Forschungsbericht 2025-38

Exploration of Physics-Informed Neural Networks for Compressible Flows in Aerodynamics

Simon Wassing

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und Strömungstechnik
Braunschweig

Forschungsbericht 2025-38

Exploration of Physics-Informed Neural Networks for Compressible Flows in Aerodynamics

Simon Wassing

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und Strömungstechnik
Braunschweig

179 Seiten
62 Bilder
21 Tabellen
132 Literaturstellen

Deutsches Zentrum für Luft- und Raumfahrt

Herausgeber:

Deutsches Zentrum
für Luft- und Raumfahrt e. V.
Wissenschaftliche Information
Linder Höhe
D-51147 Köln

ISSN 1434-8454
ISRN DLR-FB-2025-38
Erscheinungsjahr 2025
DOI: 10.57676/cqme-zr79

Aerodynamik, tiefe neuronale Netze, kompressible Strömung, partielle Differentialgleichungen, numerische Simulation, Stoßwellen, physik-informierte Neuronale Netze

Simon Wassing
DLR, Institut für Aerodynamik und Strömungstechnik, Braunschweig

Untersuchung von Physik-informierten neuronalen Netzen für kompressible Strömungen in der Aerodynamik
Technische Universität Braunschweig

Die Lösung von partiellen Differentialgleichungen ist für viele Disziplinen in Wissenschaft und Technik von Bedeutung. Luftströmungen, z.B. über Tragflächen, können mit Hilfe nichtlinearer Transportgleichungen, wie den kompressiblen Navier-Stokes-Gleichungen oder, unter Vernachlässigung viskoser Effekte, den kompressiblen Euler-Gleichungen, modelliert werden. Numerische Methoden werden verwendet, um aerodynamischen Strömungsphänomene einschließlich Stoßwellen, deren Erfassung besonders schwierig ist, vorherzusagen. Daraus lassen sich die resultierenden aerodynamischen Kräfte und Momente ableiten. Daher sind diese numerischen Verfahren zu einem wichtigen Werkzeug für den aerodynamischen Entwurf von Fluggeräten geworden. Etablierte Verfahren, wie die finite Volumen Methode, lösen die Gleichungen an diskreten Punkten in der Strömungsdomäne. Diese Methoden erfordern feine Gitter, die zu Millionen von Freiheitsgraden führen, was selbst für die heutige Rechenhardware eine Herausforderung darstellt. Insbesondere in Szenarien, die ein mehrfaches Aufrufen des Strömungslösers erfordern, wie z.B. bei der Entwurfsoptimierung, können die kumulativen Rechenkosten zu aufwendig sein. Alternative numerische Verfahren, die künstliche neuronale Netze als globale kontinuierliche Ansatzfunktionen für die Lösungsannäherung verwenden, gewinnen in letzter Zeit als Löser partieller Differentialgleichungen an Bedeutung. Diese so genannten physik-informierten neuronalen Netze sind tiefe neuronale Netze, die mit einer Kostenfunktion trainiert werden, welche die partiellen Differentialgleichungen direkt einbezieht. Insbesondere für technische Anwendungen ist dieser Ansatz vielversprechend, da ein einziges tiefes neuronales Netz viele klassische Simulationen in Szenarien ersetzen könnte, die ein mehrfaches Aufrufen des Strömungslösers erfordern. Allerdings ist die Auflösung von Stoßwellen eine besondere Herausforderung für diesen Ansatz.

Diese Arbeit untersucht die Anwendung von physikalisch informierten neuronalen Netzen als alternativen Ansatz zur Lösung partieller Differentialgleichungen mit einem Fokus auf kompressible Strömungen, die durch die Euler-Gleichungen beschrieben werden. Inspiriert von klassischen Methoden der numerischen Strömungsmechanik werden Stabilisierungstechniken für Stoßwellen entwickelt, die auf künstlicher Viskosität und Sensorfunktionen basieren und mit Netztransformationen kombiniert werden. Diese Techniken werden an kompressiblen Strömungsproblemen im Unterschall, Transschall und Überschall validiert und liefern im Vergleich zu, mit klassischen Verfahren berechneten, Referenzlösungen eine angemessene Vorhersagegenauigkeit. Darüber hinaus werden Parametrisierungen der Randbedingungen und der Geometrie in den Eingaberaum des Netzwerks integriert, was die kontinuierliche Annäherung von Lösungen im gesamten Parameterraum ermöglicht. Die parametrischen Modelle erreichen eine ähnliche Vorhersagegenauigkeit wie ihre nicht-parametrischen Gegenstücke und können, nachdem sie trainiert wurden, mit verschwindend geringem Aufwand wiederholt evaluiert werden.

Die entwickelten Techniken bringen den Stand der Technik von physikalisch informierten neuronalen Netzen in der Aerodynamik erheblich voran, insbesondere für transsonische Strömungsprobleme. Diese Arbeit zeigt, dass künstliche Viskosität ein wertvolles numerisches Werkzeug sein kann, das Lösungen einschließlich Stöße zuverlässig stabilisiert und die Genauigkeit der Methode bei transsonischen und Überschallströmungen verbessert. Darüber hinaus wird veranschaulicht, dass parametrische physikalisch informierte neuronale Netzmodelle vielversprechende Alternativen für Anwendungen darstellen, die einen mehrfachen Aufruf des Strömungslösers erfordern. Während physikalisch informierte neuronale Netze zum jetzigen Zeitpunkt etablierte numerische Verfahren in Bezug auf Recheneffizienz und Genauigkeit nicht übertreffen können, so eignen Sie sich hervorragend für parametrische Problemstellungen. Mit zukünftigen Entwicklungen und Erweiterungen, um komplexere, industriell relevante Geometrien und die Reynolds-gemittelten Navier-Stokes-Gleichungen zu berücksichtigen, hat der Ansatz das Potenzial ein wertvolles Werkzeug für den aerodynamischen Entwurf und andere technischen Anwendungen zu werden.

Aerodynamics, deep neural networks, compressible flows, partial differential equations, numerical simulation, shock waves, physics-informed neural networks

(Published in English)

Simon Wassing
German Aerospace Center (DLR), Institute of Aerodynamics and Flow Technology, Braunschweig

Exploration of Physics-Informed Neural Networks for Compressible Flows in Aerodynamics

Technische Universität Braunschweig

The solution of partial differential equations is relevant for many disciplines in science and engineering. Airflow, for example over wings, can be modeled using nonlinear transport equations, such as the compressible Navier-Stokes equations or, in the simplified inviscid case, the compressible Euler equations. Numerical methods are used to predict the aerodynamic flow patterns including shock waves, which are particularly challenging to capture, and the resulting aerodynamic forces and moments. Hence, these numerical techniques have become a crucial tool for aerodynamic vehicle design. To this end, well-established methods, such as finite volume methods, predict the solution at discrete points in the domain. These methods require fine grids resulting in millions of degrees of freedom, which is challenging even today's computational hardware. Especially in multi-query scenarios, such as design exploration and optimization, the cumulative computational cost of performing numerous individual simulations can be prohibitive. Alternative numerical methods are recently gaining popularity as partial differential equation solvers, employing artificial neural networks as global continuous ansatz functions for the solution approximation. These so-called physics-informed neural networks are deep neural networks trained with a loss function that directly incorporates partial differential equations. Especially for engineering applications, this approach holds promise because a single deep neural network could substitute many classical simulations in multi-query scenarios. However, the capturing of shock waves is particularly challenging for the neural network-based approach.

This work explores the application of physics-informed neural networks as an alternative approach for solving partial differential equations with a focus on compressible flow applications, governed by the Euler equations. Inspired by classical computational fluid dynamics methods, stabilization techniques for shock waves, based on artificial viscosity and sensor functions are developed and combined with mesh transformations. These techniques are validated on compressible flow problems in subsonic, transonic and supersonic conditions, yielding reasonable prediction accuracies. Furthermore, parametrizations of the boundary conditions and geometry are integrated into the input space of the network, enabling the approximation of solutions across the entire parameter space. The parametric models obtain similar prediction accuracies as their non-parametric counterparts and once trained, can be evaluated at negligible computational cost.

The developed techniques significantly advance the state of the art of physics-informed neural network models in aerodynamics, especially for applications in transonic flows. This work shows that artificial viscosity can be a valuable tool that reliably stabilizes solutions including shocks and improves the accuracy of the method when dealing with transonic and supersonic flows. In addition, it is illustrated that parametric physics-informed neural network models are promising candidates for multi-query scenarios. While physics-informed neural network can, at this point, not outperform established numerical techniques in terms of computational efficiency and accuracy, they do excel in approximating parametric problems. With future developments and extensions to accommodate more complex, industrially relevant geometries and the Reynolds-averaged Navier Stokes equations, the approach has the potential to become a valuable tool for applications in aerodynamic design and other engineering applications.

TU Braunschweig – Niedersächsisches
Forschungszentrum für Luftfahrt

Berichte aus der Luft- und Raumfahrttechnik

Forschungsbericht 2025-18

Exploration of Physics-Informed Neural Networks for Compressible Flows in Aerodynamics

Simon Wassing

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und Strömungstechnik
Braunschweig

Diese Veröffentlichung wird gleichzeitig in der Berichtsreihe „NFL - Forschungsberichte“ geführt.

Diese Arbeit erscheint gleichzeitig als von der Fakultät für Maschinenbau der Technischen Universität Carolo-Wilhelmina zu Braunschweig zur Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigte Dissertation.

Exploration of Physics-Informed Neural Networks for Compressible Flows in Aerodynamics

Von der Fakultät für Maschinenbau
der Technischen Universität Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von:	Simon Wassing
geboren in:	Paderborn
eingereicht am:	10.07.2025
mündliche Prüfung am:	7.10.2025
Vorsitz:	Prof. Dr.-Ing. Ulrich Römer
Gutachter:	Prof. Dr. Stefan Görtz
Gutachter:	Prof. Dr.-Ing. Carsten Schilde

Acknowledgments

First, I would like to thank my doctoral advisor, Prof. Dr. Stefan Görtz, for his trust and for allowing me to complete my PhD in his department. Despite his many responsibilities, Stefan has consistently shown his enthusiasm for my work. He has demonstrated that he deeply cares about the well-being of his employees, including his PhD students.

In addition, I would like to extend my special thanks to Dr. Philipp Bekemeyer and Prof. Dr. habil. Stefan Langer. Without both of you, this work would not have been possible. From the first day at DLR, Philipp's invaluable management skills as a team lead, his technical insight, and his guidance have made it a pleasure to be part of the SUM team. Your drive and ambition have been inspiring to me as a young researcher, and I am proud to call you my team lead, colleague, and friend. Similarly, Stefan's mathematical and technical understanding of the subject has shaped my work in a way that nothing else has. I have learned so much from you and continue to do so on a daily basis. With your academic rigor and uncompromising approach to research, you have truly been a role model for me throughout my time at DLR. Despite the numerous small setbacks I have encountered, both Stefan and Philipp have always been open and supportive, listening to my struggles and encouraging me to persevere. I cannot thank you enough for that.

I want to thank all current and former members of the SUM team. Special thanks go to Anna Kiener, who went through the whole PhD experience alongside me. I always found it reassuring to share my experiences with someone who was going through the same things and to be able to support each other. As a roommate, Mario has been a steady source of support in my everyday life, and I have always felt at home in our shared office. I want to thank Derrick for the fruitful collaborations on SMARTy and neural networks. I also want to express my gratitude for the time that I got to work together with Mateus. He was one of the kindest people I ever met, and it pains me that he is no longer with us. My heartfelt thanks also go to all my colleagues who have supported me throughout this journey. Your collaborative spirit and open-mindedness in the department have always made it easy to learn something new and to get help when needed. I also want to acknowledge the people behind the scenes, including Conny Delfs, Stefan Pomaska, and Andreas Ernst. I could not count how many times Conny has helped me out with administrative processes or how many times Stefan has swiftly fixed an issue with my laptop.

I am also very thankful for all the wonderful friends I made in DLR along the way, including Miquel, Fabian, Wojciech, Jörn, Malte, Eric, David and Thomas who have always made it a pleasure to come to work but also to spend quality time together outside of work. Especially Jesús became one of my best friends, and I look forward to finally having more time to go hiking together or check out the newest café in town. I would also like to thank all my friends back home, including Hendrik, Annika, and Marina, as well as all my friends in Göttingen, for their support.

Special gratitude also goes to the family. From an early age, my dad excited me about science, computers, and math and always encouraged me in my academic pursuits. My mom was always my biggest supporter, and without her love and motivation, I could not have done it. My brother has always had my back, and I am so proud of him and his achievements. I also want to thank

Maria and Manfred, who supported my studies from an early day on and always helped me to achieve my goals.

Lastly, I want to thank Anjali. Like no one else, you have supported me on this path. Despite all the doubts I had, you never failed to make me see the light at the end of the tunnel. Especially over the last few months, with your unwavering encouragement and support, you were the one who kept me going, and I cannot wait to make up for all the weekends we missed together.

Summary

The solution of partial differential equations is relevant for many disciplines in science and engineering. Airflow, for example over wings, can be modeled using nonlinear transport equations, such as the compressible Navier-Stokes equations or, in the simplified inviscid case, the compressible Euler equations. Numerical methods are used to predict the aerodynamic flow patterns including shock waves, which are particularly challenging to capture, and the resulting aerodynamic forces and moments. Hence, these numerical techniques have become a crucial tool for aerodynamic vehicle design. To this end, well-established methods, such as finite volume methods, predict the solution at discrete points in the domain. These methods require fine grids resulting in millions of degrees of freedom, which is challenging even today's computational hardware. Especially in multi-query scenarios, such as design exploration and optimization, the cumulative computational cost of performing numerous individual simulations can be prohibitive. Alternative numerical methods are recently gaining popularity as partial differential equation solvers, employing artificial neural networks as global continuous ansatz functions for the solution approximation. These so-called physics-informed neural networks are deep neural networks trained with a loss function that directly incorporates partial differential equations. Especially for engineering applications, this approach holds promise because a single deep neural network could substitute many classical simulations in multi-query scenarios. However, the capturing of shock waves is particularly challenging for the neural network-based approach.

This work explores the application of physics-informed neural networks as an alternative approach for solving partial differential equations with a focus on compressible flow applications, governed by the Euler equations. Inspired by classical computational fluid dynamics methods, stabilization techniques for shock waves, based on artificial viscosity and sensor functions are developed and combined with mesh transformations. These techniques are validated on compressible flow problems in subsonic, transonic and supersonic conditions, yielding reasonable prediction accuracies. Furthermore, parametrizations of the boundary conditions and geometry are integrated into the input space of the network, enabling the approximation of solutions across the entire parameter space. The parametric models obtain similar prediction accuracies as their non-parametric counterparts and once trained, can be evaluated at negligible computational cost.

The developed techniques significantly advance the state of the art of physics-informed neural network models in aerodynamics, especially for applications in transonic flows. This work shows that artificial viscosity can be a valuable tool that reliably stabilizes solutions including shocks and improves the accuracy of the method when dealing with transonic and supersonic flows. In addition, it is illustrated that parametric physics-informed neural network models are promising candidates for multi-query scenarios. While physics-informed neural network can, at this point, not outperform established numerical techniques in terms of computational efficiency and accuracy, they do excel in approximating parametric problems. With future developments and extensions to accommodate more complex, industrially relevant geometries and the Reynolds-averaged Navier Stokes equations, the approach has the potential to become a valuable tool for applications in aerodynamic design and other engineering applications.

Übersicht

Die Lösung von partiellen Differentialgleichungen ist für viele Disziplinen in Wissenschaft und Technik von Bedeutung. Luftströmungen, z.B. über Tragflächen, können mit Hilfe nichtlinearer Transportgleichungen, wie den kompressiblen Navier-Stokes-Gleichungen oder, unter Vernachlässigung viskoser Effekte, den kompressiblen Euler-Gleichungen, modelliert werden. Numerische Methoden werden verwendet, um aerodynamischen Strömungsphänomene einschließlich Stoßwellen, deren Erfassung besonders schwierig ist, vorherzusagen. Daraus lassen sich die resultierenden aerodynamischen Kräfte und Momente ableiten. Daher sind diese numerischen Verfahren zu einem wichtigen Werkzeug für den aerodynamischen Entwurf von Fluggeräten geworden. Etablierte Verfahren, wie die finite Volumen Methode, lösen die Gleichungen an diskreten Punkten in der Strömungsdomäne. Diese Methoden erfordern feine Gitter, die zu Millionen von Freiheitsgraden führen, was selbst für die heutige Rechenhardware eine Herausforderung darstellt. Insbesondere in Szenarien, die ein mehrfaches Aufrufen des Strömungslösers erfordern, wie z.B. bei der Entwurfsoptimierung, können die kumulativen Rechenkosten zu aufwendig sein. Alternative numerische Verfahren, die künstliche neuronale Netze als globale kontinuierliche Ansatzfunktionen für die Lösungsannäherung verwenden, gewinnen in letzter Zeit als Löser partieller Differentialgleichungen an Bedeutung. Diese so genannten physik-informierten neuronalen Netze sind tiefe neuronale Netze, die mit einer Kostenfunktion trainiert werden, welche die partiellen Differentialgleichungen direkt einbezieht. Insbesondere für technische Anwendungen ist dieser Ansatz vielversprechend, da ein einziges tiefes neuronales Netz viele klassische Simulationen in Szenarien ersetzen könnte, die ein mehrfaches Aufrufen des Strömungslösers erfordern. Allerdings ist die Auflösung von Stoßwellen eine besondere Herausforderung für diesen Ansatz.

