, p) extracted from a single kinetic simulation for the test dataset, with Knudsen number $\varepsilon \simeq 0.13$. Only 10 out of the 20 entries are shown for clarity.

u_x, u_y, p) are then taken as

$$\begin{aligned}
\rho(x,y) &= \text{Softplus}(g(x,y;\omega)),\\
u_x(x,y) &= g(x,y;\omega'),\\
u_y(x,y) &= g(x,y;\omega''),\\
T(x,y) &= \tfrac{1}{2}\text{Softplus}(g(x,y;\omega''')).
\end{aligned}$$

The softplus function $X \mapsto \log(1+e^X)$ is used to make the density and pressure positive, and the $\frac{1}{2}$ coefficient is here to decrease the variance of the velocity distribution in f, so that velocity amplitudes spread over a smaller interval.

The Knudsen number is taken as the square of a sample of the uniform distribution on $[0.1, 1]$, and thus lies between 0.01 and 1. Using the data of the whole simulations would be very much redondant and thus unnecessarily expensive, so instead we pick some recording times randomly, similarly to the Knudsen number but between 0 and 1.

We generate two datasets in this way: a training dataset with 200 simulations, and a test dataset with 40 simulations. Both use a space resolution of 100×100, and 20 times per simulations are recorded, resulting in datasets with 4000 and 800 entries respectively, each entry consisting of 8 features (ρ, u_x, u_y, p, Π_{xx}, Π_{xy}, q_x, q_y) computed on a 100×100 grid. The Knudsen number used for each entry is also saved to be used as an input for the neural networks. Figure 3.3.2 gives an example of inputs $(\rho, \mathbf{u}, p)$ generated this way, and the corresponding outputs $(\Pi, \mathbf{q})$ can be seen Figure 3.4.2.

3.3.3 Data processing and augmentation

In this work we use very little pre-processing since we only apply a scaling to the input so that each input feature (ρ, u_x, u_y, p) has a standard deviation of 1 in the training dataset. As for post-processing, we only use smoothing in order to cope with stability issues, as discussed in section 3.3.4.

In order to exploit the expensive kinetic simulations as much as possible, as well as encouraging the neural networks to learn the symmetries we would expect them to have, we augment the training data using some simple transformations to the physical domain, namely the transformations generated by the rotation of $\frac{\pi}{2}$ and the reflexion w.r.t the y-axis. These allow to artificially increase the size of the dataset by a factor 8. Appendix 3.C gives the effect of these transformations on each feature.

3.3.4 Stabilization techniques

As we show in the results section, the neural network closure makes the numerical scheme mostly unstable when used as is, due to negative pressures occuring. We experimented with different techniques in order to mitigate this issue, that we describe in this section. The numerical results regarding

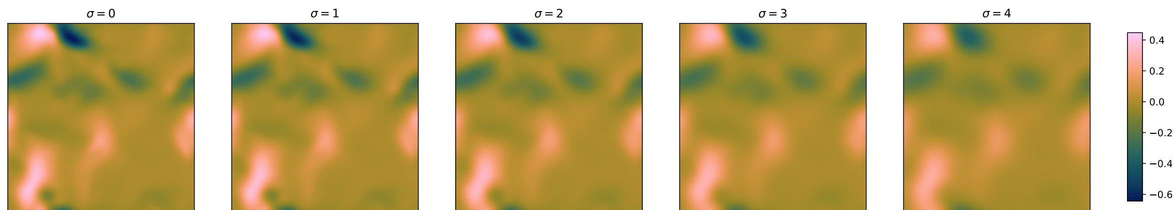


Figure 3.3.3 – An example of π_{xx} from the test dataset smoothed with kernels of different standard deviations σ. Knudsen number $\varepsilon \simeq 0.14$, time $t \simeq 0.11$.

stability and accuracy of the model when using these techniques are discussed in sections 3.4.2 and 3.4.3 respectively.

The first technique is the smoothing of the neural networks' outputs, similarly to what we did in our first work in 1D [2], that aims at removing noise from the closure. This smoothing is performed via a convolution with a gaussian kernel, whose standard deviation σ can be adjusted to put more or less smoothing. Figure 3.3.3 shows the effect of this smoothing on an example with different standard deviations.

As the smoothing is not enough to stabilize the model, we also tweak the viscosity coefficient μ in the numerical flux of our scheme (cf appendix 3.A.2), initially set to 1.1. Increasing this coefficient makes the numerical scheme more diffusive, and less prone to produce negative pressures when used with the neural network.

Since both of these techniques lower the accuracy of the resulting numerical solution, we also experiment training neural networks on more restricted ranges of Knudsen numbers. We slice the $[0.01, 1]$ range in three parts: $[0.01, 0.05]$, $[0.05, 0.2]$ and $[0.2, 1]$. We thus train 3 new sets of neural networks, each on their respective sub-dataset of the training dataset.

Finally, we also compare our techniques to more drastic methods, namely preventing the pressure from dropping below a given threshold (set to 10^{-4}), and switching from our order 2 MUSCL scheme to a scheme of order 1.

3.4 Numerical results

3.4.1 Accuracy on test dataset

As described in section 3.3.2, our test datasets comes from 40 kinetic simulations, each with a different Knudsen number $\varepsilon \in [0.01, 1]$. From each simulation are extracted $(\rho, \mathbf{u}, p, \Pi, \mathbf{q})$ at 20 different times in $[0, 1]$, resulting in 800 entries. The evaluation of the different neural network architectures is performed by comparing the accuracy of each architecture on this test dataset. Figure 3.4.1 shows the results in relative L^2 norm for our best candidate for each type of architecture. In view of these results, we choose to use the V-Net architecture in the rest of the paper, as it has a significant advantage on earlier times. We thus use 4 different neural networks, one for each of the output features π_{xx}, π_{xy}, q_x and q_y.

In Figure 3.4.2, we show examples of predictions on 10 entries from a single simulation. Note that each prediction is performed with input data from the kinetic simulation, and these predictions are not used in the fluid model yet.

3.4.2 Stability of the fluid model

In this section we consider the use of the neural network closure in the fluid model, and give numerical results regarding the necessary stabilization techniques described in section 3.3.4. The way we evaluate the efficacy of these techniques is by testing them on 70 test cases, that consist of simulations from 10 different initial conditions and 7 different Knudsen numbers $\varepsilon \in \{0.01, 0.02, 0.05, 0.10, 0.20, 0.50, 1.00\}$. These initial conditions are generated in the same way as for the training and test datasets. The results are shown Figure 3.4.3, and more details are given in the following paragraphs.

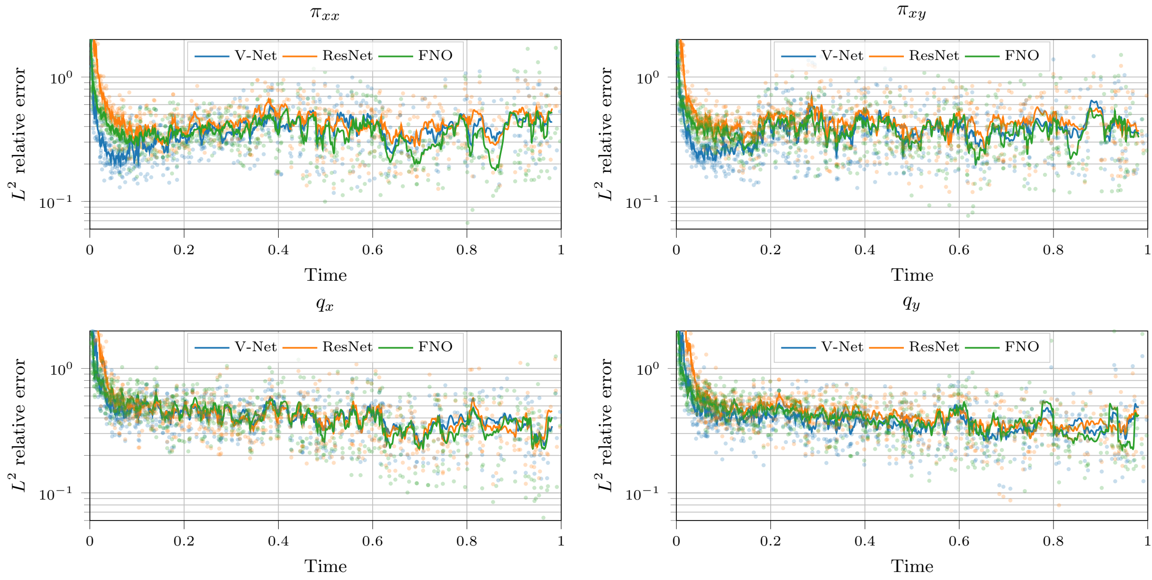


Figure 3.4.1 – Accuracy of different architectures on each of the 800 entries of the test dataset, plotted in function of time. The lines correspond to a moving average of 7 consecutive points.

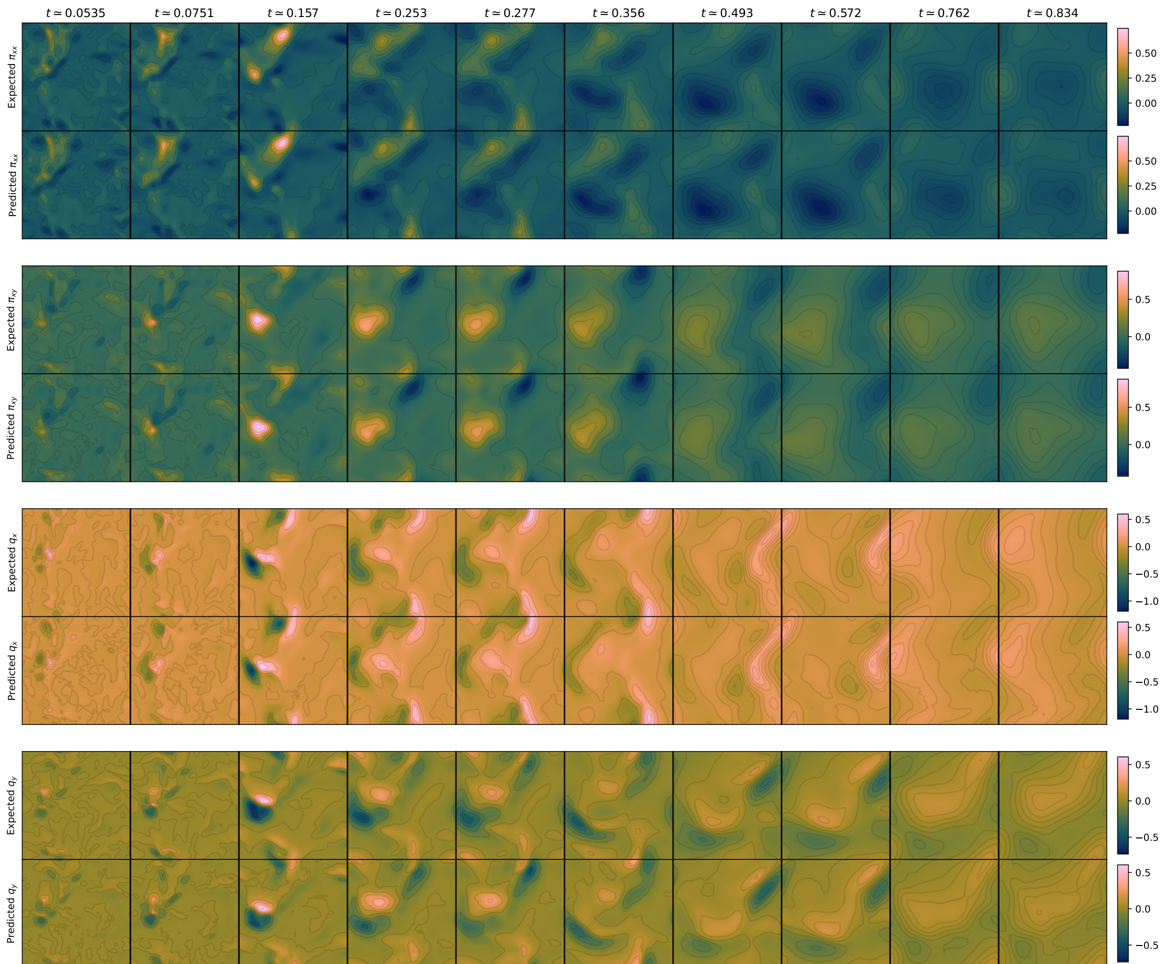


Figure 3.4.2 – Example of predictions for Π and $\mathbf{q}$ with the neural networks, compared to the expected outputs. The input data for these predictions come from a single kinetic simulation with Knudsen number $\varepsilon \simeq 0.13$, and are shown Figure 3.3.2.

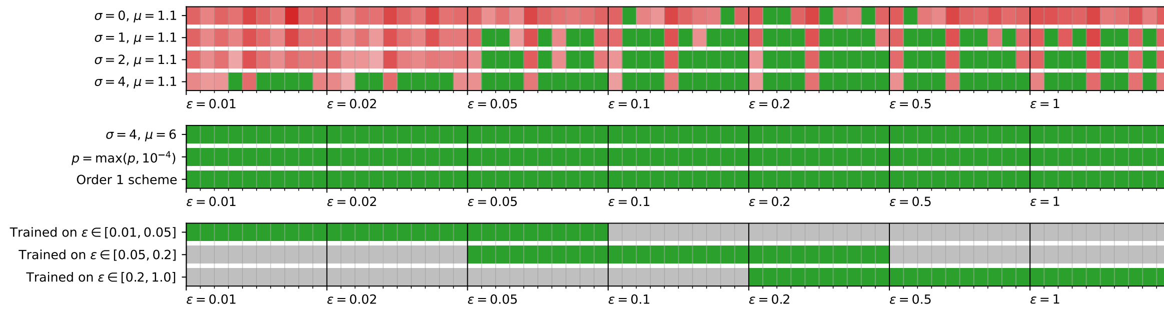


Figure 3.4.3 – Stability results for different models on 70 test cases. Green indicates stability, red unstability. σ is the standard deviation of the gaussian kernel for smoothing the closure, μ is the viscosity coefficient of the numerical flux, and p is the pressure.

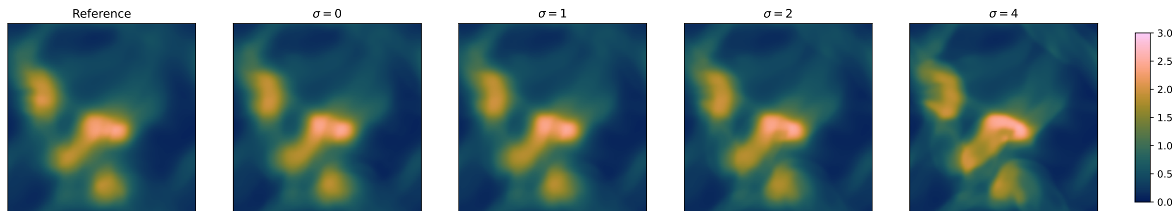


Figure 3.4.4 – Density ρ at time $t = 0.2$ of the solution computed with more or less smoothing applied to the closure. For this illustration an initial condition that do not lead to instabilities was picked, with Knudsen number $\varepsilon = 0.1$.

The first thing to note from the results in Figure 3.4.3 is the lack of stability of the model when no stabilization technique is applied: as can be seen in the first line of the Figure, only 7 out of 70 simulations are carried to term, while the others end prematurly due to negative pressures. In particular, not a single simulation is stable for the three smallest Knudsen numbers and for the biggest one.

The first attempt at stabilizing the model relies on the smoothing of the neural networks' outputs. The improvement is significant as the stability goes from 10% to 43% with little smoothing ($\sigma = 2$), and up to 70% with more smoothing ($\sigma = 4$). Since smoothing Π and $\mathbf{q}$ tends to bring them closer to zero, the resulting solution is more Euler-like as σ increases (see Figure 3.4.4), which could explain the better performance of the smoothed closure for small Knudsen numbers ε.

As a complement to the smoothing, the increase of the viscosity coefficient allows to get up to 100% stability on our 70 test cases, when this coefficient is set to 6 instead of 1.1. As opposed to smoothing the closure, increasing the viscosity makes the solution smoother, as illustrated Figure 3.4.5.

Other methods that allow to get 100% stability include the use of a scheme of order 1, provided that a little smoothing is added ($\sigma = 2$), and preventing the pressure from dropping below 10^{-4}. Obviously all these techniques have an impact on the quality of the resulting solution, which we discuss in the next section.

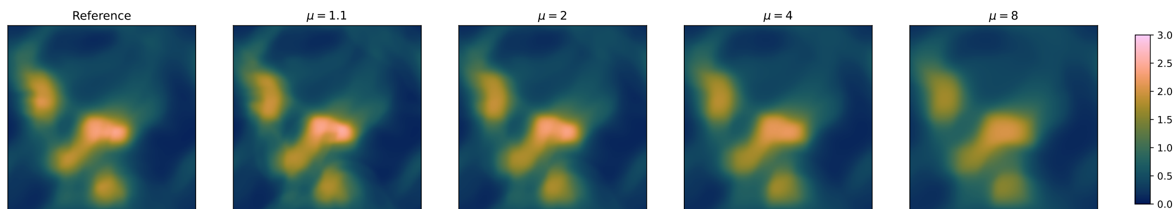


Figure 3.4.5 – Density ρ at time $t = 0.2$ of the solution computed with more or less viscosity μ in the numerical flux, and smoothing with $\sigma = 2$. For this illustration an initial condition that do not lead to instabilities was picked, with Knudsen number $\varepsilon = 0.1$.

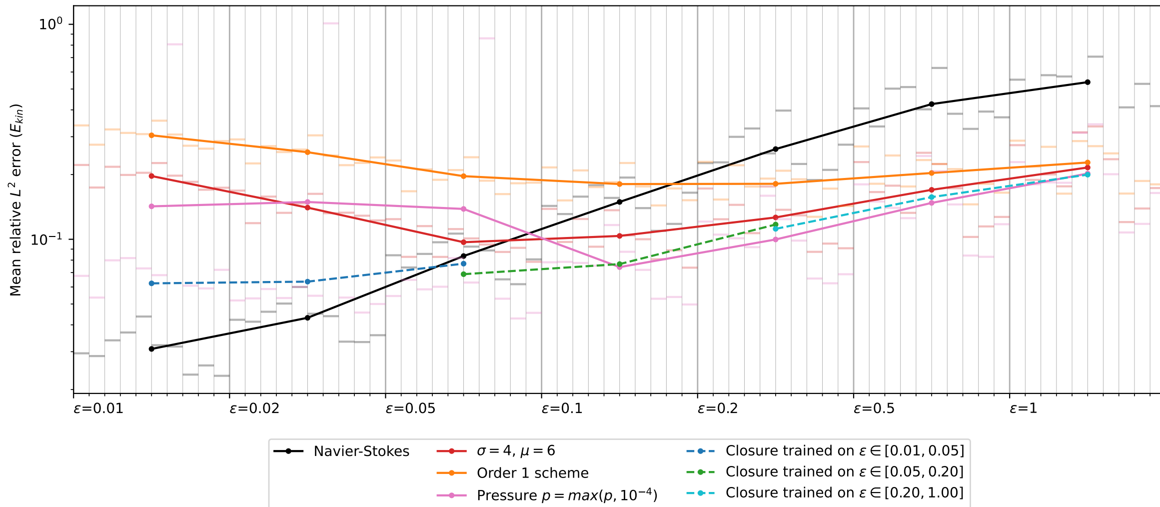


Figure 3.4.6 – Accuracy of different models on 70 test cases, consisting of simulations with 10 different initial conditions and 7 different Knudsen numbers. The lines show the average per Knudsen number. For clarity, the details are not shown for the closures trained on smaller ranges of Knudsen numbers.

Finally, by using neural networks trained on a smaller range of Knudsen numbers, stability can be achieved with less smoothing and less viscosity. On $[0.05, 0.2]$ and $[0.2, 1]$, we use $\sigma = 3$ and $\mu = 3$, parameters with which the closure trained on the whole dataset is not stable. The $[0.01, 0.05]$ range gets a particular treatment, obtained after empirical experimentations: Π is smoothed with $\sigma = 1$ and $\mathbf{q}$ with $\sigma = 4$, and we use a local viscosity that depends on the pressure p, in order to avoid adding unnecessary diffusion. These adjustments are not mandatory for stability, but lead to significant benefits regarding accuracy, as shown in the next section.

An important remark on these results is that the parameters for these stabilization techniques are all resolution dependant. In particular, when increasing the resolution to 200×200 instead of 100×100, none of the method given above provide stability.

3.4.3 Accuracy of the fluid model

In order to measure the accuracy of the fluid model with neural network closure, we use as baseline the solution given by the fluid model with closure coming directly from the kinetic model. Because of numerical effects, this solution is not exactly the same as the kinetic solution. However, for small Knudsen numbers and at a given resolution, it may actually be closer to the exact solution.

Figure 3.4.6 shows the relative error in norm L^2, averaged over time, on the kinetic energy

$$E_{\text{kin}} = \tfrac{1}{2}\rho\|\mathbf{u}\|^2,$$

for the stable models introduced in the previous section. As can be seen in this Figure, the neural network closure with smoothing and increased viscosity only outperfoms the Navier-Stokes closure for Knudsen numbers above 0.1. Using a scheme of order 1 lowers the accuracy of the model significantly, because it adds too much diffusion, as shown Figure 3.4.7. Preventing the pressure from going below 10^{-4} mostly gives good results, better than Navier-Stokes even for $\varepsilon = 0.05$, but with a risk of important failure: Figure 3.4.8 shows an example of such failure, which corresponds to the outlier that can be seen in Figure 3.4.6 for $\varepsilon = 0.02$. Finally, training the neural networks on smaller ranges of Knudsen numbers allows to get lower errors. This improvement is mostly due to the lower requirement for stabilization, in particular for the smallest Knudsen numbers.

3.5 Conclusion

This work has been devoted to the construction of a data-driven closure for the Boltzmann equation in two dimensions, focused on transitional regimes of collisions with Knudsen numbers between 0.01

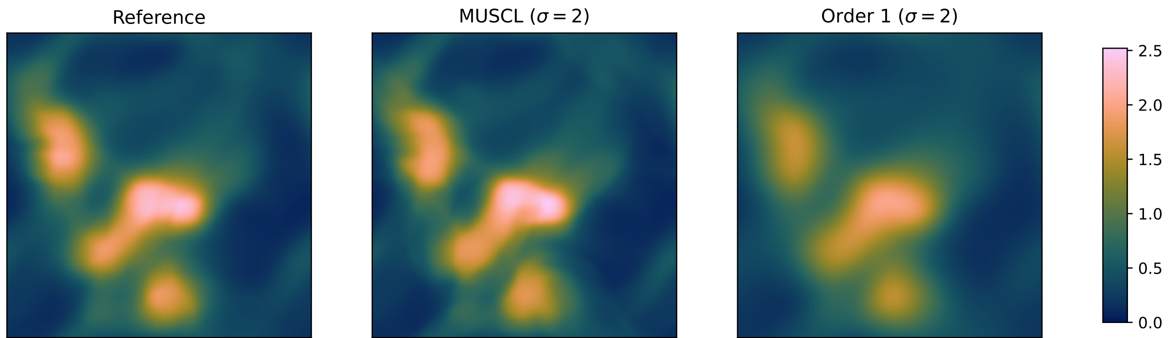


Figure 3.4.7 – Example of solutions (density ρ) at $t = 0.2$, with Knudsen number $\varepsilon = 0.1$, showing the excessive diffusion of the order 1 scheme compared to the order 2 MUSCL scheme.

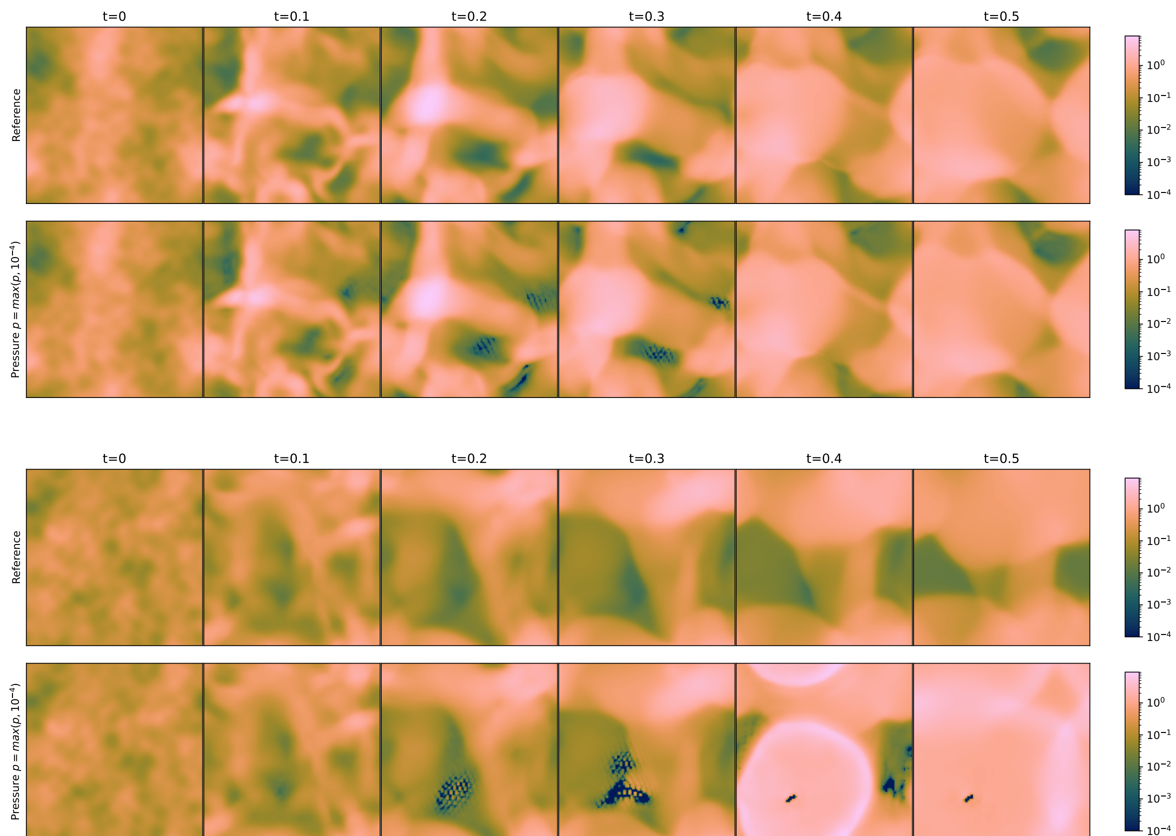


Figure 3.4.8 – Example of solutions (pressure p) that can be obtained using the network closure and stabilizing by preventing the pressure from dropping below 10^{-4}. Top: successful simulation, with the pressure going back above 10^{-4} after some time, matching the reference solution. Bottom: example of failure where the method produces a huge perturbation.

and 1. To this end, we have used convolutional neural networks for their performance with images, since we are interested in similar data, namely solutions of partial differential equations discretized on uniform grids. We have first generated a training and a testing datasets, from kinetic simulations with random initial conditions and Knudsen numbers. We have then trained and evaluated neural networks with different architectures on these datasets, allowing us to pick the most promising one: a V-Net architecture with about 90k weights. The final step has been the integration of the neural network based closure to the numerical scheme for the fluid model, which proved to be more challenging than in our previous work in one dimension. We have thus been forced to combine multiple stabilization techniques: smoothing of the closure, increased viscosity coefficient in the numerical flux, and slicing of the Knudsen numbers' interval into three smaller intervals. We have provided numerical results to account for the performance of our method, both in terms of stability and accuracy, demonstrating a lower error than the Navier-Stokes closure for Knudsen numbers above 0.05.

The main limit of our work is the complicated relationship between the closure and the numerical scheme with regard to stability. Although we ended up with a setup that seems to be working, it comes with multiple drawbacks. First, the tweaking of the stabilization techniques is empirical, which can be quite tedious and, above all, offers no guarantee of stability. Secondly, both the smoothing of the closure and the increase of the viscosity coefficient tend to lower the accuracy of the model, wether by distorting the closure or by adding diffusion to the numerical scheme. And last but not least, none of these techniques generalizes to different resolutions. In our previous work in one dimension, although the neural network was trained on a given resolution, it was able to perform well with other resolutions provided a resampling of its input to the training resolution. That is not the case here: even with this resampling, the model is no longer stable when refining the discretization. Thus a new effort for adapting the stabilization techniques would be required for each new resolution, and would likely result in more deterioration of the closure and the numerical scheme, defeating the purpose of the finer grid.

On a positive note, the use of the fluid model with a neural network based closure did prove less expensive than kinetic simulations, contrary to what we obtained in one dimension. For the models with reasonable viscosity coefficients μ, such as the models using closures trained on $\varepsilon \in [0.05, 0.2]$ or $\varepsilon \in [0.2, 1]$, the simulations are sped up by a factor 10. For models using more viscosity, such as the model with $\sigma = 4$ and $\mu = 6$, the CFL condition imposes a smaller timestep, decreasing the speedup factor to 2. These results were obtained using the same GPU to perform both the kinetic and fluid simulations, however the performance of the kinetic model may have been limited by the memory of said GPU.

In future work, different approaches can be investigated to adress the issues mentioned above. Neural networks based on Fourier neural operators (FNOs) may prove more robust with regard to changes in resolution, provided that the stability issues are solved at the training resolution. To this end, another algorithm could be used to train the data-driven closure: instead of measuring the loss by comparing the output of the neural network to the expected $(\Pi, \mathbf{q})$, we could compare directly the solution obtained with the neural network $(\rho, \mathbf{u}, p)$ to the expected solution, on a given number of timesteps. This way the neural network based closure would be integrated to the numerical scheme from the start, making stability issues an integral part of the training. This method would also have the added benefit of giving access to the resulting pressure during the training, making easy the addition of a penalization term for low pressures responsible for instabilities.

Acknowledgements The Scientific colour map batlow (Crameri 2018) is used in this study to prevent visual distortion of the data and exclusion of readers with colourvision deficiencies (Crameri et al., 2020).

Bibliography

[1] Andrea Beck, David Flad, and Claus-Dieter Munz. "Deep neural networks for data-driven LES closure models". en. In: *Journal of Computational Physics* 398 (Dec. 2019), p. 108910. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.108910. URL: https://www.sciencedirect.com/science/article/pii/S0021999119306151.

[2] Léo Bois et al. "A neural network closure for the Euler-Poisson system based on kinetic simulations". en. In: *Kinetic & Related Models* 15.1 (2022), p. 49. DOI: 10.3934/krm.2021044. URL: https://www.aimsciences.org/article/doi/10.3934/krm.2021044.

[3] D. Burnett. "The Distribution of Velocities in a Slightly Non-Uniform Gas". en. In: *Proceedings of the London Mathematical Society* s2-39.1 (1935), pp. 385–430. ISSN: 1460-244X. DOI: 10.1112/plms/s2-39.1.385. URL: https://onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-39.1.385.

[4] Mario De Florio et al. "Physics-Informed Neural Networks for rarefied-gas dynamics: Poiseuille flow in the BGK approximation". en. In: *Zeitschrift für angewandte Mathematik und Physik* 73.3 (May 2022), p. 126. ISSN: 1420-9039. DOI: 10.1007/s00033-022-01767-z. URL: https://doi.org/10.1007/s00033-022-01767-z.

[5] Mario De Florio et al. "Physics-informed neural networks for rarefied-gas dynamics: Thermal creep flow in the Bhatnagar–Gross–Krook approximation". In: *Physics of Fluids* 33.4 (Apr. 2021), p. 047110. ISSN: 1070-6631. DOI: 10.1063/5.0046181. URL: https://doi.org/10.1063/5.0046181.

[6] Harold Grad. "On the kinetic theory of rarefied gases". en. In: *Communications on Pure and Applied Mathematics* 2.4 (1949), pp. 331–407. ISSN: 1097-0312. DOI: 10.1002/cpa.3160020403. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160020403.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]* (Dec. 2015). URL: http://arxiv.org/abs/1512.03385.

[8] Marius Kurz and Andrea Beck. "A machine learning framework for LES closure terms". In: *arXiv:2010.03030 [physics]* (Oct. 2020). URL: http://arxiv.org/abs/2010.03030.

[9] C. David Levermore. "Moment closure hierarchies for kinetic theories". en. In: *Journal of Statistical Physics* 83.5 (June 1996), pp. 1021–1065. ISSN: 1572-9613. DOI: 10.1007/BF02179552. URL: https://doi.org/10.1007/BF02179552.

[10] R. Li, E. Lee, and T. Luo. "Physics-informed neural networks for solving multiscale mode-resolved phonon Boltzmann transport equation". en. In: *Materials Today Physics* 19 (July 2021), p. 100429. ISSN: 2542-5293. DOI: 10.1016/j.mtphys.2021.100429. URL: https://www.sciencedirect.com/science/article/pii/S2542529321000900.

[11] Ruiyang Li, Eungkyu Lee, and Tengfei Luo. "Physics-Informed Deep Learning for Solving Coupled Electron and Phonon Boltzmann Transport Equations". In: *Physical Review Applied* 19.6 (June 2023). Publisher: American Physical Society, p. 064049. DOI: 10.1103/PhysRevApplied.19.064049. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.19.064049.

[12] Zongyi Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. May 2021. DOI: 10.48550/arXiv.2010.08895. URL: http://arxiv.org/abs/2010.08895.

[13] Qin Lou, Xuhui Meng, and George Em Karniadakis. “Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation”. en. In: *Journal of Computational Physics* 447 (Dec. 2021), p. 110676. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2021.110676. URL: https://www.sciencedirect.com/science/article/pii/S0021999121005714.

[14] Chenhao Ma et al. “Machine Learning Surrogate Models for Landau Fluid Closure”. In: *Physics of Plasmas* 27.4 (Apr. 2020), p. 042502. ISSN: 1070-664X, 1089-7674. DOI: 10/gjt9ck. URL: http://arxiv.org/abs/1909.11509.

[15] Romit Maulik et al. “Neural network representability of fully ionized plasma fluid model closures”. In: *Physics of Plasmas* 27.7 (July 2020), p. 072106. ISSN: 1070-664X, 1089-7674. DOI: 10/gjt9cj. URL: http://arxiv.org/abs/2002.04106.

[16] Sean T. Miller et al. “Neural-network based collision operators for the Boltzmann equation”. en. In: *Journal of Computational Physics* 470 (Dec. 2022), p. 111541. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.111541. URL: https://www.sciencedirect.com/science/article/pii/S0021999122006039.

[17] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation”. In: *arXiv:1606.04797 [cs]* (June 2016). URL: http://arxiv.org/abs/1606.04797.

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *arXiv:1505.04597 [cs]* (May 2015). URL: http://arxiv.org/abs/1505.04597.

[19] Steffen Schotthöfer et al. *Neural network-based, structure-preserving entropy closures for the Boltzmann moment system.* Jan. 2022. DOI: 10.48550/arXiv.2201.10364. URL: http://arxiv.org/abs/2201.10364.

[20] Tianbai Xiao and Martin Frank. “Using neural networks to accelerate the solution of the Boltzmann equation”. en. In: *Journal of Computational Physics* 443 (Oct. 2021), p. 110521. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2021.110521. URL: https://www.sciencedirect.com/science/article/pii/S0021999121004162.

[21] Yajun Zhu, Chengwen Zhong, and Kun Xu. “Ray effect in rarefied flow simulation”. en. In: *Journal of Computational Physics* 422 (Dec. 2020), p. 109751. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109751. URL: https://www.sciencedirect.com/science/article/pii/S0021999120305258.

Appendix

3.A Numerical considerations

3.A.1 Kinetic model

For the discretization of the phase space, we use a uniform grid with 100×100 cells for the physical space and a special designed mesh with 22400 cells shown Figure 3.A.1 for the velocity space, as proposed in [21] to prevent ray effect from occuring. This discretization in velocity gives good results and seems appropriate to our problem, since we use Maxwellian initial conditions with mean velocity around the origin. For the numerical resolution of the Boltzmann equation, we use a time splitting method: at each iteration, the advection part $\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = 0$ is solved using a finite volume scheme of order 2 (MUSCL), and the relaxation part $\partial_t f = \frac{1}{\varepsilon}(M(f) - f)$ is solved using an implicit Euler scheme, that can be expressed explicitly. The advection part and the relaxation part are respectively local in velocity and local in space, so each can be parallelized for faster computations at the cost of higher memory requirements.

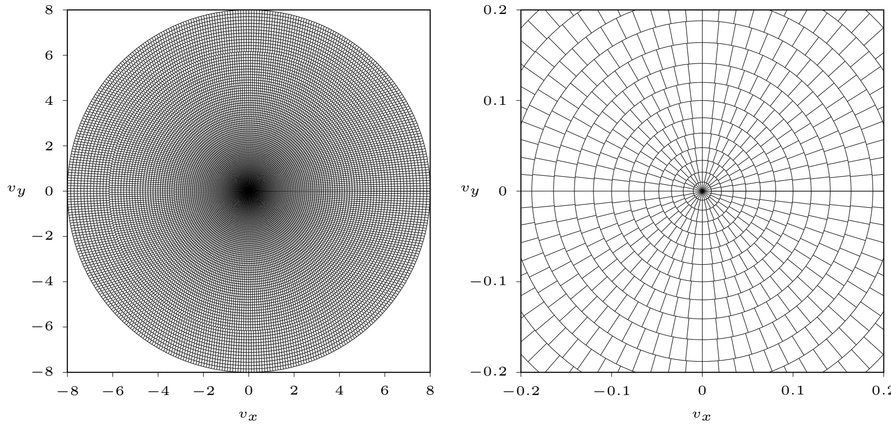


Figure 3.A.1 – Mesh used for the discretization of the velocity space (22400 cells). The values used in practice are the centers of the cells.

3.A.2 Fluid model

The fluid system can be written under hyperbolic form:

$$\partial_t U + \partial_x f_x(U) + \partial_y f_y(U) = 0,$$

with

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ p + \frac{1}{2}\rho\|\mathbf{u}\|^2 \end{bmatrix},$$

$$f_x(U) = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p + \pi_{xx} \\ \rho u_x u_y + \pi_{xy} \\ 2pu_x + \frac{1}{2}\rho\|\mathbf{u}\|^2 u_x + \pi_{xx}u_x + \pi_{xy}u_y + q_x \end{bmatrix},$$

$$f_y(U) = \begin{bmatrix} \rho u_y \\ \rho u_x u_y + \pi_{xy} \\ \rho u_y^2 + p - \pi_{xx} \\ 2pu_y + \frac{1}{2}\rho\|\mathbf{u}\|^2 u_y + \pi_{xy}u_x - \pi_{xx}u_y + q_y \end{bmatrix}.$$

We solve this system numerically using a finite volume method, with numerical flux

$$(F_x)^n_{i+\frac{1}{2},j} = \frac{1}{2}\left(f_x(U^n_{i,j}) + f_x(U^n_{i+1,j})\right) - \mu\frac{S^n_{i+\frac{1}{2},j}}{2}\left(U^n_{i+1,j} - U^n_{i,j}\right),$$
$$(F_y)^n_{i,j+\frac{1}{2}} = \frac{1}{2}\left(f_y(U^n_{i,j}) + f_y(U^n_{i,j+1})\right) - \mu\frac{S^n_{i,j+\frac{1}{2}}}{2}\left(U^n_{i,j+1} - U^n_{i,j}\right),$$

where

$$S^n_{i+\frac{1}{2},j} = \max\left(\|\mathbf{u}^n_{i,j}\| + \sqrt{2\frac{p^n_{i,j}}{\rho^n_{i,j}}}, \|\mathbf{u}^n_{i+1,j}\| + \sqrt{2\frac{p^n_{i+1,j}}{\rho^n_{i+1,j}}}\right),$$
$$S^n_{i,j+\frac{1}{2}} = \max\left(\|\mathbf{u}^n_{i,j}\| + \sqrt{2\frac{p^n_{i,j}}{\rho^n_{i,j}}}, \|\mathbf{u}^n_{i,j+1}\| + \sqrt{2\frac{p^n_{i,j+1}}{\rho^n_{i,j+1}}}\right).$$

The coefficient viscosity μ is set to 1.1 unless stated otherwise.

In practice, we use an order 2 MUSCL scheme with minmod limiter and corrections.

3.B Derivation of the fluid equations

The Boltzmann equation:

$$\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f = \frac{1}{\varepsilon}(M(f) - f)$$

Conservation of mass

By integrating the Boltzmann equation w.r.t the velocity $\mathbf{v}$, we get the equation for the conservation of mass:

$$\begin{aligned} & \partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f = \tfrac{1}{\varepsilon}\big(M(f) - f\big) \\ \implies\quad & \int \left(\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f\right) d\mathbf{v} = \int \tfrac{1}{\varepsilon}\big(M(f) - f\big)\, d\mathbf{v} \\ \implies\quad & \partial_t\left(\int f\, d\mathbf{v}\right) + \nabla_{\mathbf{x}}\cdot\left(\int f\mathbf{v}\, d\mathbf{v}\right) = \tfrac{1}{\varepsilon}\left(\int M(f)\, d\mathbf{v} - \int f\, d\mathbf{v}\right) \\ \implies\quad & \partial_t\rho + \nabla_{\mathbf{x}}\cdot(\rho\mathbf{u}) = \tfrac{1}{\varepsilon}\left(\rho - \rho\right) \\ \implies\quad & \partial_t\rho + \nabla_{\mathbf{x}}\cdot(\rho\mathbf{u}) = 0 \end{aligned}$$

3.B.1 Conservation of momentum

By multiplying by $\mathbf{v}$ the Boltzmann equation and integrating w.r.t $\mathbf{v}$, we get the equation for the conservation of momentum:

$$\begin{aligned}
&\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f = \tfrac{1}{\varepsilon}\big(M(f)-f\big)\\
\Longrightarrow\quad &\int \big(\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f\big)\mathbf{v}\,d\mathbf{v} = \int \tfrac{1}{\varepsilon}\big(M(f)-f\big)\mathbf{v}\,d\mathbf{v}\\
\Longrightarrow\quad &\partial_t\left(\int f\mathbf{v}\,d\mathbf{v}\right) + \nabla_{\mathbf{x}}\cdot\left(\int f\mathbf{v}\otimes\mathbf{v}\,d\mathbf{v}\right) = \tfrac{1}{\varepsilon}\left(\int M(f)\mathbf{v}\,d\mathbf{v} - \int f\mathbf{v}\,d\mathbf{v}\right)\\
\Longrightarrow\quad &\partial_t(\rho\mathbf{u}) + \nabla_{\mathbf{x}}\cdot(\rho\mathbf{u}\otimes\mathbf{u}+\mathbb{P}) = \tfrac{1}{\varepsilon}\left(\rho\mathbf{u}-\rho\mathbf{u}\right)\\
\Longrightarrow\quad &\partial_t(\rho\mathbf{u}) + \nabla_{\mathbf{x}}\cdot(\rho\mathbf{u}\otimes\mathbf{u}+pI+\Pi) = 0
\end{aligned}$$

$$\begin{aligned}
&\int f\mathbf{v}\otimes\mathbf{v}\,d\mathbf{v}\\
&= \int f\big((\mathbf{v}-\mathbf{u})\otimes(\mathbf{v}-\mathbf{u}) + \mathbf{v}\otimes\mathbf{u} + \mathbf{u}\otimes\mathbf{v} - \mathbf{u}\otimes\mathbf{u}\big)\,d\mathbf{v}\\
&= \int f(\mathbf{v}-\mathbf{u})\otimes(\mathbf{v}-\mathbf{u})\,d\mathbf{v} + \left(\int f\mathbf{v}\,d\mathbf{v}\right)\otimes\mathbf{u} + \mathbf{u}\otimes\left(\int f\mathbf{v}\,d\mathbf{v}\right) - \left(\int f\,d\mathbf{v}\right)\mathbf{u}\otimes\mathbf{u}\\
&= \mathbb{P} + \rho\mathbf{u}\otimes\mathbf{u} + \mathbf{u}\otimes\rho\mathbf{u} - \rho\mathbf{u}\otimes\mathbf{u}\\
&= \mathbb{P} + \rho\mathbf{u}\otimes\mathbf{u}
\end{aligned}$$

3.B.2 Conservation of energy

By multiplying by $\frac{|\mathbf{v}|^2}{2}$ the Boltzmann equation and integrating w.r.t $\mathbf{v}$, we get the equation for the conservation of energy:

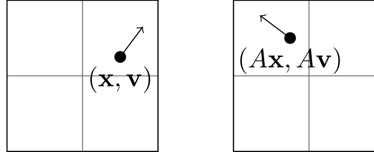
$$\begin{aligned}
&\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f = \tfrac{1}{\varepsilon}\big(M(f)-f\big)\\
\Longrightarrow\quad &\tfrac{1}{2}\int \big(\partial_t f + \mathbf{v}\cdot\nabla_{\mathbf{x}} f\big)|\mathbf{v}|^2\,d\mathbf{v} = \tfrac{1}{2}\int \tfrac{1}{\varepsilon}\big(M(f)-f\big)|\mathbf{v}|^2\,d\mathbf{v}\\
\Longrightarrow\quad &\partial_t\left(\tfrac{1}{2}\int f|\mathbf{v}|^2\,d\mathbf{v}\right) + \nabla_{\mathbf{x}}\cdot\left(\int f|\mathbf{v}|^2\mathbf{v}\,d\mathbf{v}\right) = \tfrac{1}{\varepsilon}\left(\int M(f)|\mathbf{v}|^2\,d\mathbf{v} - \int f|\mathbf{v}|^2\,d\mathbf{v}\right)\\
\Longrightarrow\quad &\partial_t\big(p+\tfrac{1}{2}\rho|\mathbf{u}|^2\big) + \nabla_{\mathbf{x}}\cdot\big(2p\mathbf{u}+\tfrac{1}{2}\rho|\mathbf{u}|^2\mathbf{u}+\Pi\mathbf{u}+\mathbf{q}\big) = 0
\end{aligned}$$

$$\begin{aligned}
&\tfrac{1}{2}\int f|\mathbf{v}|^2\,d\mathbf{v}\\
&= \tfrac{1}{2}\int f\big(|\mathbf{v}-\mathbf{u}|^2 + 2\mathbf{v}\cdot\mathbf{u} - |\mathbf{u}|^2\big)\,d\mathbf{v}\\
&= \tfrac{1}{2}\int f|\mathbf{v}-\mathbf{u}|^2\,d\mathbf{v} + \left(\int f\mathbf{v}\,d\mathbf{v}\right)\cdot\mathbf{u} - \tfrac{1}{2}\left(\int f\,d\mathbf{v}\right)|\mathbf{u}|^2\\
&= p + \rho\mathbf{u}\cdot\mathbf{u} - \tfrac{1}{2}\rho|\mathbf{u}|^2\\
&= p + \tfrac{1}{2}\rho|\mathbf{u}|^2
\end{aligned}$$

$$\begin{aligned}
&\tfrac{1}{2}\int f|\mathbf{v}|^2\mathbf{v}\,d\mathbf{v}\\
&= \tfrac{1}{2}\int f\big(|\mathbf{v}-\mathbf{u}|^2(\mathbf{v}-\mathbf{u}) + |\mathbf{v}-\mathbf{u}|^2\mathbf{u} - |\mathbf{u}|^2\mathbf{v} + 2(\mathbf{v}\cdot\mathbf{u})\mathbf{v}\big)\,d\mathbf{v}\\
&= \tfrac{1}{2}\int f|\mathbf{v}-\mathbf{u}|^2(\mathbf{v}-\mathbf{u})\,d\mathbf{v} + \left(\tfrac{1}{2}\int f|\mathbf{v}-\mathbf{u}|^2\,d\mathbf{v}\right)\mathbf{u} - \tfrac{1}{2}|\mathbf{u}|^2\left(\int f\mathbf{v}\,d\mathbf{v}\right) + \left(\int f\mathbf{v}\otimes\mathbf{v}\,d\mathbf{v}\right)\mathbf{u}\\
&= \mathbf{q} + p\mathbf{u} - \tfrac{1}{2}\rho|\mathbf{u}|^2\mathbf{u} + (\rho\mathbf{u}\otimes\mathbf{u} + pI + \Pi)\mathbf{u}\\
&= \mathbf{q} + p\mathbf{u} - \tfrac{1}{2}\rho|\mathbf{u}|^2\mathbf{u} + \rho|\mathbf{u}|^2\mathbf{u} + p\mathbf{u} + \Pi\mathbf{u}\\
&= 2p\mathbf{u} + \tfrac{1}{2}\rho|\mathbf{u}|^2\mathbf{u} + \Pi u + \mathbf{q}
\end{aligned}$$

3.C Data augmentation

We want to augment our dataset by "looking" at our simulations from different perspectives, or equivalently by applying different transformations to the underlying domain.



Let us consider an isometry $A \in \mathcal{O}(2,\mathbb{R})$. For a given solution f, we denote by f^A the function

$$f^A : (\mathbf{x},\mathbf{v},t) \mapsto f(A\mathbf{x},A\mathbf{v},t).$$

Let us express the moments of f^A in function of those of f. We will then show that f^A is a solution of the Boltzmann equation with initial condition f_0^A.

3.C.1 Density

$$\begin{aligned}
\rho^A(\mathbf{x},t) &= \int f^A(\mathbf{x},\mathbf{v},t)\,d\mathbf{v}\\
&= \int f(A\mathbf{x},A\mathbf{v},t)\,d\mathbf{v}\\
&= \int f(A\mathbf{x},\mathbf{v},t)\,d\mathbf{v} \quad \text{(change of variables)}\\
&= \rho(A\mathbf{x},t)
\end{aligned}$$

3.C.2 Mean velocity

$$\begin{aligned}
\rho^A(\mathbf{x},t)\mathbf{u}^A(\mathbf{x},t) &= \int f^A(\mathbf{x},\mathbf{v},t)\mathbf{v}\,d\mathbf{v}\\
&= \int f(A\mathbf{x},A\mathbf{v},t)\mathbf{v}\,d\mathbf{v}\\
&= \int f(A\mathbf{x},\mathbf{v},t)A^{-1}\mathbf{v}\,d\mathbf{v} \quad \text{(change of variables)}\\
&= A^{-1}\int f(A\mathbf{x},\mathbf{v},t)\mathbf{v}\,d\mathbf{v} \quad \text{(linearity of the integral)}\\
&= A^{-1}\rho(A\mathbf{x},t)\mathbf{u}(A\mathbf{x},t)
\end{aligned}$$

and thus

$$\mathbf{u}^A(\mathbf{x},t) = A^{-1}\mathbf{u}(A\mathbf{x},t)$$

3.C.3 Pressure tensor

$$\begin{aligned}
\mathbb{P}^A(\mathbf{x},t) &= \int f^A(\mathbf{x},\mathbf{v},t)(\mathbf{v}-\mathbf{u}^A(\mathbf{x},t)) \otimes (\mathbf{v}-\mathbf{u}^A(\mathbf{x},t))\, d\mathbf{v} \\
&= \int f(A\mathbf{x},A\mathbf{v},t)(\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t)) \otimes (\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t))\, d\mathbf{v} \\
&= \int f(A\mathbf{x},\mathbf{v},t)(A^{-1}\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t)) \otimes (A^{-1}\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t))\, d\mathbf{v} \quad \text{(change of variables)} \\
&= \int f(A\mathbf{x},\mathbf{v},t)(A^{-1}(\mathbf{v}-\mathbf{u}(A\mathbf{x},t))) \otimes (A^{-1}(\mathbf{v}-\mathbf{u}(A\mathbf{x},t)))\, d\mathbf{v} \\
&= \int f(A\mathbf{x},\mathbf{v},t)A^{-1}((\mathbf{v}-\mathbf{u}(A\mathbf{x},t)) \otimes (\mathbf{v}-\mathbf{u}(A\mathbf{x},t)))(A^{-1})^T\, d\mathbf{v} \\
&= A^{-1}\left(\int f(A\mathbf{x},\mathbf{v},t)((\mathbf{v}-\mathbf{u}(A\mathbf{x},t)) \otimes (\mathbf{v}-\mathbf{u}(A\mathbf{x},t)))\, d\mathbf{v}\right)(A^{-1})^T \quad \text{(linearity of the integral)} \\
&= A^{-1}\left(\mathbb{P}(A\mathbf{x},t)\right)(A^{-1})^T
\end{aligned}$$

and in particular

$$p^A(\mathbf{x},t) = \tfrac{1}{2}\mathrm{Tr}(\mathbb{P}^A(\mathbf{x},t)) = \tfrac{1}{2}\mathrm{Tr}(\mathbb{P}(A\mathbf{x},t)) = p(A\mathbf{x},t)$$

and

$$\Pi^A(\mathbf{x},t) = \mathbb{P}^A(\mathbf{x},t) - p^A(\mathbf{x},t)I = A^{-1}\Pi(A\mathbf{x},t)(A^{-1})^T$$

3.C.4 Heat flux

$$\begin{aligned}
\mathbf{q}^A(\mathbf{x},t) &= \tfrac{1}{2}\int f^A(\mathbf{x},\mathbf{v},t)|\mathbf{v}-\mathbf{u}^A(\mathbf{x},t)|^2(\mathbf{v}-\mathbf{u}^A(\mathbf{x},t))\, d\mathbf{v} \\
&= \tfrac{1}{2}\int f(A\mathbf{x},A\mathbf{v},t)|\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t)|^2(\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t))\, d\mathbf{v} \\
&= \tfrac{1}{2}\int f(A\mathbf{x},\mathbf{v},t)|A^{-1}\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t)|^2(A^{-1}\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t))\, d\mathbf{v} \quad \text{(change of variables)} \\
&= \tfrac{1}{2}\int f(A\mathbf{x},\mathbf{v},t)|A^{-1}(\mathbf{v}-\mathbf{u}(A\mathbf{x},t))|^2(A^{-1}(\mathbf{v}-\mathbf{u}(A\mathbf{x},t)))\, d\mathbf{v} \\
&= \tfrac{1}{2}\int f(A\mathbf{x},\mathbf{v},t)|(\mathbf{v}-\mathbf{u}(A\mathbf{x},t))|^2(A^{-1}(\mathbf{v}-\mathbf{u}(A\mathbf{x},t)))\, d\mathbf{v} \quad (A \text{ isometry}) \\
&= A^{-1}\tfrac{1}{2}\int f(A\mathbf{x},\mathbf{v},t)|(\mathbf{v}-\mathbf{u}(A\mathbf{x},t))|^2(\mathbf{v}-\mathbf{u}(A\mathbf{x},t))\, d\mathbf{v} \quad \text{(linearity of the integral)} \\
&= A^{-1}\mathbf{q}(A\mathbf{x},t)
\end{aligned}$$

3.C.5 f^A is solution to the Boltzmann equation

First let us note that from the previous equalities we get

$$
\begin{aligned}
M(f^A)(\mathbf{x},\mathbf{v},t) &= \frac{\rho^A(\mathbf{x},t)}{2\pi T^A(\mathbf{x},t)} \exp\left(-\frac{|\mathbf{v}-\mathbf{u}^A(\mathbf{x},t)|^2}{2T^A(\mathbf{x},t)}\right) \\
&= \frac{\rho(A\mathbf{x},t)}{2\pi T(A\mathbf{x},t)} \exp\left(-\frac{|\mathbf{v}-A^{-1}\mathbf{u}(A\mathbf{x},t)|^2}{2T(A\mathbf{x},t)}\right) \\
&= \frac{\rho(A\mathbf{x},t)}{2\pi T(A\mathbf{x},t)} \exp\left(-\frac{|A\mathbf{v}-\mathbf{u}(A\mathbf{x},t)|^2}{2T(A\mathbf{x},t)}\right) \quad (A \text{ isometry}) \\
&= M(f)(A\mathbf{x},A\mathbf{v},t)
\end{aligned}
$$

And thus:

$$
\begin{aligned}
\partial_t\left[f^A(\mathbf{x},\mathbf{v},t)\right] + \mathbf{v}\cdot\nabla_{\mathbf{x}}\left[f^A(\mathbf{x},\mathbf{v},t)\right] &= \partial_t\left[f(A\mathbf{x},A\mathbf{v},t)\right] + \mathbf{v}\cdot\nabla_{\mathbf{x}}\left[f(A\mathbf{x},A\mathbf{v},t)\right] \\
&= \left(\partial_t f\right)(A\mathbf{x},A\mathbf{v},t) + \mathbf{v}\cdot\left(A\left(\nabla_{\mathbf{x}} f\right)(A\mathbf{x},A\mathbf{v},t)\right) \\
&= \left(\partial_t f\right)(A\mathbf{x},A\mathbf{v},t) + A\mathbf{v}\cdot\left(\nabla_{\mathbf{x}} f\right)(A\mathbf{x},A\mathbf{v},t) \\
&= \tfrac{1}{\varepsilon}\left(M(f)(A\mathbf{x},A\mathbf{v},t) - f(A\mathbf{x},A\mathbf{v},t)\right) \\
&= \tfrac{1}{\varepsilon}\left(M(f^A)(\mathbf{x},\mathbf{v},t) - f^A(\mathbf{x},\mathbf{v},t)\right)
\end{aligned}
$$

which proves that f^A is solution of the Boltzmann equation.

3.C.6 Application to data augmentation

We consider the following transformations, that preserve the physical domain $[-1,1]\times[-1,1]$:

Identity	Rotation $\frac{\pi}{2}$	Rotation π	Rotation $\frac{3\pi}{2}$
$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}$	$\begin{pmatrix}-1 & 0\\ 0 & -1\end{pmatrix}$	$\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}$
Reflexion	Rotation $\frac{\pi}{2}$ + Reflexion	Rotation π + Reflexion	Rotation $\frac{3\pi}{2}$ + Reflexion
$\begin{pmatrix}-1 & 0\\ 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$	$\begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix}$

Here are the corresponding data to use:

Identity	Rotation $\frac{\pi}{2}$	Rotation π	Rotation $\frac{3\pi}{2}$
$\rho(x,y,t)$	$\rho(-y,x,t)$	$\rho(-x,-y,t)$	$\rho(y,-x,t)$
$u_x(x,y,t)$	$u_y(-y,x,t)$	$-u_x(-x,-y,t)$	$-u_y(y,-x,t)$
$u_y(x,y,t)$	$-u_x(-y,x,t)$	$-u_y(-x,-y,t)$	$u_x(y,-x,t)$
$p(x,y,t)$	$p(-y,x,t)$	$p(-x,-y,t)$	$p(y,-x,t)$
$\pi_{xx}(x,y,t)$	$-\pi_{xx}(-y,x,t)$	$\pi_{xx}(-x,-y,t)$	$-\pi_{xx}(y,-x,t)$
$\pi_{xy}(x,y,t)$	$-\pi_{xy}(-y,x,t)$	$\pi_{xy}(-x,-y,t)$	$-\pi_{xy}(y,-x,t)$
$q_x(x,y,t)$	$q_y(-y,x,t)$	$-q_x(-x,-y,t)$	$-q_y(y,-x,t)$
$q_y(x,y,t)$	$-q_x(-y,x,t)$	$-q_y(-x,-y,t)$	$q_x(y,-x,t)$
Reflexion	Rotation $\frac{\pi}{2}$ + Reflexion	Rotation π + Reflexion	Rotation $\frac{3\pi}{2}$ + Reflexion
$\rho(-x,y,t)$	$\rho(y,x,t)$	$\rho(x,-y,t)$	$\rho(-y,-x,t)$
$-u_x(-x,y,t)$	$u_y(y,x,t)$	$u_x(x,-y,t)$	$-u_y(-y,-x,t)$
$u_y(-x,y,t)$	$u_x(y,x,t)$	$-u_y(x,-y,t)$	$-u_x(-y,-x,t)$
$p(-x,y,t)$	$p(y,x,t)$	$p(x,-y,t)$	$p(-y,-x,t)$
$\pi_{xx}(-x,y,t)$	$-\pi_{xx}(y,x,t)$	$\pi_{xx}(x,-y,t)$	$-\pi_{xx}(-y,-x,t)$
$-\pi_{xy}(-x,y,t)$	$\pi_{xy}(y,x,t)$	$-\pi_{xy}(x,-y,t)$	$\pi_{xy}(-y,-x,t)$
$-q_x(-x,y,t)$	$q_y(y,x,t)$	$q_x(x,-y,t)$	$-q_y(-y,-x,t)$
$q_y(-x,y,t)$	$q_x(y,x,t)$	$-q_y(x,-y,t)$	$-q_x(-y,-x,t)$

3.D Smoothing implementation

In this work we perform such smoothing using a convolution with a gaussian kernel. This gaussian kernel is characterized by its standard deviation $\sigma\Delta x$, where Δx is the spatial step of the closure's discretization, supposed to be the same in the x and y direction. The gaussian kernel is discretized on $[-\lceil 3\sigma\rceil \Delta x, +\lceil 3\sigma\rceil \Delta x] \times [-\lceil 3\sigma\rceil \Delta x, +\lceil 3\sigma\rceil \Delta x]$ with spatial step Δx, so as to coincide with the discretization of the closure and to entail 3 standard deviations of the gaussian. Finally the kernel is normalized so that the convolution performs a weighted average. Thus the weights $w_{i,j}$ read as follows:

$$x_i = i\Delta x, \quad y_j = j\Delta x, \quad w_{i,j} = \frac{e^{\frac{-x_i^2-y_j^2}{2\sigma^2\Delta x^2}}}{\sum_{i,j} e^{\frac{-x_i^2-y_j^2}{2\sigma^2\Delta x^2}}}, \quad i,j \in \{-\lceil 3\sigma\rceil, ..., -1, 0, 1, ..., \lceil 3\sigma\rceil\}$$

or, equivalently,

$$x_i = i, \quad y_j = j, \quad w_{i,j} = \frac{e^{\frac{-x_i^2-y_j^2}{2\sigma^2}}}{\sum_{i,j} e^{\frac{-x_i^2-y_j^2}{2\sigma^2}}}, \quad i,j \in \{-\lceil 3\sigma\rceil, ..., -1, 0, 1, ..., \lceil 3\sigma\rceil\}$$

For the smoothed closure to be of the same size as the raw closure, the latter is applied a periodic padding (consistent with the periodic boundary conditions) of size $\lceil 3\sigma\rceil$ before the convolution.

Chapter 4

An optimal control deep learning method to design artificial viscosities for Discontinuous Galerkin schemes

This chapter was recently submitted for publication as an article, and a preprint is available online: Léo Bois, Emmanuel Franck, Laurent Navoret, and Vincent Vigon. "An optimal control deep learning method to design artificial viscosities for Discontinuous Galerkin schemes". 2023. hal-04213057 (https://arxiv.org/pdf/2309.11795.pdf)

Abstract

In this paper, we propose a method for constructing a neural network viscosity in order to reduce the non-physical oscillations generated by high-order Discontiuous Galerkin (DG) methods. To this end, the problem is reformulated as an optimal control problem for which the control is the viscosity function and the cost function involves comparison with a reference solution after several compositions of the scheme. The learning process is strongly based on gradient backpropagation tools. Numerical simulations show that the artificial viscosities constructed in this way are just as good or better than those used in the literature.

4.1 Introduction

In computational fluid dynamics, Discontinuous Galerkin (DG) methods can be used as an alternative to finite volumes (FV) methods when a high order of convergence is desired. Indeed, by using polynomials coupled with the weak form of the equation to approximate the solution, DG methods allow to reach arbitrarily high orders of convergence [7, 14]. However, when the solution exhibits shocks or strong gradients, these high order methods introduce non-physical oscillations, which can deteriorate the accuracy of the solution and lead to stability issues. Since shocks and strong gradients easily appear in non-linear conservative systems, even from continuous initial conditions, countermeasures are required to stabilize DG methods.

Classical approaches to reduce oscillations and stabilize DG methods are based on slope limiters, filtering techniques or artificial viscosity methods. Slope limiter methods were initially developed for FV methods and then adapted to DG schemes: they use a troubled-cell indicator to identify cells with oscillations and then define fluxes at the cells interfaces with second order polynomial approximation and total variations diminishing property [6, 5]. An alternative is to consider WENO (Weighted Essentially Non Oscillating) type reconstruction to take advantage of the full high-order approximation in the identified troubled cells and their neighbours [26, 17, 25]. Regarding filtering techniques, they involve applying a linear filter using the modal representation of the solution locally in each cell to smooth out the solution [13, 12]. Finally the artificial viscosity method consists in adding a non-linear viscous term to the equation, which makes the solution smoother, thus saving the DG methods from situations that they cannot handle correctly. The viscous term can then be tuned with a local

coefficient, which should only be activated in problematic areas and should vanish as the characteristic length of the mesh tends to zero.

In this paper, we will focus on artificial viscosity methods. A few different approaches have emerged to deal with different problems. For instance, the artificial viscosity can be function of the divergence of the velocity field [15], function of the modal decay of the solution in each cell [16] or function of the entropy production [10]. A comparative study of some models has been carried out in [24], and shows that these different models behave differently from each other, and that the most suitable model may depend on the test case considered. In addition, each of these models relies on some parameters that have to be adjusted empirically, making the process of finding the right viscosity coefficient even more difficult.

A more recent approach consists in exploiting the capabilities of neural networks in pattern recognition to design data-driven tools. In short, neural networks can be described as non-linear functions with many parameters —from thousands to millions— that can be adjusted using gradient descent in order to optimize a given criterion. Examples of this approach can be found in several related topics: neural networks are used as troubled-cell indicator for high order schemes in [18], as classifiers of functions' regularity to control oscillations in spectral methods in [19], or as predictor of the degree of reconstruction for MOOD algorithm in [3]. Last but not least, in [8] the authors design an artificial viscosity for DG methods using neural networks. In all these examples, the authors use *supervised learning*, where the criterion to optimize is an error between the output of the neural network and a given target. In [8] for instance, for each test case of the training dataset, the target is set to the artificial viscosity model (among a given set of options) that performs best for this specific test case. This method makes the neural network converge toward a kind of good interpolation between known models. However supervised learning in this way is not necessarily the best option, or may not even be possible in some other contexts. Indeed, an appropriate target is not always available; and when it is, as is the case in the example previously described, using it as a target will not allow the neural network to explore new designs.

In this paper we use a different way to train parameterized functions in numerical schemes, that is not bounded by these limitations. We train the neural network directly in the numerical scheme, and compare the resulting numerical solution to a target solution, instead of considering the output of the neural network itself. This approach relieves us from any prior expectation on what the output of the neural network should look like and only focuses on the result, while having the additional advantage to include effects of the neural network through many iterations of the scheme computing the gradient by automatic differentiation framework. We speak about "differentiable physic approach" [22]. This approach have been very recently used to learn discretization [1, 9].