Diese Arbeit untersucht die Anwendung von physikalisch informierten neuronalen Netzen als alternativen Ansatz zur Lösung partieller Differentialgleichungen mit einem Fokus auf kompressible Strömungen, die durch die Euler-Gleichungen beschrieben werden. Inspiriert von klassischen Methoden der numerischen Strömungsmechanik werden Stabilisierungstechniken für Stoßwellen entwickelt, die auf künstlicher Viskosität und Sensorfunktionen basieren und mit Netztransformationen kombiniert werden. Diese Techniken werden an kompressiblen Strömungsproblemen im Unterschall, Transschall und Überschall validiert und liefern im Vergleich zu, mit klassischen Verfahren berechneten, Referenzlösungen eine angemessene Vorhersagegenauigkeit. Darüber hinaus werden Parametrisierungen der Randbedingungen und der Geometrie in den Eingaberaum des Netzwerks integriert, was die kontinuierliche Annäherung von Lösungen im gesamten Parameterraum ermöglicht. Die parametrischen Modelle erreichen eine ähnliche Vorhersagegenauigkeit wie ihre nicht-parametrischen Gegenstücke und können, nachdem sie trainiert wurden, mit verschwindend geringem Aufwand wiederholt evaluiert werden.

Die entwickelten Techniken bringen den Stand der Technik von physikalisch informierten neuronalen Netzen in der Aerodynamik erheblich voran, insbesondere für transsonische Strömungsprobleme. Diese Arbeit zeigt, dass künstliche Viskosität ein wertvolles numerisches Werkzeug sein kann, das Lösungen einschließlich Stöße zuverlässig stabilisiert und die Genauigkeit der Methode bei transsonischen und Überschallströmungen verbessert. Darüber hinaus wird veranschaulicht, dass parametrische physikalisch informierte neuronale Netzmodelle vielversprechende Alternativen für Anwendungen darstellen, die einen mehrfachen Aufruf des Strömungslösers erfordern. Während physikalisch informierte neuronale Netze zum jetzigen Zeitpunkt etablierte numerische Verfahren in Bezug auf Recheneffizienz und Genauigkeit nicht übertreffen können, so eignen Sie sich hervorragend für parametrische Problemstellungen. Mit zukünftigen Entwicklungen und Erweiterungen, um komplexere, industriell relevante Geometrien und die Reynolds-gemittelten Navier-Stokes-Gleichungen zu berücksichtigen, hat der Ansatz das Potenzial ein wertvolles Werkzeug für den aerodynamischen Entwurf und andere technischen Anwendungen zu werden.

Contents

List of Figures

List of Tables

List of Publications

This work is, in parts, based on the following articles and conference proceedings:

- Wassing, Simon; Langer, Stefan; Bekemeyer, Philipp (2025): “Physics-Informed Neural Networks for Inviscid Transonic Flows around an Airfoil.” In: *Physics of Fluids.* DOI: 10.1063/5.0276518.
- Wassing, Simon; Langer, Stefan; Bekemeyer, Philipp (2025): “Adopting Computational Fluid Dynamics Concepts for Physics-Informed Neural Networks.” In: *AIAA SCITECH 2025 Forum. AIAA SCITECH 2025 Forum.* Orlando, FL. Reston, Virginia: American Institute of Aeronautics and Astronautics, DOI: 10.2514/6.2025-0269.
- Wassing, Simon; Langer, Stefan; Bekemeyer, Philipp (5th - 9th 2022): “Parametric Compressible Flow Predictions using Physics-Informed Neural Networks.” In: *8th European Congress on Computational Methods in Applied Sciences and Engineering.*, 5th - 9th Jun 2022, Oslo, Norway, DOI: 10.23967/eccomas.2022.217.
- Wassing, Simon; Langer, Stefan; Bekemeyer, Philipp (2024): “Physics-informed neural networks for parametric compressible Euler equations.” In: *Computers & Fluids* 270, S. 106164. DOI: 10.1016/j.compfluid.2023.106164.
- Siegl, Pia; Wassing, Simon; Mieth, Dirk Markus; Langer, Stefan; Bekemeyer, Philipp (2024): “Solving transport equations on quantum computers—potential and limitations of physics-informed quantum circuits.” In: *CEAS Aeronautical Journal* DOI: 10.1007/s13272-024-00774-2.
- Bekemeyer, Philipp; Bertram, Anna; Hines Chaves, Derrick A.; Dias Ribeiro, Mateus; Garbo, Andrea; Kiener, Anna; Sabater, Christian; Stradtner, Mario; Wassing, Simon; Widhalm, Markus; Goertz, Stefan; Jaeckel, Florian; Hoppe, Robert; Hoffmann, Nils (2022): “Data-Driven Aerodynamic Modeling Using the DLR SMARTy Toolbox.” In: *AIAA AVIATION 2022 Forum.* AIAA AVIATION 2022 Forum. Chicago, IL & Virtual. Reston, Virginia: American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2022-3899.

Nomenclature

Abbreviations

2D	two-dimensional
3D	three-dimensional
AD	automatic differentiation
ADAM	adaptive momentum estimation algorithm
AI	artificial intelligence
AV	artificial viscosity
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
CFD	computational fluid dynamics
DNS	direct numerical simulation
GPU	graphics processing unit
HPC	high performance computing
JST	Jameson-Schmidt-Turkel
L-BFGS	low-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
MT	mesh transformation
NACA	National Advisory Committee for Aeronautics
NN	neural network
ODE	ordinary differential equation
PDE	partial differential equation
PINN	physics-informed neural network
POD	proper orthogonal decomposition
RANS	Reynolds-averaged Navier-Stokes
ROM	reduced order model
SGD	stochastic gradient descent algorithm
SMARTy	surrogate modeling for aero data toolbox in python

Variables

ϵ	small scalar value > 0
Re	Reynolds number
$\boldsymbol{q}$	velocity vector
ρ	density
L	characteristic length
μ_{d}	dynamic viscosity
ω	frequency
V	a general volume
ψ	general conserved quantity
∇	Nabla operator
C_p	coefficient of pressure/pressure coefficient
e	eccentricity of ellipse

Symbols - Scalar Conservation Laws

t	time coordinate
x	spatial coordinate (1D)
F	flux function
Ω	domain
T	maximum of analyzed time interval
u	PDE solution (1D)
u_+	right side of shock solution
u_-	left side of shock solution
u_μ	solution with viscous regularization
ϕ	test function
$s(t)$	shock position (1D)
Φ	entropy funciton
Ψ	entropy flux function

Symbols - Euler equations

ρ	density
u	x-velocity

v	y-velocity
E	energy
p	pressure
κ	ratio of specific heats ($\kappa = 1.4$ for air)
$\boldsymbol{W}$	vector of conserved quantities
$\boldsymbol{F}_x$	flux function in x-direction
$\boldsymbol{F}_y$	flux function in y-direction
$\bar{F}$	numerical flux function
α	angle of attack
M	Mach number
c	speed of sound
$\boldsymbol{D}$	dissipative term
α	angle of attack
M	Mach number
Λ	sum of largest eigenvalues of flux Jacobians
$k^{(2)}$	constant in JST-scheme
$k^{(4)}$	constant in JST-scheme
$s^{(2)}$	sensor second difference
$s^{(4)}$	sensor fourth difference
Υ	pressure sensor JST-scheme
A	flux jacobian in x-direction
B	flux jacobian in y-direction
T	diagonalization of flux jacobians
λ_i	eigenvalues of flux jacobians $i = 1 \ldots 4$

Symbols - Neural Networks

$\hat{\boldsymbol{u}}$	neural network output vector
D	number of layers in fully connected NN
N_k	number of neurons in layer k
f^k	layer function in layer k
σ^k	activation function in layer k

b^k	bias vector in layer k
w^k	weight matrix in layer k
$\boldsymbol{r}$	input vector
tanh	hyperbolic tangent
ReLu	rectified linear unit
$\tanh_{\text{adap}}$	layer-wise adaptive hyperbolic tangent activation
u	general target function
$\boldsymbol{\theta}$	vector of trainable network parameters
N_θ	number of trainable network paramters
$\mathcal{L}$	loss function
$\tilde{\mathcal{L}}_i$	stochastic loss function in iteration i
N_{batch}	batch size
X	random purtubation of index vector
m	first moment estimate ADAM
v	second moment estimate ADAM
β_1	first moment decay rate ADAM
β_2	second moment decay rate ADAM
τ	learning rate
g	general function in AD example
v	sub operation AD example
$\bar{v}$	adjoint of v
$Q(i)$	child nodes of operation i
$\boldsymbol{z}$	vector of weight weight matrix for normalization
ϱ	magnitude of weight matrix vector for normalization
$\hat{u}_{\text{fe}}$	output vector of Fourier embedding
D_{fe}	number of Fourier features
ϕ	Fourier feature frequency
σ_ϕ	standard deviation of random sampling of frequencies

Symbols - Physics-Informed Neural Networks

$\mathcal{R}$	residual operator

$\mathcal{B}$	boundary condition operator
$\mathcal{I}$	initial condition operator
$\mathcal{L}_k$	loss term k
N	number of training/collocation points
λ_k	loss term weight of loss term k
$\hat{u}$	prediction (1D)
ζ	parameter of PDE
$\hat{\boldsymbol{w}}$	prediction of primitive variable vector in Euler Eqs.
l_w	interval between dynamic loss term weighting upddate iterations
β_w	updating rate dynamic loss term weighting
Λ_{dw}	updating function for dynamic weights
$\mathcal{F}$	mesh transformation
Σ	computational domain
Σ_{h}	grid in computational domain
Ω_{h}	grid in physical domain
ξ	first curvilinear coordinate
η	second curvilinear coordinate
J	Jacobian of mesh transformation
τ	time coordinate in mesh transformation
μ	artificial viscosity
ν	artificial viscosity factor
$\tilde{\nu}$	prescribed artificial viscosity factor
$I_{\rho,\rho E}$	$\mathrm{diag}(1,0,0,1)$
s	sensor function
s_{shock}	shock sensor function
s_{stag}	stagnation sensor function
$k^{(0)}_{\mathrm{shock}}$	shock sensor threshold
$k^{(1)}_{\mathrm{shock}}$	shock sensor sensitivity
$k^{(0)}_{\mathrm{stag}}$	stagnation sensor threshold
$k^{(1)}_{\mathrm{stag}}$	stagnation sensor sensitivity

Symbols - Error Metrics

MAE	mean absolute error
$RMAE$	relative mean absolute error
R_2	coefficient of determination/R^2-score

Subscripts

R	residual
B	boundary condition
I	initial condition
$\boldsymbol{\theta}$	depending on/with respect to trainable parameters
∞	far-field
ob	on aerodynamic object/boundary
per	periodic
0	initial value
μ	viscosity
x	first spatial Cartesion coordinate
y	second spatial Cartesian coordinate
t	time coordinate
ξ	first curvilinear coordinate
η	second curvilinear coordinate

Statement on the use of Artificial Intelligence

In the writing of this work, generative artificial intelligence (AI) tools were used to improve the language and style-of-writing for less than 10% of the total amount of text. All sections, where AI was used, were double checked and modified by the author by hand, to ensure the correctness of all statements made. None of the technical content was generated by generative AI.

Acknowledgment of Computational Resources

The author gratefully acknowledges the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARA is partially funded by “Saxon State Ministry for Economic Affairs, Labour and Transport” and “Federal Ministry for Economic Affairs and Climate Action”. This work was also supported by the Helmholtz Association’s Initiative and Networking Fund on the HAICORE@KIT partition.

1. Introduction

With technological advancements, driven by our innate desire to explore, humans have built progressively larger and faster modes of transport. In addition, efficiency has become an increasingly important factor in the vehicle design. Thus, understanding aerodynamics, the study of the motion of air and its interactions with solid bodies, becomes paramount. The fundamental forces of aerodynamics have a critical influence on vehicular motion and are thus pivotal factor for the design process. Only through research on this subject, are we able to understand and utilize these forces to our advantage. For example in trains or cars, only by reducing drag can we voyage at high velocities. However, one of the most remarkable applications of aerodynamics is flight. For millennia, humans have been fascinated by the ability of certain animals to fly and tried to emulate them. Through progress in our understanding of aerodynamics, we have been able to achieve that dream. Decades of observations, trial and error and simultaneous progress in mathematics was necessary to obtain a deep understanding of the principles of flight. Today, mathematical models of the underlying physics act as the basis for large portions of the aircraft development process. At the heart of these mathematical models lie so-called *partial differential equations* (PDEs). Many processes in physics, engineering, and other mathematical sciences can be modeled by differential equations. Broadly speaking, they can be seen as mathematical abstractions of certain principles in physics and other disciplines that include the rate of change of some unknown function. We usually refer to these unknown functions as solutions because they often are the key to our understanding of that process and allow the prediction of how a system is behaving and possibly evolving in time. Perhaps one of the most well known differential equation is Newton's second law of motion, which states that the rate of change of the momentum of a point mass is equal to the force acting on it. This deceptively simple equation lies at the core of classical mechanics. Solution of the equations allow the prediction of motion of objects like particles, projectiles, cars, planes, planets and stars. Notably, even relatively simple equations, like the equations of motion, give rise to highly complex dynamics. For a mere three-body system, a closed form solution is already unknown in many cases. The observation that a rather concise differential equation gives rise to solutions of vast complexity is not a singular incident but a rather common observation when dealing with differential equations. Especially for partial differential equations which contain derivatives with respect to different independent variables, exact solutions are often only known under special boundary conditions. Boundary conditions are constraints on the solution of the differential equation imposed at the boundary of a domain. A problem posed by a differential equation and additional boundary conditions is called a *boundary value problem*.

The Navier-Stokes equations are a system of non-linear partial differential equations, describing the motion of fluids [1, Ch. 15]. They are essentially the reformulation of the conservation of mass, energy and momentum for continuous media. These equations are considered to be an accurate and versatile model for the mechanics of gasses and liquids. In essence, they contain the essential physical principles which describe all acting forces in fluid flows and their interaction. Hence, in the context of aerodynamics, the solution of the Navier-Stokes equations under certain boundary and initial conditions allows for a precise prediction of the forces acting on arbitrarily shaped objects subject to airflow. Unfortunately, solutions of these equations are notoriously difficult to obtain. Exact solutions are only known for trivial geometries, usually at modest flow speeds and not in three spatial dimensions (see e.g. [1, Ch. 16]). However, when dealing with complex engineering tasks, these trivial solutions are rarely useful because they cannot describe the resulting flows around non-trivially shaped bodies or at realistic speeds. Since exact solutions are unattainable for most engineering applications, numerical approximations are necessary.

The numerical approximation of solutions to the Navier-Stokes equations is a significant challenge in and of itself. Methods trying to approximate solutions to the Navier-Stokes equations and other related equations of fluid dynamics are the quintessence of computational fluid dynamics (CFD). In general in CFD, numerical and mathematical simulation methods are used to analyze problems involving fluid flows such as gases and liquids. The most popular approaches for solving the Navier-Stokes equations perform a discretization of the system in space and time (see for example [2, Ch. 4-5] or [3, Ch. 3]). This means that instead of using continuous functions to approximate the solution in the spatial and temporal dimensions, these dimensions are discretized into a finite number of points on a grid and a numerical solver tries to approximate the values of the solution function at these discrete points. To fully resolve important flow features, finely resolved grids are required, corresponding to a large number of degrees of freedom. Two main factors contribute to the high grid densities required in aerodynamics. First, in the wall-normal direction near the aerodynamic surfaces, boundary layers warrant a high grid density. Boundary layers are the region near a wall where the tangential flow velocity, which is zero on the walls themselves, approaches the incoming flow speed and where the flow is dominated by friction [1, Ch. 1.11]. These regions are characterized by steep gradients in wall normal direction and hence require a high point density in this direction. A significant portion of grid nodes is typically required to accurately capture the gradients in the boundary layer. This increases the computational effort significantly. Secondly, turbulence is also an important contributor to the large number of degrees of freedom required for the discretization. Turbulent flows exhibit irregular chaotic and in-stationary variations in the flow quantities such as the velocity and pressure [4, p. 104] down to the Kolmogorov length scale. Whether a flow is turbulent depends on a few quantities, which can be summed up in the dimensionless parameter called the *Reynolds number*:

$$Re = \frac{|\boldsymbol{q}|\ L\ \rho}{\mu_{\mathrm{d}}}, \tag{1.1}$$

with a characteristic fluid velocity $\boldsymbol{q}$ vector, a characteristic length scale L, the fluid density ρ and the dynamic viscosity μ_{d}. For moderate Reynolds numbers, the flow is *laminar*, meaning that no significant mixing between neighboring particles occurs [4,

p. 104]. However, when a certain critical Reynolds number Re_{crit} is exceeded, we see the emergence of turbulence. The critical Reynolds number is different for every geometry. The required mesh resolution to properly capture all turbulent length scales for a large scale problem like an aircraft far exceeds the capabilities of current computational hardware. Since a full solution of the Navier-Stokes equations, a so-called direct numerical simulation (DNS), is infeasible for the geometries and high Reynolds numbers encountered in aeronautics, simplifications of the full set of equations are used in practice.

A common simplification is the assumption of an incompressible fluid. Here, it is assumed that the local density is constant in the whole domain and does not change over time. This assumption is reasonable if only small variations in fluid density occur. As a general rule of thumb for when incompressibility is applicable, one can consider the Mach number. It represents the ratio of the absolute flow velocity $\boldsymbol{q}$ and the speed of sound c.

$$M = \frac{|\boldsymbol{q}|}{c}. \tag{1.2}$$

As a rough rule of thumb, compressibility effects are oftentimes negligible at Mach numbers of $M \ll 0.3$ [4, p. 107]. From the continuity equation, it follows that the divergence of the flow velocity vector vanishes in the entire domain. Under the assumption of incompressibility, the density is not a variable of the solution. Hence the complexity of the system is reduced by one variable. However, the incompressible Navier-Stokes equations are almost equally challenging to solve. The proof of existence of smooth exact solutions is one of the Millenium Prize Problems [5], underlining the difficulty and scientific relevance. As for the compressible equations, numerical solutions are also difficult to obtain due to the emergence of turbulence and the requirement of very fine grids to capture all turbulent length scales. In addition, many modern aircraft fly at Mach numbers $M \geq 0.7$, where compressibility effects play a significant role. Hence, incompressibility is not a reasonable simplification for these problems.