This approach can be see as an optimal control approach, which we design a closed-loop control of the system to fit the reference solution. A comparable approach is reinforcement learning, which is equivalent to optimal control without using the temporal transition scheme, but only examples of transitions. To our knowledge, this approach has been used for the construction of limiters in [20] and for adjusting the weights in WENO schemes [23].

One of the main ingredients of this approach is the use of deep learning frameworks (like TensorFlow or PyTorch), not only for the implementation of the neural network, but extended to the implementation of the whole numerical scheme. Indeed, the optimization of the parameters is performed with a gradient descent algorithm, which requires the computation of the gradient of the error. Since the error involves the numerical solution produced through many iterations of the parameterized numerical scheme, all the computations need to be differentiated. By implementing the numerical scheme in a deep learning framework, the computation of this highly complex gradient can be fully automated, making the optimization algorithm fairly easy to code. However, the complexity of the gradient is in itself an obstacle to the proper functionning of the algorithm, both because of its high computational cost and because of potential gradient instability. In this paper, we propose an algorithm to adress both of these limitations, and provide results of this algorithm when applied to the design of a neural network viscosity for DG schemes.

The remaining of this paper is organized as follows. In section 4.2, we introduce a general framework for the approach we used in this work. In section 4.3, we describe in details our implementation of this framework to the design of an artificial viscosity for DG schemes in 1D. Finally, section 4.4 gives numerical results for the advection equation, Burgers' equation and Euler's equation, with considerations on the influence of some parameters.

4.2 An optimal control method for parameterized schemes

In this section we describe the method we use to construct an artificial viscosity. Since this method is not specific to this problem, we describe it in general terms, as a general framework to optimize parameters in a numerical scheme for partial differential equations. In our application, these parameters are the weights of a neural network designed to output the coefficient for the artificial viscosity.

4.2.1 Optimization problem

As an example, let us consider a general hyperbolic equation

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0,$$

with $\mathbf{U} : \mathbb{R}^d \times \mathbb{R}_+^* \to \mathbb{R}^s$ the vector of unknowns and $\mathbf{F} : \mathbb{R}^s \to \mathbb{R}^{s\times d}$ the flux of the equation. Let us consider a given discretization of space (finite volumes, DG, WENO schemes, etc.) and time (explicit Euler, Runge-Kutta, etc.), resulting in a numerical scheme of the form

$$U^{n+1} = S(U^n, \pi(U^n)) = S_\pi(U^n), \tag{4.1}$$

where U^n is the approximation of the solution $\mathbf{U}$ at time t^n, S_π is an iteration of the numerical scheme on one timestep, and π is a part of the numerical scheme that can be modified to serve as a control to the scheme. For instance, π could be a slope limiter for a finite volumes method, a process to compute the weights of a WENO scheme, or —as in this paper— an artificial viscosity coefficient for a discontinuous Galerkin method, among many other possibilities.

In order to express the problem as an optimization problem, we denote by $V_\pi^N(U^0)$ the quantity

$$V_\pi^N(U^0) = \mathcal{L}\big(S_\pi(U^0),\ S_\pi^2(U^0),\ \ldots,\ S_\pi^N(U^0)\big), \tag{4.2}$$

where S_π^n corresponds to n iterations of the scheme

$$S_\pi^n(U^0) = (\underbrace{S_\pi \circ \cdots \circ S_\pi}_{n \text{ times}})(U^0),$$

and $\mathcal{L}$ is a generic cost function. Thus $V_\pi^N(U^0)$ is a cost function depending on a discrete solution made of N successive iterations of the numerical scheme S_π. It can be for example an error committed by the scheme compared to a reference solution a penalization of the control.

Since there is usually little hope to find a control π that minimizes $V_\pi^N(U_0)$ for all possible initial conditions U_0, we focus on a specific distribution of initial conditions $\mathbb{P}$, and consider the following optimization problem:

$$\min_\pi \int V_\pi^N(U_0)\, d\mathbb{P}(U_0). \tag{4.3}$$

In order to find a solution to this optimization problem, we choose to parameterize π with a set of parameters θ, for instance by implementing π_θ as a neural network. The optimization problem thus becomes

$$\min_\theta J(\theta) = \min_\theta \int V_{\pi_\theta}^N(U_0)\, d\mathbb{P}(U_0). \tag{4.4}$$

This problem is similar to the optimization problem solved by policy gradient methods in reinforcement learning [21].

4.2.2 Gradient descent and back-propagation

Our approach to solve the optimization problem (4.4) is to use a mini-batch gradient descent algorithm, relying on automatic differentiation for the computation of the gradient. The gradient descent algorithm consists in starting from an arbitrary set of parameters, and iteratively improve it by performing updates of the form

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta),$$

or any other alternative, e.g with momentum, Adam, and so on. The *mini-batch* version of the algorithm consists in replacing the gradient $\nabla_\theta J(\theta)$ by an approximation using a Monte-Carlo method,

meaning that the integral over the distribution $\mathbb{P}$ of initial conditions is replaced by a sum over a sample $(U_1^0, ..., U_K^0)$ of $\mathbb{P}$:

$$\nabla_\theta J(\theta) \simeq \sum_{k=1}^{K} \nabla_\theta V_{\pi_\theta}^N(U_k^0).$$

In principle this approximation does not prevent the convergence of the algorithm while making it much faster. In our case, if the distribution $\mathbb{P}$ is infinite, such an approximation is even required for the computation of the gradient to be possible.

From here, the difficulty lies in the computation of $\nabla_\theta V_{\pi_\theta}^N(U^0)$ for a given U^0. Figure 4.3.2 shows the computational graph of this quantity, from which can be derived the following formulae:

$$\big(\nabla_\theta V_{\pi_\theta}^N\big) = \sum_{n=1}^{N} \big(\nabla_\theta S_{\pi_\theta}^n\big)\big(\nabla_{U^n}\mathcal{L}\big),$$

$$\big(\nabla_\theta S_{\pi_\theta}^{n+1}\big) = \big(\nabla_\theta(S_{\pi_\theta} \circ S_{\pi_\theta}^n)\big) = \big(\nabla_\theta S_{\pi_\theta}^n\big)\big(\nabla_U S_{\pi_\theta}\big) + \big(\nabla_\theta S_{\pi_\theta}\big),$$

where, for any function $g(x)$, the gradient $\nabla_x g$ refers to the transpose of its Jacobian matrix. Assuming that all these quantities are well defined, meaning that both $\mathcal{L}$ and S_{π_θ} are differentiable, we thus obtain a way to compute the gradient $\nabla_\theta V_{\pi_\theta}^N(U^0)$.

In practice, all these computations are done automatically. Indeed, in the same way that deep learning frameworks (e.g Tensorflow, Pytorch) allow to automatically compute the gradient of a neural network w.r.t its parameters using the *backpropagation* algorithm, the same frameworks can be used to compute the gradient of any parameterized numerical scheme S_{π_θ} w.r.t θ, provided that S_{π_θ} is implemented using the differentiable functions of the framework. More than that, the backpropagation algorithm can be applied to any number of iterations of the numerical scheme, and even to the complete computational graph for $V_{\pi_\theta}^N(U^0)$ shown in Figure 4.3.2. In particular, this method illustrates one way these deep learning frameworks can prove useful for optimization tasks in scientific computing.

Let us conclude this section with a few observations on this approach to solve problem (4.4). A first advantage of this optimization algorithm lies in the fact that it does not require any reference π, since the error is not computed on the output of π_θ directly, but instead on the numerical solution that stems from π_θ. A second advantage is that the optimized function $J(\theta)$ can take into account many iterations of the numerical scheme S_{π_θ}, thus including effects of the control π_θ that would go unnoticed on shorter time scales. In our application to an artificial viscosity, these long time effect would be the diffusion related to a too high viscosity, as opposed to the short term oscillations related to a too low viscosity. Finally, note that this method contrasts with classical reinforcement learning algorithms in that this method leverages our knowledge of the transition process between two successive states, here U^n and U^{n+1}. Indeed, classical reinforcement learning usually build implicitly (model free approaches) or explicitly (model based approaches) an approximation of this transition process by analyzing examples of transitions, whereas in our case the full knowledge of this transition process, ie of the numerical scheme, can be used to compute the gradient of $J(\theta)$ directly.

4.2.3 Optimization on sub-trajectories using the reference solutions

Let us mention two obstacles to the application of the method described above. The first one stems from the *depth* of the computational graph when the number of iterations N grows higher and higher. Indeed, as N increases, the computation of $\nabla_\theta V_{\pi_\theta}^N$ becomes not only more and more expensive, but also more and more subject to gradient instability issues, similarly to very deep neural networks. The second obstacle is that although the algorithm does not require a reference control π, it does rely on a function $\mathcal{L}$ that quantifies the error of a numerical solution, and which may not be easy to determine.

One way we have found to partially address both of these issues is to use a reference numerical scheme S_{ref}, accurate and robust, and the reference solutions $U_{\text{ref}}^1, ..., U_{\text{ref}}^N$ provided by it. First it helps with the measure of the error committed by U_θ, by providing an expected result to compare with. But we can also use the reference solutions to limit the number of iterations the gradient actually goes through to a given number $m < N$, by replacing problem (4.4) by:

$$\min_\theta J(\theta) = \int \sum_{n=0}^{N-m} V_\theta^m(U_{\text{ref}}^n)\, d\mathbb{P}(U_0) = \min_\theta \int \sum_{n=0}^{N-m} V_\theta^m(S_{\text{ref}}^n(U_0))\, d\mathbb{P}(U_0). \tag{4.5}$$

In this formulation of the problem, instead of minimizing the error on an entire trajectory with N iterations, we minimize the sum of the errors on all the sub-trajectories with m iterations, starting from a point in the reference solution. This process thus limits the size of the computational graph for $J(\theta)$, while still allowing the parameterized scheme S_{π_θ} to be trained on data at times arbitrarily far from $t = 0$, which would not be the case if we simply picked a small N. Note that for the computation of $\nabla_\theta J(\theta)$ with this new formulation, the sum over the sub-trajectories is also approximated by a Monte-Carlo method, similarly to the integral over the initial conditions. This approach does not allow to capture very long time effects of the control but only on medium size time sequences. For a number of applications such as the construction of limiters or viscosity this seems to be sufficient.

4.2.4 Algorithm

The whole method is described in Algorithm 1. Note that, as mentioned in the previous section, both the integral over the initial conditions and the sum over the sub-trajectories of length m in (4.5) are approximated by a Monte-Carlo method, resulting in a kind of double mini-batch gradient descent algorithm. Something not mentioned in this algorithm for the sake of simplicity, but very useful to track the progression of the training, is the computation of a *validation loss* at the end of each epoch, consisting in the evaluation of $J(\theta)$ on a set of sub-trajectories generated at the beginning of the training. Also note that in this algorithm, S_{π_θ} and S_{ref} could actually consist of several iterations of the corresponding numerical schemes, so that the actual timestep Δt satisfies some stability conditions. Equivalently, we could say that the error $\mathcal{L}(U^n, \cdots, U^{n+m})$ could be computed on a subset of instants, thus lowering the memory requirements for the storage of the reference solutions.

Algorithm 1: training algorithm

```
1  Start from a random set of parameters θ
2  for each episode do
3      Generate random initial conditions (U_1^0, ..., U_K^0) ~ P
4      Compute reference trajectories from U_k^0 up to S_ref^N(U_k^0) for all k ∈ {1, ..., K}
5      for each epoch do
6          Randomly select a set I of indices (k, n) ∈ {1, ..., K} × {0, ..., N−m}
7          Compute sub-trajectories from S_ref^n(U_k^0) up to S_πθ^m(S_ref^n(U_k^0)) for all (k, n) ∈ I
8          Compute J(θ) = Σ_{(k,n)∈I} V_πθ(S_ref^n(U_k^0))
9          Update parameters θ with ∇J(θ)
10     end
11 end
```

4.3 Design of an artificial viscosity for discontinuous Galerkin schemes

This section is dedicated to our application of the method previously described to the design of an artificial viscosity for discontinuous Galerkin schemes in one dimension. Sections 4.3.1 and 4.3.2 describe the problem and the key elements of the method, like the numerical scheme, the control π, the cost function $\mathcal{L}$. Then sections 4.3.3 to 5.2 give some details on the implementation.

4.3.1 Discontinuous Galerkin method and artificial viscosity

In this application, we are interested in using discontinuous Galerkin (DG) schemes to solve hyperbolic equations of the form

$$\partial_t \mathbf{U} + \partial_x \mathbf{F}(\mathbf{U}) = 0, \tag{4.6}$$

with $\mathbf{U} : \mathbb{R}_+ \times [x_{\min}, x_{\max}] \to \mathbb{R}^s$ the conservative variables, and $\mathbf{F} : \mathbb{R}^s \to \mathbb{R}^s$ the physical flux.

In order to discretize equation (4.6) with a discontinuous Galerkin method, we consider a spatial mesh of the interval $[x_{\min}, x_{\max}]$ made of n_x cells of equal length Δx, and introduce a basis of polynomials $(\phi_1, ..., \phi_p)$ of degree at most $p - 1$ on the reference interval $[-1, 1]$. Assuming that the

components of $\mathbf{U}$ are polynomials of degree at most $p-1$ on each cell, with no constraint of continuity at the interfaces of the cells, the i-th variable on the j-th cell can be written

$$\mathbf{U}_{i,j}(x,t) = \sum_{k=1}^{p} U_{i,j,k}(t)\phi_k(\hat{x}), \quad 1 \leq i \leq s, 1 \leq j \leq n_x,$$

involving the change of variable to the reference interval: $\hat{x} = -1+2\left((x-x_{\min}) \mod \Delta x\right)/\Delta x \in [-1,1]$. Assuming that $\mathbf{F}(\mathbf{U})$ are also polynomials of degree at most $p-1$ on each cell, it has a similar decomposition with coefficients $(F_{i,j,k})$. Then, integrating (4.6) against each of the ϕ_k leads to the following semi-discrete weak formulation:

$$\frac{dU}{dt}M + (F^\star - FS) = 0, \tag{4.7}$$

where $M = \left(\int_{-1}^{1} \phi_k\phi_\ell\right)_{k,\ell}$, $S = \left(\int_{-1}^{1} \phi_k\partial_x\phi_\ell\right)_{k,\ell}$, and $F^\star$ involves the estimated values at the interfaces of the cells, using a local Lax-Friedrichs flux. Here, the product $\frac{dU}{dt}M$ is to be understood as

$$\left(\frac{dU}{dt}M\right)_{i,j,k} = \sum_{\ell} \frac{dU_{i,j,\ell}}{dt}M_{\ell,k}.$$

Finally, a Runge-Kutta method is used for the time integration of (4.7). We refer to [14] for more details.

An important benefit of discontinuous Galerkin schemes is that they can be made to converge in $O(\Delta x^p)$ for any arbitrary order p, by using polynomials of high enough degree ($p-1$ in one dimension). However, when the solution exhibits strong gradients or shocks, high-order DG schemes produce oscillations as those shown in Figure 4.3.1, which can ruin the accuracy of the scheme and produce fatal instabilities (e.g negative pressure in the Euler equations). For this reason, a method that is sometimes used consists in adding an artificial viscosity term to the equation to solve, in order to smooth out the solution:

$$\partial_t\mathbf{U} + \partial_x\mathbf{F}(\mathbf{U}) = \partial_x(\mu\partial_x\mathbf{U}). \tag{4.8}$$

The artificial viscosity above depends on a coefficient $\mu = \mu(x,t) \in \mathbb{R}$ that can locally increase or decrease the amount of smoothing, and that is expected to vanish as the length Δx of the cells tends to zero to recover the original equation asymptotically. In practice, since the places where the viscosity is needed depend on the solution, the viscosity coefficient is taken as a function of $\mathbf{U}$:

$$\mu = \pi(\mathbf{U}).$$

Denoting $\mathbf{G} = \mu\partial_x\mathbf{U} = \pi(\mathbf{U})\partial_x\mathbf{U}$ and $(G_{i,j,k})$ its coefficients in the discontinuous polynomial basis, the discontinuous Galerkin scheme now reads:

$$\begin{aligned} &GM = \pi(U)\,(U^\star - US), \\ &\frac{dU}{dt}M + \big((F^\star - FS) - (G^\star - GS)\big) = 0. \end{aligned}$$

where $U^\star$ and $G^\star$ involve the estimated values at the cells interfaces using a centered numerical flux. After time discretization, still with a Runge-Kutta method, we have thus completely defined the numerical scheme $U^{n+1} = S_\pi(U^n)$.

Function π is the one that we intend to design with the use of the method described in the previous section. As discussed in the introduction, some models for π can already be found in the literature, and [24] compare some of them. In the result section, we compare our own viscosity to two of these models, referred to as the derivative-based (DB) and highest modal decay (MDH) models respectively, briefly described in appendix 4.A.

4.3.2 Definition of the cost function

To design the cost function $\mathcal{L}$ used to determine the control π, we compare the associated numerical solution $U^1, \cdots, U^N$ (or of a sub-trajectory $U^n, \cdots, U^{n+m}$) to a reference solution U_{ref} by summing a

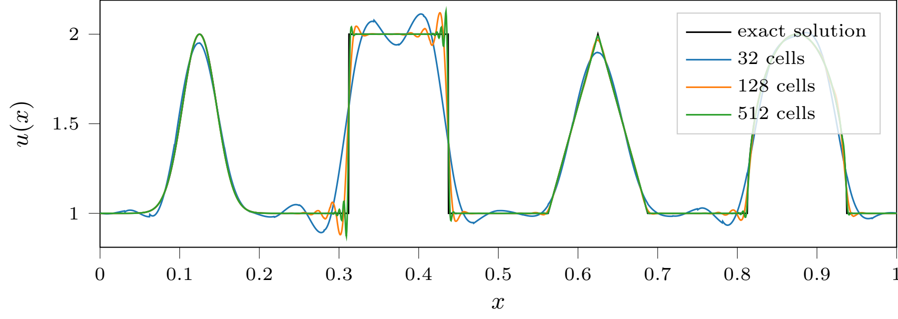


Figure 4.3.1 – Example of oscillations with discontinuous Galerkin schemes. Linear advection with periodic boundary conditions, solutions after one period. All solutions were obtained using a DG scheme of order 4.

local-in-time cost function C over iterations:

$$\mathcal{L}(U^1, \dots, U^N) = \sum_{n=1}^{N} C(U^n, U^n_{\text{ref}}).$$

For the reference solution, we use the numerical solution of a second-order MUSCL scheme on a fine grid, which ensures that the reference solution is both accurate and oscillation-free. The local-in-time cost function C, also concerned with both the accuracy of the solution and the presence of oscillations, is taken as a combination of three terms:

$$C(U^n, U^n_{\text{ref}}) = \omega_{\text{osc}}\, C_{\text{osc}}(U^n, U^n_{\text{ref}}) + \omega_{\text{acc}}\, C_{\text{acc}}(U^n, U^n_{\text{ref}}) + \omega_{\text{visc}}\, C_{\text{vis}}(U^n).$$

For simplicity, we give below the expression of each term in the scalar case. In case of a system, we simply take the average cost.

The aim of the first term is to detect the numerical oscillations. After some testing, we have obtained interesting results with the following $W^{2,1}$ semi-norm:

$$C_{\text{osc}}(U^n, U^n_{\text{ref}}) = \Delta x_{\text{ref}} \sum_i \left\| D_{xx}(\Pi_{\text{ref}}(U^n))_i - D_{xx}(U^n_{\text{ref}})_i \right\|_1 ,$$

where, for any approximate quantity $\mathbf{V}$, $\mathbf{V}_i$ refers to its value in cell i, $\Pi_{\text{ref}}(U^n)$ is the projection of the piecewise polynomial solution of the DG scheme on the fine mesh of the reference FV scheme, and $D_{xx}(U)_i = \frac{1}{\Delta x^2_{\text{ref}}}(U_{i-1} - 2U_i + U_{i+1})$ a finite-difference second derivative. One can obviously use other measures of oscillations or costs penalizing positivity losses or violations of the local maximum principle.

The second term measures the accuracy of the scheme and is given by the discrete L^1 norm of the difference between U^i and U^i_{ref}:

$$C_{\text{acc}}(U^n, U^n_{\text{ref}}) = \Delta x_{\text{ref}} \sum_i \left\| \Pi_{\text{ref}}(U^n)_i - (U^n_{\text{ref}})_i \right\|_1 ,$$

We compare the two solutions on the fine grid in order to highlight the oscillations.

Finally, since the artificial viscosity is a non-physical process, it is natural to look for the smallest viscosity which still allows to kill the oscillations. To do so, we use as the third term an L^2 penalisation:

$$C_{\text{vis}}(U^n) = \left\| \pi_\theta(U^n) \right\|_2^2 ,$$

with the norm computed directly from the piecewise polynomial viscosity. This cost is standard in optimal control problems.

Finally, a good starting point for the weights ω_{osc}, ω_{acc} and ω_{visc} could be such that all three terms contribute about as much to the overall error, but further empirical tweaking is necessary to get to the best compromise between diffusion and oscillations, as illustrated in the result section.

4.3.3 Neural network viscosity function

To apply Algorithm 1, it remains to define the neural network used for the viscosity function $\pi_\theta(U)$ and how it is used in the scheme S_π.

Neural network architecture. In this work we use a residual neural networks (ResNet) as introduced in [11], with adequate padding and no pooling so that the size of the output is the same as the input. This is a standard architecture for deep convolutionnal neural network. The hyper-parameters of this architecture are its *depth*, i.e. the number of blocks, its *width*, i.e. the number of filters per convolution, and the *kernel size* of these convolutions. We got good results with a very small version of it depicted in Figure 4.3.5: one block, width 16 and kernel size 3, for a total of about 2000 trainable parameters. We use the rectified linear unit (ReLU) activation function, except for the last layer that uses the softplus activation function. Also the last layer is initialized with kernel zero and constant bias -3 so that the initial output of the neural network is a constant vector with value $\text{softplus}(-3) \simeq 0.02$. The purpose of this initialization is to start the training with a reasonable viscosity that makes the numerical scheme stable.

Pre-processing and post-processing The raw input for the neural network is the approximated solution U^n at a given time, which comes as a tensor of values at each quadrature point of each cell. The cells are of equal length but the quadrature points are not uniformly distributed across the cells, which results in an overall non uniform discretization of the solution. Since convolutional neural networks –as the one we use– are not adapted to non uniform discretization, the input needs to be encoded in some way before being fed to the neural network. We opted for a concatenation between the value of the solution at the quadrature point and the relative position of the said quadrature point in the cell, in the form of a one-hot encoding: the first quadrature point of the cell is mapped to the vector $(1, 0, ..., 0) \in \mathbb{R}^p$, the second to $(0, 1, 0, ..., 0) \in \mathbb{R}^p$, and so on, p being the number of quadrature points per cell. Figure 4.3.4 gives an example of this encoding on a single variable.

Notably, the input of the neural network does not include information about the resolution of the solution, and therefore the artificial viscosity produced by the neural network is only adapted to the resolution the neural network has been trained with. In order to use the neural network with different resolution, we multiply the ouptut by a scaling factor, the same way it is done in [8]. This scaling factor s is constant across each cell and involves the size of the cell Δx as well as the jumps of the solution at its interfaces $[\![U]\!]_L$ and $[\![U]\!]_R$:

$$s = \min\{\Delta x, \max\{|[\![U]\!]_L|, |[\![U]\!]_R|\}\}$$

Also, this scaling helps the artificial viscosity getting closer to zero where the solution is smooth, which prevents unnecessary diffusion. Figure 4.3.4 depicts the whole process for the computation of the viscosity on a fictive cell.

Integration in the numerical scheme Since the evaluation of the neural network is relatively expensive compared to the rest of the numerical scheme, we choose to compute the artificial viscosity only once at the beginning of the timestep, as illustrated in Figure 4.3.3. Thus, we do not update its value at each stage of the Runge-Kutta method. We found that this simplification allowed faster computing with no perceptible loss of accuracy.

4.3.4 Training data

In this work we try and learn from initial conditions that have a general form, expressed as partial Fourier series:

$$U^0 : x \in [0, 1] \mapsto \sum_{i=0}^{20} \tfrac{a_n}{n} \cos(2\pi n x) + \tfrac{b_n}{n} \sin(2\pi n x), \tag{4.9}$$

with coefficients a_n and b_n following a uniform distribution on $[-1, 1]^s$. Of course, it is possible to use other type of dataset without difficulties. For instance, for the Euler equations (see Section 4.4.3), we will use this kind of initialization on the primitive variables instead of the conservative ones. Positive initial conditions can be necessary for some variables: in this case, we subtract to the above functions their minima and add a small positive value $\varepsilon = 0.1$.

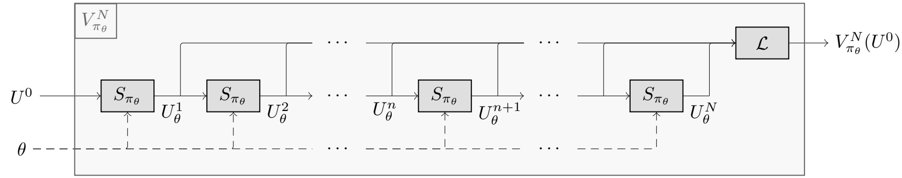


Figure 4.3.2 – View of the computational graph for V_{π_θ}.

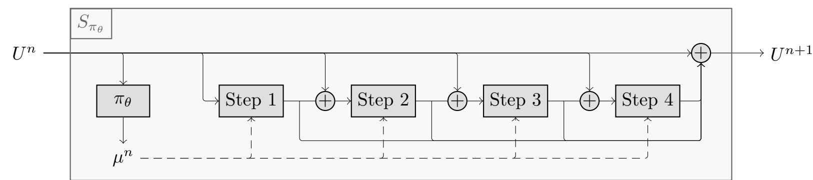


Figure 4.3.3 – Computations graph for one iteration of the Runge-Kutta 4 scheme. The artificial viscosity is computed only once and used at each step.

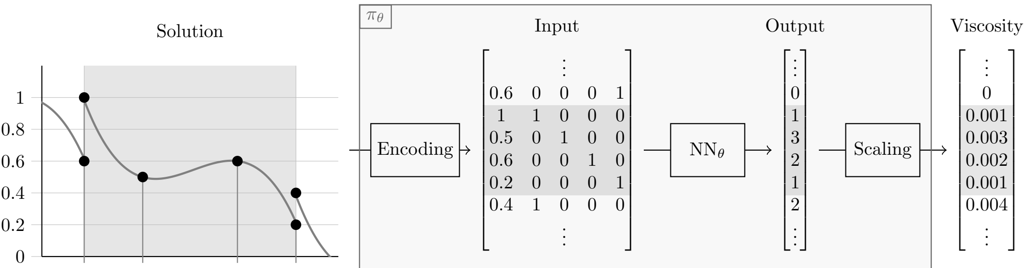


Figure 4.3.4 – Computing of the viscosity with a neural network. The input variable is encoded and given to the neural network, whose output is scaled to produce the artificial viscosity.

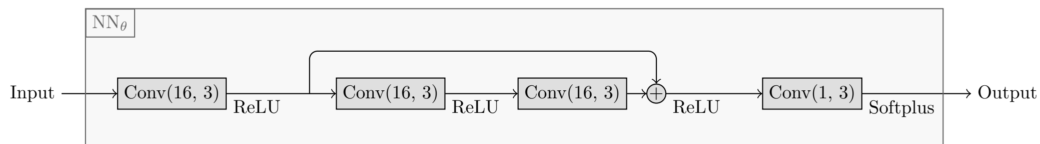


Figure 4.3.5 – The architecture we use to compute the artificial viscosity: a small ResNet with only one block. Conv(w, k) represents a 1D convolution with w filters of size k.

As the neural network is non-local, the learned viscosity may depend on the solutions generated during the training. In particular, if the network is trained with one particular equation, it may not perform as well on another equation. However, it would be possible to train the network directly on several equations, even if it has not been done in this work.

4.4 Numerical Results

In the following three sections we give numerical results for three different equations : the advection equation, Burgers' equation and Euler's system respectively. We give some details regarding the training and the influence of some parameters in the advection case, and then simply give the results for Burgers and Euler.

In all the numerical results presented below, we use the following parameters unless stated otherwise:

- DG scheme: order $p = 4$, Gauss-Lobatto quadrature points, 32 cells on $[0, 1]$, timestep $\Delta t = 1e - 5$, RK4 discretization in time,
- Reference FV scheme: order 2 (MUSCL), 2048 cells on $[0, 1]$, timestep $\Delta t = 10^{-5}$, RK2 discretization in time,
- Entire trajectories of $N = 4096$ iterations, sub-trajectory of $m = 512$ iterations,
- $K = 8$ initial conditions per episode,
- 20 batches of size 16 per episode, arbitrary high number of episodes.

Values for the weights ω_{osc}, ω_{acc} and ω_{visc} will be specified for each test-cases.

4.4.1 Advection equation

We will start by validating the approach on the advection equation given by

$$\begin{cases} \partial_t \rho + a\, \partial_x \rho = 0, \\ \rho(t = 0, x) = \rho_0(x), \end{cases}$$

where $\rho : \mathbb{R}_+ \times [0, 1] \to \mathbb{R}$ is the advected density and $a \in \mathbb{R}$ is a constant velocity that we take equal to 1. We consider periodic boudary conditions For the initial condition ρ_0, we test the viscosity on a common composite function with different kinds of discontinuities:

$$\rho_0(x) = 1 + \begin{cases} e^{-((x-0.125)/0.03)^2} & \text{if } x < 0.25, \\ 1, & \text{if } 5/16 \le x < 7/16, \\ 1 - \left|(x - \frac{5}{8}) \times 16\right| & \text{if } 9/16 \le x < 11/16, \\ \sqrt{1 - (16x - 14)^2} & \text{if } 13/16 \le x < 15/16, \\ 0 & \text{otherwise.} \end{cases} .$$

In order to notice the effects in long time of the different viscosities, we consider the solution after two periods at $t = 2$. We take advantage of the simplicity of the problem to discuss the effect of the hyper parameters of the optimization problem, i.e. the weights ω_{osc}, ω_{acc} and ω_{visc} involved in the cost function and the size m of the sub-trajectories.

For simplicity we start by training an artificial viscosity using only two of the three terms in the loss. Since the term in C_{osc} on the one hand and the terms in C_{acc} and C_{visc} on the other seem adversarial, we consider using only C_{osc} and C_{visc} ($\omega_{\text{acc}} = 0$), or only C_{osc} and C_{acc} ($\omega_{\text{visc}} = 0$). Let us consider the first case. Figure 4.4.1 shows a typical training in these conditions: at first, the neural network greatly decreases the amount of viscosity, before adding some back in order to find a better compromise between the two parts of the loss. In order to visualize the resulting viscosity and solution, the neural network is applied with a specific test case, but note that the training was done with random initial conditions as described in section 4.3.4.

An important factor in the equilibrium reached is the value chosen for the weights ω_{osc}, ω_{visc} and ω_{acc}. Figure 4.4.2 illustrates this by showing the resulting solutions and viscosities when ω_{osc} is set to 10^{-5} and when ω_{visc} varies (ω_{acc} still being set to zero). As expected, when the L^2 penalization increases, the viscosity gets smaller and smaller, which results in less diffusion but more oscillations. Indeed, as observed in Table 4.4.1, both C_{osc} and L^∞ error increases with ω_{visc}. Figure 4.4.3 and Table 4.4.2 show what happens when it is ω_{visc} which is set to zero and ω_{acc} which varies, ω_{osc} still being set to 10^{-5}. The results are similar, but show more diffusion overall.

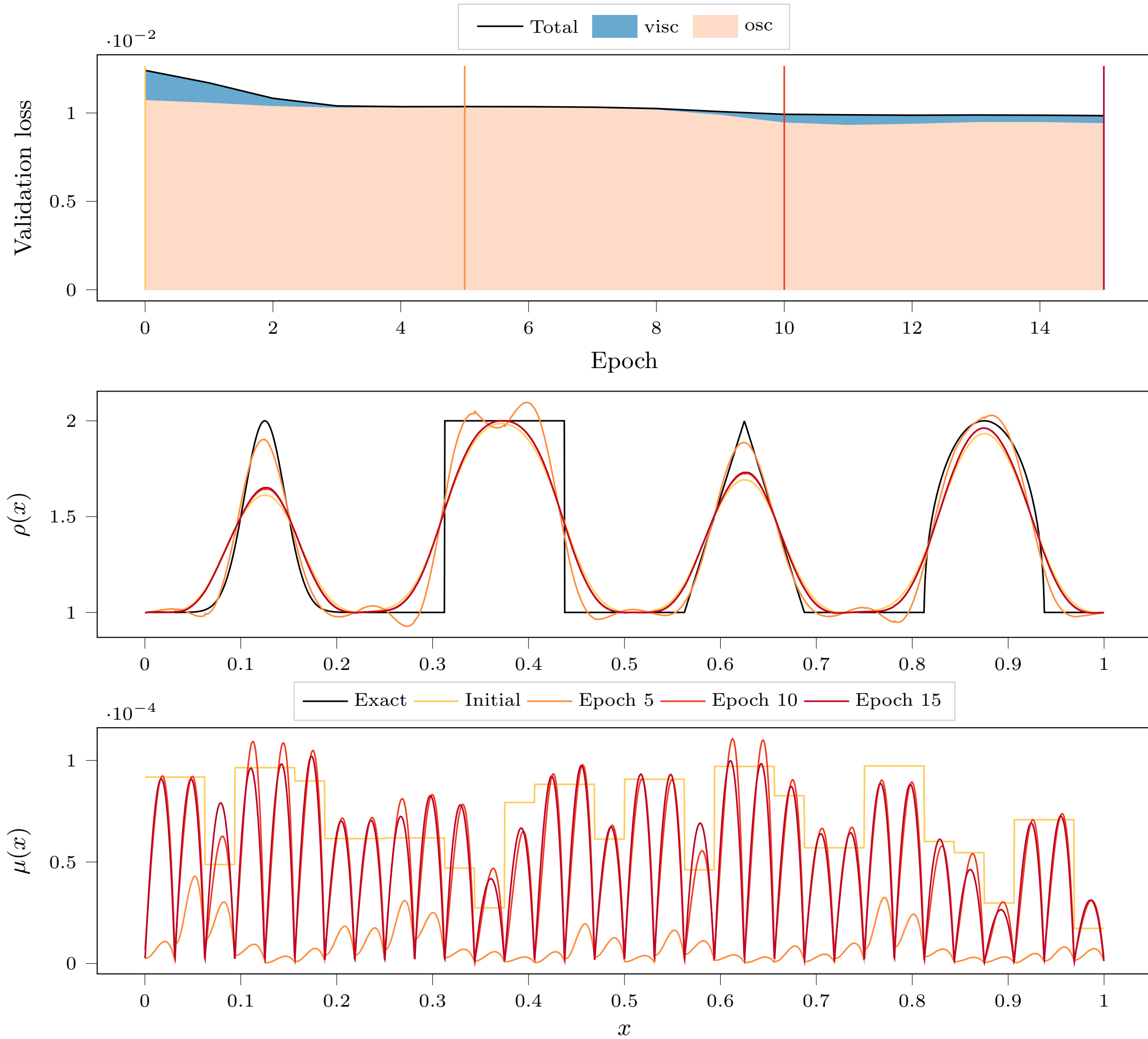


Figure 4.4.1 – (Advection) Top: Evolution of the validation loss during training. The contribution of the different terms are shown in color. Middle and bottom: Solution (middle) and viscosity (bottom) on a test case using the neural network viscosity at different points in its training. The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

Model	ω_{osc}	ω_{acc}	ω_{visc}	C_{osc}	C_{acc}	C_{visc}	L^2	L^∞
DG NN	10^{-5}	0	$2 \cdot 10^3$	9.32e+03	1.10e-01	1.73e-10	5.91e-02	5.26e-01
	10^{-5}	0	$4 \cdot 10^3$	9.36e+03	9.32e-02	1.03e-10	5.95e-02	5.34e-01
	10^{-5}	0	$6 \cdot 10^3$	9.41e+03	8.35e-02	7.44e-11	5.97e-02	5.37e-01
	10^{-5}	0	$8 \cdot 10^3$	9.45e+03	7.57e-02	5.85e-11	5.98e-02	5.43e-01

Table 4.4.1 – (Advection - variation ω_{visc}) Errors for each model presented in Figure 4.4.2.

Model	ω_{osc}	ω_{acc}	ω_{visc}	C_{osc}	C_{acc}	C_{visc}	L^2	L^∞
DG NN	10^{-5}	0.2	0	9.26e+03	1.38e-01	3.47e-10	5.86e-02	5.18e-01
	10^{-5}	0.8	0	9.27e+03	1.28e-01	2.87e-10	5.88e-02	5.26e-01
	10^{-5}	1.6	0	9.25e+03	1.28e-01	2.94e-10	5.88e-02	5.26e-01
	10^{-5}	3.2	0	9.27e+03	1.19e-01	2.55e-10	5.89e-02	5.22e-01

Table 4.4.2 – (Advection - variation ω_{acc}) Errors for each model presented in Figure 4.4.3.

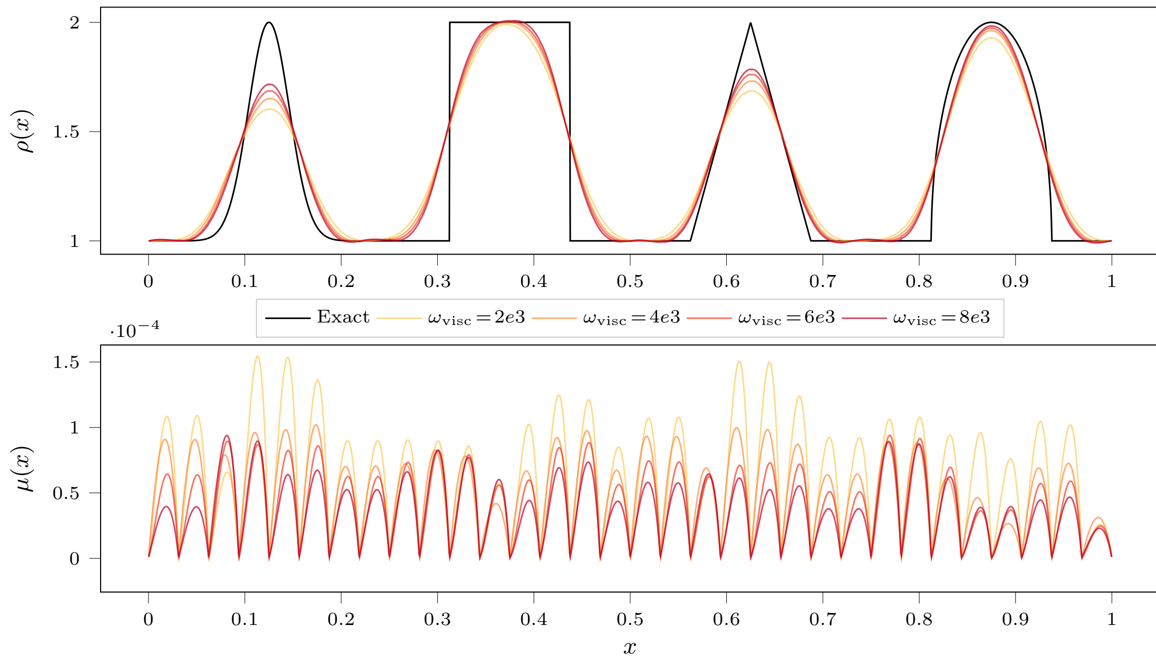


Figure 4.4.2 – (Advection - variation ω_{visc}) Solution (top) and viscosity (bottom) on a test case with neural network viscosities obtained with different weights ω_{visc}, the other two weights being set to $\omega_{\text{osc}} = 10^{-5}$ and $\omega_{\text{acc}} = 0$. The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

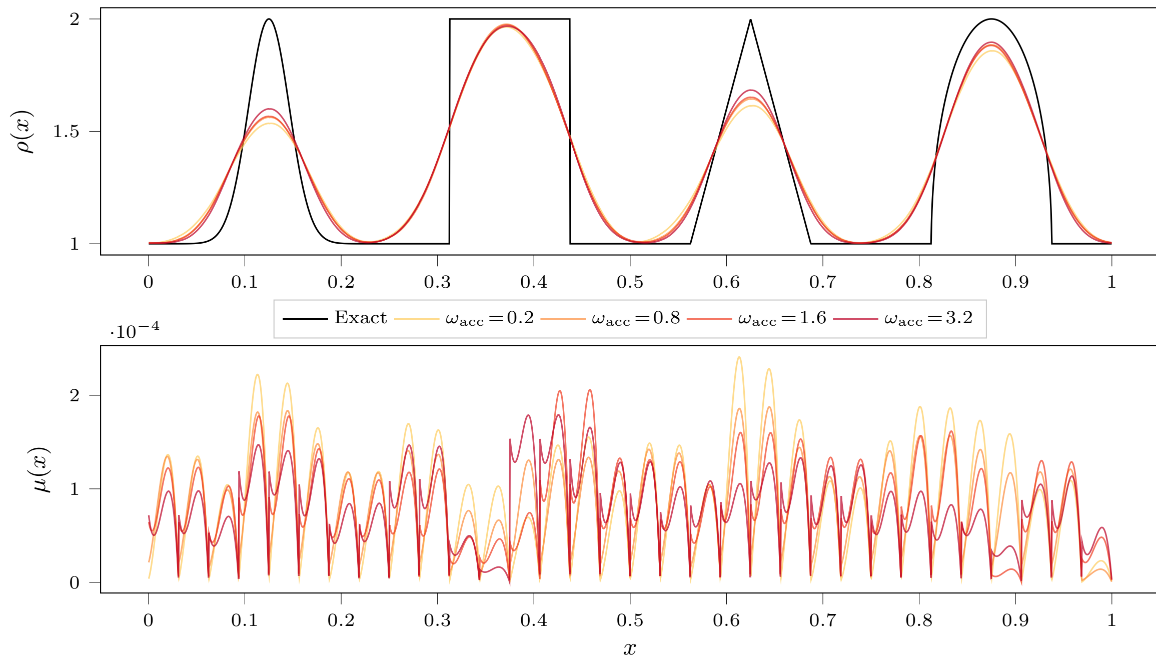


Figure 4.4.3 – (Advection - variation ω_{acc}) Solution (top) and viscosity (bottom) on a test case with neural network viscosities obtained with different weights ω_{acc}, the other two weights being set to $\omega_{\text{osc}} = 10^{-5}$ and $\omega_{\text{visc}} = 0$. The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

Another important parameter of the algorithm is the number m of iterations in a sub-trajectory, on which the gradient of the loss is computed. Picking a big value for m makes the computation

of the gradient more expensive, but allows to include more long-term effects of the viscosity. Figure 4.4.4 shows some resulting viscosities with different values for m. It would seem that as m increases, the model becomes less and less diffusive, because more diffusion appears in longer sub-trajectories and affect the gradient. In order to learn a minimal viscosity that will eliminate the oscillations it is therefore important to optimize our network on long enough trajectories so that the effect of the diffusion is clearly visible in the cost functions. Otherwise, the method learns a too large viscosity since its negative effect will not be visible enough in the cost functions. Also, a smaller m do not necessarily decrease the training time even if the computation of the gradient is cheaper since the number of steps of gradient descent may be larger.

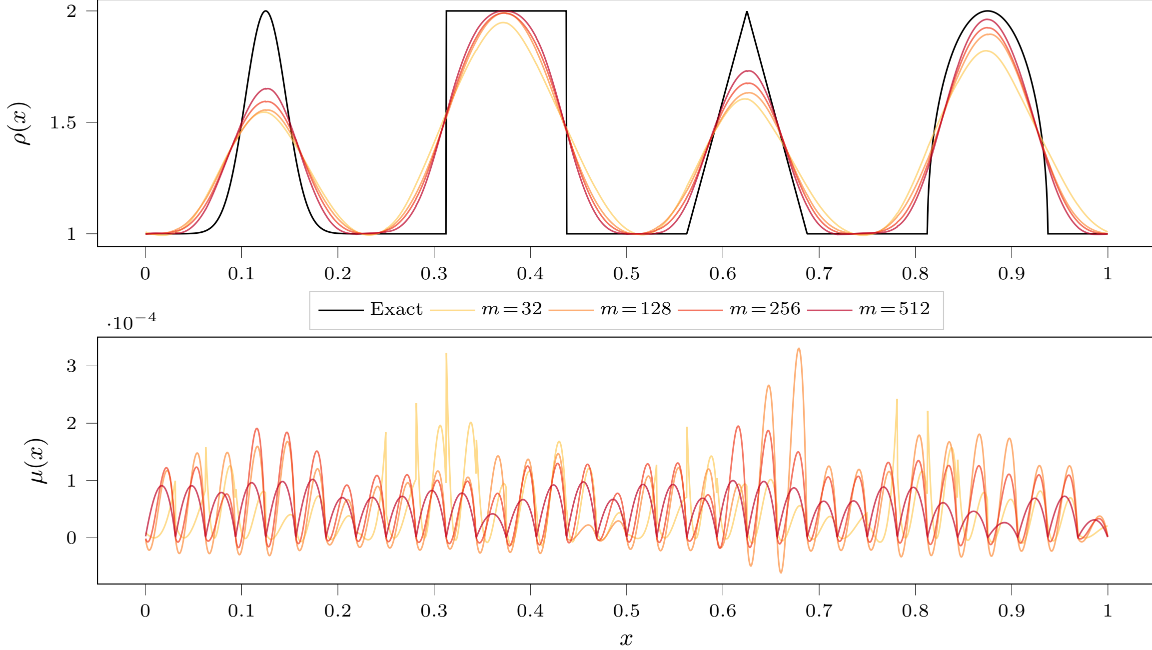


Figure 4.4.4 – (Advection - variation m) Solution (top) and viscosity (bottom) on a test case with neural network viscosities obtained using sub-trajectories with different length m. The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

Finally, we compare our viscosity to two reference viscosities: the "derivative-based" (DB) viscosity, and the "highest modal decay" (MDH) viscosity, described in appendix 4.A. Figure 4.4.5 shows the result with 32 cells for the discontinuous Galerkin scheme, which is the same resolution as the one used for the training of the neural network. Interestingly enough, our viscosity generalises pretty well to other resolutions, thanks to the scaling described in section 4.3.3. As an illustration, Figures 4.4.6, 4.4.7 and 4.4.8 compare the same three viscosities but used with 64, 128 and 256 cells respectively. Note that in these three examples, the model "DG NN" uses the same viscosity as in Figure 4.4.5, trained on 32 cells only. The results on the different figures and given by Table 4.4.3 show that the neural network viscosity gives the best compromise between accuracy and oscillations or between L^2 and L^∞ errors. Indeed, the MDH method has a lower L^2 error but oscillates more and has a larger L^∞ error. The DB approach is clearly more diffusive for this long time problem.

4.4.2 Burgers equation

We now consider the Burgers equation given by

$$\begin{cases} \partial_t \rho + \partial_x \left(\dfrac{\rho^2}{2} \right) = 0, \\ \rho(t=0,x) = \rho_0(x), \end{cases}$$

with $\rho : \mathbb{R}_+ \times [0,1] \to \mathbb{R}$ and complemented with periodic boundary conditions. The network, the hyper-parameters and the training process are exactly the same as for the advection equation, with

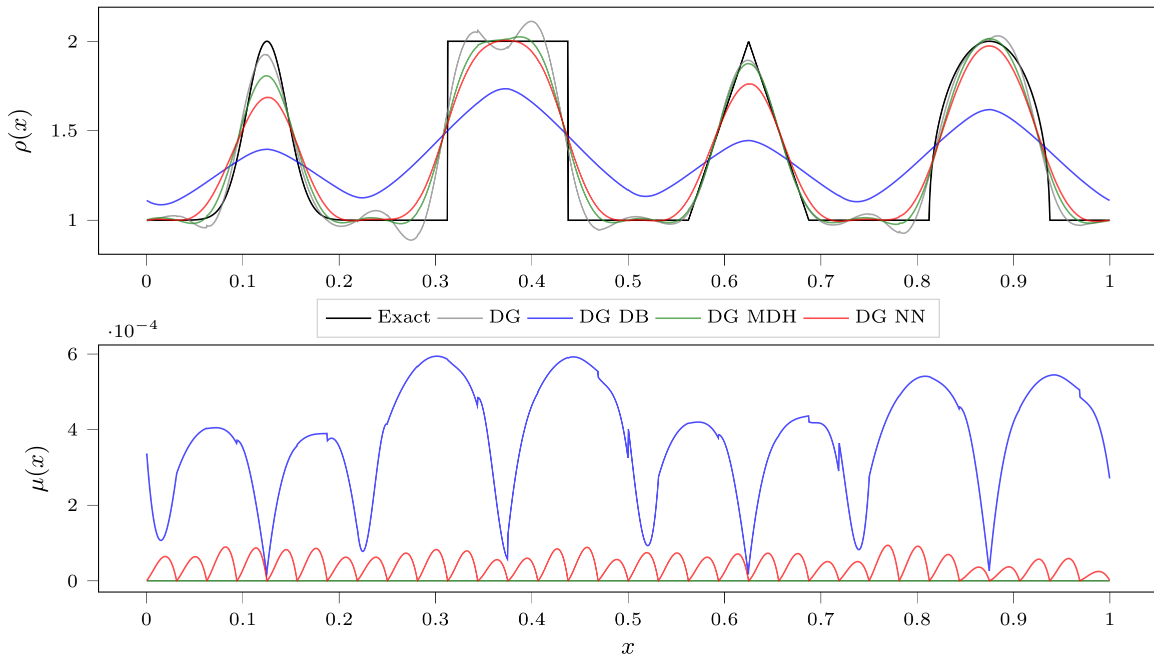


Figure 4.4.5 – (Advection - comparison with DB and MDH) Solution (top) and viscosity (bottom) on a test case with different viscosities: no viscosity (DG), derivative-based viscosity (DG DB), highest modal decay viscosity (DG MDH) and our neural network based viscosity (DG NN). The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

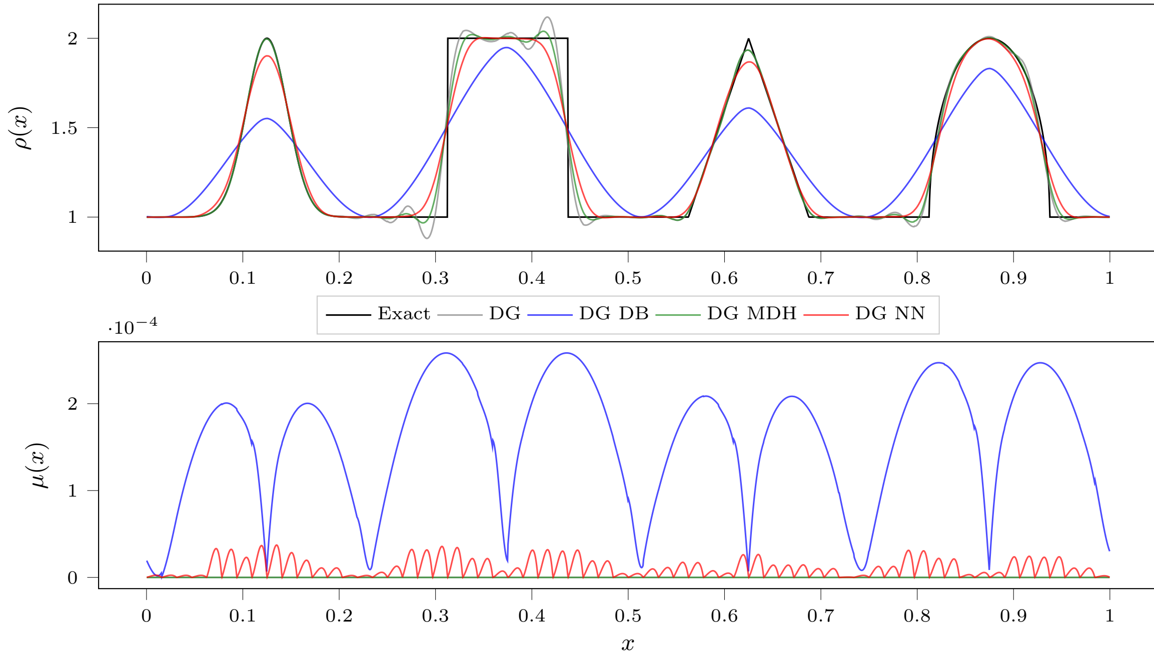


Figure 4.4.6 – (Advection - comparison with DB and MDH) Solution (top) and viscosity (bottom) on a test case with 64 cells instead of the usual 32. The different viscosities used are the derivative-based (DB), the highest modal decay (MDH) and our neural network based viscosity (NN). The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

coefficients $\omega_{\text{acc}} = 0.5$, $\omega_{\text{osc}} = 10^{-5}$ and $\omega_{\text{visc}} = 5$. In order to avoid any issues with non-entropic solutions the DG scheme could converge to, we only consider positive functions in the dataset. The remarks made on the hyper-parameters, the learning in the previous section on advection remains

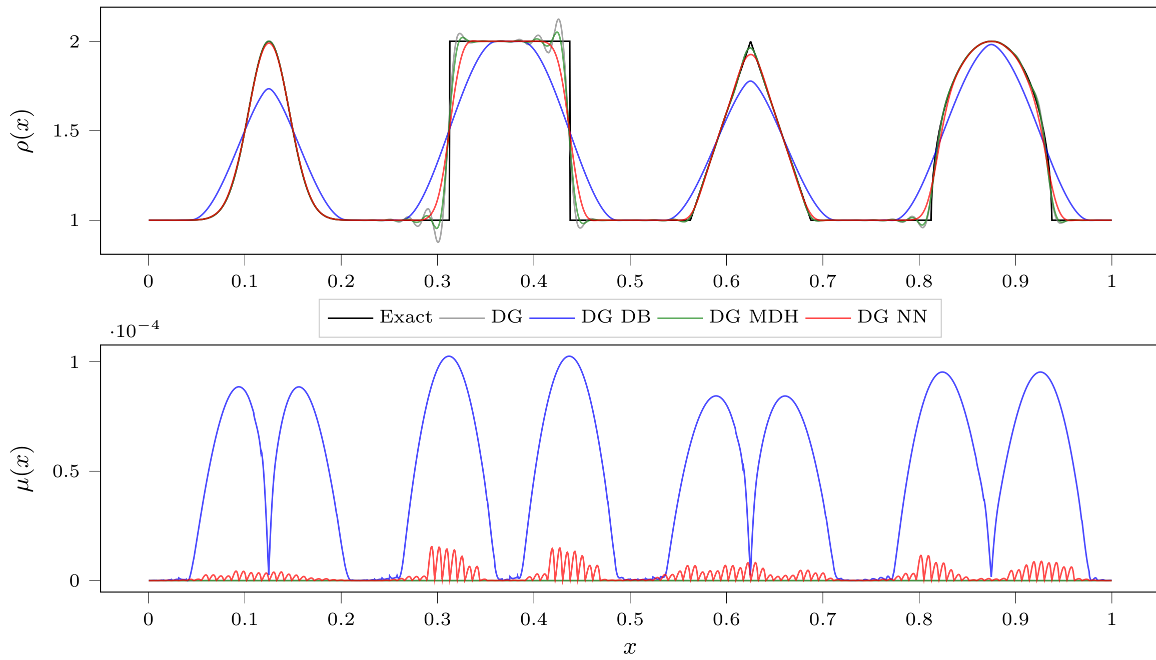


Figure 4.4.7 – (Advection - comparison with DB and MDH) Solution (top) and viscosity (bottom) on a test case with 128 cells instead of the usual 32. The different viscosities used are the derivative-based (DB), the highest modal decay (MDH) and our neural network based viscosity (NN). The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

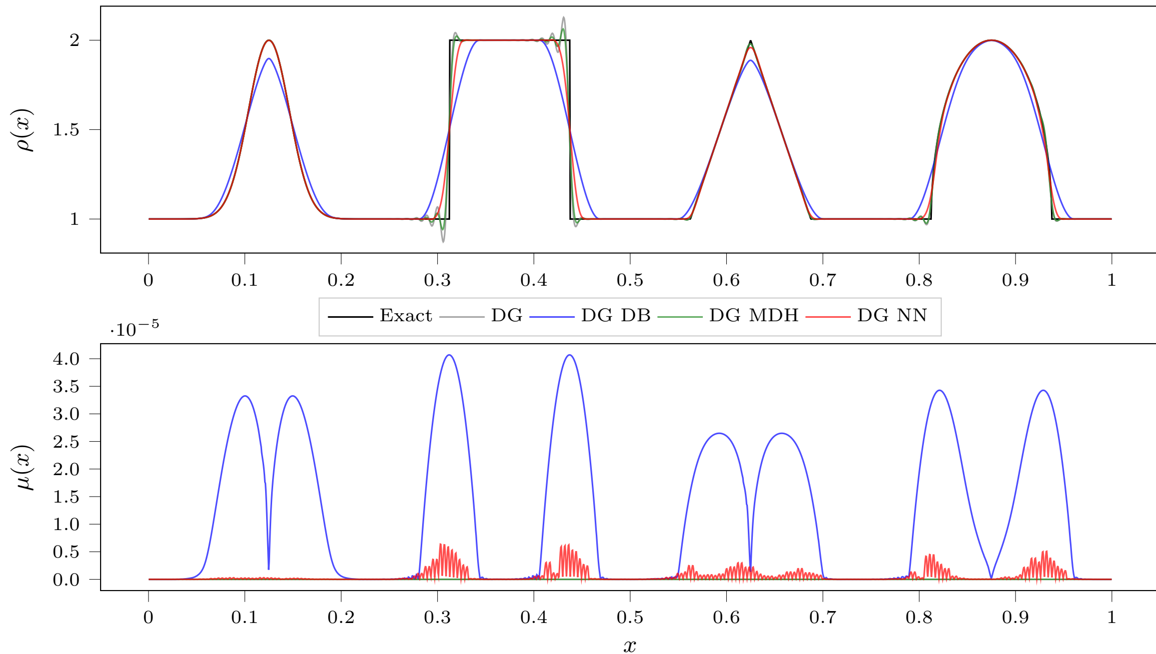


Figure 4.4.8 – (Advection - comparison with DB and MDH) Solution (top) and viscosity (bottom) on a test case with 256 cells instead of the usual 32. The different viscosities used are the derivative-based (DB), the highest modal decay (MDH) and our neural network based viscosity (NN). The test case uses periodic boundary conditions and consists of two periods (ie final time $t = 2$).

valid here. We therefore propose to give direct results comparing a learned viscosity with classical viscosities. To do this, we consider an initial condition that has not been used in the training phase of

Model	Cells	C_{osc}	C_{acc}	C_{visc}	L^2	L^∞
DG	32	9.97e+03	5.24e-02	0.00e+00	9.03e-03	6.05e-01
	64	9.89e+03	2.62e-02	0.00e+00	4.71e-03	5.95e-01
	128	1.01e+04	1.36e-02	0.00e+00	2.52e-03	5.81e-01
	256	1.05e+04	7.23e-03	0.00e+00	1.37e-03	5.60e-01
DG DB	32	9.18e+03	2.49e-01	3.30e-07	7.82e-02	6.04e-01
	64	9.18e+03	1.56e-01	6.11e-08	3.94e-02	5.10e-01
	128	9.21e+03	8.57e-02	7.16e-09	1.86e-02	5.06e-01
	256	9.22e+03	4.36e-02	6.94e-10	9.13e-03	5.02e-01
DG MDH	32	9.57e+03	5.94e-02	0.00e+00	1.22e-02	5.57e-01
	64	9.56e+03	2.49e-02	0.00e+00	5.37e-03	5.57e-01
	128	9.69e+03	1.26e-02	0.00e+00	2.75e-03	5.50e-01
	256	1.00e+04	6.60e-03	0.00e+00	1.45e-03	5.37e-01
DG NN	32	9.41e+03	8.35e-02	4.76e-09	1.78e-02	5.37e-01
	64	9.30e+03	4.14e-02	3.96e-10	8.48e-03	5.22e-01
	128	9.27e+03	2.20e-02	2.86e-11	4.98e-03	5.08e-01
	256	9.34e+03	1.28e-02	3.00e-12	3.11e-03	4.94e-01

Table 4.4.3 – (Advection - comparison with DB and MDH) Errors for each model presented in Figure 4.4.5.

the neural network viscosity:

$$\rho_0(x) = 1 + \sin(2\pi x), \quad x \in [0, 1],$$

with final time $t = 1$.

On Figures 4.4.9 and 4.4.10, we observe as before that the classical DG method without viscosity term generates large oscillations closed to the discontinuity. Contrary to the transport case, the MDH method is here the more diffusive method. Note that the MDH viscosity acts at the beginning of the simulation and then vanishes as the approximate solution becomes smooth. The neural network and the DB methods gives very similar results with less numerical diffusion and small oscillations. Note that the neural network is slightly less oscillating at the bottom of the discontinuity. In conclusion, this test-case shows that the neural network viscosity still provides good results for such a non-linear equation with generates discontinuities.

4.4.3 Euler system

Finally we present results for the Euler system:

$$\begin{cases} \partial_t \rho + \partial_x (\rho u) = 0, \\ \partial_t(\rho u) + \partial_x \left(\rho u^2 + p\right) = 0, \\ \partial_t E + \partial_x (Eu + pu) = 0, \end{cases}$$

where $\rho : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ denotes the density, $u : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ the velocity, $p : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ the pressure and $E : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ the energy. The system is completed with a perfect gas law, resulting in the following relation :

$$E = \frac{p}{\gamma - 1} + \frac{\rho u^2}{2},$$

where γ is the adiabatic constant taken equal to 1.4 here. Once again we use the same parameters as before, the only difference being that π_θ now has three inputs, one for each conservative variable. The output is still a single viscosity coefficient $\mu(x)$, since the same viscosity is applied to each equation. The neural network is trained using the coefficients $\omega_{\text{acc}} = 0$, $\omega_{\text{osc}} = 10^{-5}$ and $\omega_{\text{visc}} = 10^3$ in the loss. The training dataset is made using initial conditions with the three variables ρ, u, p chosen according to (4.9) with the correction to ensure the positivity of the density and the pressure.

We compare the different viscosity approaches on two classical test cases: the Sod problem and the Shu-Osher problem. In these test cases the DG scheme without viscosity is unstable and therefore is not presented.

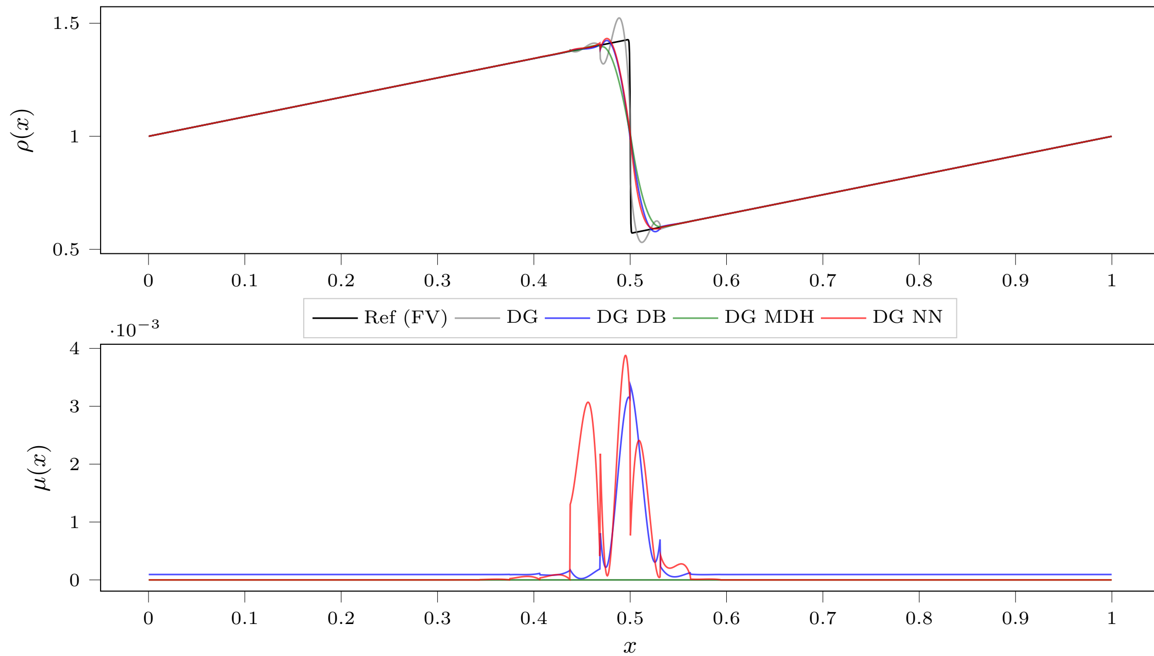


Figure 4.4.9 – (Burgers - comparison with DB and MDH) Solution (top) and viscosity (bottom) of different models for Burgers equation with 32 cells

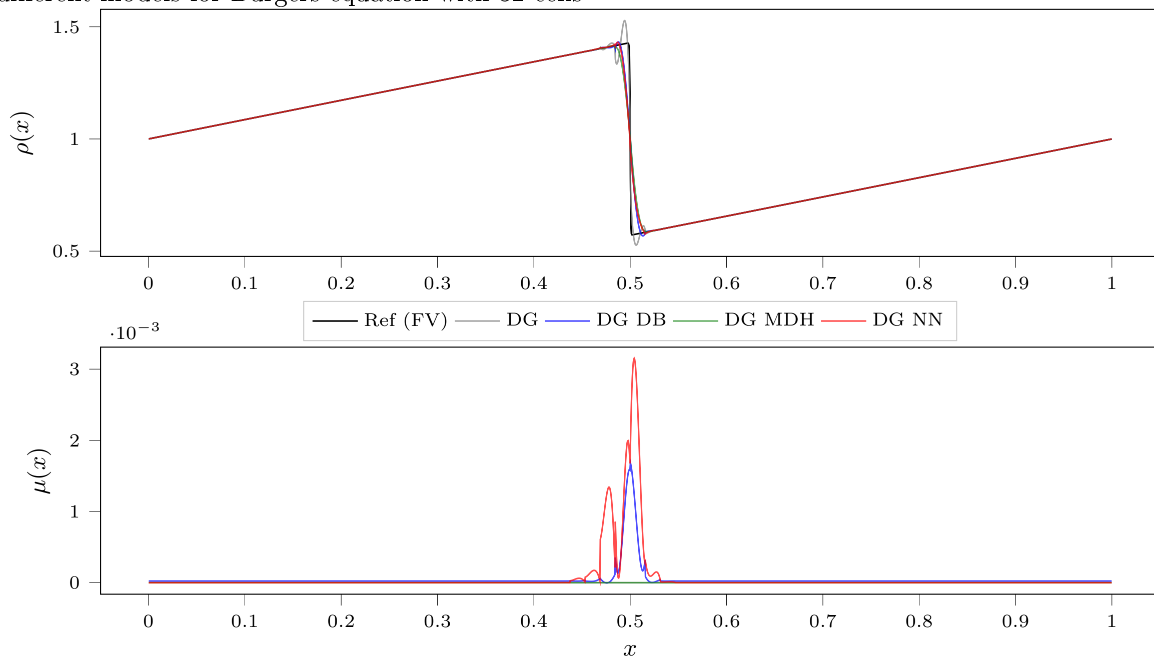


Figure 4.4.10 – (Burgers - comparison with DB and MDH) Solution (top) and viscosity (bottom) of different models for Burgers equation with 64 cells

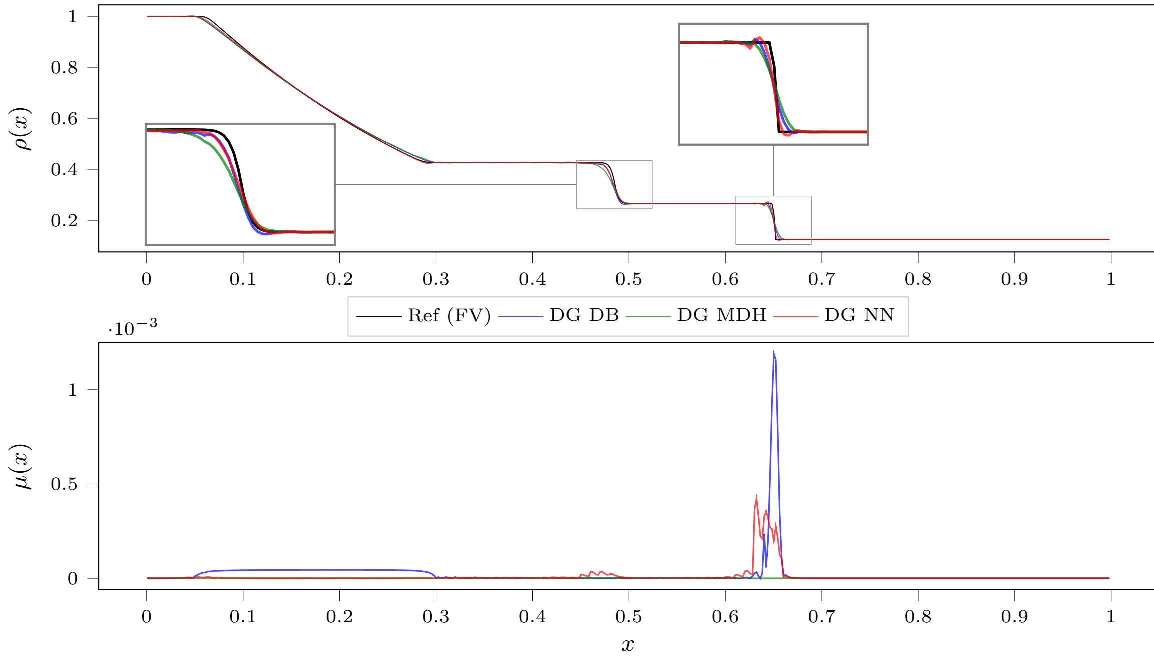


Figure 4.4.11 – (Euler - Sod) Sod test case, 100 cells

The Sod test-case uses the initial condition

$$(\rho_0, u_0, p_0)(x) = \begin{cases} (1, 0, 1), & \text{if } x < 0.5 \\ (0.125, 0, 0.1). & \text{otherwise} \end{cases}$$

on the interval $[0, 1]$ with final time $t = 0.2$. We consider also Dirichlet boundary conditions. On Figure 4.4.11, we compare the different schemes associated with the different viscosity models on a mesh with 100 cells. As for the Burgers equation, the MDH viscosity provides the worst results. This problem can be explained by the fact that the hyper-parameters of the MDH method, taken from [24], may not be optimized to this specific test-case. The result between the DB model and the neural network model are close. Our approach seems better in the contact wave and a little bit more oscillating on the shock. On a grid with 200 cells 4.4.12, the neural network viscosity seems slightly more accurate for all the different components of the solution. It is confirmed by the errors presented in Table 4.4.4, which both L^2 and L^∞ errors are smaller on the two meshes.