Perhaps the most important set of equations with practical use for today's numerical simulations are the Reynolds-Averaged Navier-Stokes (RANS) equations [6]. They are derived by splitting the flow quantities into a time-averaged and a fluctuating part in a process called Reynolds-averaging in the incompressible and Favre-averaging in the compressible case. After rearranging the resulting equations, all fluctuating quantities can be contained in a single term that describes the effects of turbulence on the averaged flow quantities. To close the system, empirical turbulence models are used to approximate this term. For many engineering applications, the numerical solution of the RANS equations present a suitable trade off between accuracy and computational effort. Due to the Reynolds-averaging, turbulent fluctuations in the flow variables do not need to be captured by the grid. Hence, the mesh resolution can be reduced significantly. Furthermore, in engineering applications, one is oftentimes interested in time-averaged forces and moments, which is exactly what the RANS equations are able to calculate. The strong gradients in boundary layer do however remain. Hence, typical grids for complex three-dimensional geometries still contain millions of points. Therefore, even RANS equations are still expensive to solve, especially for complex three-dimensional geometries and transient flows. Depending on the application, they can be solved in the simplified incompressible variant or in the compressible variant, which is routinely done for aeronautical applications.

A further simplification are the compressible Euler equations, which can be obtained by entirely neglecting the viscous terms in the compressible Navier-Stokes equations [2, Ch. 2]. As previously mentioned, compressibility effects play a crucial role in the approximation of high-speed flows in aerodynamics. Modern transport aircraft routinely fly at transonic flow conditions. This means that the speed of air with respect to the aircraft locally exceeds the speed of sound or in mathematical terms, that $M > 1$. In these conditions, we see abrupt changes in the pressure, density and velocities, which are called shock waves. An exemplary simulation of a shock for a flow around an airfoil is shown in Fig. 1.1. For transport aircraft, these shocks form, for instance, at the suction side of the wing, leading to significant changes in drag and lift. . Hence, it is crucial for an accurate simulation model to take these effects into account. The mathematical model thus needs to consider compressibility. As previously discussed, the compressible Navier-Stokes equations are even more complex than their incompressible counter part, which is also true for the compressible RANS equations. Consequently, in many cases, the compressible Euler equations offer an attractive compromise between model complexity and accuracy. They neglect the viscous terms in the Navier-Stokes equations and therefore can not model phenomena originating from viscous effects. Examples for this are boundary layers and turbulence. However, they do predict the occurrence of crucial compressibility effects like different kinds of expansion and shock waves [1, Ch.7-13]. From a mathematical perspective the compressible Euler equations are a nonlinear system of conservation laws. Even for this simplified model, a sound foundational mathematical theory for the existence and uniqueness of solutions in three dimensions does not exist [7]. Nevertheless, much of the underlying knowledge that is available for scalar conservation laws has been applied to the Euler equations in practice.

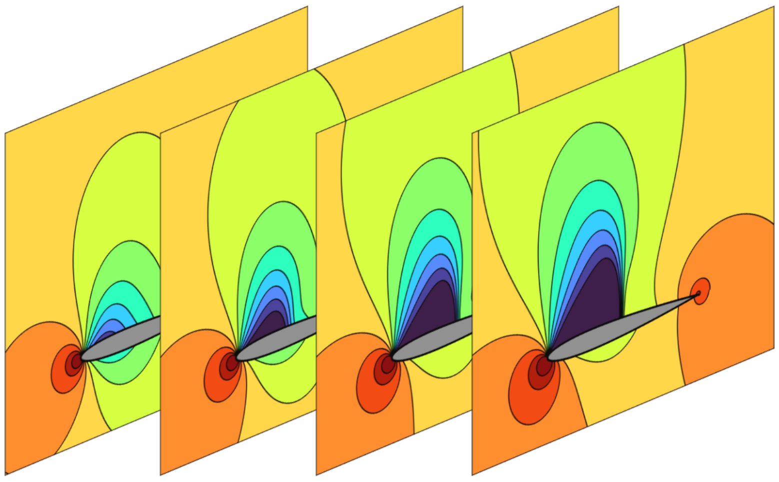


Figure 1.1.: Visualization of pressure contours for an inviscid transonic flow at $M = 0.72$ around the NACA 0012 airfoil with an angle of attack of $\alpha \in \{1, 2, 3, 4\}^\circ$ (from left to right). The results were obtained, using a physics-informed neural network, as presented later in Ch. 6.

Using simplified equations such as the RANS equations, reasonably accurate simulations of aerodynamics can be obtained in a few seconds for simple problems and a few days for complex three-dimensional problems, when using modern computational hardware. In addition, simplified models, such as the Euler equations for instance, can for example give valuable insights into the expected shock wave dynamics at more manageable computational efforts. However, in the aircraft design process, many different geometric variations need to be investigated to obtain an optimal design. Furthermore, different scenarios in the whole flight envelope need to be analyzed, further adding to the number of evaluations of the PDE solver. Hence, in recent years, surrogate models have become an important tool when parameter variations or rapid turnaround times [8, 9] are required. Surrogate models aim to mimic the outputs of a full-order model, such as a CFD solver, based on a finite number of data samples. Classical system identification approaches, used for applications in aerospace for decades [10], are an example of surrogate models. Furthermore, regression models such as radial basis functions, polynomial interpolation and Gaussian processes have been used to design capable data-driven surrogates with fast evaluation times [11, 12]. When a surrogate aims to predict an entire solution field, one often refers to it as a reduced-order model (ROM) [13]. ROMs often also make use of projection-based methods or some other form of lower-dimensional representation to approximate the full order model, the most popular choice being proper orthogonal decomposition (POD) [13]. One can also distinguish between non-intrusive ROMs, which are purely based on data, and intrusive ROMs, which also incorporate information from the underlying differential equations in some form during the prediction stage. One example of an intrusive ROM is the work by Zimmermann and Görtz [14] who enhance the predictions of a data-driven POD-ROM, by minimizing the residual of the predicted flow states evaluated with a CFD solver within the subspace spanned by the POD coefficients. By incorporating the residual evaluation, they find the combination of POD coefficients which yields the lowest CFD residual. Their method is validated on steady subsonic and transonic flows around an airfoil. The residual minimization approach was later also applied to unsteady problems by Bekemeyer et al. [15]. Notably, as later discussed, in this work will also investigate an approach where the PDE residual is minimized to obtain accurate flow predictions, although without additional data and without the restriction to a pre-computed subspace. Moreover, the method we utilize does not need access to the residual information during the prediction stage, making it, strictly speaking, not an intrusive ROM.

Recently, neural networks (NNs) have emerged as a highly capable alternative [16, 8] for surrogate and reduced order modeling. The biological networks of neurons, found in the brain, inspired the creation of the NN algorithms that are used today. NNs can be seen as a general parametrized function consisting of multiple layers of simple nonlinear units, the neurons. Information is processed though multiple layers of these neurons connected through adjustable weight matrices. These adjustable parameters are adapted during a training process. During the training, a loss function, measuring the deviation between the network prediction and target values, is minimized. For the minimization, iterative gradient-based optimization algorithms can be used. The gradients can be calculated efficiently using the chain rule of derivatives in an algorithm called back-propagation [17] (see Ch. 2.3). Using this algorithm, neural networks can be trained to approximate arbitrary functions. Furthermore, it has been shown that certain neural networks have universal function approximation capabilities. Roughly speaking, this means that a sufficiently large neural network exists that can approximate any function fulfilling certain smoothness criteria up to an arbitrary closeness (see e.g. [18]). Notably, this does not guarantee that the correct parameters can be learned.

While NNs have been known for many decades, they gained wider recognition when convolutional NNs first achieved top scores in image recognition contests in the early 2010s [19]. This success of NNs was facilitated by so-called deep NN architectures [20], where the term “deep” refers to an architecture with multiple hidden layers. There is no consensus on which kind of networks are considered deep, but it is generally agreed that more than one hidden layer is required. Furthermore, the utilization of graphics processing units (GPUs) has been vital for efficiently training deep network architectures which frequently contain millions of parameters. Lastly, sufficiently large labeled datasets were required, allowing for an effective training and circumventing over-fitting. Deep learning models were already actively used throughout the 2010s in science and industry [20] for image processing tasks and represented an active topic of research. However, the year 2022 marked another surge in the use of deep learning methods. With the release of the generative chatbot "ChatGPT", deep learning based models, which are able to generate human-conversation like responses, gained wider public recognition. These so-called large language models are based on the transformer architecture [21], contain billions of trainable parameters and are trained on massive text-databases. In the research community, the development of neural network models has been facilitated by powerful and user friendly open-source software libraries such as PyTorch [22] and Tensorflow [23]. These libraries can be used to easily set-up new model architectures, directly including automatic differentiation [24] (AD) and calculations on GPUs to accelerate the training process.

Neural network architectures are flexible in the sense that they can essentially be used for most data-driven tasks in machine learning by adapting the model architecture to the problem at hand. They excel at learning structures in high dimensional data and nonlinear relations [20]. Due to these properties, Neural networks have also become an important technique for surrogate modeling in aerodynamics. Due to the nonlinearity of the underlying PDEs, problems in fluid dynamics are typically nonlinear. This is for instance evident when approximating transonic flows, exhibiting shock waves. As for example shown by Hines and Bekemeyer [25], classical surrogate models such as proper orthogonal decomposition with interpolation, struggle to accurately approximate the pressure field in the vicinity of the shock wave due to the localized nonlinearity. In contrast, even simple neural network based architectures, such as a fully-connected feed-forward neural network, achieve improved prediction accuracies at these locations. Moreover, different types of advanced neural network architectures can handle other complexities arising in the context of aerodynamics. Encoder-Decoder-type architectures can effectively learn lower-dimensional representations of the high-dimensional data coming from the millions of nodes used in the grids of CFD simulations [26]. Furthermore, graph neural networks can exploit the graph structure of the grids and natively exchange information between neighboring nodes enhancing the accuracy of the models [25]. Other special types of architectures, such as recurrent neural networks or transformers are designed to learn from sequential data. Hence, they are well suited for transient problems in aerodynamics [27]. However, one key disadvantages of NNs is that they typically require a large database to be effectively trained. In sparse data scenarios, they are oftentimes outperformed by classical reduced order and surrogate modeling techniques. Hence, their reliability is often limited, when the training data base is insufficient. In addition, as a purely data-driven black-box model, it is also difficult to interpret the reliability of a trained model, since they are typically prone to over-fitting.

As simulation data is costly to obtain and usually sparse, new methods have emerged, aiming to directly combine the strengths of traditional numerical PDE solvers and data-driven modeling techniques, such as neural networks. These so-called scientific machine learning approaches have the goal to be more accurate because they directly incorporate physical knowledge into the model, while also being as flexible and numerically efficient as classical data-driven methods [28]. One of these approaches is the physics-informed neural network (PINN) [29].

The PINN approach essentially uses a standard deep neural network to approximate the solution to some problem, involving physical laws governed by partial differential equations. In comparison to standard NN models, it is however not only trained on data of the solution but also on the physical laws themselves. This is achieved by constructing a composite loss function, which is the sum of both data-driven and physics-driven terms. While the data-driven terms measure the agreement of the model with the data, the physics-driven terms can measure how well the model prediction conforms with the differential equation(s). To obtain this estimate, the fully differentiable structure of the NN is used to calculate the partial derivatives of the predicted solution using the chain rule of derivatives. These calculation can be be efficiently vectorized for many point locations in the domain, using the so-called automatic differentiation (AD) algorithms, also called algorithmic differentiation, which are also used for back-propagation. The partial derivatives are plugged into the differential equation which should equal 0 for a solution of the PDE. An approximation of the solution will however only yield a value $\neq 0$. Hence, this so-called residual of the differential equation can be squared and used as a loss term for the optimization of the neural network parameters. When combined with other data driven terms it can act as a physics-based regularization, aiming to obtain a model which conforms with the physics of the problem. However, when no data is available, PINNs can also be used to solve the full boundary-value problem. To this end, one also has to enforce the boundary conditions of the problem. This can, for example, be achieved using additional loss terms. To evaluate the residual loss, representative training or *collocation points* need to be selected inside of the domain as well as on the boundaries for the boundary loss terms. These points are often randomly sampled.

In aerodynamics, PINNs can potentially fill the gap between fast but inaccurate surrogate models and accurate but costly CFD solvers. In the following, the development of PINNs over recent decades is discussed in more detail and the immense potential attributed to PINNs is showcased, based on a selection of applications from various fields of science. General modifications and improvements of the methodology are also discussed. Lastly, we focus on the application of PINNs in fluid dynamics and especially aerodynamics.

1.1. Origins of PINNs and Today's Applications

Already in 1994, Dissanayake and Phan-Thien [30] published a first attempt of utilizing a neural network as a global ansatz function for the solution of boundary value problems. Surprisingly, even these first attempts show remarkable similarities with the approaches that are used today. They reformulate the Cauchy problem as an unconstrained optimization problem by designing a composite loss function with different terms for boundary and initial conditions, respectively. In addition, similarly to the popular L-BFGS [31] optimization algorithm, which is used today, they utilize a quasi-Newton optimizer to minimize the resulting loss function. Using this method, they solve two linear PDEs, obtaining satisfactory approximation accuracy considering the limited resolution due to computational limitations of that time. In contrast to today's approaches, they calculate the spatial derivatives using finite differences, leading to additional truncation errors. In the later work of Lagaris et al. [32], the authors explicitly derive the analytical derivatives of the network output with respect to the coordinate inputs based on the chain rule and use these to optimize the network parameters. They also introduce a constructed ansatz function which automatically fulfills the boundary conditions without imposing them as soft constraints via additional loss terms. For the optimization, they also employ a quasi-Newton optimizer, namely the BFGS algorithm [33, pp. 136-143]. They validate the method on a suite of simple ODEs, a linear PDE in two dimensions and a nonlinear PDE in two dimensions. In a subsequent article [34], the authors extend the ansatz-encoded boundary conditions to non-trivial geometries by fitting a radial-basis network to the geometry at hand.

While many of these fundamental concepts underlying PINNs are still used in today's models, it took almost two decades before these algorithms again received significant attention. Facilitated by readily available powerful consumer grade GPUs, interest in neural network algorithms was reignited in the 2010s, leading to the development of powerful and developer-friendly software libraries such as Tensorflow [23] and Pytorch [22], optimized for computations on GPUs. Starting around 2017, a research group at Brown university, lead by George Kardianakis, re-popularized neural network based algorithms for PDE-based problems. They introduced a unified framework, similar to the previous approaches, and called it a *physics-informed neural network* [35, 36, 29]. These articles by Raissi, Perdiakis and Kardianakis can be seen as the seminal work which popularized the method in many fields of scientific computing. They reintroduce the methodology and demonstrate on an extensive suite of test cases many of the key features of the methodology. Besides the approximation of solutions to Cauchy problems, they also highlight the applicability to solve *inverse problems*. For inverse problems, some form of data of a PDE solution or even measured data needs to be available at the start of the training. This data is then integrated as a data driven loss term, which is combined with the residual based loss terms to draw conclusions about unknowns in the set-up of the forward problem itself. This can, for example, be unknown boundary conditions or a PDE parameter whose value is unknown. In their work, this approach is, for example, used to identify the pressure field and PDE parameters such as the viscosity in the incompressible Navier-Stokes equation for the vortex shedding past a circular cylinder.

Combining these promising results with the fact that a basic PINN is straightforward to implement and apply, the method quickly drew attention in many fields of scientific computing. At the time of writing, the publication by Raissi et al. [29] has been cited more than 7000 times, clearly showing the significant interest the methodology has received in recent years. With the amount of work that has since been published on the methodology, it is very rapidly evolving and it is not possible to cover all of the recent developments within the scope of this work. A broader overview is given by the review articles [28, 37, 38], highlighting various fields of applications. Since partial differential equations are ubiquitous in many fields of science, possible use cases for PINNs are equally diverse. PINNs are for example applied in biomedicine, geophysics, fluid mechanics, material science, system theory, chemical engineering, astrophysics, nuclear physics and astronomy, just to name a few [38].

In the following, a small selection of recently published work, utilizing the PINN methodology, is highlighted to give an impression of the versatility of this approach in various fields. In the context of geophysics, Waheed et al. [39] employed PINNs to solve the factorized Eikonal equation. The Eikonal equation is a hyperbolic PDE which arises in various context. In seismology it can for example be used to model the travel time of seismic waves, helping to localize wave sources or for seismic inversion where seismic reflection data is used to create a quantitative model of the subsurface [40]. Here, the authors utilize PINNs as an alternative PDE solver. They utilize adaptive loss term weighting [41] and localized adaptive activation functions [42], obtaining reasonable accuracies on different two-dimensional test cases. They also find that transfer learning can be used to reuse a trained model which is retrained for new source locations. The authors see potential in the methodology for speeding up more complex 3D simulations. Note that in this example, PINNs are used as a PDE solver to approximate the solution to a particular Cauchy problem. This use-case is often referred to as the *forward problem*.