Model	Cells	C_{osc}	C_{acc}	C_{visc}	L^2	L^∞
DG DB	100	3.13e+02	2.47e-03	2.04e-08	3.36e-05	5.08e-02
	200	3.04e+02	1.12e-03	2.48e-09	1.03e-05	3.93e-02
DG MDH	100	2.86e+02	2.88e-03	0.00e+00	5.27e-05	5.43e-02
	200	3.00e+02	1.24e-03	0.00e+00	1.27e-05	3.90e-02
DG NN	100	3.30e+02	1.36e-03	4.97e-09	1.48e-05	4.21e-02
	200	2.94e+02	5.86e-04	7.09e-10	2.92e-06	3.11e-02

Table 4.4.4 – Errors for each model on the Sod test case.

The second test case is the Shu-Osher test case, whose initial condition is given by:

$$(\rho_0, u_0, p_0)(x) = \begin{cases} (3.857143, 2.629369, 10.333333) & \text{if } x < -4 \\ (1 + 0.2\sin(5x), 0, 1) & \text{otherwise} \end{cases},$$

on the interval $[-5, 5]$ with final time $t = 1.8$. The solution is composed of several smooth oscillations and a discontinuity. As before, we compare the different approaches on a given mesh with 200 cells. The results in Figure 4.4.14 and Table 4.4.5 shows that our model and the DB model gives very similar results with a slight advantage for the DB model.

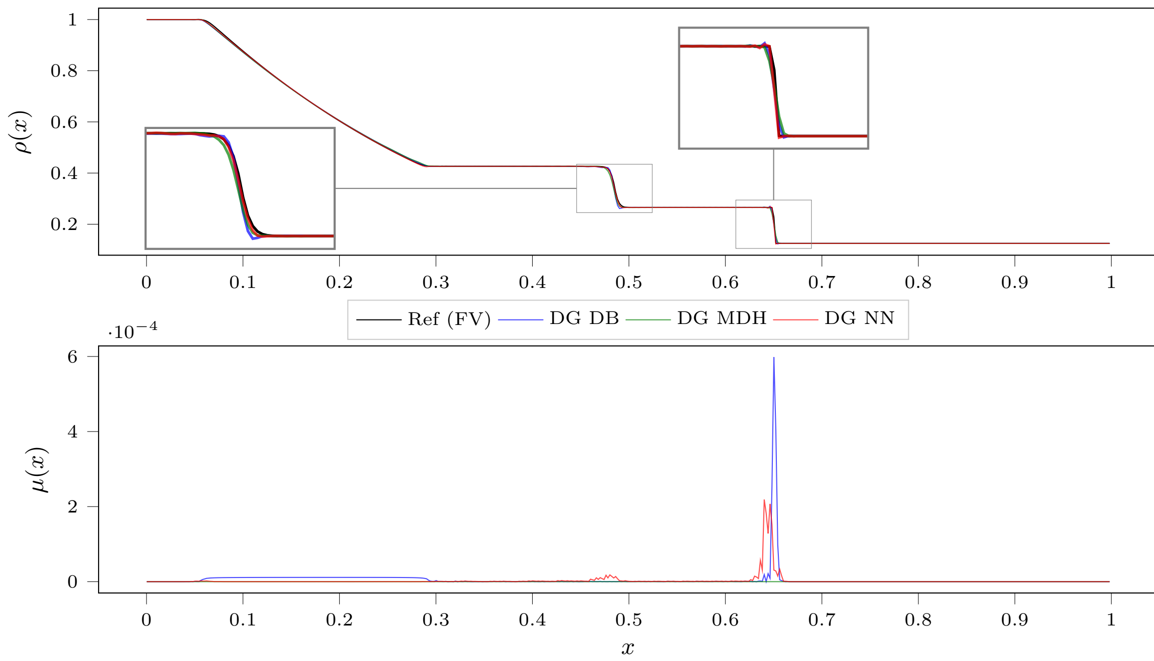


Figure 4.4.12 – (Euler - Sod) Sod test case, 200 cells

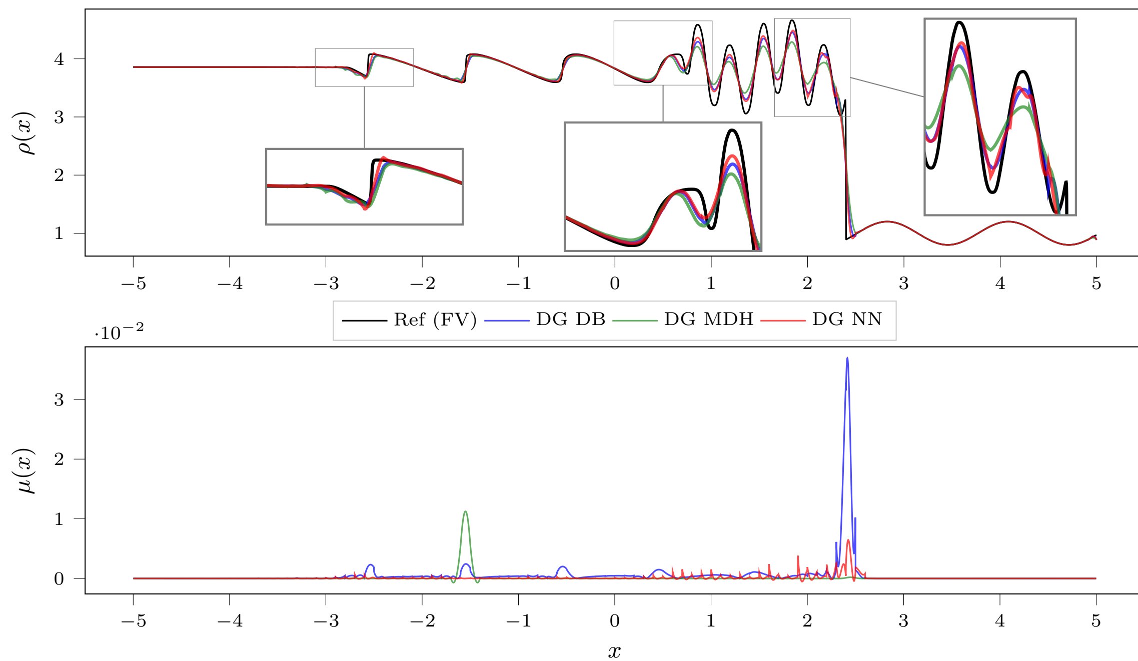


Figure 4.4.13 – (Euler) Shu-Osher test case, 100 cells

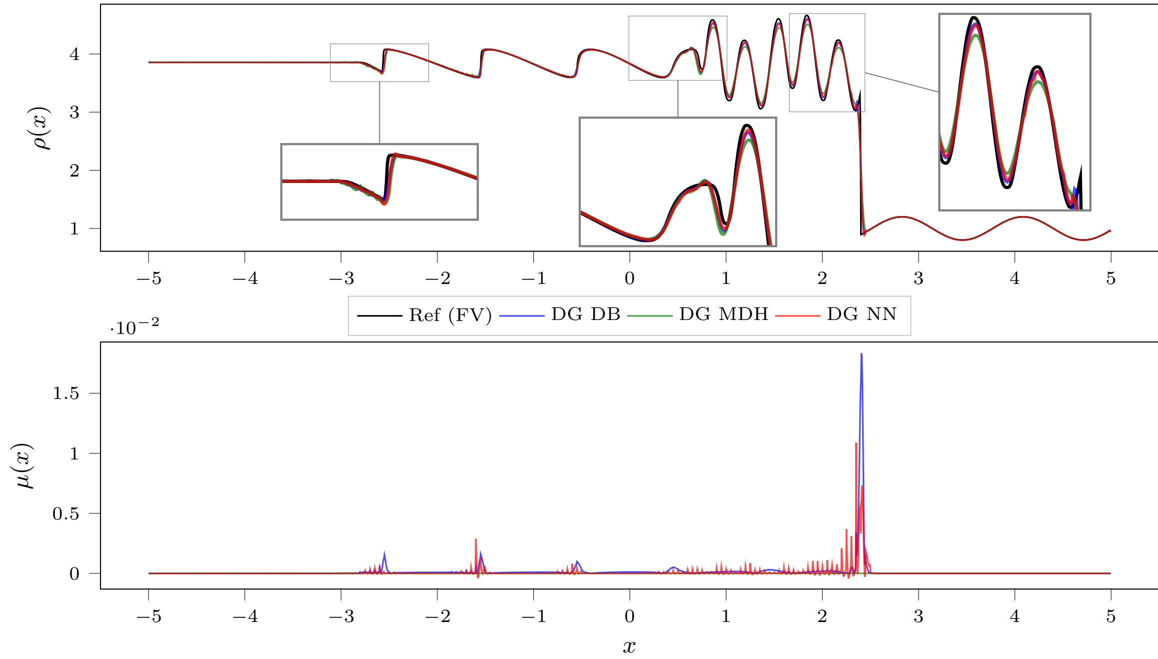


Figure 4.4.14 – (Euler) Shu-Osher test case, 200 cells

Model	Cells	C_{osc}	C_{acc}	C_{visc}	L^2	L^∞
DG DB	100	2.47e+03	4.09e-01	1.73e-04	1.01e-01	1.18e+00
	200	2.40e+03	1.61e-01	2.16e-05	2.92e-02	1.08e+00
DG MDH	100	2.37e+03	5.76e-01	1.98e-05	1.79e-01	1.31e+00
	200	2.25e+03	2.53e-01	6.00e-13	5.21e-02	1.25e+00
DG NN	100	2.38e+03	3.49e-01	5.46e-06	8.24e-02	1.22e+00
	200	2.42e+03	1.71e-01	5.29e-06	2.95e-02	1.23e+00

Table 4.4.5 – (Euler) Errors for each model on the Shu-Osher test case.

4.5 Conclusion

In this paper, we propose an optimal control approach to optimize a parametric numerical scheme based on its effect after several iterations. The method is a simple gradient method to optimize a given cost function, where the gradient is calculated across a large number of iterations by automatic differentiation. We apply it to the construction of an artificial viscosity for DG methods for one-dimensional hyperbolic equations. The numerical results on different simulations show that the obtained neural network viscosities result in equivalent or better results compared with classical artificial viscosities (Derivative Based or Highest Model Decay viscosities).

There are several possible ways to extend this work. First, non-physical oscillations have so far been detected with the semi-norm $W^{2,1}$ of the error with respect to the reference solution. Another possibility would be to design a data-driven detector of the non-physical oscillations like in [2].

It will also be naturally important to extend this work to 2D/3D problems. Note however that a major difficulty comes from the number of iterations taken into account in the computation of the gradient. In our one-dimensional problem, we succeed in considering up to 1000 time steps. However, this was possible because of the coarse meshes and small networks. For two-dimensional problems, the sizes of the mesh and the network may be larger and the memory resources may be saturated. To overcome this difficulty, the method could be coupled with a reinforcement approach [23] or a neural ODE method [4], for which the gradient are computed by duality.

Finally, the same methodology could also be applied to other problems like estimating optimal slope limiters or WENO stencils.

Bibliography

[1] Y. Bar-Sinai et al. "Learning data-driven discretizations for partial differential equations". In: *Proceedings of the National Academy of Sciences* 116.31 (2019), pp. 15344–15349. DOI: 10.1073/pnas.1814058116. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1814058116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1814058116.

[2] A.D. Beck et al. "A neural network based shock detection and localization approach for discontinuous Galerkin methods". In: *J. Comput. Phys.* 423 (2020), p. 109824.

[3] A. Bourriaud, R. Loubère, and R. Turpault. "A priori neural networks versus a posteriori MOOD loop: a high accurate 1D FV scheme testing bed". In: *J. Sci. Comput.* 84.2 (2020), pp. 1–36.

[4] R.T.Q. Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems* 31 (2018).

[5] B. Cockburn, S.-Y. Lin, and C.-W. Shu. "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one-dimensional systems". In: *J. Comput. Phys.* 84.1 (1989), pp. 90–113.

[6] B. Cockburn and C.-W. Shu. "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework". In: *Mathematics of computation* 52.186 (1989), pp. 411–435.

[7] Bernardo Cockburn and Chi-Wang Shu. "Runge–Kutta discontinuous Galerkin methods for convection-dominated problems". In: *J. Scient. Comput.* 16 (2001), pp. 173–261.

[8] N. Discacciati, J.S. Hesthaven, and D. Ray. "Controlling oscillations in high-order Discontinuous Galerkin schemes using artificial viscosity tuned by neural networks". In: *J. Comput. Phys.* 409 (2020), p. 109304.

[9] Gi. Dresdner et al. "Learning to correct spectral methods for simulating turbulent flows". In: *arXiv preprint arXiv:2207.00556* (2022).

[10] J.-L. Guermond, R. Pasquetti, and B. Popov. "Entropy viscosity method for nonlinear conservation laws". In: *J. Comput. Phys.* 230.11 (2011), pp. 4248–4267.

[11] K. He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[12] J.S. Hesthaven. *Numerical methods for conservation laws: From analysis to algorithms*. SIAM, 2017.

[13] J.S. Hesthaven and R. Kirby. "Filtering in Legendre spectral methods". In: *Mathematics of Computation* 77.263 (2008), pp. 1425–1452.

[14] J.S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[15] A. Mani, J. Larsson, and P. Moin. "Suitability of artificial bulk viscosity for large-eddy simulation of turbulent flows with shocks". In: *J. Comput. Phys.* 228.19 (2009), pp. 7368–7374.

[16] P.-O. Persson and J. Peraire. "Sub-Cell Shock Capturing for Discontinuous Galerkin Methods". In: *44th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 2006. DOI: 10.2514/6.2006-112.

[17] J. Qiu and C.-W. Shu. "Hermite WENO schemes and their application as limiters for Runge–Kutta discontinuous Galerkin method: one-dimensional case". In: *J. Comput. Phys.* 193.1 (2004), pp. 115–135.

[18] D. Ray and J.S. Hesthaven. “An artificial neural network as a troubled-cell indicator”. In: *J. Comput. Phys.* 367 (2018), pp. 166–191.

[19] L. Schwander, D. Ray, and J.S. Hesthaven. “Controlling oscillations in spectral methods by local artificial viscosity governed by neural networks”. In: *J. Comput. Phys.* 431 (2021), p. 110144.

[20] A. Schwarz et al. “A Reinforcement Learning Based Slope Limiter for Second-Order Finite Volume Schemes”. In: *PAMM* 23.1 (2023), e202200207.

[21] D. Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

[22] N. Thuerey et al. “Physics-based deep learning”. In: *arXiv preprint arXiv:2109.05237* (2021).

[23] Y. Wang et al. “Learning to discretize: solving 1D scalar conservation laws via deep reinforcement learning”. In: *arXiv preprint arXiv:1905.11079* (2019).

[24] J. Yu and J.S. Hesthaven. “A study of several artificial viscosity models within the Discontinuous Galerkin framework”. In: *Communications In Computational Physics* 27.5 (2020), pp. 1309–1343.

[25] Z. Zhao, Y. Chen, and J. Qiu. “A hybrid Hermite WENO scheme for hyperbolic conservation laws”. In: *J. Comput. Phys.* 405 (2020), p. 109175.

[26] X. Zhong and C.-W. Shu. “A simple weighted essentially nonoscillatory limiter for Runge–Kutta discontinuous Galerkin methods”. In: *J. Comput. Phys.* 232.1 (2013), pp. 397–415.

Appendix

4.A Reference artificial viscosity models

In the result section, we compare our viscosity to two models of reference, that we briefly describe here in the context of a discontinuous Galerkin scheme of order p in one dimension.

The first one is the simplest one, refered to as the derivative-based (DB) model in the comparative study [24], and reads

$$\pi_{\mathrm{DB}}(\mathbf{U}) = \min(\mu_\beta, \mu_{\max}), \quad \mu_\beta = c_\beta(\tfrac{\Delta x}{p-1})^2|\partial_x u|, \quad \mu_{\max} = c_{\max}\tfrac{\Delta x}{p-1}\max_{\text{cell}}|s|,$$

where u is the unique variable in the scalar case and the velocity for the Euler equation, s is the local wave speed, and c_β and $c_{\max}$ are empirical parameters, set to 1 and 0.5 respectively.

The second one is refered to as the highest modal decay (MDH) model in [24] and was first proposed in [16]. In this model, the viscosity is computed from the variable ρ which refers to the unique variable in the scalar case, and to the density for the Euler equation. The MDH model relies on a modal expansion of ρ in each cell,

$$\rho(x,t) = \sum_{k=0}^{p-1} \hat{\rho}_k(t)\psi_k(x), \quad \psi_k \text{ Legendre polynomials on the cell considered,}$$

and more specifically on the ratio between the norm of the highest mode and the overall norm:

$$r = \log_{10}\frac{\|\hat{\rho}_{p-1}\psi_{p-1}\|_{L^2}^2}{\|\rho\|_{L^2}^2}.$$

The viscosity is then taken smoothly increasing with r from 0 to $\mu_{\max}$ as follows:

$$\pi_{\mathrm{MDH}}(\mathbf{U}) = \mu_{\max}\begin{cases} 0 & \text{if } r < r_0 - c_K \\ \frac{1}{2}\left(1+\sin\frac{\pi(r-r_0)}{2c_K}\right) & \text{if } r_0 - c_K < r < r_0 + c_K \\ 1 & \text{otherwise} \end{cases}$$

The threshold r_0 depends on the order p as

$$r_0 = -\big(c_A + 4\log_{10}(p-1)\big),$$

and c_A and c_K are empirical parameters set to 2.5 and 0.2 respectively. These computations give a value for the viscosity coefficient on each cell, which is interpolated by a polynomial of degree 2 that has this value in the middle of the cell, and the average value between the two cells involved at the interfaces, resulting in a continuous function.

Chapter 5

Neural network based relaxation matrix for vectorial Lattice-Boltzmann methods in 2D

This chapter focuses on the vectorial Lattice-Boltzmann method introduced in chapter 1, where references can be found for more details. The first section below dives into the details of the specific implementation that we consider in this work, so we briefly remind the general method here first. Lattice-Boltzmann (LBM) methods come from the observation that some systems, like the Navier-Stokes equations, can be derived from a kinetic model —a model that describes a set of particles by their statistical distribution in the phase space. In order to solve a given system of equations, Lattice-Boltzmann methods consist in finding and solving such a kinetic model from which the said system can be derived. Furthermore, this kinetic model can use a distribution of particles with only a few specific velocities, which is leveraged by LBM methods to produce relatively inexpensive numerical methods. The relationship between the variables of the kinetic model and those of the original equations can take two forms: the original LBM method mimics the kinetic theory by considering the different moments in velocity of a unique distribution of particles ; on the other hand, the *vectorial* LBM method generalises the approach, by considering the density (0th-order moment) of as many distributions as there are variables —resulting in a bigger kinetic model. In all cases, the evolution of the distributions is described by a Boltzmann equation with a BGK-like collision operator (Bhatnagar–Gross–Krook), meaning that the collisions of the particles is modelled by a relaxation to a local equilibrium state. This local equilibrium state is key to the method, as it dictates the system of equations that derives from the kinetic model, and should therefore be chosen accordingly. Finally, the relaxation to this equilibrium can be weighted by a matrix, refered to as the *relaxation matrix* below, in order to improve both the accuracy and the stability of the numerical method. The design of such a relaxation matrix is the focus of this work.

The remainder of this chapter is structured as follows. Section 5.1 describes the kinetic model used in our implementation of the vectorial LBM method, while section 5.2 describes its discretization and the impact of the relaxation matrix on the resulting scheme. Section 5.3 introduces neural networks and the training algorithm we use in order to design a relaxation matrix. Finally section 5.4 shows early results for the linear advection equation, Burgers equation and the wave equations.

5.1 Lattice-Boltzmann equations

In this work we consider 2D hyperbolic equations of the form

$$\partial_t \bar{\mathbf{u}} + \partial_x \mathbf{f}_x(\bar{\mathbf{u}}) + \partial_y \mathbf{f}_y(\bar{\mathbf{u}}) = 0, \tag{5.1}$$

with s equations, where $\bar{\mathbf{u}} : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}^s$ is the unknown, and $\mathbf{f}_x, \mathbf{f}_y : \mathbb{R}^s \to \mathbb{R}^s$ is a given flux. In order to numerically solve such an equation, we are interested in vectorial Lattice-Boltzmann (LBM) methods, focusing in particular on the relaxation matrix to improve on the standard formulation. Vectorial Lattice-Boltzmann methods introduce a distribution of particles for each conservative variable,

and solve a kind of Boltzmann equation with BGK collision operator, resulting in a bigger system of linear advection equations with a relaxation source term:

$$\partial_t \mathbf{F} + \boldsymbol{\lambda} \cdot \nabla \mathbf{F} = \frac{1}{\varepsilon} R(\mathbf{u})(\mathbf{F}^{\mathrm{eq}}(\mathbf{u}) - \mathbf{F}). \tag{5.2}$$

In this work we consider a method that uses 4 velocities, so the velocities $\boldsymbol{\lambda}$ and the unknown $\mathbf{F}$ can be written

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \boldsymbol{\lambda}_3 \\ \boldsymbol{\lambda}_4 \end{bmatrix}, \boldsymbol{\lambda}_i = (\lambda_{ix}, \lambda_{iy}) \in \mathbb{R}^2 \quad \text{and} \quad \mathbf{F} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \mathbf{F}_3 \\ \mathbf{F}_4 \end{bmatrix}, \mathbf{F}_i : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}^s.$$

With these notations, the advection term is to be understood as

$$\boldsymbol{\lambda} \cdot \nabla \mathbf{F} = \begin{bmatrix} \lambda_{1x}\, \partial_x \mathbf{F}_1 + \lambda_{1y}\, \partial_y \mathbf{F}_1 \\ \lambda_{2x}\, \partial_x \mathbf{F}_2 + \lambda_{2y}\, \partial_y \mathbf{F}_2 \\ \lambda_{3x}\, \partial_x \mathbf{F}_3 + \lambda_{3y}\, \partial_y \mathbf{F}_3 \\ \lambda_{4x}\, \partial_x \mathbf{F}_4 + \lambda_{4y}\, \partial_y \mathbf{F}_4 \end{bmatrix}$$

In the relaxation term, the variable $\mathbf{u}$ denotes the quantity $\mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$. This quantity is the one intended to tend towards the solution $\bar{\mathbf{u}}$ of the hyperbolic system (5.1) when the parameter $\varepsilon > 0$ tends to zero. This goal imposes a certain form for the equilibrium function $\mathbf{F}^{\mathrm{eq}} : \mathbb{R}^s \to \mathbb{R}^s$ that we discuss below. Finally, the matrix $R(\mathbf{u}) \in \mathbb{R}^{4s \times 4s}$ consists of additional degrees of freedom that can be used to improve the accuracy of the method. The design of this matrix, or rather its numerical counterpart, is the focus of this work.

Moments of $\mathbf{F}$ and $\mathbf{F}^{\mathrm{eq}}$ For the analysis of equation (5.2), it is convenient to express $\mathbf{F}$ in a different basis, in order to replace $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{F}_3$, $\mathbf{F}_4$ by four other variables, of which one should be $\mathbf{u}$. To this end, in addition to $\mathbf{u} = \sum_{i=1}^4 \mathbf{F}_i$, we introduce the three following moments of $\mathbf{F}$:

$$\mathbf{v}_x = \sum_{i=1}^{4} \lambda_{ix} \mathbf{F}_i, \quad \mathbf{v}_y = \sum_{i=1}^{4} \lambda_{iy} \mathbf{F}_i, \quad \text{and} \quad \mathbf{e} = \sum_{i=1}^{4} (\lambda_{ix}^2 - \lambda_{iy}^2) \mathbf{F}_i.$$

The first two arise naturally on the left-hand side of the equation on $\mathbf{u}$ obtained by adding the equations on $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{F}_3$ and $\mathbf{F}_4$. The last one, $\mathbf{e}$, is a possibility among others that can simplify some expressions when (5.2) is expressed in the basis $(\mathbf{u}, \mathbf{v}_x, \mathbf{v}_y, \mathbf{e})$. We also denote by $\mathbf{u}^{\mathrm{eq}}$, $\mathbf{v}_x^{\mathrm{eq}}$, $\mathbf{v}_y^{\mathrm{eq}}$, $\mathbf{e}^{\mathrm{eq}}$ the moments of $\mathbf{F}^{\mathrm{eq}}(\mathbf{u})$, and by M the passage matrix such that

$$M\mathbf{F} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{e} \end{bmatrix} \quad \text{and} \quad M\mathbf{F}^{\mathrm{eq}}(\mathbf{u}) = \begin{bmatrix} \mathbf{u}^{\mathrm{eq}}(\mathbf{u}) \\ \mathbf{v}_x^{\mathrm{eq}}(\mathbf{u}) \\ \mathbf{v}_y^{\mathrm{eq}}(\mathbf{u}) \\ \mathbf{e}^{\mathrm{eq}}(\mathbf{u}) \end{bmatrix}.$$

Necessary conditions for $\mathbf{F}^{\mathrm{eq}}(\mathbf{u})$ can now be found by formally taking the limit $\varepsilon \to 0$ in (5.2) after multiplying it by ε. Assuming that $R(\mathbf{u})$ is invertible, this results in $\mathbf{F} = \mathbf{F}^{\mathrm{eq}}(\mathbf{u})$. When the latter is true, $\mathbf{u}^{\mathrm{eq}}(\mathbf{u}) = \mathbf{u}$ and the equation on $\mathbf{u}$ mentioned above then reads

$$\partial_t \mathbf{u} + \partial_x \mathbf{v}_x^{\mathrm{eq}}(\mathbf{u}) + \partial_y \mathbf{v}_y^{\mathrm{eq}}(\mathbf{u}) = 0,$$

which makes $\mathbf{u}$ solution of equation (5.1) provided that

$$\mathbf{v}_x^{\mathrm{eq}}(\mathbf{u}) = \mathbf{f}_x(\mathbf{u}), \quad \text{and} \quad \mathbf{v}_y^{\mathrm{eq}}(\mathbf{u}) = \mathbf{f}_y(\mathbf{u}).$$

It only remains to define $\mathbf{e}^{\mathrm{eq}}$ to characterize $\mathbf{F}^{\mathrm{eq}}$, which we can pick equal to zero. We thus have

$$\mathbf{F}^{\mathrm{eq}}(\mathbf{u}) = M^{-1} \begin{bmatrix} \mathbf{u} \\ \mathbf{f}_x(\mathbf{u}) \\ \mathbf{f}_y(\mathbf{u}) \\ 0 \end{bmatrix}$$

Choice of $\boldsymbol{\lambda}$ In this work we use velocities in the four cardinal directions with an amplitude $\lambda > 0$:

$$\boldsymbol{\lambda}_1 = \begin{bmatrix} +\lambda \\ 0 \end{bmatrix}, \boldsymbol{\lambda}_2 = \begin{bmatrix} -\lambda \\ 0 \end{bmatrix}, \boldsymbol{\lambda}_3 = \begin{bmatrix} 0 \\ +\lambda \end{bmatrix}, \boldsymbol{\lambda}_4 = \begin{bmatrix} 0 \\ -\lambda \end{bmatrix}.$$

We now have both specified the local equilibrium $\mathbf{F}^{\text{eq}}$, through the choice of a set of moments and their value at equilibrium, and the velocities $\boldsymbol{\lambda}$. In the next section we turn to the discretization of the equation, to derive a numerical scheme and shed light on the relaxation matrix, the only element of equation (5.2) yet to be defined.

5.2 Numerical scheme

In order to solve equation (5.2), we use a time-splitting method and solve alternatively the advection part $\partial_t \mathbf{F} + \boldsymbol{\lambda} \cdot \nabla \mathbf{F} = 0$ and the relaxation part $\partial_t \mathbf{F} = \frac{1}{\varepsilon} R(\mathbf{u})(\mathbf{F}^{\text{eq}}(\mathbf{u}) - \mathbf{F})$. Denoting by $A(\Delta t)$ and $R(\Delta t)$ the operators performing one timestep for the advection and relaxation part respectively, an iteration of the numerical scheme S then reads

$$S(\Delta t) = R(\Delta t) \circ A(\Delta t).$$

Advection A specificity of the Lattice-Boltzmann method is that the advection part is solved exactly by the operator A. Given that the exact solution of the advection part satisfies

$$\begin{aligned}
\mathbf{F}_1(x, y, t + \Delta t) &= \mathbf{F}_1(x - \lambda \Delta t, y, t), \\
\mathbf{F}_2(x, y, t + \Delta t) &= \mathbf{F}_2(x + \lambda \Delta t, y, t), \\
\mathbf{F}_3(x, y, t + \Delta t) &= \mathbf{F}_3(x, y - \lambda \Delta t, t), \\
\mathbf{F}_4(x, y, t + \Delta t) &= \mathbf{F}_4(x, y + \lambda \Delta t, t),
\end{aligned}$$

this is made possible by setting $h = \lambda \Delta t$ ie $\Delta t = \frac{h}{\lambda}$, where $h = \Delta x = \Delta y$ denotes the size of the spatial discretization. For $A(\Delta t)$ to perform the advection of $\mathbf{F}$ on a single timestep is then simply a matter of shifting each $\mathbf{F}_i$ of one cell in the right direction. Obviously ghost cells have to be added on one side of the (rectangular) domain for each $\mathbf{F}_i$, in a way that complies with the boundary conditions of the problem considered.

Relaxation For the relaxation part, we use the following operator:

$$R(\Delta t)(\mathbf{F}) = \mathbf{F} + \Omega(\mathbf{u})(\mathbf{F}^{\text{eq}}(\mathbf{u}) - \mathbf{F}),$$

where $\Omega(\mathbf{u}) \in \mathbb{R}^{4s \times 4s}$ is a matrix whose design is the aim of this paper, and is discussed in more details below. Note that for instance the choice $\Omega(\mathbf{u}) = \frac{\Delta t}{\varepsilon} R(\mathbf{u})$ corresponds to an explicit Euler scheme, but in what follows we define Ω independently of R and ε, which we have not specified anyway. For the analysis of the residual error, whose result is given below, we assume that the matrix $\Omega(\mathbf{u})$ acts seperately on the moments $\mathbf{v} = (\mathbf{v}_x, \mathbf{v}_y)$ and $\mathbf{e}$, by having the following form:

$$\Omega(\mathbf{u}) = M^{-1} \left[\begin{array}{c|cc|c} 0 & 0 & 0 & 0 \\ \hline 0 & \multicolumn{2}{c|}{\multirow{2}{*}{$\Omega_{\mathbf{v}}(\mathbf{u})$}} & 0 \\ 0 & & & 0 \\ \hline 0 & 0 & 0 & \Omega_{\mathbf{e}}(\mathbf{u}) \end{array} \right] M, \quad \Omega_{\mathbf{v}} : \mathbb{R}^s \to \mathbb{R}^{2s \times 2s}, \Omega_{\mathbf{e}} : \mathbb{R}^s \to \mathbb{R}^{s \times s}, \tag{5.3}$$

with $\Omega_{\mathbf{v}}(\mathbf{u})$ and $\Omega_{\mathbf{e}}(\mathbf{u})$ invertible. Note that the value of what would be $\Omega_{\mathbf{u}}(\mathbf{u})$ in the first block, here set to zero, has no incidence on the result since it is multiplied by $\mathbf{u}^{\text{eq}}(\mathbf{u}) - \mathbf{u} = 0$.

Residual error in the limit $\Delta t \to 0$ Under the assumptions that $\Omega(\mathbf{u})$ has the form above and that the numerical solution is smooth enough, it can be shown (cf appendix 5.A) that the residual error on $\mathbf{u}$ behaves as

$$\partial_t \mathbf{u} + \partial_x \mathbf{f}_x(\mathbf{u}) + \partial_y \mathbf{f}_y(\mathbf{u}) = \Delta t \ \nabla\cdot(D(\bar{\mathbf{u}})\nabla\bar{\mathbf{u}}) + O(\Delta t^2), \tag{5.4}$$

with

$$D(\bar{\mathbf{u}}) = \left(\Omega_{\mathbf{v}}(\bar{\mathbf{u}})^{-1} - \tfrac{1}{2}I\right)\left(\tfrac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\bar{\mathbf{u}})\right), \tag{5.5}$$

where $\bar{\mathbf{u}}$ is the exact solution of (5.1), $\nabla\bar{\mathbf{u}} = \begin{bmatrix}\partial_x\bar{\mathbf{u}} & \partial_y\bar{\mathbf{u}}\end{bmatrix}^T$ and $\mathrm{Jac}(\mathbf{f})^{②}$ denotes the following matrix:

$$\mathrm{Jac}(\mathbf{f})^{②} = \begin{bmatrix}\mathrm{Jac}(\mathbf{f}_x)^2 & \mathrm{Jac}(\mathbf{f}_x)\mathrm{Jac}(\mathbf{f}_y)\\ \mathrm{Jac}(\mathbf{f}_y)\mathrm{Jac}(\mathbf{f}_x) & \mathrm{Jac}(\mathbf{f}_y)^2\end{bmatrix}.$$

For the matrix $\frac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\bar{\mathbf{u}})$ to always be positive, the amplitude λ of the velocities needs to satisfy

$$\frac{\lambda^2}{2} \geq \mu_{\max},$$

where $\mu_{\max}$ denotes the highest value, over space and time, of the maximal eigenvalue of $\mathrm{Jac}(\mathbf{f})^{②}(\bar{\mathbf{u}})$ —which is also the maximal eigenvalue of $\mathrm{Jac}(\mathbf{f}_x)(\bar{\mathbf{u}})^2 + \mathrm{Jac}(\mathbf{f}_y)(\bar{\mathbf{u}})^2$. Considering that $\Delta t = \frac{h}{\lambda}$, this is equivalent to the CFL-like condition $\Delta t \leq \frac{h}{\sqrt{2\mu_{\max}}}$.

Numerical relaxation matrix $\Omega(\mathbf{u})$ A standard choice for the relaxation matrix $\Omega(\mathbf{u})$ is that of a constant scalar matrix:

$$\Omega(\mathbf{u}) = \omega I, \quad \omega \in [1, 2].$$

In the particular case where $\omega = 2$, the factor $\Omega_{\mathbf{v}}(\bar{\mathbf{u}})^{-1} - \frac{1}{2}I$ in the residual error cancels out, resulting in a scheme of order 2. For $\omega < 2$, the numerical scheme is of order 1, with more and more diffusion as ω decreases. The counterpart for the order 2 scheme with $\omega = 2$ resides in the oscillations that appear for discontinuous solutions.

The first alternative that we consider in this paper is the use of a scalar but local matrix Ω:

$$\Omega(\mathbf{u}) = \omega(\mathbf{u})I, \quad \omega : \mathbb{R}^s \to [1, 2].$$

Such a relaxation matrix could perform better than the previous one by picking small values of ω near discontinuities to prevent the oscillations, but high values in smooth regions to limit the diffusion. For better detection of shocks, the input of ω could include the value of $\mathbf{u}$ in the neighbouring cells as well.

The second alternative that we consider is the use of full matrices for $\Omega_{\mathbf{v}}$ and $\Omega_{\mathbf{e}}$. This option is meant to tackle another issue that can arise. If the matrix $\mathrm{Jac}(\mathbf{f})^{②}(\bar{\mathbf{u}})$ is ill-conditionned, since the parameter λ adapts to the maximal eigenvalue, then the numerical scheme is bound to be excessively diffusive in the directions with lower eigenvalues. Such issue of ill-conditionning can only be resolved by a multiplication with a non-scalar matrix, thus requiring a non-scalar $\Omega_{\mathbf{v}}$ to compensate and lower the diffusion in the necessary directions. The impact of $\Omega_{\mathbf{e}}$ is not visible in our analysis of the residual error up to order 1, but our numerical experimentations showed that it could not simply be taken as a constant scalar matrix without leading to stability issues. Finally, in this case also, the input of $\Omega_{\mathbf{v}}$ and $\Omega_{\mathbf{e}}$ could include the value of $\mathbf{u}$ in the neighbouring cells.

5.3 Data-driven relaxation matrix

The aim of this work is to design a mapping between the numerical solution $\mathbf{u}$ on a given cell (plus the neighbouring ones eventually) and the matrix Ω to use in this cell for the relaxation part of the numerical scheme. Our approach relies on the use of neural networks to perform this mapping, and the use of automated differentiation of the numerical scheme to optimize the parameters of the neural network with a gradient descent algorithm.

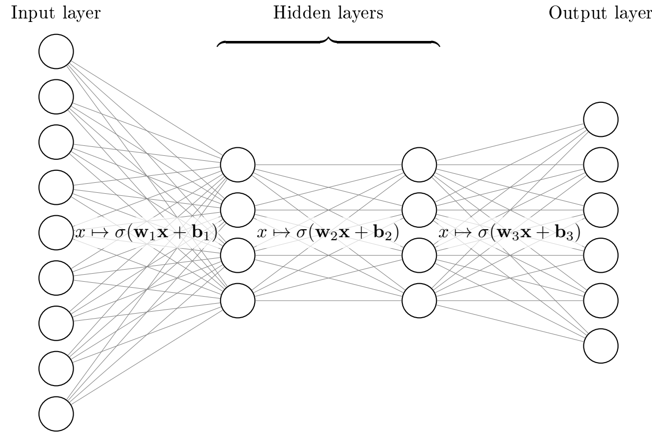


Figure 5.3.1 – Illustration of a *multilayer perceptron* with an input layer of size 9, an output layer of size 6, and two hidden layers of size 4.

Neural network A neural network π_θ is a parameterized function, whose purpose is to explore a vast space of functions by varying its parameters θ, in the hope of finding one that performs well at the desired task. Many architectures have been proposed for neural networks to be able to learn different tasks, but for this work we use a very elementary one: the multilayer perceptron (MLP), or fully connected neural network. What this means is that the neural network consists of several successive *layers*, as illustrated Figure 5.3.1, performing the following operation:

$$\mathbf{x} \mapsto a(\mathbf{w}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{x} \in \mathbb{R}^m$ is the input of the layer (and the output of the previous layer), $\mathbf{w} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^n$ are called the weights and bias respectively and contain the parameters of the layer, and $a : \mathbb{R} \to \mathbb{R}$ is a non-linear *activation* function that is applied term by term. The *size* of the layer refers to the size n of its output.

In our case, the neural network has $s \times k \times k$ inputs, where s is the number of physical conserved variables, and k is the size of the stencil, typically $k \in \{1, 3, 5\}$. We then add 2 layers of size 4 unless stated otherwise, with activation function $a = \tanh$ the hyperbolic tangent function. The output $\mathbf{y}$ is provided by a last layer with no activation function, and whose size depends on the desired form for Ω. For a scalar Ω, the output is a scalar $y \in \mathbb{R}$, and the relaxation matrix is taken as

$$\Omega = (1 + \sigma(y))I,$$

where σ denotes the sigmoid function $\sigma : x \mapsto \frac{1}{1+e^{-x}}$, so that $(1 + \sigma(y))$ falls between 1 and 2. The architecture for the scalar case is illustrated Figure 5.3.2. For a Ω with full blocks $\Omega_\mathbf{v}$ and $\Omega_\mathbf{e}$, the output $\mathbf{y}$ consists of $5s^2 + 1$ coefficients, that we put together to form two matrices $\mathbf{y}_\mathbf{v} \in \mathbb{R}^{2s\times 2s}$ and $\mathbf{y}_\mathbf{e} \in \mathbb{R}^{s\times s}$ with a remaining coefficient $y \in \mathbb{R}$. We then define $\Omega_\mathbf{v}$ and $\Omega_\mathbf{e}$ as perturbations from the scalar case:

$$\Omega_\mathbf{v} = (1 + \sigma(y))I + \mathbf{y}_\mathbf{v} \quad \text{and} \quad \Omega_\mathbf{e} = (1 + \sigma(y))I + \mathbf{y}_\mathbf{e}.$$

In particular, the neural network is easily initialized to output a scalar matrix, which is necessary to avoid stability issues. The architecture for the non-scalar case is illustrated Figure 5.3.3.

Finally, although the neural network is an MLP at the scale of the cell, it can be interpreted —and implemented— as a convolutional neural network at the scale of the whole mesh, with kernel size k on the first layer, and 1 on all the others. Also, for a stencil $k > 1$, a padding of the input is necessary for the output to be defined on the borders of the mesh. This padding is done by adding as many ghost cells as necessary around the domain, in a way that complies with the boundary conditions of the problem. For simplicity, we use periodic boundary conditions unless stated otherwise.

Optimization of the parameters Once the neural network set up and its parameters initialized, the space of parameters has to be explored via an adapted algorithm. Given the dimension of the space to explore, this algorithm is usually the minimization of a *loss function* with a gradient descent method,

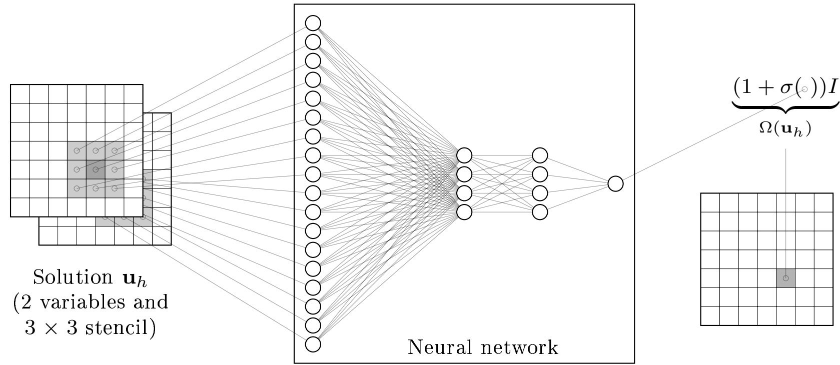


Figure 5.3.2 – Example of neural network, for a system of 2 equations solved using a scalar relaxation matrix.

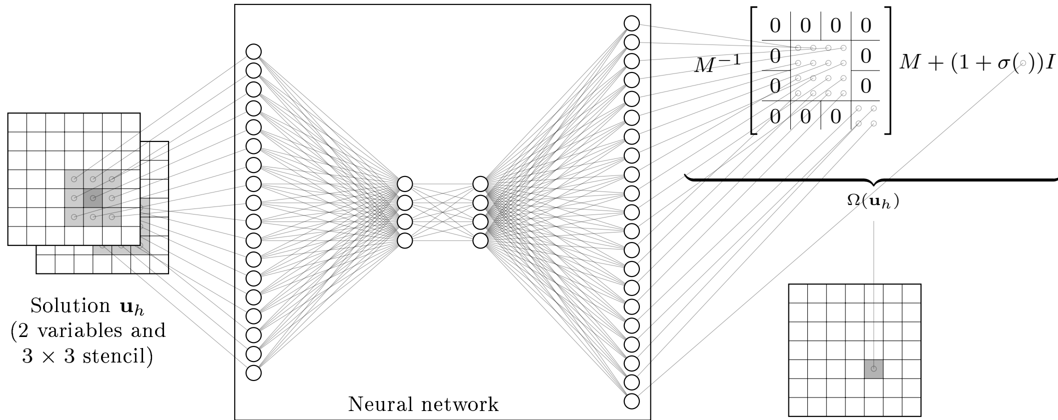


Figure 5.3.3 – Example of neural network, for a system of 2 equations solved using relaxation matrix with full matrices $\Omega_{\mathbf{v}}$ and $\Omega_{\mathbf{e}}$.

which leverages the ability of automated differentiation to compute the derivatives of neural networks with respect to their parameters. The typical framework for that is that of supervised learning, where the loss function $\mathcal{L}$ to minimize is taken as a sum of errors between outputs of the neural network and the desired outputs, over a dataset for which these desired outputs are known:

$$\mathcal{L}(\theta) = \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}} \|\pi_\theta(\mathbf{x}_i) - \mathbf{y}_i\|,$$

with $\|.\|$ a given norm, and $\mathcal{D}$ the dataset. The optimization of the parameters θ is then done with an iterative process that broadly consists in computing the gradient $\nabla_\theta \mathcal{L}(\theta)$ and updating θ with a small increment in the opposite direction:

$$\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta).$$

In our situation however, there is no dataset of optimal relaxation matrices to learn from, as what we aim to get is an upgrade of already existing techniques. Thus we cannot pick a loss directly on the output of the neural network, and instead have to measure its performance another way. What we do have is the possibility to compute reference solutions $\mathbf{u}_{\text{ref}}$ for a given set of test cases, wether using a robust numerical scheme on a fine mesh, or evaluating the exact solution when known. We can then compare the numerical solutions $\mathbf{u}_\theta$ obtained with the numerical scheme using π_θ to these reference solutions. Considering such training on M test cases, the loss would then read

$$\mathcal{L}(\theta) = \sum_{m=1}^{M} \|\mathbf{u}_{\theta,m} - \mathbf{u}_{\text{ref},m}\|,$$

with again $\|.\|$ a certain norm. For the numerical results given section 5.4, we use the L^1 norm, averaged over time. Note that this loss function involves the numerical scheme in addition to the

neural network, which therefore also needs to be compatible with automated differentiation for the computation of $\nabla_\theta \mathcal{L}$. This is easily achieved by implementing the numerical scheme using the same framework as for the neural network, e.g Tensorflow or Pytorch. Also note that the computation of the gradient requires applying the chain rule over all the operations performed both by the neural network and the numerical scheme, timestep after timestep, resulting in a very large expression. Automated differentiation is a necessary tool for the evaluation of this expression.

But however helpful automated differentiation can be, the evaluation of such a complex gradient still requires significant computational resources. A natural way of controlling the computational cost of the method is to limit the number n of consecutive timesteps performed to compute the loss, and thus the depth of the expression for the loss and its gradient. The loss function can then be based on the sum of the errors made on each sub-trajectory of size n starting from all possible points in time:

$$\mathcal{L}(\theta) = \sum_{m=1}^{M} \sum_{i=0}^{N-n} \|\mathbf{u}_{\theta,m}^{[t^i, t^{i+n}]} - \mathbf{u}_{\text{ref},m}^{[t^i, t^{i+n}]}\|,$$

where N denotes the total number of timesteps in the reference solutions, $\mathbf{u}_{\text{ref}}^{[t^i, t^{i+n}]}$ denotes the reference solution on all timesteps from t^i to t^{i+n}, and $\mathbf{u}_{\theta}^{[t^i, t^{i+n}]}$ denotes the numerical solution obtained with the LBM scheme using π_θ, starting from the reference solution at timestep t^i and on all timesteps up to t^{i+n}. In practice, the whole sum over i needs not be computed, and instead can be approximated with a Monte-Carlo method. To do so we draw a random set of times t^i for each test case, which is renewed for each gradient step. We call *batch size* the number of t^i drawn for each test case at each gradient step. In practice, this batch size does not need to be high to yield good results, and often a batch size of 1 is enough. All the steps of the training method described above are summed up in algorithm 2.

Algorithm 2: Training algorithm

1. Choose M initial conditions for the training
2. Choose the total number N of timesteps for the reference solutions
3. Compute the reference solutions $\{\mathbf{u}_{\text{ref},m}\}_{m=1}^{M}$ on all timesteps t^0, ..., t^N
4. Build the neural network π_θ and initialize its parameters θ
5. Choose the number n of consecutive timesteps for the training n
6. **while** *True* **do**
7.   **for** $k \in \{1, ..., \textit{batch size}\}$ **do**
8.   **for** $m \in \{1, ..., M\}$ **do**
9.   Draw a random starting time $t^{i(m)} \in \{t^0, ..., t^{N-n}\}$
10.   Compute the numerical solution $\mathbf{u}_{\theta,i}^{[t^{i(m)}, t^{i(m)+n}]}$
11.   **end**
12.   Compute the loss for this sample $\mathcal{L}_k(\theta) = \sum_{m=1}^{M} \|\mathbf{u}_{\theta,m}^{[t^{i(m)}, t^{i(m)+n}]} - \mathbf{u}_{\text{ref},m}^{[t^{i(m)}, t^{i(m)+n}]}\|$
13.   **end**
14.   Compute the loss for this batch $\mathcal{L}(\theta) = \sum_{k=1}^{\text{batch size}} \mathcal{L}_k(\theta)$
15.   Update the parameters θ with $\nabla \mathcal{L}(\theta)$
16.   Compute the validation loss to track progress and stop training if necessary
17. **end**

An important question regarding models trained on a given dataset is that of the generalization ability of the model, ie its ability to perform well on data other than that from the training dataset. In our case this could mean at least four things: (1) simulations with other initial conditions, (2) simulations on more consecutive timesteps than the number n used for the training, (3) simulations with other spatial resolution than h, and (4) simulations with another parameter λ. In order to measure the generalization ability of the model and avoid any overfitting to the training data, a validation loss is evaluated alongside the training loss $\mathcal{L}$. This validation loss is taken as the sum of the errors of the LBM scheme with π_θ, using different initial conditions from the training ones, and computed from t^0 up to t^N ; thus leaving aside criteria (3) and (4). Whenever the validation loss stops decreasing or

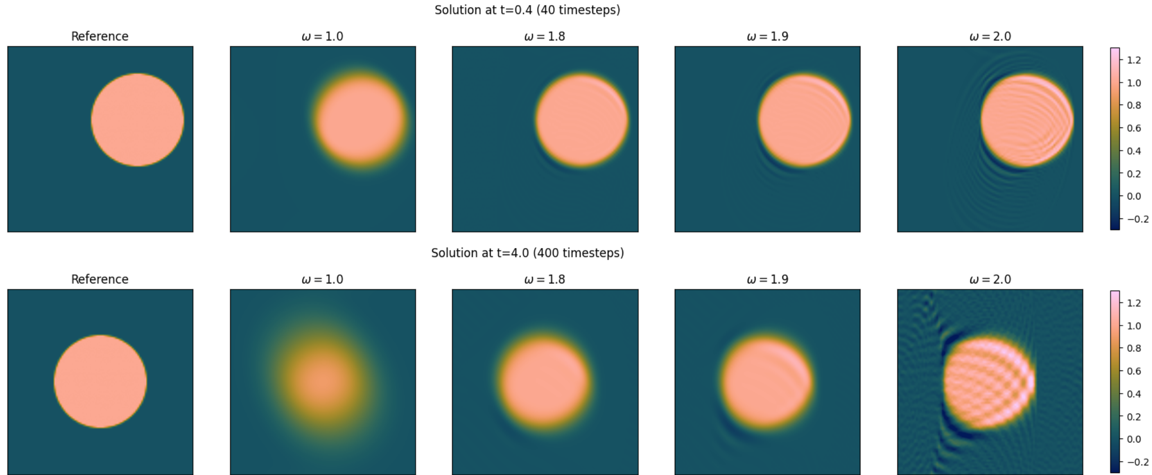


Figure 5.4.1 – (Toy example - Constant scalar relaxation) Numerical solution of the linear advection equation at $t = 0.4$ (top) and $t = 4$ (bottom) with different values for the constant ω.

even explodes due to stability issues on the long time, the training can be stopped and the parameters that yielded the lower validation loss can be saved.

5.4 Numerical results

5.4.1 Toy example

The first results presented here were obtained for the advection equation

$$\partial_t \bar{u} + \lambda_x \partial_x \bar{u} + \lambda_y \partial_y \bar{u} = 0, \tag{5.6}$$

on the domain $[-1, 1]^2$ using periodic boundary conditions. In particular, we look at the case

$$\lambda_x = 1, \lambda_y = 0.5,$$

and pick the amplitude λ for the four LBM velocities as $\lambda = 2\max(|\lambda_x|, |\lambda_y|) = 2$. The reference solution consists of the exact solution evaluated on a fine 400×400 mesh and projected on the coarser 100×100 mesh used for the numerical scheme. With this 100×100 mesh, we have $h = 0.2$ and $\Delta t = 0.1$.

In this toy example we are interested in the ability of a neural network relaxation to outperform constant scalar relaxation matrices on a given test case, regardless of the resulting relaxation's ability to generalize. This is why neural networks are trained on a single test case, and evaluated on the same test case. This test case consists in the advection of the indicator function of a disk.

We first take a look at constant scalar relaxations $\Omega(\mathbf{u}) = \omega I$. Figure 5.4.1 shows the numerical solution obtained with different values of the constant from 1 to 2. In the case $\omega = 1$, we can see that the numerical scheme is the least diffusive in the direction of the advection, and the most diffusive in the orthogonal direction, as predicted by eigen values and eigen vectors of the diffusion matrix $D(u)$, here equal to

$$D(u) = \left(\frac{1}{\omega} + \frac{1}{2}\right) \begin{bmatrix} \lambda_x & -\lambda_y \\ \lambda_y & \lambda_x \end{bmatrix}^{-1} \begin{bmatrix} \frac{\lambda^2}{2} - (\lambda_x^2 + \lambda_y^2) & 0 \\ 0 & \frac{\lambda^2}{2} \end{bmatrix} \begin{bmatrix} \lambda_x & -\lambda_y \\ \lambda_y & \lambda_x \end{bmatrix}$$

In the other cases, the oscillations caused by the discontinuity of the solution get more and more pronounced as ω gets closer to 2. Because of these oscillations, the optimal ω is not $\omega = 2$, the value for which the numerical scheme is of order 2, but a smaller value. Figure 5.4.2 shows that the optimal ω lies between 1.75 and 1.95, depending on the norm of reference but also on the final time considered. Indeed, a lower ω induces more diffusion, which is increasingly penalizing as time passes.

We now consider scalar but local relaxation matrices $\Omega(\mathbf{u}) = \omega(\mathbf{u})I$, with $\omega(\mathbf{u})$ computed by a neural network. Figure 5.4.3 shows the result of a training carried out on this single test case,

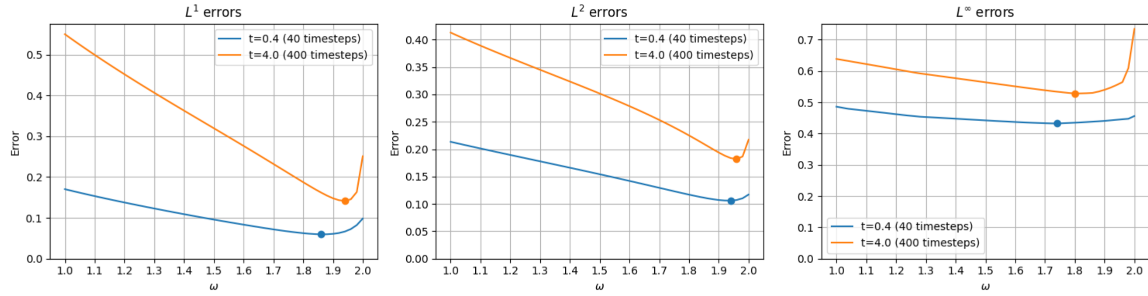


Figure 5.4.2 – (Toy example - Constant scalar relaxation) Errors of the numerical solution w.r.t the value for the constant ω, at $t = 0.4$ and $t = 4$, with different norms. The marker indicates the minimum value of the curve.

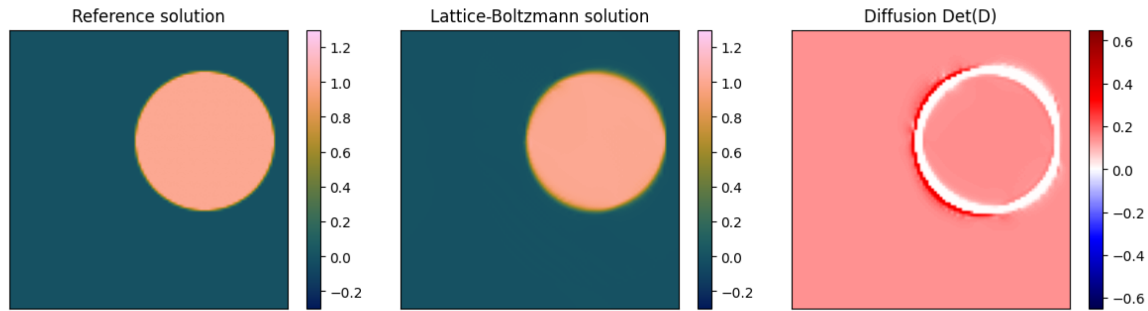


Figure 5.4.3 – (Toy example - Network scalar relaxation) Reference solution (left), Lattice-Boltzmann solution (middle) and determinant of the diffusion matrix (5.5) (right) for a scalar relaxation computed with a neural network, after 40 timesteps ($t = 0.4$). Here the value of ω is related to $\mathrm{Det}(D)$ by the relation $\mathrm{Det}(D) = \frac{3}{2}(\frac{1}{\omega} - \frac{1}{2})^2$.

with a loss computed on $n = 40$ consecutive timesteps. In this example, the neural network uses a 3×3 stencil. As could be expected, the value of the relaxation is set by the neural network so that there is no excessive diffusion on the edge of the disk in order to preserve its sharpness, but enough diffusion on the inside and the outside of the disk in order to prevent the oscillations. This results in a significant improvement on the constant scalar relaxation, as can be seen in Table 5.4.1. The latter provides quantitative results comparing different scalar relaxations, obtained by varying the stencil of the neural network and the number n of timesteps used in the loss. What stems from these results is that there seems to be no need to increase the stencil beyond 3×3 for this test case, and that the number n of timesteps should be at least 20, with little gain from going beyond.

Model	L^1	L^2	L^∞
Constant $\omega = 1.9$	6.08e-02	1.07e-01	4.41e-01
1×1 stencil, $n = 40$ timesteps	4.06e-02	8.63e-02	4.77e-01
3×3 stencil, $n = 40$ timesteps	3.19e-02	7.08e-02	3.41e-01
5×5 stencil, $n = 40$ timesteps	3.20e-02	7.14e-02	3.37e-01
7×7 stencil, $n = 40$ timesteps	3.28e-02	7.09e-02	3.38e-01
3×3 stencil, $n = 5$ timesteps	4.32e-02	7.83e-02	3.89e-01
3×3 stencil, $n = 20$ timesteps	3.34e-02	6.97e-02	3.19e-01
3×3 stencil, $n = 40$ timesteps	3.19e-02	7.08e-02	3.41e-01
3×3 stencil, $n = 80$ timesteps	3.14e-02	7.39e-02	3.80e-01

Table 5.4.1 – (Toy example - Network scalar relaxation) Errors at $t = 0.4$ (40 timesteps) using scalar relaxations trained with different parameters.

Finally, we turn to non-scalar relaxations computed by neural network. Figure 5.4.4 shows the solution obtained with such a relaxation. Once again, the diffusion is low on the edge but higher

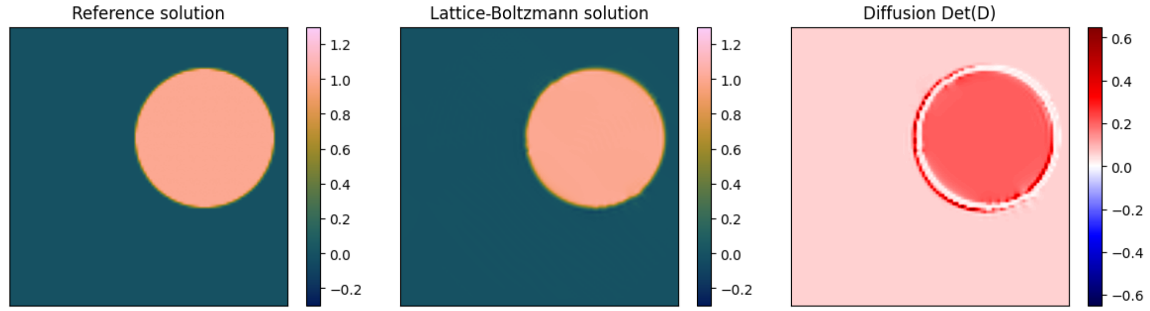


Figure 5.4.4 – (Toy example - Network non-scalar relaxation) Reference solution (left), Lattice-Boltzmann solution (middle) and determinant of the diffusion matrix (5.5) (right) for a matrix relaxation computed with a neural network, after 40 timesteps ($t = 0.4$).

around it, especially in the trail of the disk, preserving a sharper edge, although a slight deformation of the disk is visible. Quantitatively, this results in an even further improvement on the constant relaxation as can be seen Table 5.4.2.

Model	L^1	L^2	L^∞
Constant $\omega = 1.9$	6.08e-02	1.07e-01	4.41e-01
1×1 stencil, $n = 40$ timesteps	4.15e-02	8.85e-02	4.58e-01
3×3 stencil, $n = 40$ timesteps	1.83e-02	4.55e-02	2.82e-01
5×5 stencil, $n = 40$ timesteps	1.56e-02	4.00e-02	3.50e-01
7×7 stencil, $n = 40$ timesteps	1.51e-02	4.09e-02	3.15e-01
3×3 stencil, $n = 5$ timesteps	nan	nan	nan
3×3 stencil, $n = 20$ timesteps	1.74e-02	4.64e-02	5.30e-01
3×3 stencil, $n = 40$ timesteps	1.83e-02	4.55e-02	2.82e-01
3×3 stencil, $n = 80$ timesteps	1.98e-02	4.94e-02	4.23e-01

Table 5.4.2 – (Toy example - Network non-scalar relaxation) Errors at $t = 0.4$ (40 timesteps) using non-scalar relaxations trained with different parameters.

For this toy example, the duration of a training lies between 1 and a few minutes depending on the network architecture (stencil and output size), the number of consecutive timesteps n in the training, and the number of steps for the gradient descent to converge.

Finally, let us mention that as expected the neural networks trained for this test case suffer from a lack of generalization ability. Notably, even with the same initial condition, the neural network based relaxations lead to inaccurate solutions in longer times. Figure 5.4.5 shows the numerical solution at $t = 4$ with the same non-scalar relaxation used for Figure 5.4.4. This solution has a much sharper edge and very little oscillations compared to the solutions obtained with constant relaxations, but the contour of the disk is no longer circular. Since this deformation pushes the solution further away from the training data, it can only get worse as time passes. However, a training with more varied initial conditions could help in that regard and make the neural network relaxations more robust. This is what we look at in the next section.