PDEs also play an important role in finance. The Black-Scholes equation is for example used to describe the price evolution of derivatives (e.g. options). More advanced models, such as the Heston model, assume stochastic volatility, leading to stochastic differential equations (SDE). Recently, Hainaut and Casas [43] applied the PINN methodology to the Heston model. The used PINN model is parametric, meaning that general input parameters of the SDE are incorporated in the input space of the network. Once trained, the resulting model can, practically in an instant, value an option at different parameter configurations, which is a fundamental advantage compared to the conventionally used spectral algorithms, which need to be rerun for new parameters. Hence, the authors note that the methodology shows potential and could even be applied to more complex derivatives in the future. In this example the PINN was again used as a PDE solver, however in a parametric fashion. One single PINN is trained to obtain the solution to a Cauchy problem for all possible parameter values in the range that it is trained in. The capability of PINNs to solve parametric forward problems will be further explored in this work, too.

PINNs have also been used in cardiac electrophysiology to approximate myocardial fiber orientation on the hearts surface, based on electroanatomical maps, which can be obtained in a clinically viable way. The orientation of myocardial fibers is relevant since electrical activation propagates anisotropically because the electrical conductivity is highest in fiber direction [44, 45]. With the help of PINNs, Herrera et al. [46] solved the arising inverse problem, again governed by the Eikonal equation, to reconstruct the surface fiber direction. The methodology is validated on synthetical 2D and 3D- data, on ex-vivo fiber, obtained with diffusion tensor imaging and on clinical data. When more than two electroanatomical maps are available, the fiber orientation can be reconstructed with reasonable error levels. Additional regularization has been shown to be crucial for solving the inverse problem. The method also shows resilience to noise. The presented technique could, for instance, be used to compile patient specific treatment plans based on the attained fiber orientations, without the need for invasive measurements. Inverse problem are often-times ill-posed and hence can be very difficult to solve. This article is an excellent display of how PINNs are well suited for these kind of scenarios.

Another recent application of PINNs lies in the field of surrogate modeling in nuclear reactor designs. To simulate a wide range of security critical failure scenarios, surrogates models are used to predict the behavior of a reactor design under different parameter configurations. Surrogate models are especially useful in safety critical scenarios, because the most accurate simulation models are too expensive to evaluate when Monte-Carlo based uncertainty quantification approaches are required. Recently, Antonelli et al. [47] showcased the use of a PINN model as a surrogate model for reactor simulations. Their model is mainly trained on a dataset obtained by a high-fidelity simulation, but the loss function is enhances with a single physics-informed loss term which describes the reactivity feedback to a change in average fuel temperature. Even the inclusion of this single equation acting as a physics-driven regularization is shown to significantly enhance the accuracy of the model and to reduce under- and overfitting in comparison to the fully data-driven model. For this application, the trained PINN model is primarily trained on data and the physics-based loss terms do not constitute a full Cauchy problem. The single physics-driven term can rather be seen as a type of regularization, which steers a data driven model into the direction of physically reasonable approximations.

Overall, we see that the PINN methodology is applied in many scientific fields, dealing with problems governed by PDEs. This adaptability is facilitated by the flexibility of PINNs in tackling different kinds of problems. We have seen that PINNs can solve a single instance of a forward problem or even parametric forward problems. Furthermore, they are also applicable to inverse problems or as a physics-based regularization for a data-driven model. Of course, in practice, the differences between these different use-cases can become somewhat blurred as PINNs are being applied to more and more complex problems. However, it is instructive to keep these key features, or capabilities of PINNs in mind.

1.2. General Modifications and State-of-the-Art

The standard PINN architecture uses one single NN as a global ansatz function for the approximation of the solution. In many ways, this a desirable property of PINNs because, unlike many other discretization based techniques, one obtains a globally smooth approximation for which values and derivatives can be easily calculated without any truncation error. However, when dealing with large scale problems, the neural network size also has to be increased significantly for the global solution approximation to be sufficiently accurate. Moseley et al. [48] demonstrate how for a simple one-dimensional differential equations with a harmonic solution, a standard PINN can approximate the solution well at a low frequency of $\omega = 1$. However, when they increase the frequency to $\omega = 15$, the neural network size also had to be increased significantly (number of parameters had to be increased by a factor of more than 200) to achieve a good approximation of the solution. Note that the increase in frequency is equivalent to an increase in domain size at the previous $\omega = 1$. The increase in network size also leads to significantly longer training times and deteriorated convergence. For such scenarios, it therefore makes sense to decompose the domain into multiple parts. In each subdomain, a separate NN can be trained, also enabling parallelization of the training process for each domain. Such domains decomposition approaches have for example been explored by Jagtap et al. [49] for conservation laws. They enforce continuity of fluxes at the domain boundaries via additional loss terms. In a similar fashion, in [50] this approach has been extended to other types of PDEs. Another approach was proposed by Moseley et al. [48], where subdomains are overlapping. The continuity between the individual sub-networks is obtained with predefined window functions which form a partition of unity. The window functions are multiplied with the individual network outputs and then summed up to obtain the global solution. This approach does not require additional loss terms at the interfaces but the overlapping domains create additional computational overhead. While domain decomposition is likely advantageous when scaling PINNs to large scale applications, it typically complicates the formulation of the model significantly and interfacing loss terms or window functions can aggravate the complexity of the optimization problem that is to be solved. For a more comprehensive overview of different PINN domain decomposition approaches, see [51].

The PINN loss function typically consists of different terms corresponding to different sub-objectives like the fulfillment of initial condition and the minimization of the PDE residual. During the training of the PINN, the magnitude of gradients of certain loss terms can be quite different, leading to a situation where practically one and not all of the objectives are learned simultaneously. This is because typical iterative optimization algorithms for neural networks identify try to identify the minimum of the loss function, based on the steepest descend direction. If the gradient of the sum of the loss terms is dominated by a single term, then the other loss terms are not considered during the optimization. This issue has been identified as one of the major failure modes of PINNs by Wang et al. [41]. On a number of test cases, the authors visualized the gradient imbalances in different loss terms. The issues can be alleviated by introducing weighting factors, multiplied with the different loss terms. They propose an adaptive weighting algorithm to adjust the loss term weights during training, based on their respective training gradient magnitudes. Different version of these adaptive weighting approaches have been proposed, all aiming to alleviate the imbalances in the different loss terms, leading to improved training convergence and accuracies [41, 52, 53, 54]. It should however be noted that these algorithms are only effective when an imbalance between the loss terms exists and they cannot alleviate other failure modes of PINNs.

Another way of avoiding imbalances between different terms of the composite loss function is to encode initial and boundary conditions directly into the ansatz function, removing the need for the additional loss terms. Historically, such ansatz-encoded boundary conditions were already explored in the early works of Lagaris [32]. For simple Dirichlet boundary conditions, the network output can simply be multiplied with a function which is non-zero inside the entire domain but vanishes on the boundary. Then, an additional function that exactly fulfills the boundary condition is added to that product. Similar approaches have also been developed for Neumann- and Robin-type boundary conditions [55]. While the ansatz-encoded boundary conditions can potentially improve the conditioning of the optimization problem [56, 55], the construction of the modified ansatz function can be challenging. Especially for Neumann-type boundary conditions for complex domains, approximate distance functions, fulfilling certain smoothness criteria in the neighborhood of the boundary, have to be constructed. Different numerical approaches for the construction of the ansatz function based on R-functions are proposed in [55]. The application to higher dimensional Neumann-type boundaries is not shown.

Alternative activation functions have also been explored as a potentially advantageous modification for PINNs. The introduction of additional trainable parameters into the activation function, which can modify the slope, has shown to improve the convergence behavior of PINNs and yields higher accuracies [57]. These so-called adaptive activation functions have been extended in various works [42, 58]. Moreover, sinusoidal activation functions have been able to improve PINN accuracies, compared to classical sigmoid-like activations [59, 60], especially when the solution exhibits certain periodic features. Recently, special activation functions have also been designed to approximate discontinuous solutions of hyperbolic conservation laws [61]. A comprehensive overview of different activation functions is presented in [62].

A general property of Neural network training is the so-called spectral bias phenomenon. Essentially, when training neural networks, lower frequency modes of the approximated functions are learned first during training while higher frequency modes are only learned after long training periods. Hence, when dealing with multi-scale problems, PINNs generally require very long training times to learn the higher frequency modes of a particular solution. For example, in aerodynamics, a PINN will approximate the smoother parts of a flow field, such as the far field, first and require rather long training times to approximate the nonlinearities, such as boundary layers or shocks. The previously mentioned domain decomposition approaches can be seen as a way to address the spectral bias phenomenon because by splitting a global domain into multiple smaller subdomains, the effective frequency of all solution modes in each subdomain is reduced. A more direct approach is the so- called Fourier embedding, originally proposed in [63] for general neural networks and later adapted by Wang et al. [64] for PINNs. Fourier embedding uses additional encoding layers, which map the inputs of the neural network into a higher dimensional space with harmonic functions at different randomly sampled frequencies. This encoding layer has no trainable parameters and enables neural networks to better approximate higher dimensional frequencies of the solution. In practice, this significantly improves the convergence of PINNs for multi-scale problems and leads to higher solution accuracy at the cost of an additional hyperparameter required for the sampling of the Fourier frequencies. Empirical evidence shows that Fourier embedding is one of the most reliable and universally applicable improvements that can be made for the PINN architecture [54].

Overall, we see that a significant amount of research has been dedicated to enhancing and modifying PINNs since their inception. The above overview of improvements and modifications is by no means exhaustive, as researchers continue to explore new avenues to augment the capabilities, accuracy, and efficiency of PINNs. The future of PINNs holds much promise, with ongoing research positioned to unlock even greater potential for this technology in tackling complex scientific and engineering challenges. For a more extensive overview on recent developments, the interested reader is referred to the review articles [37, 38]. In the following, the focus is shifted back on applications in fluid dynamics and aerodynamics and an overview on different PINN works in this field are discussed.

1.3. Physics-Informed Neural Networks in the Context of Fluid Dynamics and Aerodynamics

Already in the original paper by Raissi et al. [29], the application of PINNs to problems in fluid mechanics gained significant attention. Based on the viscous Burgers equation, the authors highlight how PINNs can be applied to solve forward problems governed by nonlinear conservation laws. For the incompressible Navier-Stokes equations, the authors also show how PINNs can solve inverse problems, such as the prediction of the pressure field and PDE parameters based on noisy measurements of the velocity field for the vortex shedding behind a cylinder. Since then, PINNs have frequently shown to be effective for solving similar inverse problems for the incompressible Navier-Stokes equations, where some variables of the flow field (e.g. pressure or the velocities) are reconstructed based on incomplete or noisy measurements of other variables and potentially with incomplete boundary conditions. For example in [65] the authors show how a PINN can be used to reconstruct pressure and velocity field based on sparse concentration measurements of a passive scalar convected by a three dimensional flow. This method can be used to reconstruct flow fields simply based on experimental snapshots of a flow field with some passive visualization agent like a dye or smoke. In a similar manner, Cai et al. [66] reconstruct the velocity and pressure fields, based on temperature measurements. The method is applied to three-dimensional convective flow over an espresso cup, where temperature measurements are obtained using tomographic background oriented Schlieren imaging. Steinfurth and Weiss [67] assimilate the mean flow field in an diffuser test-section, based on particle-image velocimetry as well as mean pressure and wall shear stress measurements. Their model uses the RANS equations to successfully reconstruct the three dimensional flow field in-between the particle image velocimetry planes. Mommert et al. [60] consider synthetic DNS data of a Rayleigh-Bénard convection cell. The authors reconstruct the temperature fields based on measurements of the velocity field and vice-versa.

The application to forward problems governed by the incompressible Navier Stokes equations, where the model is only trained based the boundary value problem itself, is limited to more simple problem, typically in the laminar regime and at low Reynolds-number. Early investigations by Jin et al. [52] showed the application of PINNs to solve simple two and three-dimensional laminar flows and a turbulent channel flow at a Reynolds number of $Re = 1000$, where the PINN is used in a small window and boundary conditions for that window are provided by DNS data. Sun et al. [56] applied the PINN methodology to laminar flows, modeling the blood flow through idealized stenotic and aneurysmal channels. They also show how ansatz-encoded boundary conditions compare favorably against loss-term encoded boundaries. Wang et al. [54] review and test different important PINN improvements, such as loss-term weighting and Fourier embedding on the lid-driven cavity flow, obtaining accurate predictions for $Re = 3200$. These results have only recently been improved by Cao et al. [68] who solve the lid driven cavity flow at $Re = 5000$, using a quasi-timstepping training methodology. This methodology aims to alleviate the ill-conditioning of the governing equations, or more specifically their Jacobian [69].

Eivazi et al. [70] investigate the use of PINNs for solving the incompressible RANS equations. They do not utilize an explicit turbulence model and rather let the neural network predict the Reynolds stresses directly. Dirichlet boundary conditions are prescribed for these additional variables at the domain boundaries. With this method, the authors obtain accurate predictions for zero pressure gradient and adverse pressure gradient boundary layer flows. For the turbulent flow around a NACA2412 airfoil at $Re = 2 \cdot 10^5$ and a periodic hill geometry with $Re = 2800$, they obtain modest L_2 errors with less than 10% for the airfoil and less than 30% for the periodic hill, respectively.

PINNs have also been explored for solving the compressible Euler equations. Initial efforts by Mao et al. [71], showed satisfactory results for simple one- and two-dimensional problems, when artificially increasing the resolution with additional collocation points around shock waves. However, significant smoothing can still be observed around the shock. They also investigate inverse problems. The available solution data inside of the domain helps to approximate the shocks more accurately, even when parameters of the PDE are unknown. However, the placement of additional points around the shock requires a priori knowledge of the shock location. While these initial results were promising, it was later revealed, that for more complex geometries and shock waves, PINNs generally struggle to approximate solutions for these equations [72, 73]. In fact, difficulties with the approximation of shock wave solutions can even be observed for simple scalar conservation laws. These types of differential equations can be seen as a prototypical equations for the more complex Euler equations, representing a system of conservation laws. Hence, a more fundamental analysis and refinement of the method based on simplified equations is required, before more complex problems for the Euler equations can be solved reliably. Fuks and Tchelepi [74] first report that even for hyperbolic conservation laws, PINNs fail to accurately approximate shock waves. They also note that the addition of a diffusive term to the PDE and the resulting smoothing of the solution can alleviate the problem albeit at the cost of less sharply resolved shocks.

This has inspired the investigation of artificial viscosity (AV) methodologies for PINNs. The fundamental idea is to modify the residual loss of the PINN by adding a small dissipative term to the PDE. This AV smooths out nonlinearities. AV is a well-known concept in classical methods to numerically solve hyperbolic equations [75]. For the compressible Euler equations and other conservation laws in general, some form of dissipation is usually required to obtain a stable solver scheme. The development of both accurate and simultaneously robust central difference schemes with AV has been an active field of research for decades in the context of conventional CFD solvers [76, 77, 78, 75, 79].

The failure of PINNs to approximate shock waves was later also analyzed in more detail for scalar conservation laws and systems of conservation laws, such as the Euler equations, by Patel et al [80]. To overcome this limitation, the authors discretize the domain into control volumes and reformulate the loss function in integral form. The resulting control-volume-PINN requires an approximate quadrature to evaluate the integrals. Furthermore, the authors introduce three different penalization methods to ensure convergence to viscosity solutions. Firstly, an AV penalization introduces a small dissipative term to the PDE to smooth out the discontinuities. Secondly, an entropy inequality penalization directly penalizes disagreements with the entropy condition thus ensuring convergence to the entropy/viscosity solution. Thirdly, a total variational diminishing loss penalizes oscillations around shock waves. Using these modifications, they are able to calculate accurate solutions for various hyperbolic conservation laws such as Burgers' equation and the one-dimensional Euler equations. Returning to a classical PINN formulation without control volumes, Coutinhou et al. [81] proposed different methods to locally determine AV levels for one dimensional PDEs. Waagenar [73] provided a detailed analysis of common failure modes arising when approximating compressible flows with PINNs and identified the issue that paradoxically, residual losses tend to increase when shocks are approximated by PINNs. Furthermore, different AV methods were tested on a variety of two dimensional stationary shock waves for the Euler equations, for example by utilizing the NN to locally predict the AV. The use of AV for PINNs will be further analyzed and expanded in this work. Besides AV, other modifications to the PINN architecture have been proposed to enable the successful approximation of conservation laws [61, 82, 83, 84, 85]. A detailed discussion of AV and other approaches is presented in Ch.3.

Besides the fundamental works on solving flows on trivial geometries, PINNs are recently also being explored for more applied problems in aerodynamics with non-trivial domains. In this context, we refer to a geometry as non-trivial, if the domain features internal boundaries (i.e. holes) with high curvature or sharp edges and/or geometrical features of drastically different length scales[1]. Cao et al. [86] have shown accurate predictions for subsonic inviscid and steady-state flows around an airfoil by combining PINNs with mesh transformations (MTs). This classical methodology is well known in the context of CFD (see e.g. [2, pp. 168-215]) and can, for instance, be used to solve PDEs for non-trivial geometries with finite differences. The general idea is to transform a curvilinear grid representing the physical domain into a regular grid in the computational domain. The boundary value problem is then solved on the regular grid in this computational domain. PINNs do not strictly require a regular grid-like point distribution. It was however demonstrated that using MT yields significant improvements in accuracy and convergence speed for typical two-dimensional aerodynamic flows around an airfoil. High point-density areas in the physical domain are stretched in the computational domain (e.g. near the airfoil surface) and low point-density areas in the physical domain are compressed in the computational domain. This mitigates some of the natural multi-scale characteristics of such aerodynamic flows. Near an airfoil, we typically have curvature effects in the flow field on very short length scales. The total domain needs to be comparatively large though (multiple chord lengths) to emulate an infinite domain where all flow quantities recede to free stream conditions. In addition, outgoing disturbances may be reflected back into the domain by the far-field boundary, impeding convergence [3, pp.262-268]. Therefore, numerical legacy methods require domain sizes around 10-100 times the chord length of the airfoil. Multi-scale problems like this have shown to be challenging for PINNs [64, 48] and the reformulation of the problem in terms of a square computational domain with equalized length scales leads to a loss landscape which is evidently far easier to navigate during training. The PINN prediction matches well with finite volume reference results even on coarse grids. In their subsequent work, Cao et al. [87] demonstrated how the PINN with mesh transformations can be used to predict the flow around a fully parameterized airfoil shape, highlighting the potential for geometry optimization. The mesh-transformation based PINNs have since also been adapted to solve other parametric PDEs for flows around airfoils, including laminar flows governed by the incompressible Navier-Stokes equations [88], and even the incompressible RANS equations [89], clearly demonstrating the major potential of parametric neural network based PDE solvers in aerodynamics. However, the authors also show that for transonic flows, the presented method is unable to accurately approximate the expected normal shock wave [86]. This is a major limitation for engineering applications in aeronautics, since, for the majority of the time, today's airplane's operate at transonic conditions.

In conclusion, the rapid advancement of PINNs in fluid dynamics has shown a diverse array of applications, including the solution of hyperbolic conservation laws, the approximation of laminar and turbulent flows and the application to first aerodynamic problems such as flows around airfoils. A more extensive overview of works in this field is given by [90, 91].

[1]To clarify this description, since this dissertation is focused on PINN development, geometries that are labeled to be non-trivial, e.g. the flow around an airfoil, would indeed be trivial for today's industrial CFD software. However, due to the current state of PINN development in literature, non-trivial geometries in a classical CFD context are beyond the scope of this work. Henceforth, geometries that are non-trivial in a classical CFD context will be referred to as *problems/geometries of industrial relevance*.

1.4. Research Questions

In the wide-ranging context of PINN research in fluid mechanics, where turbulence, compressibility and multi-scale phenomena pose significant simulation challenges, key questions remain unanswered. Especially when dealing with compressible flows, the applicability, accuracy, and robustness of the PINN method seems to not be sufficiently investigated. Within this scope, this work has the ambition to answer the following questions:

1. **Are classical numerical techniques and concepts for solving PDEs relevant for PINNs?**
2. **What can be the role of PINNs in compressible aerodynamics?**
   a) To which extent can PINNs solve forward problems and how do they compare to classical CFD solvers?
   b) Can PINNs solve complex parametric problems?

While PINNs represent an alternative direction for approximation of solutions to the PDEs of aerodynamics, one should also not ignore the fact that an extensive suite of legacy methods and existing theory is available for the solution of conservation laws. This catalog of knowledge needs to be adapted, because unlike classical discretization based methods, PINNs use a global, parametrized ansatz function. Hence, it is unclear if classical principles are still applicable to PINNs. However, by transferring familiar numerical concepts, the standard PINN approach can possibly be enhanced and benefit from existing knowledge which is the focus of research question 1.

As previously described, PINNs can be employed for different kinds of problems involving partial differential equations, including forward, inverse or parametric problems. However, it is unclear whether they are in fact applicable to the numerically challenging PDEs encountered in aerodynamics and in what scenarios they are advantageous, compared to PDE solvers and reduced order models in use today. Hence, question 2 focuses on investigating main use-cases of the PINN methodology. The use as a solver for classical Cauchy problems, is of major importance in aerodynamics and will be investigated in question 2.(a). For that investigation, this work focuses on conservation laws, and especially the compressible Euler equations as a model for an inviscid compressible fluid. Comparisons to legacy methods such as the finite volume method, the de-facto scientific and industrial standard for solving conservation laws, are of interest. The central criteria of these comparisons are the robustness of the method, the computational effort and the numerical accuracy. For sub-question (b) we shift the attention to parametric problems. The rather unique ability of PINNs to solve parametric problems could be expedient for aerodynamic applications since many-query scenarios are very common in design-space exploration and optimization scenarios.

1.5. Outline

The remainder of this thesis is structured as follows. **Chapter 2** starts with a recap of the mathematical basics of conservation laws, mainly focusing on the notion of a solution. As it turns out, this notion is non-trivial for conservation laws and also has important implication for the design of numerical algorithms for the approximation of the solutions. Certain solutions of conservation laws, that can indeed be observed in nature, are not solutions in a classical mathematical sense. Therefore, the formal definition of a solution needs to be widened to include these so-called weak solutions. However, weak solutions are not-unique and additional criteria need to be introduced to identify the solution that resembles the behavior that we observe in nature. As we will see, the definition of these criteria for a physically reasonable solution, is intrinsically linked to the concept of viscosity. This motivates, the usage of AV as a central concept for the novel algorithms developed in this work. Moreover, the chapter briefly discusses the finite-volume method as a classical numerical method for solving conservation laws and how it also can make use of artificial viscosity. Furthermore, the concept of a neural networks is introduced, including the mathematical formulation, standard optimization/training algorithms and important modifications and improvements. Finally, the chapter delves into the PINN methodology, outlining the general model architecture, the imposition of boundary conditions, the construction of the loss function and important extensions.

Chapter 3 explores why standard PINNs have difficulties to approximate solutions to conservation laws and what modifications can be made to overcome these limitations. Initially, a mathematical argument is presented which illustrates for a simple Riemann problem, why standard PINNs are unable to approximate shock waves. In the following, based on the theory for conservation laws in the previous chapter, different novel alterations of the PINN architecture for solving conservation laws are introduced. These novel methods aim to stabilize the training of PINNs by utilizing artificial viscosity. As an initial motivation, they are showcased, based on the inviscid Burgers equation representing a simple nonlinear conservation law and then further extended for the compressible Euler equations.

In the following the newly introduced methods are applied to a number of test cases for the compressible Euler equations. **Chapter 4** applies one of the newly developed methods, the PINN with adaptive scalar AV to a flow around a cylinder at subsonic conditions. It also shows how the method can be extended towards parametric problems. All results are compared against finite volume reference solutions.

In the following **Chapter 5**, a similar model with adaptive scalar AV is used to predicts an oblique shock wave at supersonic conditions. The models is also used for a parametrized version of the problem with variable Mach number and compared with analytical reference results. In **Chapter 6** a second model variant, the PINN with sensor-controlled AV, is used to simulate flows around airfoils. Initially, we show that, thanks to a mesh-transformation, the model is able to accurately predict subsonic flows around these non-trivial geometries. Afterwards, the chapter is focusing on transonic conditions. In these conditions, the emergence of normal shock waves is a crucial aerodynamic phenomenon which is challenging for numerical methods to capture accurately. With the help of the sensor-controlled AV, the PINN model is also able to predict the normal shock waves. Again, the results are evaluated in comparison to finite-volume references and the method is extended to parametric problems.

Chapter 7 summarizes the key findings of this work. Furthermore the research questions are revisited and it is discussed how they have been addressed in this work. To conclude this thesis, in **Chapter 8**, the findings are put into a broader context, emphasizing the implications of this work also with regards to other recent developments in this dynamic research field. The chapter also covers recommendations for possible future investigations, building upon this work.

2. Conservation Laws of Fluid Mechanics and Solution Methods

The mathematical intricacies of conservation laws have been studied for decades and represent the foundation of current CFD solvers used in industrial applications. This chapter will revisit some of these mathematical fundamentals. It is not surprising that these principles are also essential for the development of suitable neural network based solution methods for these kind of problems. As a point of comparison, finite volume methods, representing a well established class of algorithms for the solution of conservation laws, will be briefly revisited. In the second part of this chapter, the focus is shifted to classical neuronal networks and how they are used as a powerful data-driven method in machine learning. Lastly, the physics-informed neural network method is introduced and several important aspects and extensions of the method are discussed in detail.

2.1. Conservation Laws

The modeling of fluids using continuum mechanics puts our understanding of physical phenomena into mathematical terms, ultimately enabling us to predict the behavior of continuous media in unseen conditions. At the core of this mathematical description is the observation that a change of a conserved quantity over time in a fixed volume can only be precipitated by a flux across its boundaries, assuming an absence of any sources in that volume. This observation gives rise to the conservation laws of continuum mechanics.

2.1.1. General Form

We consider a general conserved quantity ψ in a domain V. In mathematical terms, a general *conservation law* is written as:

$$\frac{d}{dt}\int_V \psi dV + \oint_{\partial V} \boldsymbol{F}(\psi) \cdot \boldsymbol{n} dS = 0, \tag{2.1}$$

where $\oint dS$ represents the surface integral and $\boldsymbol{n}$ is the outward facing normalized normal vector of the control volume's boundary ∂V. In the context of fluid mechanics, ψ could be the mass density of the fluid, while V is a fixed volume, a so-called control volume, through which the fluid flows. Assuming that ψ is continuously differentiable, we can apply Gauss's theorem. In the vanishing volume limit $\mathrm{vol}(V) \to 0$ we obtain the differential form:

$$\frac{\partial \psi}{\partial t} + \nabla \cdot \boldsymbol{F}(\psi) = 0, \tag{2.2}$$

with the divergence operator $\nabla \cdot (\cdot)$. Eq. (2.2) is a partial differential equation for the unknown ψ. If ψ is a vector-valued variable, we speak of a system of conservation laws. Besides the term *conservation law*, (2.2) is called an equation *in conservation form*. Certain equations may be rewritten in non-conservative form, where factors appear outside of the differential operator. Notably, contrary to the differential form, the integral form may admit solutions of ψ that are not differentiable. We will consider such weak solutions in more detail in the following section.

Many of the governing equations of fluid mechanics for the conserved quantities such as the density, momentum and energy can be written in this form. Furthermore, one of the most prominently used numerical methods in computational fluid mechanics, the *finite volume method*, utilizes conservation laws. The method splits the computational domain into small control volumes and computes the quantities inside of these volumes by means of Eq. (2.1).

2.1.2. Scalar Conservation Laws

In the following, we consider scalar conservation laws starting with a linear transport equation, followed by the nonlinear Burgers equation. Using these equations as examples, important theoretical properties and concepts are introduced.

Characteristics and Weak Solutions

The first conservation law considered is a *linear transport equation*

$$\frac{\partial u(x,t)}{\partial t} + c\frac{\partial u(x,t)}{\partial x} = 0, \;\; (x,t) \in \mathbb{R} \times \mathbb{R}^+, \;\; x \neq 0, \tag{2.3}$$

for the unknown function $u : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$. The equation can be seen as a model of a passive scalar quantity that is advected with constant velocity c. Let us consider an initial value problem, given by Eq. (2.3) for $(x,t) \in \mathbb{R} \times \mathbb{R}^+$ and an initial condition:

$$u(x,0) = u_0(x). \tag{2.4}$$

For differentiable u_0 a solution is given by:

$$u(x,t) = u_0(x - c \cdot t). \tag{2.5}$$

This can be confirmed by plugging Eq. (2.5) into Eq. (2.3). The solution to this problem and other conservation laws can for example be obtained using the *method of characteristics* [7, Ch. 3.2].

Definition 1 (Characteristic Curve)**.** For nonlinear first-order PDEs, *characteristic curves* or, in short, *characteristics* are curves along which the PDE is converted into a system of ordinary differential equations (ODEs).

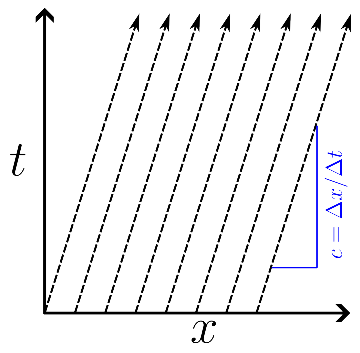


Figure 2.1.: Characteristic lines for the linear transport equation. The velocity c is given by the slope of the characteristics.

For Eq. (2.3), we look for characteristics $(x(s), t(s))$ parametrized by a new variable s. The resulting ODE has the form

$$\frac{d}{ds}u(x(s), t(s)) = D(u, x(s), t(s)), \tag{2.6}$$

with the unknown right hand side D. Using the chain rule we obtain

$$\frac{d}{ds}u(x(s), t(s)) = \frac{\partial u}{\partial t}\frac{\partial t}{\partial s} + \frac{\partial u}{\partial x}\frac{\partial x}{\partial s}. \tag{2.7}$$

Comparing this to Eq. (2.3), we set

$$\frac{dx(s)}{ds} = c \tag{2.8}$$

$$\frac{dt(s)}{ds} = 1. \tag{2.9}$$

For Eq. (2.6), it follows that

$$\frac{d}{ds}u(x(s), t(s)) = \frac{\partial u(x,t)}{\partial t} + c\frac{\partial u(x,t)}{\partial x} = 0. \tag{2.10}$$

Hence, along the characteristics, the solution is constant. Eqs. (2.8)-(2.10) are the resulting system of ordinary differential equations. From Eqs. (2.8)-(2.9) we obtain

$$\begin{aligned} t &= s \\ x(s) = x(t) &= x_0 + c \cdot t. \end{aligned} \tag{2.11}$$

Since u is constant along the characteristics, we have

$$u(x(s), t(s)) = u(x(0), 0) = u_0(x_0) = u_0(x - c \cdot t), \tag{2.12}$$

giving us the solution to the initial-boundary value problem. A schematic plot of the characteristics of the linear transport equation is shown in Fig. 2.1.

Next we consider the *inviscid Burgers equation.* In conservative form, the equation is given by:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial(u^2)}{\partial x} = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0. \tag{2.13}$$

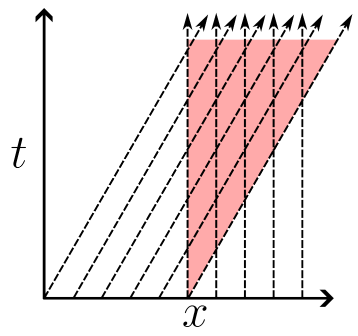


Figure 2.2.: Characteristics for Burgers' equation for the Riemann problem $u_\mathrm{l} = 1$ and $u_\mathrm{r} = 0$. The characteristics intersect in the marked area.

We can again apply the method of characteristics. Following the same steps as for the linear transport equation in Eqs. (2.6)-(2.10), we obtain the system of ODEs:

$$\begin{aligned} \frac{dx(s)}{ds} &= u(s) \\ \frac{dt(s)}{ds} &= 1 \\ \frac{du(s)}{ds} &= 0. \end{aligned} \tag{2.14}$$

Again the solution is constant along the characteristics and $s = t$. However, contrary to the linear equation, the solution of $x(t)$ depends on u. When choosing piecewise constant initial conditions, we can however make certain assertions about the solution. In constant regions we can integrate Eq. (2.14):

$$x(t) = u_0(x_0)t + x_0. \tag{2.15}$$

Let us now consider a special initial value problem, the so-called *Riemann problem*. It is given by initial data which is constant in space, except for a single discontinuity:

$$u_0(x) = \begin{cases} u_\mathrm{l}, & x \leq 0 \\ u_\mathrm{r}, & x > 0. \end{cases}, \qquad u_\mathrm{l} \neq u_\mathrm{r} \tag{2.16}$$

First, we analyze the case for $u_l = 1$ and $u_r = 0$. The characteristics, given by Eq. (2.15), are shown in Fig. 2.2. We can see, they intersect in the marked area. Hence, in this area, our classical solution is not uniquely defined, because it is supposed to take on two different values (u_l and u_r) at once. Notably, even without discontinuous initial conditions such intersections of characteristics appear for the Burgers equation, when the initial condition contains negative derivatives. This is due to the fact that the characteristics are traveling towards each other at such locations. In such cases, a unique smooth solution of the problem does not exist. We have to consider so-called weak (or integral) solutions to the problem to admit also discontinuous solutions.

Definition 2 (Weak Solution)**.** A function $u(x,t) : (\mathbb{R} \times \mathbb{R}^+) \to \mathbb{R}$ is called a weak solution to the scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} = 0, \tag{2.17}$$

if it fulfills the following identity:

$$\int\limits_{\mathbb{R}\times\mathbb{R}^+} \left(u\frac{\partial \phi}{\partial t} + F(u)\frac{\partial \phi}{\partial x} \right) dxdt + \int\limits_{\mathbb{R}} u_0(x)\phi(x,0)dx = 0, \tag{2.18}$$

for all continuously differentiable test-functions $\phi : (\mathbb{R} \times \mathbb{R}^+) \to \mathbb{R}$ with compact support [7, pp. 150-155].

Eq. (2.18) can be obtained from Eq. (2.17) by multiplying with a test function ϕ and integrating by parts. All boundary terms vanish at $\pm\infty$ since $\lim\limits_{x\to\infty} = \lim\limits_{x\to-\infty} = 0$ due to the compact support of ϕ, leaving only the second integral in Eq. (2.18). Note how the derivatives have been transferred from u to the test function ϕ. Hence, Eq. (2.18) gives us a notion of solution candidates, even when u i not smooth. The set of weak solutions contains the classical differentiable solutions but also admits certain candidates of discontinuous solutions.