5.4.2 Linear advection equation

In this section we take a look at the advection of indicator functions of smoothed polygonal shapes as illustrated Figure 5.4.6. We consider the same equation as in section 5.4.1, with the same velocity $(\lambda_x, \lambda_y) = (1, \frac{1}{2})$, and we use the same amplitude $\lambda = 2$ for the LBM scheme. Contrary to the previous section, here we do care about the generalization ability of the neural network, both in terms of initial conditions and final time. For this reason we use 4 initial conditions for the training, and 4 other initial conditions for the evaluation of the results, shown Figure 5.4.6. The reference solutions are computed up to $t = 4$ (400 timesteps), to allow comparisons on longer times. For the reference solutions, the exact solutions are evaluated on a finer 400×400 mesh and projected on the 100×100 mesh used for

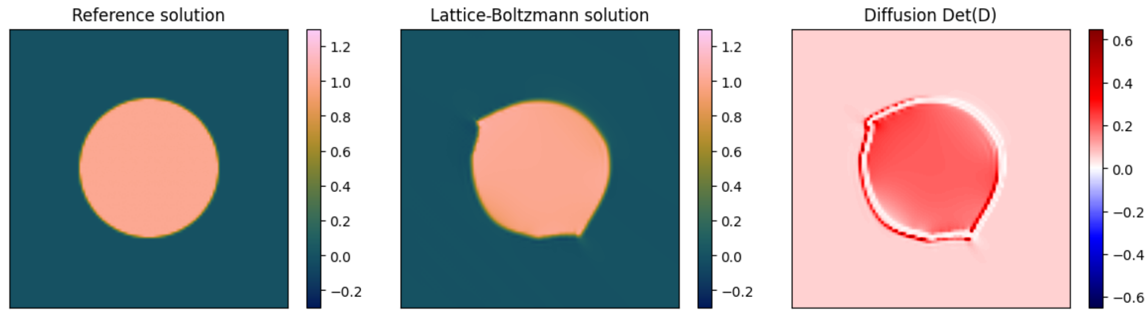


Figure 5.4.5 – (Toy example - Network non-scalar relaxation) Reference solution (left), Lattice-Boltzmann solution (middle) and determinant of the diffusion matrix (5.5) (right) for a non-scalar relaxation computed with a neural network, after 400 timesteps ($t = 4$).

Figure 5.4.6 – (Linear advection) Initial conditions for the test cases of section 5.4.2. Top row: test cases used for the training. Bottom row: test cases used for the validation.

the lattice-Boltzmann method, as in the previous section.

Similarly to the previous section, the discontinuity in the solutions makes the constant scalar relaxations $\Omega(\mathbf{u}) = \omega I$ produce oscillations when ω gets close to 2, resulting in an optimal value closer to 1.9, as shown in Figures 5.4.7 and 5.4.8.

Table 5.4.3 shows the errors of scalar relaxations trained with different parameters. Here the size of the stencil seems to have little impact on the results as long as it is at least 3×3. The same can be said for the number n of timesteps used for the training, that yields similar results for the values 40, 60 and 80. Regarding the strategy found by the neural networks, it is similar to the previous section: the diffusion is set to zero along the edge to preserve its sharpness, and increased on each side of the edge to prevent the oscillations. Although the numerical solution obtained with these models has sharper edges and less oscillations than with a constant relaxation (which results in lower numerical errors), the numerical solution develops unwanted artefacts in the long time, as can be seen in the comparison Figure 5.4.10.

Non-scalar relaxations are harder to train, as they are more prone to instability issues. This is particularly apparent in what amounts to overfitting: after the first steps of gradient descent, while the training loss keeps decreasing, the validation loss —computed with other initial conditions but also up to 400 timesteps— greatly increases or even explodes. For $n = 40$ timesteps in the training, this phenomenon happens after only a few dozen steps of gradient descent, and imposes a very early stopping of the training. However, increasing n seems to mitigate the issue, and yields better results. The numerical errors obtained on the validation test cases are summed up in Table 5.4.4.

The lack of stability is a consequence of the absence of constraint on the relaxation matrix other than its global shape (5.3). For instance, neural networks can learn relaxations that lead to negative diffusion in some places, as illustrated Figure 5.4.9. However this absence of constraint is also what allows the neural networks to explore different strategies freely, and the same relaxation that produces

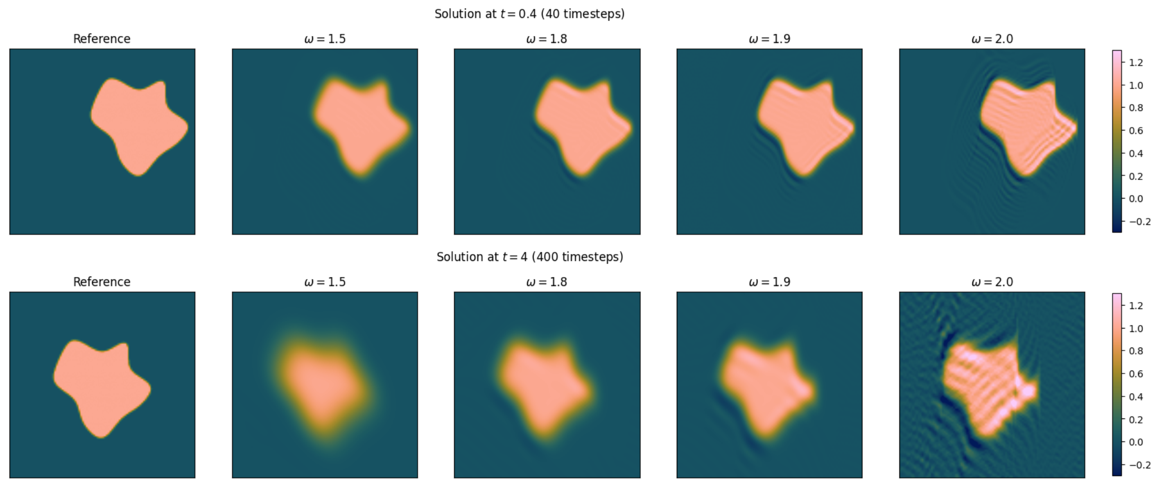


Figure 5.4.7 – (Linear advection - Constant scalar relaxation) Numerical solution of the linear advection equation at $t = 0.4$ (top) and $t = 4$ (bottom) with an initial condition from the validation test cases, using different values for the constant ω.

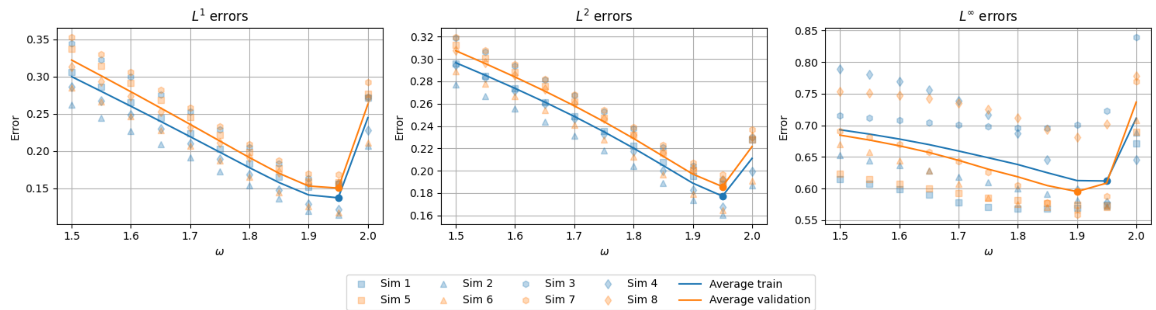


Figure 5.4.8 – (Linear advection - Constant scalar relaxation) Errors at $t = 4$ of the numerical solution w.r.t the value for the constant ω. The marker on the lines indicates the minimum value.

Model	L^1		L^2		L^∞	
	$t = 0.4$	$t = 4.0$	$t = 0.4$	$t = 4.0$	$t = 0.4$	$t = 4.0$
Constant $\omega = 1.9$	6.35e-02	1.53e-01	1.08e-01	1.97e-01	4.58e-01	5.95e-01
1×1, $n = 40$	4.38e-02	1.19e-01	8.93e-02	1.61e-01	5.07e-01	8.63e-01
3×3, $n = 40$	3.33e-02	1.01e-01	7.23e-02	1.33e-01	4.07e-01	6.61e-01
5×5, $n = 40$	3.35e-02	1.08e-01	7.19e-02	1.39e-01	3.92e-01	7.01e-01
7×7, $n = 40$	3.37e-02	1.11e-01	7.09e-02	1.44e-01	3.77e-01	7.27e-01
3×3, $n = 20$	3.64e-02	1.30e-01	7.11e-02	1.44e-01	4.05e-01	7.88e-01
3×3, $n = 40$	3.33e-02	1.01e-01	7.23e-02	1.33e-01	4.07e-01	6.61e-01
3×3, $n = 60$	3.32e-02	9.72e-02	7.38e-02	1.33e-01	4.11e-01	6.48e-01
3×3, $n = 80$	3.29e-02	8.85e-02	7.52e-02	1.33e-01	4.32e-01	6.45e-01
3×3, $n = 100$	3.32e-02	8.95e-02	7.58e-02	1.34e-01	4.30e-01	6.41e-01

Table 5.4.3 – (Advection equation - Network scalar relaxation) Average errors on the 4 validation test cases, both at $t = 0.4$ (40 timesteps) and $t = 4$ (400 timesteps), using scalar relaxations with different parameters.

negative diffusion is actually one of the best obtained in this section: it corresponds the one with a 5×5 stencil with $n = 60$. Figure 5.4.10 shows the numerical solutions obtained with this relaxation as well as others, on the four validation test cases. Once again, non-scalar relaxations produce very sharp edges and no oscillations, but tend to distort the initial shape. However, the one with a 3×3 stencil and $n = 80$ seems to be more robust in that regard, since its distortion appears to be mostly the straightening of the edges.

Model	L^1		L^2		L^∞	
	$t^{40} = 0.4$	$t^{400} = 4.0$	$t^{40} = 0.4$	$t^{400} = 4.0$	$t^{40} = 0.4$	$t^{400} = 4.0$
Constant $\omega = 1.9$	6.35e-02	1.53e-01	1.08e-01	1.97e-01	4.58e-01	5.95e-01
3×3, $n = 40$	2.17e-02	nan	5.29e-02	nan	4.76e-01	nan
5×5, $n = 40$	4.78e-02	1.74e-01	9.49e-02	2.83e-01	4.85e-01	1.73e+00
7×7, $n = 40$	5.33e-02	1.46e-01	1.01e-01	1.94e-01	4.54e-01	6.01e-01
3×3, $n = 60$	1.96e-02	5.44e-02	4.94e-02	1.28e-01	4.91e-01	1.06e+00
5×5, $n = 60$	1.21e-02	3.96e-02	3.19e-02	9.39e-02	3.21e-01	9.80e-01
7×7, $n = 60$	2.16e-02	6.62e-02	5.26e-02	1.50e-01	5.40e-01	1.36e+00
3×3, $n = 80$	1.29e-02	3.00e-02	3.53e-02	9.05e-02	3.68e-01	9.24e-01
5×5, $n = 80$	2.13e-02	5.95e-02	5.25e-02	1.35e-01	5.37e-01	1.31e+00
7×7, $n = 80$	2.27e-02	5.58e-02	5.96e-02	1.34e-01	6.76e-01	1.11e+00

Table 5.4.4 – (Advection equation - Network non-scalar relaxation) Average errors on the 4 validation test cases, both at $t = 0.4$ (40 timesteps) and $t = 4$ (400 timesteps), using non-scalar relaxations with different parameters.

In the end, the network based relaxations lead to very little oscillations, if any, and are much less diffusive than the constant relaxations ; especially the non-scalar ones that can produce very sharp edges. However, the network based relaxations can produce distorted solutions, in ways that are difficult to predict, and that could lead to instabilities in longer times than tested here.

5.4.3 Burgers equation

In this section we look at the Burgers equation

$$\partial_t \bar{u} + \lambda_x \partial_x \tfrac{\bar{u}^2}{2} + \lambda_y \partial_y \tfrac{\bar{u}^2}{2} = 0, \tag{5.7}$$

with again $\lambda_x = 1$, $\lambda_y = 0.5$, and using periodic boundary conditions on the domain $[-1, 1]^2$. We pick $\lambda = 2.2$ in order to have a bit of margin since here the wave speed depends on the solution. We use the same 100×100 mesh as in the previous sections. Once again, we use 4 initial conditions for the training, and 4 different ones for validation. These initial conditions consists of different gaussians

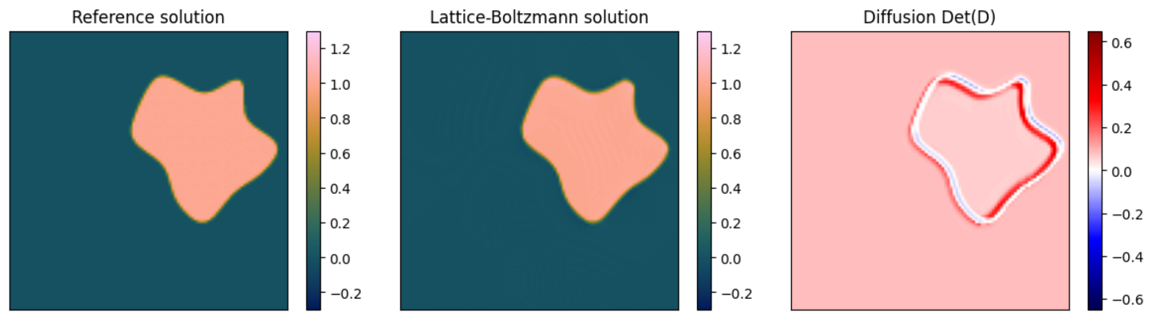


Figure 5.4.9 – (Linear advection - Network non-scalar relaxation) Reference solution (left), Lattice-Boltzmann solution with a non-scalar relaxation (middle) and corresponding diffusion Det(D) (5.5). Negative diffusion can be seen on the edge of the shape.

Figure 5.4.10 – (Linear advection) Solutions at $t = 4$ (400 timesteps) for the 4 validation test cases, obtained with different relaxations.

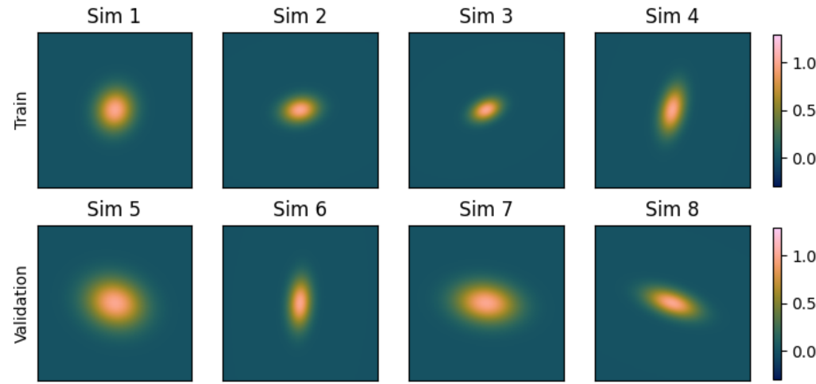


Figure 5.4.11 – (Burgers - Gaussians) Initial conditions for the test cases of section 5.4.3. Top row: test cases used for the training. Bottom row: test cases used for the validation.

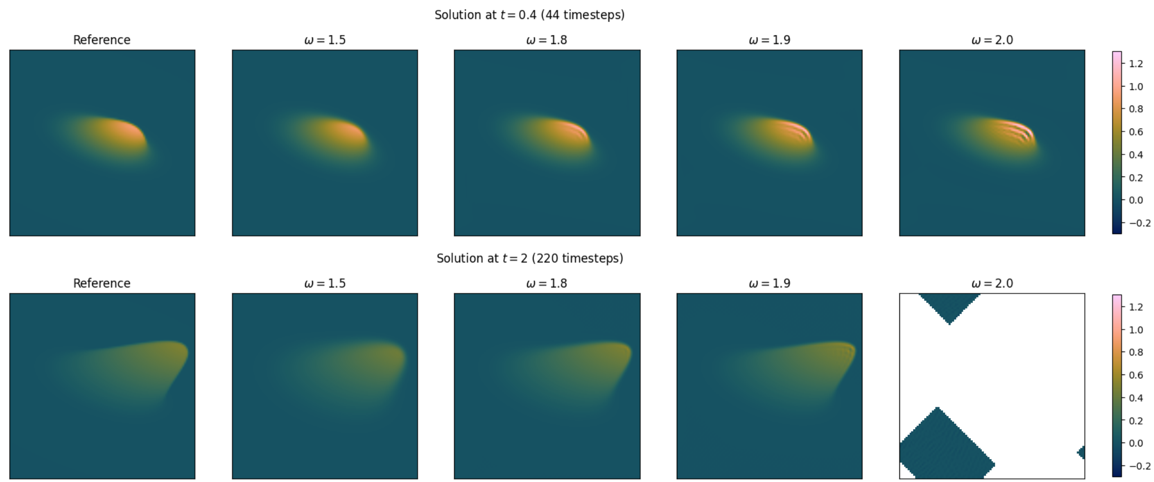


Figure 5.4.12 – (Burgers - Constant scalar relaxation) Numerical solution of the linear advection equation at $t = 0.4$ (top) and $t = 2$ (bottom) with an initial condition from the validation test cases, using different values for the constant ω.

generated randomly and shown Figure 5.4.11. The reference solutions are computed with a finite volumes MUSCL scheme on a finer 400×400 mesh then projected on the 100×100 mesh, and up to $t = 2$ (220 timesteps).

Figure 5.4.12 shows the numerical solutions obtained with a constant scalar relaxation. Once again, the closer to 2 ω is, the more oscillations can be seen in the solution, but the less diffusion. This time however, the numerical scheme is unstable at $\omega = 2$. The errors on each test case at $t = 2$ (220 timesteps) are shown Figure 5.4.13 (up to $\omega = 1.95$), and suggest an optimal value for ω around 1.9.

Contrary to the advection equation, the solutions of Burgers equation do change over time by more than a simple translation. As a consequence, the starting time t^i in algorithm 2 does make a difference on what the neural network learns from a given sub-trajectory. Thus the batch size becomes a relevant parameter than can be used to average the gradient over more than M (here 4) sub-trajectories. However, Table 5.4.5 shows very little improvement, if any, when using a batch size bigger than 1. For this reason, most of the results shown in this section use a batch size of 1. Regarding the other parameters, a 5×5 stencil seems to be the most suited to the scalar relaxation, while the non-scalar relaxation benefits from a larger 7×7 stencil (Table 5.4.6) ; for the number n of training timesteps, the non-scalar relaxation gives lower errors whith higher values, but the results are more mixed with the scalar relaxation where only the error in norm L^1 decreases as n increases.

Figure 5.4.14 compares the solution obtained with a scalar relaxation and a non-scalar relaxation. Both are visually similar, but the non-scalar relaxation seems to use less diffusion overall. These

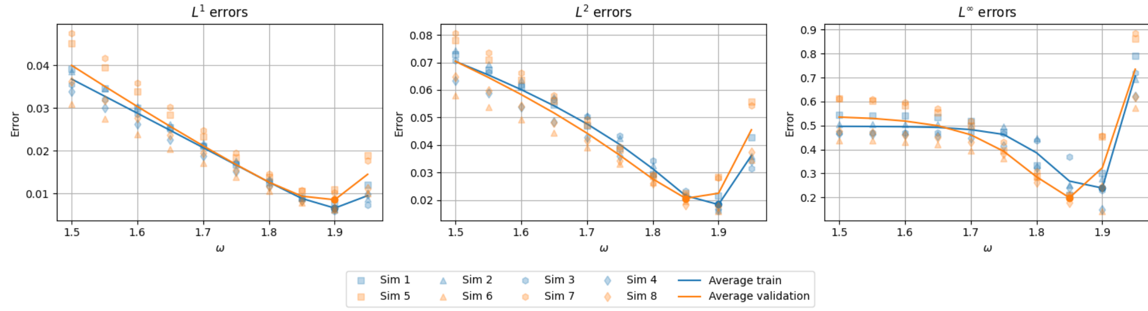


Figure 5.4.13 – (Burgers - Constant scalar relaxation) Errors of the numerical solution w.r.t the value for the constant ω at $t = 2$. The marker on the lines indicates the minimum value.

Model	L^1		L^2		L^∞	
	$t^{44} = 0.4$	$t^{220} = 2.0$	$t^{44} = 0.4$	$t^{220} = 2.0$	$t^{44} = 0.4$	$t^{220} = 2.0$
Constant $\omega = 1.9$	6.21e-03	8.53e-03	2.26e-02	2.24e-02	2.54e-01	3.23e-01
3×3, $n = 44$, batch 1	3.38e-03	5.06e-03	1.24e-02	2.06e-02	1.82e-01	3.71e-01
3×3, $n = 44$, batch 5	3.32e-03	5.44e-03	1.32e-02	2.14e-02	1.95e-01	3.72e-01
3×3, $n = 44$, batch 10	3.31e-03	5.51e-03	1.26e-02	1.93e-02	1.85e-01	3.42e-01
1×1, $n = 44$	4.12e-03	6.14e-03	1.37e-02	2.13e-02	1.83e-01	2.87e-01
3×3, $n = 44$	3.38e-03	5.06e-03	1.24e-02	2.06e-02	1.82e-01	3.71e-01
5×5, $n = 44$	2.84e-03	5.06e-03	1.10e-02	1.54e-02	1.61e-01	2.68e-01
7×7, $n = 44$	5.10e-03	6.02e-03	1.70e-02	1.88e-02	2.14e-01	2.76e-01
3×3, $n = 5$	5.31e-03	6.98e-03	1.99e-02	1.98e-02	2.29e-01	2.69e-01
3×3, $n = 11$	4.94e-03	7.56e-03	1.52e-02	2.66e-02	2.01e-01	3.51e-01
3×3, $n = 22$	4.70e-03	5.83e-03	1.69e-02	1.66e-02	1.99e-01	1.84e-01
3×3, $n = 44$	3.38e-03	5.06e-03	1.24e-02	2.06e-02	1.82e-01	3.71e-01
3×3, $n = 88$	3.32e-03	4.74e-03	1.30e-02	1.86e-02	1.84e-01	3.28e-01

Table 5.4.5 – (Burgers - Network scalar relaxations) Errors using different scalar relaxations, after 44 and 220 timesteps. The quantity shown is an average on the four validation test cases. $k \times k$ refers to the stencil of the relaxation, n is the number of consecutive timesteps in the training, and batch is the batch size, set to 1 unless stated otherwise.

Model	L^1		L^2		L^∞	
	$t^{44} = 0.4$	$t^{220} = 2.0$	$t^{44} = 0.4$	$t^{220} = 2.0$	$t^{44} = 0.4$	$t^{220} = 2.0$
Constant $\omega = 1.9$	6.21e-03	8.53e-03	2.26e-02	2.24e-02	2.54e-01	3.23e-01
1×1, $n = 44$	4.96e-03	6.70e-03	1.43e-02	1.84e-02	1.67e-01	2.47e-01
3×3, $n = 44$	3.63e-03	5.00e-03	1.19e-02	1.37e-02	1.57e-01	1.70e-01
5×5, $n = 44$	3.85e-03	4.82e-03	1.31e-02	1.29e-02	1.76e-01	1.76e-01
7×7, $n = 44$	3.35e-03	3.75e-03	1.27e-02	1.17e-02	1.68e-01	1.41e-01
3×3, $n = 11$	6.61e-03	1.62e-02	2.03e-02	4.30e-02	2.27e-01	6.04e-01
3×3, $n = 22$	6.05e-03	1.33e-02	1.76e-02	3.36e-02	1.98e-01	4.49e-01
3×3, $n = 44$	3.63e-03	5.00e-03	1.19e-02	1.37e-02	1.57e-01	1.70e-01
3×3, $n = 88$	3.36e-03	4.00e-03	1.18e-02	1.20e-02	1.59e-01	1.60e-01

Table 5.4.6 – (Burgers - Network non-scalar relaxations) Errors using different non-scalar relaxations, after 44 and 220 timesteps. The quantity shown is an average on the four validation test cases. $k \times k$ refers to the stencil of the relaxation and n is the number of consecutive timesteps in the training.

relaxations are compared to a constant relaxation in Figure 5.4.15 on all 4 validation test cases. The network based relaxations have less oscillations, if any, with no apparent downside to it, which is coherent with the errors from Tables 5.4.5 and 5.4.5 that show an improvement by a factor 2 approximatively.

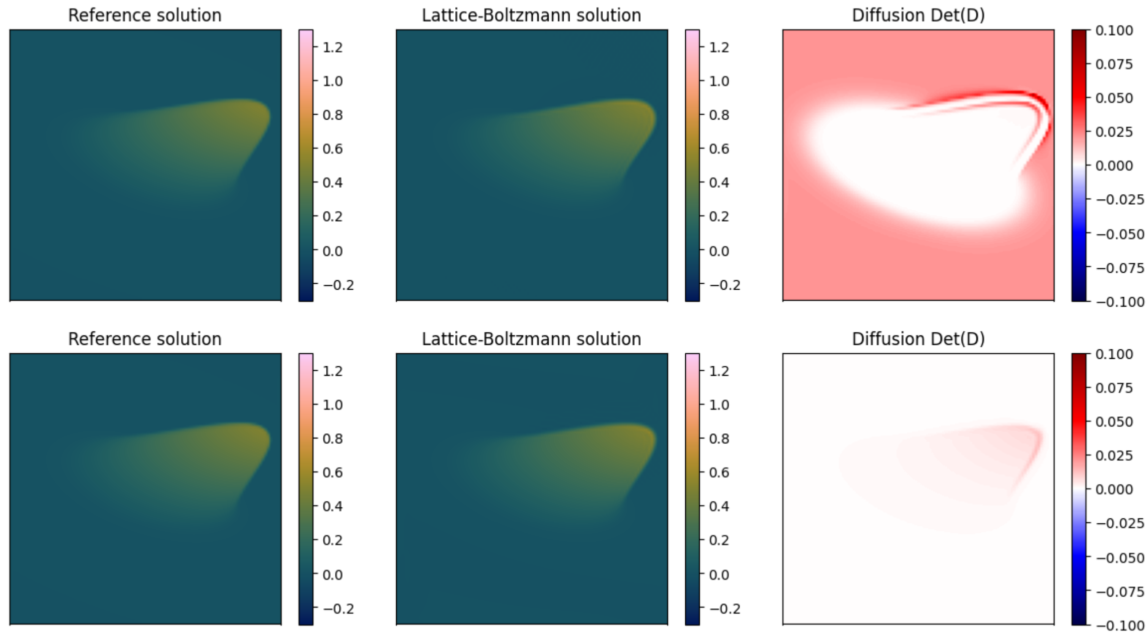


Figure 5.4.14 – (Burgers - Scalar relaxation) Reference solution (left), Lattice-Boltzmann solution with a network based relaxation (middle) and corresponding diffusion (5.7) (right), at $t = 2$. Top: scalar relaxation. Bottom: non-scalar relaxation.

5.4.4 Wave equation

In this section we look at our first system of equations:

$$\partial_t p + \nabla \cdot \mathbf{v} = 0$$
$$\partial_t \mathbf{v} + \nabla p = 0$$

which is of the form (5.1) with $\bar{\mathbf{u}} = \begin{pmatrix} p \\ \mathbf{v} \end{pmatrix}$. As in section 5.4.1, we only consider a single test case ; the initial condition is the indicator function of a disk shifted by 1 for p, and zero for $\mathbf{v}$, on the domain $[0, 1]^2$. We use a 200×200 mesh and $\lambda = 2$, resulting in $h = 0.005$ and $\Delta t = 0.0025$. The solutions are computed up to $t = 0.15$, which requires 60 timesteps.

Figure 5.4.16 shows the numerical solution obtained for p with a constant relaxation. As always, the lower the value of ω, the more diffusive the solution, and the higher the value of ω, the more oscillations. The evolution of the error on p in function of ω can be seen Figure 5.4.17, and suggests an optimal value around $\omega = 1.9$ for the errors in norm L^1 and L^2.

The neural network based relaxations are trained on all the $n = 60$ timesteps of the simulation, in order to show how much improvement can be made with local and optionally non-scalar relaxations. Table 5.4.7 compare relaxations with different parameters. The optimal stencil for the scalar relaxation seems to be 3×3, allowing to decrease the error in norm L^1 by almost a factor 2 ; on the contrary, increasing the size of the stencil up to 7×7 seems to prevent the network from learning, as it performs the same as the constant relaxation. For the non-scalar relaxations, a 3×3 stencil also performs well, while a 5×5 results in a lower error in norm L^1, but higher errors in norms L^2 and L^∞. In both cases, the error in norm L^1 is decreased by about a factor 6. The numerical solution given by some of these relaxations are compared in Figure 5.4.18: while the constant relaxation leads to significant oscillations and quite some diffusion, the scalar relaxation shows no oscillations but still some slight diffusion, and the non-scalar relaxation also shows no oscillations and very little diffusion.

In table 5.4.7 we also look at different architectures than our usual one that uses two hidden layers of size 4 (refered to as the 4×4 architecture in the table). Increasing the size of the neural network, wether by increasing the size of the hidden layers or by increasing the number of hidden layers, does allow to reach slightly lower errors in norm L^1 (which is the minimized criterion), but without any significant impact on the errors in norm L^2 or L^∞, maybe suggesting some kind of overfitting. Since

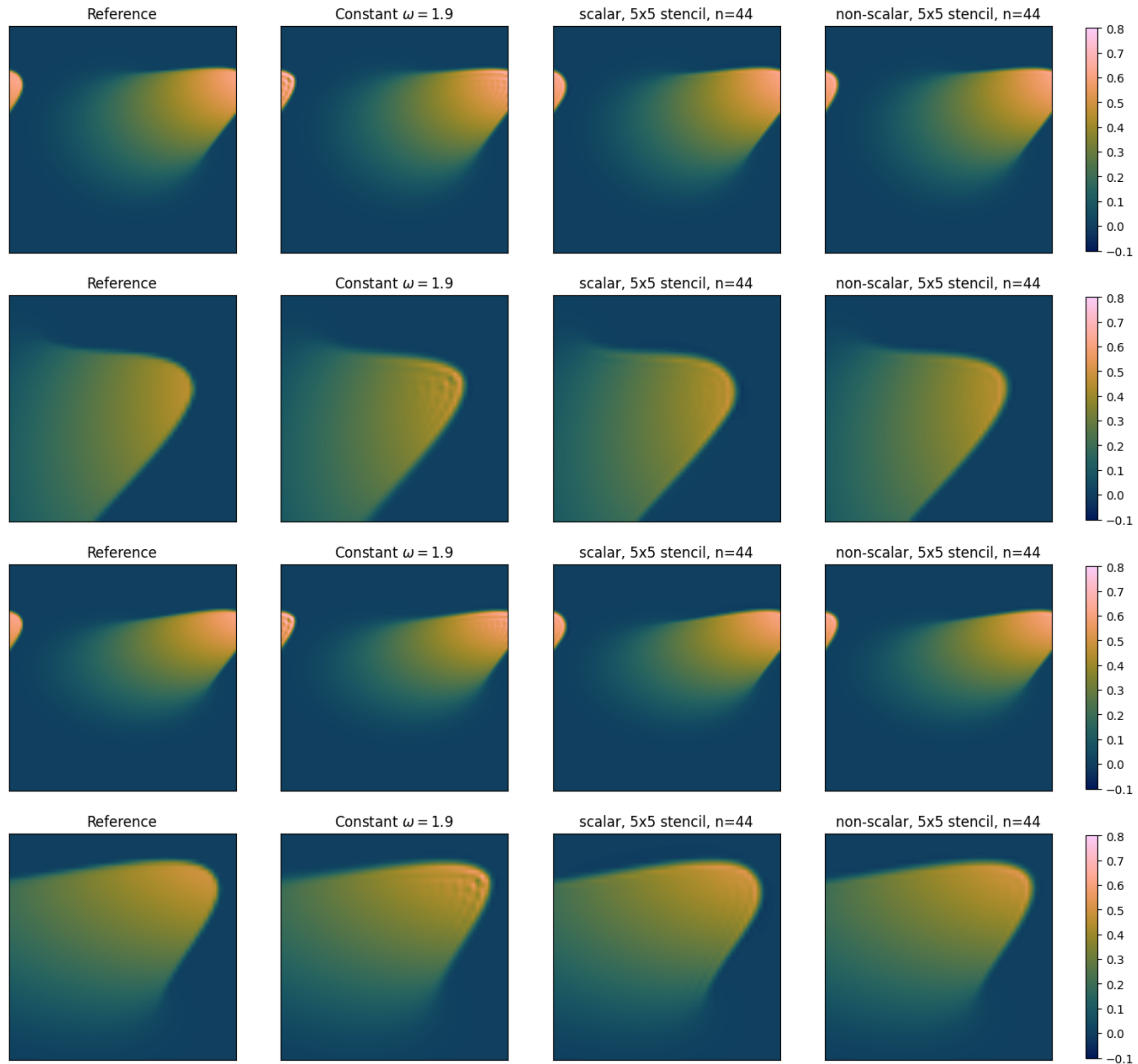


Figure 5.4.15 – (Burgers - different relaxations) Numerical solutions at $t = 2$ obtained for the 4 validation test cases with different relaxations. The second and fourth rows show an enlargement of the discontinuous portion of the solution.