Let us now consider a weak solution with a discontinuity. We analyze an open subset $V \subset \mathbb{R} \times \mathbb{R}^+$ which is separated into two regions $V_- \subset V$ and $V_+ \subset V$, separated by a curve C. The solution u is smooth in V_- and V_+ but discontinuous across the curve C. The discontinuity is parametrized by $x = s(t)$ and travels with speed $ds/dt \equiv \dot{s}(t)$. In this case, ϕ is a test function with compact support in V. Applying Eq. (2.18) to V gives:

$$\int_{V_+} \left(u\frac{\partial \phi}{\partial t} + F(u)\frac{\partial \phi}{\partial x} \right) dxdt + \int_{V_-} \left(u\frac{\partial \phi}{\partial t} + F(u)\frac{\partial \phi}{\partial x} \right) dxdt = 0, \tag{2.19}$$

where the second integral in Eq. 2.18 vanishes due to the compact support of ϕ in V. This can again be integrated by parts:

$$\begin{aligned} &- \int_{V_+} \left(\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} \right) \phi dxdt - \int_{V_-} \left(\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} \right) \phi dxdt \\ &+ \int_{\partial V_+} \left(u_+(t)n_t + F(u_+)n_x \right) \phi ds + \int_{\partial V_-} \left(u_-(t)n_t + F(u_-)n_x \right) \phi ds, \end{aligned} \tag{2.20}$$

where the $\boldsymbol{n} = (n_t, n_x) = (-\dot{s}(t), 1)$ is the normal vector across the discontinuity and u_+ and u_- are the limits of u from either side of the discontinuity. The first two integrals contain Eq. (2.17) which is satisfied pointwise in these smooth regions. Hence, we are left with:

$$\int_C \left(\dot{s}(t)(u_+(t) - u_-(t)) - (F(u_+) - F(u_-)) \right) \phi ds = 0. \tag{2.21}$$

For arbitrary test functions, the integral vanishes if $\dot{s}(t)(u_+(t) - u_-(t)) - (F(u_+) - F(u_-) = 0$. This gives the following relation for the shock speed:

$$\dot{s}(t) = \frac{F(u_+) - F(u_-)}{u_+(t) - u_-(t)}, \tag{2.22}$$

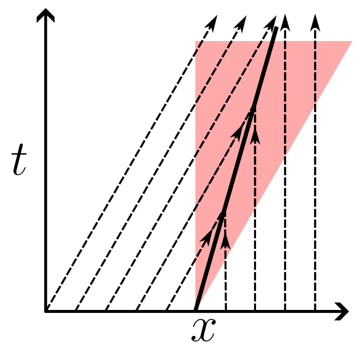


Figure 2.3.: Characteristics for Burgers' equation for the Riemann problem $u_\mathrm{l} = 1$ and $u_\mathrm{r} = 0$. The Rankine-Hugoniot condition allows for a shock to connect both sides of the Riemann problem. The previously intersecting characteristics run into the shock.

called the Rankine-Hugoniot condition. Eq. (2.22) gives an indication which curves $s(t)$ are admissible shock waves for a weak solution u. We can apply Eq. (2.22) to the previously analyzed Riemann problem, (Eq. (2.16)) for the Burgers equation with $F(u) = u^2/2$, $u_l = 1$ and $u_r = 0$:

$$\dot{s}(t) = \frac{F(0) - F(1)}{0 - 1} = \frac{-1^2}{2 \cdot (-1)} = \frac{1}{2}. \tag{2.23}$$

Hence, we obtain a weak solution to the problem:

$$u(x,t) = \begin{cases} 1, & x < \frac{1}{2}t \\ 0, & x > \frac{1}{2}t \end{cases}. \tag{2.24}$$

In Fig. 2.3 we can see that the whole domain is covered by the characteristics. They run into the contact line, meaning that the solution is uniquely defined by the initial condition u_0.

We will now analyze a second Riemann problem, given be the initial data $u_\mathrm{l} = 0$ and $u_\mathrm{r} = 1$. The domain is not fully covered by the characteristics, originating from the initial condition, as shown by the blue area in Fig. 2.4. As before, we may construct a solution with a shock to cover the missing area:

$$u(x,t) = \begin{cases} 0, & x < \frac{1}{2}t \\ 1, & x > \frac{1}{2}t, \end{cases} \tag{2.25}$$

which indeed fulfills the Rankine-Hugoniot condition. However, when looking at the solution in Fig. 2.4 (a), we see characteristics originating from the contact line. This is nonphysical, since information should always originate from the initial condition and not be created by the shock. Furthermore, we can come up with additional weak solutions with intermediate states, fulfilling the Rankine-Hugoniot conditions, such as:

$$u(x,t) = \begin{cases} 0, & x < \frac{1}{4}t \\ \frac{1}{2}, & \frac{1}{4}t < x < \frac{3}{4}t \\ 1, & x < \frac{3}{4}t, \end{cases} \tag{2.26}$$

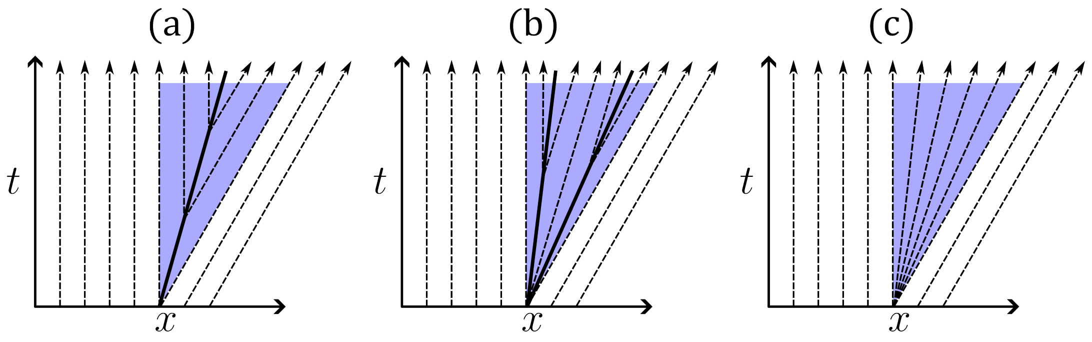


Figure 2.4.: Characteristics for different weak solutions for Burgers' equation for the Riemann problem $u_\mathrm{l} = 0$ and $u_\mathrm{r} = 1$. The Rankine-Hugoniot conditions admit for example the shocks (a) and (b) to fill the marked area. The physically reasonable solution is however given by the rarefaction wave (c).

shown in Fig. 2.4 (b). Evidently, our notion of a solution is not sufficient for describing unique physically reasonable solutions. Here, the term physically reasonable means that the solution resembles the behavior of observable systems, for example in fluid flows. Weak solutions for scalar conservation laws are not unique. To find a physically reasonable solution, we need to exclude nonphysical, weak solutions with characteristics that originate from the contact line. Hence, we postulate that a physically reasonable discontinuity should fulfill the condition:

$$F'(u_-) > \dot{s}(t) > F'(u_+), \tag{2.27}$$

called the *Lax entropy condition*. For a uniformly convex flux function, such as for the Burgers' equation, F' is strictly increasing. Hence, all discontinuities with $u_- < u_+$ are not allowed. This means that for the Burgers' equation, for all Riemann problems with $u_r > u_l$, we do not encounter any shock waves. The physically correct solution to the problem is a so-called *rarefaction solution*. For $u_\mathrm{l} = 0$ and $u_\mathrm{r} = 1$ this solution is given by:

$$u(x,t) = \begin{cases} u_\mathrm{l}, & x \leq 0 \\ \frac{x}{t}, & 0 < x < t \\ u_\mathrm{r}, & x > t. \end{cases} \tag{2.28}$$

The rarefaction solution is shown in Fig. 2.4 (c). It is a continuous, weak solution to the Riemann problem. For the full derivation, see e.g. [92, p. 25]. One can generalize this notion of an entropy conditions to non-convex fluxes. However, it only gives an indication if a certain shock wave is admissible. In the next step, we generalize the point-wise entropy condition.

Entropy Solutions and the Vanishing Viscosity Connection

As before, we consider a general scalar conservation law:

$$\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} = 0. \tag{2.29}$$

Let $\Phi : \mathbb{R} \longrightarrow \mathbb{R}$ be a strictly convex function ($\Phi''(u) > c > 0$ for some constant c) called an *entropy function*. Then the corresponding *entropy-flux* Ψ is defined as:

$$\Psi(u) = \int_0^u F'(s)\Phi'(s)ds. \tag{2.30}$$

Eq. (2.30) also implies:

$$\Psi'(u) = F'(u)\Phi'(u). \tag{2.31}$$

Suppose that u is a smooth solution to Eq. (2.29). Then we can calculate:

$$\begin{aligned} &\frac{\partial \Phi(u)}{\partial t} + \frac{\partial \Psi(u)}{\partial x} \\ =&\Phi'(u)\frac{\partial u}{\partial t} + \Psi'\frac{\partial u}{\partial x} \\ =&-\Phi'(u)\frac{\partial F(u)}{\partial x} + \Psi'\frac{\partial u}{\partial x} \\ =&(-\Phi'(u)F'(u) + \Psi'(u))\frac{\partial u}{\partial x} = 0. \end{aligned} \tag{2.32}$$

The entropy function Φ with the entropy-flux Ψ satisfy a scalar conservation law themselves. For practical applications, Φ sometimes corresponds to the negative of the physical entropy. For weak solutions, the entropy is however not conserved and Eq. (2.32) [7, Ch. 11.4.2] becomes:

$$\frac{\partial \Phi(u)}{\partial t} + \frac{\partial \Psi(u)}{\partial x} \leq 0. \tag{2.33}$$

The entropy may increase, for example over shock waves. The usage of the entropy and entropy-flux pair (Φ, Ψ) finally allows us to define an adequate notion of a solution that is both unique and physically reasonable:

Definition 3 (Entropy Solution)**.** A function $u(x,t) \in L^\infty$ is called an entropy solution of Eq. (2.29) if it satisfies the following conditions:

(i) u is a weak solution of Eq. (2.29), i.e. it satisfies the identity (2.18) for all continuously differentiable test-functions with compact support $\phi : (\mathbb{R} \times \mathbb{R}^+) \longrightarrow \mathbb{R}$.

(ii) It satisfies the entropy condition:

$$\int\limits_{\mathbb{R}\times\mathbb{R}^+} \Phi(u)\frac{\partial \phi}{\partial t} + \Psi(u)\frac{\partial \phi}{\partial x} dxdt \geq 0, \tag{2.34}$$

for all entropy/entropy-flux pairs (Φ, Ψ).

In Def. 3 (ii), we express Eq. (2.33) in a weak fashion. As for example shown by Evans [7, ch. 11.4.3], the entropy solution is unique.

Def. 3 is actually motivated by the concept of vanishing viscosity solution, as we will demonstrate in the following. For this we will consider the system:

$$\frac{\partial u_\mu(x,t)}{\partial t} + \frac{\partial F(u_\mu)}{\partial x} = \mu\frac{\partial^2 u_\mu}{\partial x^2}, \quad \mu > 0. \tag{2.35}$$

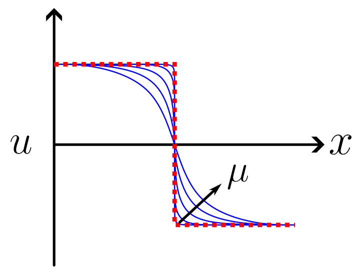


Figure 2.5.: Schematic demonstration of dissipative term in Eq. (2.35). Discontinuities are more smoothed out with higher values of μ.

This is a pertubation of Eq. (2.17), turning it into a convection-diffusion equation. It can be shown that for such systems, the solutions are C^∞ [92]. The dissipative term on the right hand side of Equation (2.35) smooths out discontinuities. This is illustrated in Fig. 2.5. To find a solution to Eq. (2.29) we now take the limit $\mu \longrightarrow 0$. The corresponding limit $u_\mu \longrightarrow u$ is expected to be the physically reasonable solution that we are looking for. It can be shown that this limit exists and that it is a weak solution to Eq. (2.29). Let us now use this idea to confirm our definition of the entropy solution. We start by multiplying Eq. (2.17) with Φ'. For notational convenience the dependence of $F(u_\mu)$, $\Phi(u_\mu)$ and $\Psi(u_\mu)$ is not explicitly shown.

$$\Phi' \frac{\partial u_\mu}{\partial t} + \Phi' \frac{\partial F}{\partial x} = \mu \cdot \Phi' \frac{\partial^2 u_\mu}{\partial x^2}. \tag{2.36}$$

We apply the chain rule of derivatives to both terms on the left hand side respectively. For the right hand side, we use $\partial^2\Phi/\partial x^2 = \Phi' \; (\partial^2 u_\mu/\partial x^2) + \Phi'' \; (\partial u_\mu/\partial x)^2$, yielding

$$\frac{\partial \Phi}{\partial t} + \Phi' F' \frac{\partial u_\mu}{\partial x} = \mu \left(\frac{\partial^2 \Phi}{\partial x^2} - \Phi'' \left(\frac{\partial u}{\partial x} \right)^2 \right). \tag{2.37}$$

Finally, we can use Eq. (2.31) and again the chain rule to obtain:

$$\frac{\partial \Phi}{\partial t} + \frac{\partial \Psi}{\partial x} = \mu \left(\frac{\partial^2 \Phi}{\partial x^2} - \Phi'' \left(\frac{\partial u}{\partial x} \right)^2 \right). \tag{2.38}$$

Since Φ is a convex function, the second term on the right hand side is greater than zero, resulting in the inequality:

$$\frac{\partial \Phi}{\partial t} + \frac{\partial \Psi}{\partial x} \leq \mu \frac{\partial^2 \Phi}{\partial x^2}. \tag{2.39}$$

Assuming that the limit $u_\mu \longrightarrow u$ exists, we obtain:

$$\frac{\partial \Phi(u)}{\partial t} + \frac{\partial \Psi(u)}{\partial x} \leq 0, \tag{2.40}$$

which is the entropy condition that we established in Def. 3.

As we will see in the following chapters, the idea of using dissipative terms as a way to obtain the entropy solutions to conservation laws is imperative for many numerical methods. In practice, non-vanishing viscosities are used in relevant parts of the domain.

Ultimately, the development of robust and accurate numerical solution methods for conservation laws often boils down to choosing appropriate viscosities μ.

In this section some important mathematical terminology concerning scalar conservation laws has been introduced. Even for these rather simple equations, a considerable theoretical framework is required to even find an appropriate notion of a solution. It has to be mentioned here that this chapter only covers fundamental aspects of the theory of conservation laws and many important concepts have only been roughly outlined. A more in-depth discussion is provided in the main references for these sections. The source for this section are the lecture notes of Siddharthra Mishra [92] and *chapter 3.4 and 11* of Evans [7].

Many problems in physics are however governed by systems of conservation laws. While many of the introduced concepts can be transferred to these systems of differential equations, a sound mathematical theory for such systems in higher dimensions is still missing. The Euler equations of fluid dynamics represents such a system of conservation laws. Except for trivial low dimensional examples, a straightforwards method to attain analytical solutions for these equations is not known. Hence, numerical approaches to obtain approximate solutions will be discussed in the following sections.

2.1.3. Euler Equations

The Euler equations are a nonlinear system of conversation laws and can be used to model the flow of an inviscid compressible fluid. For a detailed derivation of the equations, see e.g. [2, Ch. 2]. A central focus of this work is the solution of boundary value problems governed by the Euler equations in two dimensions:

$$\frac{\partial \boldsymbol{W}}{\partial t} + \frac{\partial \boldsymbol{F}_x}{\partial x} + \frac{\partial \boldsymbol{F}_y}{\partial y} = 0,$$
$$\boldsymbol{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad \boldsymbol{F}_x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u E + p u \end{pmatrix}, \quad \boldsymbol{F}_y = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v E + p v \end{pmatrix}, \tag{2.41}$$

with the density ρ, the flow velocities in x- and y-direction u and v and the total energy E. The system is closed by the equations of state for an ideal gas giving the following relation for the total energy:

$$E = \frac{1}{\kappa - 1}\frac{p}{\rho} + \frac{u^2 + v^2}{2}, \tag{2.42}$$

where κ is the ratio of specific heats. For air, we have $\kappa = 1.4$. Here, we are interested in steady-state solutions, satisfying $\partial \boldsymbol{W}/\partial t = 0$. The solution methods, developed in this work predict a vector of the four independent variables typically given by (ρ, u, v, E). One can choose the energy E as the fourth primitive variable or the pressure p, which are related by Eq. (2.42). The integration of the pressure field over the surface can be used to calculate the forces acting on aerodynamic objects. Hence, it is typically the quantity of interest. In engineering applications, instead of the pressure p, one often considers the pressure coefficient

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty(u_\infty^2 + v_\infty^2)}, \tag{2.43}$$

where the the ∞ subscript indicates the reference values in the free-stream (i.e. without any disturbances). The pressure coefficient is a non-dimensional quantity. We solve the conservative form of the equations. In the conservative form, the fluxes $\boldsymbol{F}_x$ and $\boldsymbol{F}_y$ are continuous in space, even across shock waves according to the conservation of mass, momentum and energy. This is advantageous for the solution of transonic flows, where discontinuities in the primitive variables may be encountered. Hence, their derivatives might not be well defined across shocks.

2.2. Finite Volume Method

Finite volume methods are numerical approaches to discretize conservation laws. For the approximate solution of conservation laws, finite volume discretizations are well established in science and industry [3, 93, 94]. For the sake of this work, finite volume methods will be used to obtain reference solutions. Moreover, the underlying theory behind this method can in many cases be adapted to the PINN methodology, as will be shown in this work and for example in [95]. Hence, in the following, the fundamentals of finite volume simulations will be introduced. For the sake of simplicity, this discussion is limited to one-dimensional scalar conservation laws. For a general introduction, the interested reader is referred to the references [3, 92, 6]. Afterwards, the Jameson-Schmidt-Turkel-scheme [76] is introduced as a standard discretization scheme for the Euler equations, based on artificial viscosity. This scheme is of particular interest for this work, because it has inspired newly developed methods for PINNs.

2.2.1. Scalar Conservation Laws

Similarly to many numerical techniques, a finite volume method relies on a spatial and temporal discretization of the domain. For simplicity, we divide the domain $V = (x_L, x_R) \times (0, T)$ into N equally large cells or control volumes of size $\Delta x = (x_R - x_L)/(N+1)$. The cell mid-points are given by $x_j = x_L + (j + 1/2)\Delta x,\ j = 0, \dots N$ and the half points, defining the cell boundaries are defined as $x_{j-1/2} = x_j - \Delta x/2$, as shown in Fig. 2.6. Note that non-uniform cells could also be used here. The fundamental idea of a finite volume method is that instead of approximating the exact solution $u(x, t)$, only averaged solution values are considered in each cell volume:

$$u_j^n = \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t^n) dx \tag{2.44}$$

By iterating through discrete time steps $t^n = n\Delta t$, the average solution in the next time step is calculated, based on the average solutions at the previous time step: $u_j^{n+1} = H(u_0^n, u_1^n, \dots, u_N^n)$, where $H(\cdot)$ is the update function of the numerical method. To find the

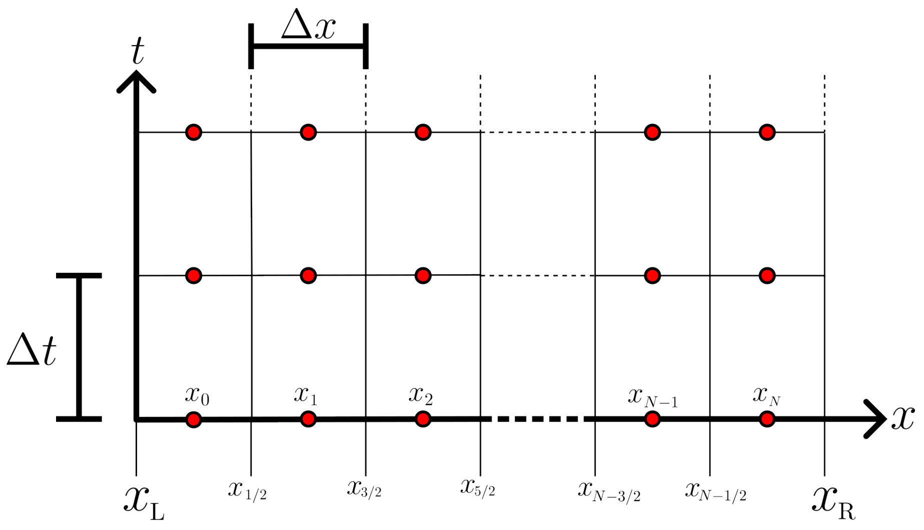


Figure 2.6.: Schematic representation of spatial discretization in finite volume method in 1D and time discretization.

update function H, we integrate a general scalar conservation law $\partial u/\partial t + \partial f(u)/\partial x = 0$ over one of the control volumes:

$$\int_{t^n}^{t^{n+1}} \int_{x_{j-1/2}}^{x_{j+1/2}} \frac{\partial u}{\partial t} dx dt + \int_{t^n}^{t^{n+1}} \int_{x_{j-1/2}}^{x_{j+1/2}} \frac{\partial f(u)}{\partial x} dx dt = 0, \tag{2.45}$$

which can be rewritten, using the fundamental theorem of calculus

$$\begin{aligned} &\int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t^{n+1}) dx - \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t^n) dx \\ &+ \int_{t^n}^{t^{n+1}} F(u(x_{j+1/2})) - \int_{t^n}^{t^{n+1}} F(u(x_{j-1/2})) dt = 0. \end{aligned} \tag{2.46}$$

With Eq. (2.44), we then obtain

$$u_j^{n+1} = u_j^n + \frac{1}{\Delta x}\left(\int_{t^n}^{t^{n+1}} F(u(x_{j+1/2})) - \int_{t^n}^{t^{n+1}} F(u(x_{j-1/2})) dt\right). \tag{2.47}$$

We define the numerical flux function

$$\bar{F}_{j-1/2} = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} F(u(x_{j-1/2}, t)) dt, \tag{2.48}$$

resulting in a concise formulation of the update rule:

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(\bar{F}_{j+1/2}^n - \bar{F}_{j-1/2}^n\right). \tag{2.49}$$

One can easily check that if $\bar{F}_{x_L} = \bar{F}_{-1/2} = 0$ and $\bar{F}_{x_R} = \bar{F}_{N+1/2} = 0$, this update rule is by construction conservative:

$$\begin{aligned}
\sum_{j=0}^{N} u_j^{n+1} &= \sum_{j=0}^{N} u_j^n - \frac{\Delta t}{\Delta x} \sum_{j=0}^{N} \left(\bar{F}^n_{j+1/2} - \bar{F}^n_{j-1/2} \right) \\
&= \sum_{j=0}^{N} u_j^n - \frac{\Delta t}{\Delta x} \left(\bar{F}^n_{1/2} - \bar{F}^n_{-1/2} + \bar{F}^n_{3/2} - \bar{F}^n_{1/2} + \cdots + \bar{F}^n_{N+1/2} - \bar{F}^n_{N-1/2} \right) \\
&= \sum_{j=0}^{N} u_j^n - \frac{\Delta t}{\Delta x} \left(\bar{F}^n_{N+1/2} - \bar{F}^n_{-1/2} \right) \\
&= \sum_{j=0}^{N} u_j^n .
\end{aligned} \tag{2.50}$$

Eq. (2.49) represents a discrete statement of conservation, similar to the general form of a conservation law (2.1). A change of the quantity u in the control volume j over one time step Δt can only be caused by a flux $\bar{F}$ of that quantity over the cell boundaries.

However, since the values of u are only known at the cell mid points u_j, to close Eq. (2.49) the numerical flux can not be evaluated directly. Hence, numeric differentiation approaches are required to evaluate the fluxes Eq. (2.48). Intuitively, one might assume that a simple central difference is a suitable way to approximate the flux

$$\bar{F}^n_{j+1/2} = \frac{F(u_j^n) + F(u_{j+1}^n)}{2}. \tag{2.51}$$

However, as it turns out, the central difference is unconditionally unstable, even for the linear transport equation (2.3). From a physical perspective, this is due to the fact that the central difference approximates the flux based on values of u in front and behind of the the cell boundary. However, since information is only propagated in one direction (for the linear transport equation with $c > 0$ to the right), the difference scheme can only be based on information upstream or the underlying physics are violated. From a mathematical perspective, it can also be shown that the central scheme is not energy bounded [92] (i.e. energy is added to the system in each iteration), which ultimately leads to divergence, confirming the previous physical argument that a central difference is not suitable.

To overcome this limitation, we can define a so-called upwind scheme that selects which neighbors flux needs to be used based on the local direction of propagation of information. For Eq. (2.3), with $F = c \cdot u$, the first order upwind flux is defined as

$$\bar{F}^n_{j+1/2} = c \begin{cases} u_j^n, & c \geq 0 \\ u_{j+1}^n, & c < 0. \end{cases} \tag{2.52}$$

Plugged into Eq. (2.49), we obtain

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x} \cdot c \begin{cases} u_j^n - u_{j-1}^n, & c \geq 0 \\ u_{j+1}^n - u_j^n, & c < 0. \end{cases} \tag{2.53}$$

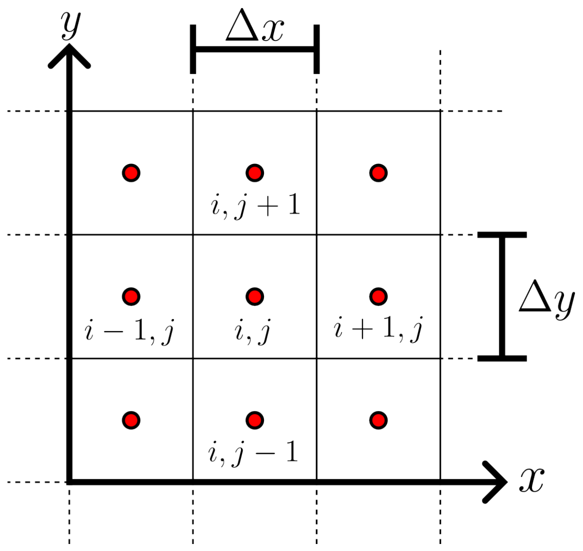


Figure 2.7.: Schematic representation of spatial discretization in finite volume method in 2D.

The upwinding scheme is indeed (conditionally) stable when choosing adequate time steps according to the CFL condition

$$\max_j |F'(u_j^n)| \leq \frac{1}{2}. \tag{2.54}$$

Eq. (2.53) can be rewritten in the equivalent form

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x} \cdot \left(c \frac{u_{j+1}^n - u_{j-1}^n}{2} + |c| \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{2} \right). \tag{2.55}$$

This formulation contains a central difference and an additional undivided second difference. This term acts in a dissipative manner. Evidently, the stable upwinding scheme is equivalent to a central difference, combined with an additional dissipative term. Many finite volume schemes for conservation laws are constituted in a similar manner, consisting of a central difference, combined with a dissipative terms. As described by Swanson and Turkel [96]: “ [...] a successful artificial dissipation model for a central difference schemes should approximate an upwind scheme in the neighborhood of shocks.”

In the following we will introduce the Jameson-Schmidt-Turkel scheme as an exemplary central difference scheme for the Euler equations.

2.2.2. Euler Equations

The Jameson-Schmidt-Turkel scheme (JST-scheme) is a central difference scheme with artificial dissipation, originally introduced for solving the compressible Euler equations [76]. The presented formulation is based on [3]. For notational convenience, we consider a two dimensional Cartesian grid with uniform control volumes $V_{i,j}$ with volume $\mathrm{vol}(V_{i,j}) = \Delta x \cdot \Delta y \subset \Omega$ (see Fig. 2.7). This approach can easily be extended to general structured [78, 96] or unstructured grids [79]. The integral form of the Euler equations for a control volume $V_{i,j}$ is given by:

$$\frac{\partial}{\partial t}\int_{V_{i,j}} \boldsymbol{W}_{i,j} dV + \oint_{\partial V_{i,j}} \vec{\boldsymbol{F}} \cdot \vec{\boldsymbol{n}} dS = 0, \tag{2.56}$$

where the vector arrows indicate a vector in Ω and $\vec{\boldsymbol{F}} = (\boldsymbol{F}_x, \boldsymbol{F}_y)$ with $\boldsymbol{W}, \boldsymbol{F}_x, \boldsymbol{F}_y$ as in Eq. (2.41). The second integral is approximated by the sum:

$$\oint_{\partial V_{i,j}} \boldsymbol{F}(\boldsymbol{W}) \cdot \boldsymbol{n} dS \approx \Delta y \left((\boldsymbol{F}_x)_{i-1/2,j} - (\boldsymbol{F}_x)_{i+1/2,j} \right) + \Delta x \left((\boldsymbol{F}_y)_{i,j+1/2} + (\boldsymbol{F}_y)_{i,j-1/2} \right). \tag{2.57}$$

As for the one-dimensional example, each numerical flux is approximated by the sum of a central difference and a dissipative term. For example:

$$(\boldsymbol{F}_x)_{i+1/2,j} \approx \Delta y F_x(\boldsymbol{W}_{i,j+1/2}) + \boldsymbol{D}_{i+1/2,j}, \tag{2.58}$$

where $\boldsymbol{W}_{i+1/2,j} = 1/2(\boldsymbol{W}_{i,j} + \boldsymbol{W}_{i+1,j})$ and with the dissipative term $\boldsymbol{D}_{i+1/2,j}$. The dissipative term consists of a blending of second differences $D^{(2)}_{i+1/2,j}$ and fourth differences $D^{(4)}_{i+1/2,j}$

$$\boldsymbol{D}_{i+1/2,j} = \Lambda_{i+1/2,j} \left(D^{(2)}_{i+1/2,j} s^{(2)}_{i,j+1/2} - D^{(4)}_{i+1/2,j} s^{(4)}_{i,j+1/2} \right). \tag{2.59}$$

The factor $\Lambda_{i+1/2,j}$ is the sum of the largest eigenvalues of the flux Jacobians, also called spectral radius of the flux Jacobians

$$\Lambda_{i+1/2,j} = \frac{1}{2} \left((\lambda_x)_{i,j} + (\lambda_x)_{i+1,j} + (\lambda_y)_{i,j} + (\lambda_y)_{i+1,j} \right), \tag{2.60}$$

with

$$(\lambda_x)_{i,j} = \left(|u_{i,j}| + c_{i,j} \right) \Delta x, \qquad (\lambda_y)_{i,j} = \left(|v_{i,j}| + c_{i,j} \right) \Delta y. \tag{2.61}$$

Fourth differences are used to obtain a second order scheme in smooth regions of the flow, while the second order difference in the viscosity of the shock result in a first order scheme, helping to avoid unwanted oscillations. The more sophisticated *roe average* [97, 92] could also be used to calculate the face value of $\Lambda_{i+1/2,j}$. The sensor $s^{(2)}$, switches on when high pressure gradients are encountered in the vicinity of shocks. In these regions, the sensor function $s^{(4)}$ switches off the fourth differences to avoid oscillations in the neighborhood of the shock. In smooth regions of the flow, the fourth differences are however required to avoid odd-even decoupling. The sensors are defined as

$$\begin{aligned} s^{(2)}_{i+1/2,j} &= k^{(2)} \max(\Upsilon_{i-1,j}, \Upsilon_{i,j}, \Upsilon_{i+1,j}, \Upsilon_{i+2,j}) \\ \Upsilon_{i,j} &= \left| \frac{p_{i+1,j} - 2p_{i,j} + p_{-1,j}}{p_{i-1,j} + p_{i,j} + p_{i+1,j}} \right| \\ s^{(4)} &= \max(0, k^{(4)} - \epsilon^{(2)}), \end{aligned} \tag{2.62}$$

with the two constants $k^{(2)}, k^{(4)}$. Typical values for these constants are $k^{(2)} = 1/2$ and $k^{(4)} = 1/64$. The difference operators $\boldsymbol{D}^{(2)}$ and $\boldsymbol{D}^{(4)}$ are standard undivided approximations of the second and fourth derivatives, applied to the vector of conserved variables

$$\begin{aligned} \boldsymbol{D}^{(2)}_{i+1,j} &= \boldsymbol{W}_{i+1,j} - \boldsymbol{W}_{i,j} \\ \boldsymbol{D}^{(4)}_{i+1,j} &= \boldsymbol{W}_{i+2,j} - 3\boldsymbol{W}_{i+1,j} + 3\boldsymbol{W}_{i,j} - \boldsymbol{W}_{i-1,j}. \end{aligned} \tag{2.63}$$

The JST-scheme applies an equal viscosity to each of the four equations, which is scaled with the largest eigenvalue of the flux Jacobians. To reduce the dissipativeness of the scheme the matrix dissipation scheme was introduced [78, 96]. Here $\Lambda_{i+1/2,j}$ is replaced with a matrix, based on the flux Jacobian, such that the viscosity in each equation is scaled by the corresponding eigenvalue. The modified version of Eq. (2.59) reads

$$\boldsymbol{D}_{i+1/2,j} = |A|_{i+1/2,j} \left(D^{(2)}_{i+1/2,j} s^{(2)}_{i,j+1/2} - D^{(4)}_{i+1/2,j} s^{(4)}_{i,j+1/2} \right). \tag{2.64}$$

The matrix $|A|_{i+1/2,j}$ is defined, as the flux Jacobian $A = \partial \boldsymbol{F}_x / \partial \boldsymbol{W}$ diagonalized with absolute values of the eigenvalues as

$$|A| = T_x |\Lambda_x| T_x^{-1}, \quad |B| = T_y |\Lambda_y| T_y, \tag{2.65}$$

with the eigendecompositions $A = T_x \Lambda_x T_x^{-1}$ and $B = T_y \Lambda_x T_y^{-1}$. The full matrices are, for example, shown in [96]. The matix dissipation scheme is comparable in accuracy to upwinding schemes but computationally similarly efficient as the JST-scheme [3, Ch. 4.3.1]. For a comparison with other schemes, see for example [75].

2.3. Neural Networks

In this section, some of the fundamental neural network (NN) concepts are introduced, starting with the architecture of a fully connected feed-forward NN. Afterwards, the properties of one of the most essential building blocks, the activation functions, are discussed in more detail. Next, the basics of relevant optimization/training algorithms are introduced. Lastly, improvements to the standard architecture, namely the *weight normalization* and the *Fourier feature embedding* methodologies are presented.