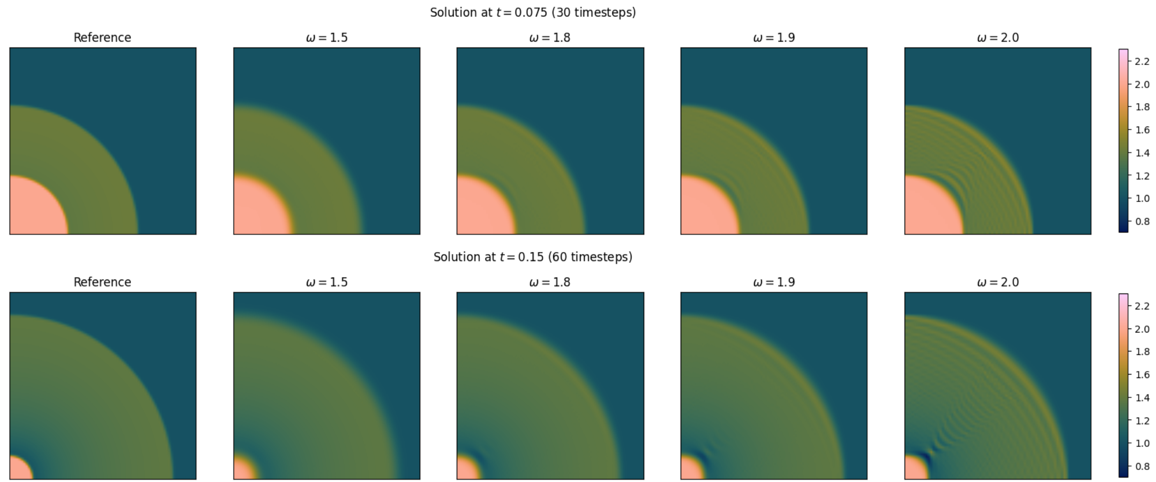


Figure 5.4.16 – (Wave - Constant scalar relaxation) Numerical solution (enlargement of the first quadrant) for p at $t = 0.075$ (top) and $t = 0.15$ (bottom), using different values for the constant ω.

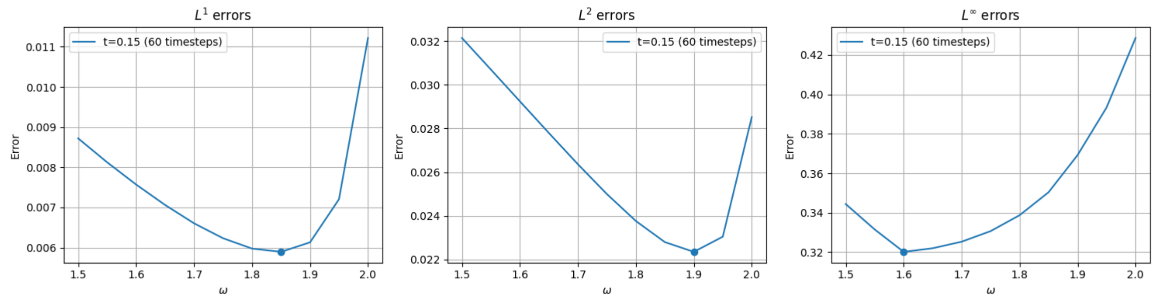


Figure 5.4.17 – (Wave - Constant scalar relaxation) Errors on p of the numerical solution w.r.t the value for the constant ω at $t = 0.15$ (60 timesteps). The marker on the lines indicates the minimum value.

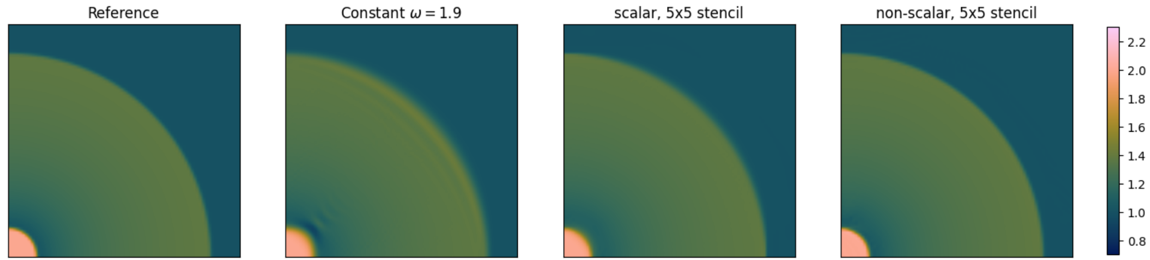


Figure 5.4.18 – (Wave - different relaxations) Numerical solutions (enlargement of the first quadrant) for p at $t = 0.15$ obtained with different relaxations.

the increase in size of the neural network comes with a higher computational cost, these results justify the use of the 4×4 architecture for this test case.

Model	L^1	L^2	L^∞
Constant $\omega = 1.9$	6.13e-03	2.24e-02	3.69e-01
Scalar, 1×1 stencil	3.77e-03	1.77e-02	2.87e-01
Scalar, 3×3 stencil	3.20e-03	1.40e-02	3.33e-01
Scalar, 5×5 stencil	3.49e-03	1.60e-02	2.67e-01
Scalar, 7×7 stencil	5.90e-03	2.31e-02	3.46e-01
Scalar, 4×4 architecture	3.20e-03	1.40e-02	3.33e-01
Scalar, 8×8 architecture	3.12e-03	1.42e-02	2.87e-01
Scalar, 16×16 architecture	3.05e-03	1.45e-02	3.15e-01
Scalar, 32×32 architecture	2.99e-03	1.42e-02	2.85e-01
Scalar, $8 \times 8 \times 8$ architecture	3.07e-03	1.43e-02	2.95e-01
Scalar, $16 \times 16 \times 16$ architecture	3.02e-03	1.42e-02	3.03e-01
Non-scalar, 3×3 stencil	9.22e-04	3.66e-03	1.24e-01
Non-scalar, 5×5 stencil	8.95e-04	4.42e-03	1.54e-01

Table 5.4.7 – (Wave - Network relaxation) Errors on p at $t = 0.15$ (60 timesteps) using relaxations trained with different parameters. The architecture refers to the size of the hidden layers in the neural network. When not specified, the stencil is 3×3 and the architecture 4×4. The number of training timesteps is $n = 60$ for all the relaxations.

5.5 Conclusion

This work has been devoted to the construction of relaxation matrices for the Lattice-Boltzmann method in two dimensions. While the standard method uses constant scalar matrices, our aim has been to design local relaxation matrices that depend on the solution, with the option of non-scalar matrices. We have relied on neural networks to perform this mapping between the solution and the relaxation matrices, with a simple fully-connected architecture (multilayer perceptron). Since the aim was to discover new relaxation matrices, a standard supervised approach was impossible. For this reason we have added the numerical scheme to the model to train, and set the criterion to optimize on the numerical solution. This method required the numerical scheme to be compatible with automated differentiation, which we have achieved by implementing it with the same framework as the neural network. We have experimented this approach on different equations: the linear advection equation, a non-linear equation (Burgers equation), and a system of linear equations (wave equations).

In all of these numerical experiments, the local relaxations performed significantly better than the standard constant relaxations, displaying much less diffusion and almost no oscillations. The non-scalar relaxation matrices further improved the results, but have proven quite prone to stability issues, making them harder to train. The numerical experiments have also looked at some parameters of the method, and have shown that the models perform best when using at least a 3×3 stencil and a few dozens consecutive training timesteps. On the other hand, increasing these parameters can only

improve the results up to a certain extent, as bigger models are more prone to overfitting or vanishing gradient issues. The former could be alleviated by training the neural network on more data, but the latter is harder to address.

The limitations of our work include the limited generalization abilities of the relaxations learned. In particular, the neural networks were trained on a given spatial resolution, with no mechanism to generalize to different resolutions. Similarly, no mechanism to adapt to different velocities λ in the Lattice-Boltzmann method has been implemented. Another limitation of our work lies in the stability issues met with non-scalar relaxation matrices. Since non-scalar matrices easily lead to instabilities, it can be difficult to find a stable one by means of a gradient descent with no specific constraints. Notably, this issue has prevented us from designing a non-scalar relaxation for Euler equations so far.

Further experimentations are needed to test the potential of our simple approach, including the use of bigger datasets and different criteria to optimize. The latter could be used to add weak constraints and improve the stability of the resulting relaxation. Strong constraints could also be added to the relaxation, eg by controlling its eigenvalues. Finally, mechanism to deal with different resolutions and velocities λ need to be implemented.

Appendix

5.A Residual error in the limit $\Delta t \to 0$

In the following we denote by $A(\Delta t)$ and $R(\Delta t)$ the operators that perform the advection and the relaxation respectively:

$$A(\Delta t)(\mathbf{F}) : (x,y) \in \mathbb{R}^2 \longmapsto \begin{bmatrix} \mathbf{F}_1(x-\lambda\Delta t, y, t) \\ \mathbf{F}_2(x+\lambda\Delta t, y, t) \\ \mathbf{F}_3(x, y-\lambda\Delta t, t) \\ \mathbf{F}_4(x, y+\lambda\Delta t, t) \end{bmatrix},$$

$$R(\Delta t)(\mathbf{F}) : (x,y) \in \mathbb{R}^2 \longmapsto \mathbf{F}(x,y) + W(\mathbf{u}(x,y))(\mathbf{F}^{\text{eq}}(\mathbf{u}(x,y)) - \mathbf{F}(x,y)),$$

and $S(\Delta t)$ the composition of the two $S(\Delta t) = R(\Delta t) \circ A(\Delta t)$. This section aims at prooving the behavior of the residual error of the numerical solution provided by the scheme S, given by equation (5.4).

For the sake of simplicity, we denote by $A(\Delta t)(\mathbf{u})$, $A(\Delta t)(\mathbf{v})$, $A(\Delta t)(\mathbf{e})$ the moments of the distribution $A(\Delta t)(\mathbf{F})$, and the same goes for $R(\Delta t)$ and $S(\Delta t)$.

Also, let us note that with our choice of velocities $\boldsymbol{\lambda}_i$, the moments of $\mathbf{F}$ are given by

$$\mathbf{u} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4, \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix} = \begin{bmatrix} \lambda\mathbf{F}_1 - \lambda\mathbf{F}_2 \\ \lambda\mathbf{F}_3 - \lambda\mathbf{F}_4 \end{bmatrix}, \quad \mathbf{e} = \lambda^2(\mathbf{F}_1 + \mathbf{F}_2 - \mathbf{F}_3 - \mathbf{F}_4).$$

The identities that follow are also used later on:

$$\lambda^2\mathbf{u} + \mathbf{e} = 2\lambda^2(\mathbf{F}_1 + \mathbf{F}_2), \quad \lambda^2\mathbf{u} - \mathbf{e} = 2\lambda^2(\mathbf{F}_3 + \mathbf{F}_4).$$

5.A.1 Expansion of one iteration of the numerical scheme

Let us get an expansion of the moments of $S(\Delta t)(\mathbf{F})$. To do so, we first expand $A(\Delta t)(\mathbf{F})$ to get

$$A(\Delta t)(\mathbf{F}) = \begin{bmatrix} \mathbf{F}_1(x-\lambda\Delta t, y) \\ \mathbf{F}_2(x+\lambda\Delta t, y) \\ \mathbf{F}_3(x, y-\lambda\Delta t) \\ \mathbf{F}_4(x, y+\lambda\Delta t) \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 - \Delta t\lambda\partial_x\mathbf{F}_1 + \Delta t^2\frac{\lambda^2}{2}\partial_{xx}\mathbf{F}_1 + O(\Delta t^3) \\ \mathbf{F}_2 + \Delta t\lambda\partial_x\mathbf{F}_2 + \Delta t^2\frac{\lambda^2}{2}\partial_{xx}\mathbf{F}_2 + O(\Delta t^3) \\ \mathbf{F}_3 - \Delta t\lambda\partial_y\mathbf{F}_3 + \Delta t^2\frac{\lambda^2}{2}\partial_{yy}\mathbf{F}_3 + O(\Delta t^3) \\ \mathbf{F}_4 + \Delta t\lambda\partial_y\mathbf{F}_4 + \Delta t^2\frac{\lambda^2}{2}\partial_{yy}\mathbf{F}_4 + O(\Delta t^3) \end{bmatrix},$$

from which we get the expansion of the moments of $A(\Delta t)(\mathbf{F})$ with the appropriate combinations:

$$\begin{aligned} A(\Delta t)(\mathbf{u}) &= \mathbf{u} - \Delta t(\nabla\cdot\mathbf{v}) + \tfrac{\Delta t^2}{4}\left(\lambda^2(\nabla\cdot\nabla\mathbf{u}) + (\partial_{xx}\mathbf{e} - \partial_{yy}\mathbf{e})\right) + O(\Delta t^3) \\ A(\Delta t)(\mathbf{v}) &= \mathbf{v} - \tfrac{\Delta t}{2}\left(\lambda^2\nabla\mathbf{u} + \begin{bmatrix} \partial_x\mathbf{e} \\ -\partial_y\mathbf{e} \end{bmatrix}\right) + O(\Delta t^2) \\ A(\Delta t)(\mathbf{e}) &= \mathbf{e} + O(\Delta t). \end{aligned}$$

Here, $(\nabla\cdot\mathbf{v})$ is to be understood as $\partial_x\mathbf{v}_x + \partial_y\mathbf{v}_y \in \mathbb{R}^s$, $\nabla\mathbf{u}$ as $\begin{bmatrix} \partial_x\mathbf{u} \\ \partial_y\mathbf{u} \end{bmatrix}$, and $(\nabla\cdot\nabla\mathbf{u})$ as $\partial_{xx}\mathbf{u} + \partial_{yy}\mathbf{u}$.

Then, we write the action of $R(\Delta t)$ on the moments $\mathbf{u}$, $\mathbf{v}$, $\mathbf{e}$. With Ω of the form (5.3), multiplying $R(\Delta t)(\mathbf{F})$ by the matrix M gives

$$\begin{aligned} R(\Delta t)(\mathbf{u}) &= \mathbf{u} \\ R(\Delta t)(\mathbf{v}) &= \mathbf{v} + \Omega_{\mathbf{v}}(\mathbf{u})\big(\mathbf{f}(\mathbf{u}) - \mathbf{v}\big) \\ R(\Delta t)(\mathbf{e}) &= (I - \Omega_{\mathbf{e}}(\mathbf{u}))\mathbf{e}. \end{aligned}$$

Replacing $\mathbf{u}$, $\mathbf{v}$, $\mathbf{e}$ by the moments of $A(\Delta t)(\mathbf{F})$ will then gives the expansion of the moments of $S(\Delta t)$. First we expand $\Omega_{\mathbf{v}}(A(\Delta t)(\mathbf{u}))$, $\Omega_{\mathbf{e}}(A(\Delta t)(\mathbf{u}))$ and $\mathbf{f}(A(\Delta t)(\mathbf{u}))$:

$$\begin{aligned} \Omega_{\mathbf{v}}(\mathbf{u} - \Delta t(\nabla \cdot \mathbf{v}) + O(\Delta t^2)) &= \Omega_{\mathbf{v}}(\mathbf{u}) - \Delta t \ \mathrm{Jac}(\Omega_{\mathbf{v}})(\mathbf{u})(\nabla \cdot \mathbf{v}) + O(\Delta t^2), \\ \Omega_{\mathbf{e}}(\mathbf{u} - \Delta t(\nabla \cdot \mathbf{v}) + O(\Delta t^2)) &= \Omega_{\mathbf{e}}(\mathbf{u}) - \Delta t \ \mathrm{Jac}(\Omega_{\mathbf{e}})(\mathbf{u})(\nabla \cdot \mathbf{v}) + O(\Delta t^2), \\ \mathbf{f}(\mathbf{u} - \Delta t(\nabla \cdot \mathbf{v}) + O(\Delta t^2)) &= \mathbf{f}(\mathbf{u}) - \Delta t \ \mathrm{Jac}(\mathbf{f})(\mathbf{u})(\nabla \cdot \mathbf{v}) + O(\Delta t^2), \end{aligned}$$

and finally we get

$$S(\Delta t)(\mathbf{u}) = \mathbf{u} - \Delta t(\nabla \cdot \mathbf{v}) + \tfrac{\Delta t^2}{4}\big(\lambda^2(\nabla \cdot \nabla \mathbf{u}) + (\partial_{xx}\mathbf{e} - \partial_{yy}\mathbf{e})\big) + O(\Delta t^3) \tag{5.8}$$

$$\begin{aligned} S(\Delta t)(\mathbf{v}) = \mathbf{v} + \Omega_{\mathbf{v}}(\mathbf{u})\big(\mathbf{f}(\mathbf{u}) - \mathbf{v}\big) - \tfrac{\Delta t}{2}\Bigg[\left(\lambda^2 \nabla \mathbf{u} + \begin{bmatrix} \partial_x \mathbf{e} \\ -\partial_y \mathbf{e} \end{bmatrix}\right) + 2\mathrm{Jac}(\Omega_{\mathbf{v}})(\mathbf{u})(\nabla \cdot \mathbf{v})\big(\mathbf{f}(\mathbf{u}) - \mathbf{v}\big) \\ + \Omega_{\mathbf{v}}(\mathbf{u})\left(2\mathrm{Jac}(\mathbf{f})(\mathbf{u})(\nabla \cdot \mathbf{v}) - \left(\lambda^2 \nabla \mathbf{u} - \begin{bmatrix} \partial_x \mathbf{e} \\ -\partial_y \mathbf{e} \end{bmatrix}\right)\right)\Bigg] + O(\Delta t^2) \end{aligned} \tag{5.9}$$

$$S(\Delta t)(\mathbf{e}) = (I - \Omega_{\mathbf{e}}(\mathbf{u}))\mathbf{e} + O(\Delta t). \tag{5.10}$$

5.A.2 Expansion of the residual error

In this section we aim at writing the expansion of the numerical solution obtained with multiple iterations of $S(\Delta t)$. To this end, we now consider $\mathbf{F} = \mathbf{F}(\Delta t, t^n)$ to be the numerical solution obtained with a given timestep Δt at a time $t^n = n\Delta t$:

$$\begin{aligned} &\mathbf{F}(\Delta t, t{=}0) = \mathbf{F}_0 \text{ (given initial condition)} \\ &\forall n{\in}\mathbb{N}, \ \mathbf{F}(\Delta t, t^{n+1}) = S(\Delta t)\big(\mathbf{F}(\Delta t, t^n)\big). \end{aligned}$$

In fact, as we need t to be independant from Δt, we consider an interpolation of the function defined above (still denoted by $\mathbf{F}$), that satisfies the equality above at any time t:

$$\forall t{\in}\mathbb{R}_+, \ \mathbf{F}(\Delta t, t + \Delta t) = S(\Delta t)(\mathbf{F}(\Delta t, t)) \tag{5.11}$$

Such an interpolation can be obtained by specifying $\mathbf{F}$ in the region $\{t < \Delta t\}$ and using the above identity iteratively to get $\mathbf{F}$ in the regions $\{n\Delta t \leqslant t < (n+1)\Delta t\}$ successively. However, here we assume that $\mathbf{F}$ has additionnal properties, namely that $\mathbf{F}$ can be continuously extended along the line $\Delta t = 0$, and that this extension is $\mathcal{C}^2$ (as a function of Δt and t) along this line. If such a function $\mathbf{F}$ exists, then in particular we can write its expansion in Δt as follows:

$$\mathbf{F}(\Delta t, t) = \mathbf{F}^0(t) + \Delta t\, \mathbf{F}^1(t) + \tfrac{\Delta t^2}{2}\mathbf{F}^2(t) + O(\Delta t^3). \tag{5.12}$$

These assumptions on $\mathbf{F}$ propagate to its moments: equation (5.11) gives

$$\forall \Delta t{\in}\mathbb{R}_+^*, \ \forall t{\in}\mathbb{R}_+, \ \mathbf{u}(\Delta t, t + \Delta t) = S(\Delta t)(\mathbf{u}(\Delta t, t)) \tag{5.13}$$

$$\mathbf{v}(\Delta t, t + \Delta t) = S(\Delta t)(\mathbf{v}(\Delta t, t)) \tag{5.14}$$

$$\mathbf{e}(\Delta t, t + \Delta t) = S(\Delta t)(\mathbf{e}(\Delta t, t)), \tag{5.15}$$

while equation (5.12) gives

$$\forall t{\in}\mathbb{R}_+, \ \mathbf{u}(\Delta t, t) = \mathbf{u}_0(t) + \Delta t\, \mathbf{u}_1(t) + \tfrac{\Delta t^2}{2}\mathbf{u}_2(t) + O(\Delta t^3) \tag{5.16}$$

$$\mathbf{v}(\Delta t, t) = \mathbf{v}_0(t) + \Delta t\, \mathbf{v}_1(t) + O(\Delta t^2) \tag{5.17}$$

$$\mathbf{e}(\Delta t, t) = \mathbf{e}_0(t) + O(\Delta t). \tag{5.18}$$

The desired result will be obtained whith the expansion of equations (5.13), (5.14), (5.15) at the limit $\Delta t \to 0$. We begin with the left hand side using (5.16), (5.17), (5.18) (the evaluation in t is implied for clarity):

$$\mathbf{u}(\Delta t, t+\Delta t) = \mathbf{u}_0 + \Delta t(\partial_t \mathbf{u}_0 + \mathbf{u}_1) + \frac{\Delta t^2}{2}(\partial_{tt}\mathbf{u}_0 + 2\partial_t \mathbf{u}_1 + \mathbf{u}_2) + O(\Delta t^3) \tag{5.19}$$

$$\mathbf{v}(\Delta t, t+\Delta t) = \mathbf{v}_0 + \Delta t(\partial_t \mathbf{v}_0 + \mathbf{v}_1) + O(\Delta t^2) \tag{5.20}$$

$$\mathbf{e}(\Delta t, t+\Delta t) = \mathbf{e}_0 + O(\Delta t). \tag{5.21}$$

For the right hand side, we use (5.16), (5.17), (5.18) in the expansion of $S(\Delta t)$, equations (5.8), (5.9), (5.10):

$$\begin{aligned} S(\Delta t)(\mathbf{u}(\Delta t, t)) = \mathbf{u}_0 + \Delta t(\mathbf{u}_1 - \nabla\cdot\mathbf{v}_0) + \frac{\Delta t^2}{2}\big(\mathbf{u}_2 - 2(\nabla\cdot\mathbf{v}_1) + \tfrac{1}{2}[\lambda^2(\nabla\cdot\nabla\mathbf{u}_0) \\ + (\partial_{xx}\mathbf{e}_0 - \partial_{yy}\mathbf{e}_0)]\big) + O(\Delta t^3) \end{aligned} \tag{5.22}$$

$$\begin{aligned} S(\Delta t)(\mathbf{v}(\Delta t, t)) &= \mathbf{v}_0 + (\Omega_{\mathbf{v}}(\mathbf{u}_0)\big(\mathbf{f}(\mathbf{u}_0) - \mathbf{v}_0\big) \\ &+ \Delta t\Bigg[\mathbf{v}_1 + \mathrm{Jac}(\Omega_{\mathbf{v}})(\mathbf{u}_0)\big(\mathbf{u}_1 - \nabla\cdot\mathbf{v}_0\big)\big(\mathbf{f}(\mathbf{u}_0) - \mathbf{v}_0\big) \\ &\quad + \Omega_{\mathbf{v}}(\mathbf{u}_0)\left(\mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\mathbf{u}_1 - \mathbf{v}_1 + \frac{1}{2}\left(\lambda^2\nabla\mathbf{u}_0 - \begin{bmatrix}\partial_x\mathbf{e}\\ -\partial_y\mathbf{e}\end{bmatrix}\right) - \mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)(\nabla\cdot\mathbf{v}_0)\right) \\ &\quad - \frac{1}{2}\left(\lambda^2\nabla\mathbf{u}_0 - \begin{bmatrix}\partial_x\mathbf{e}\\ -\partial_y\mathbf{e}\end{bmatrix}\right)\Bigg] + O(\Delta t^2) \end{aligned} \tag{5.23}$$

$$S(\Delta t)(\mathbf{e}(\Delta t, t)) = (I - \Omega_{\mathbf{e}}(\mathbf{u}_0))\mathbf{e}_0 + O(\Delta t) \tag{5.24}$$

By identifying the constant terms on the left and right hand side in (5.13), (5.14), (5.15), and using that $\Omega_{\mathbf{v}}$ and $\Omega_{\mathbf{e}}$ are assumed invertible, it comes:

$$\begin{aligned} (\mathbf{u}_0 &= \mathbf{u}_0) \\ \mathbf{v}_0 &= \mathbf{f}(\mathbf{u}_0) \\ \mathbf{e}_0 &= 0. \end{aligned}$$

By identifying the terms in Δt in (5.13), (5.14), and using the previous equalities, we get:

$$\partial_t \mathbf{u}_0 + \nabla\cdot\mathbf{f}(\mathbf{u}_0) = 0 \tag{5.25}$$

$$\partial_t \mathbf{f}(\mathbf{u}_0) = \Omega_{\mathbf{v}}(\mathbf{u}_0)\left[\mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\mathbf{u}_1 - \mathbf{v}_1 + \frac{\lambda^2}{2}\nabla\mathbf{u}_0 - \mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)(\nabla\cdot\mathbf{f}(\mathbf{u}_0))\right] - \frac{\lambda^2}{2}\nabla\mathbf{u}_0 \tag{5.26}$$

Since

$$\mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)(\nabla\cdot\mathbf{f}(\mathbf{u}_0)) = \begin{pmatrix}\mathrm{Jac}(\mathbf{f}_x)(\mathbf{u}_0)\\ \mathrm{Jac}(\mathbf{f}_y)(\mathbf{u}_0)\end{pmatrix}\begin{pmatrix}\mathrm{Jac}(\mathbf{f}_x)(\mathbf{u}_0) & \mathrm{Jac}(\mathbf{f}_y)(\mathbf{u}_0)\end{pmatrix}\nabla\mathbf{u}_0 = \mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\nabla\mathbf{u}_0,$$

and

$$\partial_t\mathbf{f}(\mathbf{u}_0) = \mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\partial_t\mathbf{u}_0 = \mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)(-\nabla\cdot\mathbf{f}(\mathbf{u}_0)) = -\mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\nabla\mathbf{u}_0,$$

equation (5.26) gives

$$\mathbf{v}_1 = \mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\mathbf{u}_1 - \big(\Omega_{\mathbf{v}}^{-1}(\mathbf{u}_0) - I\big)\big(\tfrac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\mathbf{f})(\mathbf{u}_0)\big)\nabla\mathbf{u}_0.$$

Let us note that we also have

$$\partial_{tt}\mathbf{u}_0 = \partial_t(-\nabla\cdot\mathbf{f}(\mathbf{u}_0)) = -\nabla\cdot(\partial_t\mathbf{f}(\mathbf{u}_0)) = \nabla\cdot(\mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\nabla\mathbf{u}_0).$$

These last two equalities can be used after identifying the terms in Δt^2 in (5.13) to get

$$\partial_t\mathbf{u}_1 + \nabla\cdot\mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\mathbf{u}_1 = \nabla\cdot\left[\big(\Omega_{\mathbf{v}}^{-1}(\mathbf{u}_0) - \tfrac{1}{2}I\big)\left(\frac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\right)\nabla\mathbf{u}_0\right]. \tag{5.27}$$

Finally, combining equations (5.25) and (5.27) gives

$$\partial_t(\mathbf{u}_0 + \Delta t\,\mathbf{u}_1) + \nabla\cdot(\mathbf{f}(\mathbf{u}_0) + \Delta t\,\mathrm{Jac}(\mathbf{f})(\mathbf{u}_0)\mathbf{u}_1) = \Delta t\,\nabla\cdot\left[\big(\Omega_{\mathbf{v}}^{-1}(\mathbf{u}_0) - \tfrac{1}{2}I\big)\left(\frac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\right)\nabla\mathbf{u}_0\right],$$

which means that

$$\partial_t\mathbf{u} + \nabla\cdot\mathbf{f}(\mathbf{u}) = \Delta t\,\nabla\cdot\left[\big(\Omega_{\mathbf{v}}^{-1}(\mathbf{u}_0) - \tfrac{1}{2}I\big)\left(\frac{\lambda^2}{2}I - \mathrm{Jac}(\mathbf{f})^{②}(\mathbf{u}_0)\right)\nabla\mathbf{u}_0\right] + O(\Delta t^2).$$

This is the desired result since $\mathbf{u}_0$ satisfies (5.25), meaning that $\mathbf{u}_0$ is the exact solution of (5.1).

Chapitre 6

Conclusion

Cette thèse a exploré plusieurs manières d'exploiter les récents progrès concernant les réseaux de neurones dans le cadre de la simulation numérique de fluides. Le chapitre 1 a notamment introduit les problèmes que tentent d'adresser les travaux effectués : d'une part la construction de fermetures pour le passage du modèle cinétique au modèle fluide, et d'autre part le design d'éléments de schémas numériques, par une approche basée sur les données issues de simulations.

Les chapitres 2 et 3 se sont intéressés à la problématique de fermeture au modèle fluide, pour l'équation de Vlasov en une dimension pour le premier, et l'équation de Boltzmann en deux dimensions pour le deuxième. Dans les deux cas, il a été possible d'entrainer des réseaux de neurones convolutifs à apprendre une correspondance entre les variables du système et les variables de la fermeture, à partir de données issues de simulations du modèle cinétique. Mais l'intégration de ces réseaux de neurones aux schémas numériques a posé des problèmes de stabilité. En dimension une, le lissage de la sortie du réseau de neurones a fourni une certaine stabilité, qui a permis de bénéficier de la fermeture avec différentes résolutions en espace. En deux dimensions, un tel lissage n'a pas suffi, et les méthodes proposées pour stabiliser le schéma numérique n'ont pas permis une utilisation de la fermeture à une résolution différente de celle des données d'apprentissage. En revanche, alors que l'ajout de réseaux de neurones dans le modèle fluide l'a rendu aussi couteux que le modèle cinétique en dimension une, en dimension deux il a permis une réduction du temps de calcul jusqu'à un facteur 10. Ceci illustre le potentiel de la méthode, sous réserve de résoudre les problèmes de stabilité mentionnés précédemment.

Le chapitre 4 a porté sur la construction d'un coefficient de viscosité artificielle pour la méthode Galerkin discontinu, en dimension une. Une attention particulière a été portée sur l'algorithme d'entrainement, qui diffère de l'apprentissage supervisé classique, pour faire porter le critère d'optimisation non pas sur la sortie du réseau de neurones directement, mais sur la solution numérique qui découle de son usage dans le schéma numérique. Ainsi aucun coefficient de viscité de référence n'est nécessaire pour l'entrainement du réseau de neurones. Cette méthode a permis l'apprentissage de coefficients de viscosité pour l'équation d'advection, l'équation de Burgers et l'équation d'Euler, avec des performances similaires aux coefficients pré-existants. Ces résultats montrent la capacité des réseaux de neurones à apprendre des quantités pour les schémas numériques, sans autre référence que des solutions approchées de l'équation étudiée.

Le chapitre 5 a employé la même approche que le chapitre 4 mais pour la construction d'une matrice de relaxation pour la méthode Lattice-Boltzmann, en dimension deux. Les résultats obtenus avec cette méthode pour l'équation d'advection, l'équation de Burgers et l'équation des ondes ont montré des gains significatifs par rapport à la relaxation standard pour la méthode Lattice-Boltzmann, aussi bien sur la diffusion que sur les oscillations. Ces gains sont dûs non seulement à l'usage de relaxations locales, mais aussi à l'usage de matrices non scalaires, qui ont pu être apprises par les réseaux de neurones malgré l'absence de référence pour de telles matrices de relaxations. Cependant des problèmes de stabilité font encore obstacle à l'apprentissage d'une relaxation non scalaire pour l'équation d'Euler.

Un point commun à ces quatre projets a été la confrontation régulière à des problèmes de stabilité lors de l'utilisation de réseaux de neurones dans les schémas numériques. Lorsque les réseaux de neurones ont été entrainés séparément du schéma numérique (chapitres 2 et 3), ces problèmes sont intervenus au moment de leur intégration ; lorsqu'ils ont été entrainés *avec* le schéma numérique (chapitres 4 et 5), les difficultés sont survenues dès l'apprentissage. Dans ce deuxième cas, malgré des précautions par exemple sur l'initialisation des réseaux de neurones, il est toujours possible que la

descente de gradient mène systématiquement à une solution instable. Ainsi il ressort de ces travaux que la mise au point de méthodes de stabilisation est un enjeu important pour l'exploitation du plein potentiel des réseaux de neurones au sein des schémas numériques.

On explore différentes applications des réseaux de neurones pour les méthodes numériques, dans le contexte de la simulation de fluides ou de plasmas. Une première application est l'apprentissage d'une fermeture pour un modèle macroscopique, à partir de données issues d'un modèle cinétique. On donne des résultats numériques pour l'équation de Vlasov-Poisson en 1D et l'équation de Boltzmann en 2D. Une deuxième application est l'apprentissage de paramètres dépendants du problème considéré dans les schémas numériques. On apprend ainsi un coefficient de viscosité artificielle pour un schéma de Galerkin discontinu, et une matrice de relaxation pour la méthode de Lattice-Boltzmann.

Different applications of neural networks for numerical methods are explored, in the context of fluid or plasma simulation. A first application is the learning of a closure for a macroscopic model, based on data from a kinetic model. Numerical results are given for the Vlasov-Poisson equation in 1D and the Boltzmann equation in 2D. A second application is the learning of problem-dependent parameters in numerical schemes. In this way, an artificial viscosity coefficient is learned for a discontinuous Galerkin scheme, and a relaxation matrix for the Lattice-Boltzmann method.

INSTITUT DE RECHERCHE MATHÉMATIQUE AVANCÉE
UMR 7501
Université de Strasbourg
CNRS
IRMA, UMR 7501
7 rue René Descartes
F-67000 STRASBOURG

Tél. 03 68 85 01 29

irma.math.unistra.fr
irma@math.unistra.fr

IRMA Institut de Recherche Mathématique Avancée
cnrs
Université de Strasbourg

IRMA 2023/10
http://tel.archives-ouvertes.fr/tel-04314459