Architecture

One of the simplest NN architectures is the feed-forward fully connected NN. It is a composition of multiple pairs of linear layers and non-linear activation functions. We consider the neural network $\hat{\boldsymbol{u}} : \mathbb{R}^{N_0} \longrightarrow \mathbb{R}^{N_D}$ with a depth of D layers. Each layer $k = 1 \dots D$ is defined by a function $f^k : \mathbb{R}^{N_{k-1}} \longrightarrow \mathbb{R}^{N_k}$ with

$$\begin{aligned} \hat{\boldsymbol{u}} &= f^D \circ \cdots \circ f^k \circ \cdots \circ f^1(\boldsymbol{r}), \\ f^k(\boldsymbol{r}) &= \sigma^k(w^k \boldsymbol{r} - b^k). \end{aligned} \tag{2.66}$$

The trainable parameters of the model are the bias vector $b^k \in \mathbb{R}^{N_k}$ and the weight matrices $w^k \in \mathbb{R}^{N_k} \times \mathbb{R}^{N_{k-1}}$. They are optimized by minimizing the loss function. The upper index k indicates the layer. The activation function $\sigma^k(\cdot)$ is further discussed in Sec. 2.3. Each layer can have a different number of nodes N_k, often called neurons. For the final layer D, a linear activation function is typically used such that the image of the output vector is $\mathbb{R}^{N_M}$. NNs are often depicted by layer-wise graphs with nodes, as for example shown in Fig. 2.8. In such a graph, the nodes represent the components of the output vector (i.e. $(f_1^k, f_2^k, \dots, f_{N_k}^k)^T = f^k$) of the neurons in layer k, while the connections represent

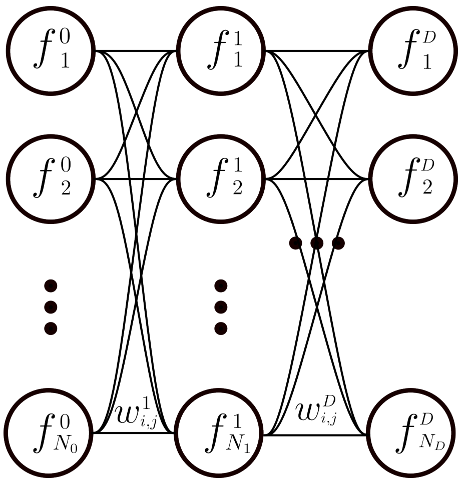


Figure 2.8.: Schematic depiction of a fully connected feed-forward NN with.

the weight matrices w^k. The fully connected architecture is at the core of many deep learning models. Even more complex architectures like convolutional NNs or transformers typically incorporate fully connected layers. An important motivation for the usage of fully connected architectures have been the universal function approximation capabilities that have been shown for these kinds of networks. Fundamentally the universal function approximation theorems state that a network, similar to (2.66) of arbitrary width N_k [18] or arbitrary depth D [98] can approximate arbitrary multivariate functions from certain function classes up to any desired accuracy. It should, however, be stressed that the universal function theorems do not make any assertions on how these approximations can be obtained. This already alludes to two major challenges. Firstly, the required depth or width of the network is unknown. These so-called *hyperparameters* typically need to be selected empirically. Secondly, the parameters of the network (the weights w^k and biases b^k) have to be determined. In practice, gradient based optimization algorithms are employed to determine optimal values for these parameters. This process is often called *training*. The underlying optimization problem is however generally non-convex and there is no guarantee to find the optimal parameters that would result in the desired exactness.

Activation Functions

An essential part of the NN architecture in Eq. (2.66) is the activation function σ. Activation functions are crucial for enhancing the expressibility of the network because they are the only non-linear building block. Hence they give NNs the ability to approximate complex nonlinear relationships, even as a global ansatz function. Additional characteristics of activation functions to consider are [62]:

1. **Boundedness**: When training NNs using gradient-based methods, activation functions with a finite range have shown to be more stable.
2. **Evaluation Cost**: Due to the iterative nature of training algorithms and the stacked architecture of the network, the activation function is evaluated many times during

the training and evaluation of the network. Therefore, a low computational effort of its evaluation and the evaluation of its derivatives is desirable.

3. **No vanishing gradients:** The image of sigmoid-like activation functions is typically bounded to a range $(-1, 1)$. Repeated application of the chain rule of derivatives leads to exponentially diminishing gradient values in the first layers of the network impeding the training.
4. **Differentiability:** For most NN applications, continuous step-function like activations with a non-differentiable point are well established. However, for the physics-informed NNs that are considered in this work, the activation needs to be continuously differentiable. This is due to the fact that higher order derivatives are required in the physics-informed loss function, leading to discontinuous jumps in the loss landscape, when activations are not continuously differentiable. These jumps are undesirable for the training of NNs.

Traditional activation functions for NNs are sigmoid-like. A typical example is the hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.67}$$

This activation is bounded ($\tanh(x) \in (-1, 1), \quad x \in \mathbb{R}$) and continuously differentiable. The derivative is limited to $\tanh'(x) \in (0, 1), \quad x \in \mathbb{R}$. Therefore, NNs with multiple layers of hyperbolic tangent activations can suffer from vanishing gradients. This can explain that the so-called rectified linear unit (ReLu$(\cdot)$) has gained popularity in the late 2010s:

$$\mathrm{ReLu}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}. \tag{2.68}$$

The ReLu activation function is not bound. However, the derivatives for $x \geq 0$ are exactly one, which can, in practice, help to avoid vanishing gradients. Furthermore, ReLu and its gradients can be evaluated more efficiently than sigmoid-like activation. On the downside, the range is not bounded for positive inputs and it is not differentiable at $x = 0$.

Recently, adaptive activation functions featuring additional trainable parameters, have gained popularity. We consider layer-wise adaptive activation functions, introduced by Jagtap et al. [42]:

$$\tanh^k_{\mathrm{adap}}(x) = \tanh\left(n\omega^k \cdot x\right), \quad k = 1, 2 \ldots, D - 1, \tag{2.69}$$

where n is a constant scaling factor and ω^k is an additional trainable parameter. The slope of the activation functions in each layer can be adapted during training. This helps to avoid vanishing gradients while maintaining the bounded range of $(0, 1)$ of the hyperbolic tangent. The computational effort per epoch is however marginally increased, compared to the normal tanh because one trainable parameter per network layer is added. In this work, the original adaptive activation function is modified as follows:

$$\tanh^k_{\mathrm{adap}}(x) = n\omega^k \tanh(x), \quad k = 1, 2 \ldots, D - 1. \tag{2.70}$$

This change allows not only for changes in the slope of the activation function but also for changes in the magnitude. Empirically, it was observed that this further improves the convergence properties. With the trainable parameter outside of the hyperbolic tangent, the boundedness is no longer guaranteed. However, in practice, this does not seem to be an issue, as the trainable parameter typically does not diverge.

Mini-Batch Training and Optimization Algorithms

Neural Networks are trained using gradient-based algorithms. Let us for example consider a regression of a function $u : \mathbb{R} \longrightarrow \mathbb{R}$. The NN is trained on a training data set, given by N sample pairs:

$$\{(x_i, y_i)| \ i = 0, \dots, N-1, \ y_i = u(x_i)\}. \tag{2.71}$$

To find optimal parameters of the NN, we solve the unconstrained optimization problem:

$$\min_{\theta} \left(\frac{1}{N} \sum_{i=0}^{N-1} \mathcal{L}(\hat{u}_{\boldsymbol{\theta}}(x_i)) \right), \quad \boldsymbol{\theta} \in \mathbb{R}^{N_\theta}, \tag{2.72}$$

with a loss function $\mathcal{L}$. The loss is typically based on a norm of the difference between the prediction and actual function value, e.g. $\mathcal{L}\hat{u}(x_i)) = \|y_i - \hat{u}(x_i)\|_2{}^2$. Here, $\boldsymbol{\theta}$ is the parameter vector of length $N_{\boldsymbol{\theta}}$, containing a concatenation of all entries of the weight matrices w^k and the biases b^k in Eq. (2.66). This vector is extended by the adaptive weights w^k when adaptive activation functions (2.70) are used.

In practice, stochastic gradient descent (SGD) algorithms are oftentimes used. The optimization is called stochastic because during each training iteration the loss function is only evaluated on a subset of the total training data, called a *batch*. The stochastic loss function for the parameters $\boldsymbol{\theta}$ in iteration i is given by:

$$\tilde{\mathcal{L}}_i(\boldsymbol{\theta}) = \frac{1}{N_{\text{batch}}} \sum_{j \in \chi_i} \mathcal{L}(\hat{u}_{\boldsymbol{\theta}}(x_j), y_j). \tag{2.73}$$

The samples of the batch χ_i are chosen as follows. Let $\tilde{\boldsymbol{X}}$ be a random permutation of the dataset index vector $\boldsymbol{X} = (0, \dots, N-1)$. Then we have:

$$\chi_i = (\tilde{X}_{i \cdot N_{\text{batch}}}, \dots, \tilde{X}_{(i+1)\cdot(N_{\text{batch}})-1}) \ \ i = 1, \dots, \lfloor \frac{N}{N_{\text{batch}}} \rfloor. \tag{2.74}$$

A random exclusive subset of N_{batch} samples of the whole dataset gets picked in each iteration until the whole dataset has been used. If N_{batch} is not a factor of N, the last batch with less than N_{batch} samples can either be used for a final iteration or be dropped as is the case in Eq. (2.74). Typically $\lfloor N/N_{\text{batch}} \rfloor$ iterations are not enough to fully train a NN until convergence. Therefore, this process is repeated and each run through the whole dataset is called an *epoch*. In the context of deep-learning, we speak of *mini-batch training* when the gradients are calculated as the mean over a subset of the whole data set. When the whole data set is used, the terms *batch training* or *fullbatch training* are commonly used.

Algorithm 1 stochastic gradient descent

```
Require: $\tau$: Step size/learning rate
Require: $\tilde{\mathcal{L}}_i(\boldsymbol{\theta})$ stochastic loss/objective function at iteration $i$
Require: $\boldsymbol{\theta}_0$: initial parameters
  $i \leftarrow 0$
  while $\boldsymbol{\theta}_i$ not converged do
      $i \leftarrow i + 1$
      $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} - \tau \boldsymbol{\nabla}_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_i(\boldsymbol{\theta}_i)$        ▷ Perform step into negative gradient direction
  end while
```

Given the stochastic loss function and initial values for $\boldsymbol{\theta}_0$, the basic SGD algorithm is given in Alg. 1. The stochastic algorithms requires less memory than regular gradient decent. In fact the amount of memory required, can be scaled by increasing or decreasing the batch size N_{batch}. When training on graphics cards, each batch of data may however have to be reloaded onto the graphics card, resulting in higher wall clock times per iteration. In addition, the stochastic nature of the algorithm is also crucial for avoiding local minima during the optimization process [99]. The step size τ is often called the *learning rate* in the machine learning context. In practice the standard SGD algorithm in Alg. 1 acts as a baseline for algorithms that are used today. One of the most popular SGD variants is the adaptive momentum estimation (ADAM) algorithm, proposed by Kingma and Ba in 2014 [100]. This algorithm extends the SGD algorithm by storing a running average of the gradients and the second moments from past updates as shown in Alg. 2. Here ϵ is a small number to avoid division by zero. The algorithm effectively scales the learning rate of each trainable parameter, based on their training history. Large variances in the gradients decrease the learning rate and high average gradients increase the learning rate. The influence of previous iterations on the moment estimates m_i and v_i is exponentially decreasing. The moments are corrected for the initialization with zero, as described in [100]. In practice ADAM has shown to be a robust choice for training NNs with large numbers of parameters and large data sets.

Besides gradient descent optimization algorithms, one might also consider quasi-Newton methods, due to their faster convergence rate. For a comparison of convergence properties of different optimization algorithms, the interested reader is referred to [33]. In practice, quasi-Newton algorithms are rarely used for training NNs due to computational limitations. The calculation (or approximation) of the Hessian that is required for these algorithms becomes a limiting factor when applying them to NNs because they often contain $N_{\boldsymbol{\theta}} \gtrsim 10^4$ parameters resulting in a Hessian with $(N_{\boldsymbol{\theta}})^2$ entries. Furthermore, many of the available algorithms are not designed to be applied to stochastic loss functions such as Eq. (2.73). In this work we consider problems with networks of intermediate size with $N_{\boldsymbol{\theta}} \lesssim 10^5$. When graphics cards with sufficiently large memory are available, such problems can be addressed, using the quasi-Newton limited memory variant of the Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) [31]. This algorithm is especially useful for large scale optimization problems, because it uses an estimate of the Hessian, based on a few vectors. Hence, its required memory only scales linearly with $N_{\boldsymbol{\theta}}$. A full description of the algorithm is beyond the scope of this work and the

Algorithm 2 Adaptive momentum estimation (ADAM) [100]

```
Require: $\tau$: Step size/learning rate
Require: $\beta_1 \in [0, 1))$: Exponential Decay rates for gradient moment estimate
Require: $\beta_2 \in [0, 1)]$: Exponential Decay rate for second moment estimate
Require: $\tilde{\mathcal{L}}_i(\boldsymbol{\theta})$ stochastic loss/objective function at iteration $i$
Require: $\boldsymbol{\theta}_0$: initial parameters
  $i \longleftarrow 0$
  $\boldsymbol{m}_0 \longleftarrow 0$
  $\boldsymbol{v}_0 \longleftarrow 0$
  while $\boldsymbol{\theta}_i$ not converged do
      $i \longleftarrow i + 1$
      $\boldsymbol{g}_i \longleftarrow \boldsymbol{\nabla}_\theta \tilde{\mathcal{L}}_i(\boldsymbol{\theta}_i)$    ▹ Get the gradients with respect to trainable parameters
      $\boldsymbol{m}_i \longleftarrow \beta_1 \cdot \boldsymbol{m}_{i-1} + (1 - \beta_1) \cdot \boldsymbol{g}_i$    ▹ Estimate the first moment with running average
      $\boldsymbol{v}_i \longleftarrow \beta_2 \cdot \boldsymbol{v}_{i-1} + (1 - \beta_2) \cdot (\boldsymbol{g}_i)^2$    ▹ Estimate the second moment with running average
      $\hat{\boldsymbol{m}}_i = \frac{\boldsymbol{m}_i}{1 - (\beta_1)^i}$    ▹ Correct the first moment for 0 initialization
      $\hat{\boldsymbol{v}}_i = \frac{\boldsymbol{v}_i}{1 - (\beta_2)^i}$    ▹ Correct the second moment for 0 initialization
      $\boldsymbol{\theta}_i \longleftarrow \boldsymbol{\theta}_{i-1} - \tau \frac{\hat{\boldsymbol{m}}_i}{\sqrt{\hat{\boldsymbol{v}}_i} + \epsilon}$    ▹ Perform step into negative gradient direction
  end while
```

interested reader is referred to the original paper [31]. Note that the basic formulation of this algorithm is unsuitable for the stochastic optimization problem, given in Eq. (2.73) and instead requires the formulation in Eq. (2.72). Further extensions of the algorithm for stochastic loss functions have been proposed in the literature [101, 102] but are currently not implemented in the deep learning libraries used in this work. Therefore, we can not benefit from the reduction in memory requirements of the stochastic formulation when using the L-BFGS algorithm and this algorithm is more prone to converge in local minima. Hence, it is often sensible to train initially, using an SGD algorithm such as ADAM. Once a pre-converged estimate for $\boldsymbol{\theta}$ is obtained, the second order L-BFGS algorithm can be used to fine-tune the results.

Automatic Differentiation

To utilize the gradient based optimization algorithms introduced in Sec. 2.3, the loss gradient $\nabla_\theta \mathcal{L}_\theta$ is essential. This gradient is calculated using so-called *automatic differentiation* (AD) also called *algorithmic differentiation.* As shown in Sec. 2.3, a NN can be seen as a composition of multiple layers of analytical functions. This would in theory allow us to derive analytic expressions for the derivatives of the loss function with respect to the network parameters. However, explicitly deriving these analytic expression, even when using computational tools such as symbolic mathematical software libraries, quickly becomes intractable for functions such as NNs. This is due to combination of different factors. It is for example obvious that the application of the product rule of derivatives, produces two terms requiring separate function evaluations. When the functions in these two terms again contain products (as is the case in deep NNs), we have a number of terms that is exponentially growing with the number of layers. To avoid explicit derivation of closed form expressions of the derivatives, automatic differentiation instead stores intermediate values of the computation that are reused for the calculation of the derivative values.

We distinguish between two basic modes of AD, forward-mode and backward mode. Let us consider a general multivariate vector valued function $\boldsymbol{g} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. We are interested in the Jacobian of the function:

$$\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix} \tag{2.75}$$

Forward mode AD allows us to calculate one column of the Jacobian per evaluation of the function, while reverse mode AD allows us to calculate one row of the Jacobian per evaluation of the function. Hence, for training NNs, where we have many inputs of the loss function (i.e. the trainable parameter vector $\boldsymbol{\theta}$) and one scalar output (i.e the value of the loss function $\tilde{\mathcal{L}}(\boldsymbol{\theta})$), it makes sense to use reverse mode AD. In the context of training NNs, the reverse-mode AD algorithm is called *backpropagation*. For a more detailed view at the forward mode and the theoretical background, the interested reader is referred to the book by Griewank and Walther [24]

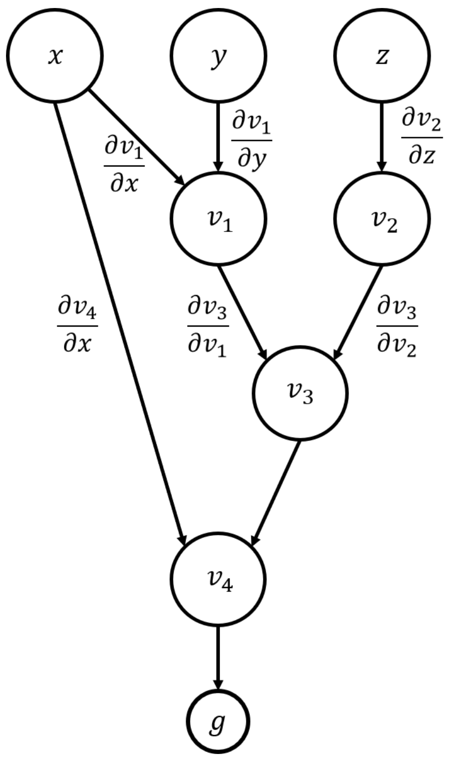
To illustrate how reverse-mode AD can be used to numerically calculate derivatives of an analytic expression, let us consider the function:

$$g(x, y, z) = (x + y)\sin(z) + x, \tag{2.76}$$

where we seek the gradient $\boldsymbol{\nabla} g = (\partial/\partial_x, \partial/\partial_y, \partial/\partial_z)^T g(x, y, z)$. With classical derivative rules, we can derive the expected expressions for the derivatives by hand:

$$\boldsymbol{\nabla} g(x, y, z) = \begin{bmatrix} \sin(z) + 1 \\ \sin(z) \\ (x + y) \cdot \cos(z) \end{bmatrix}, \tag{2.77}$$

which we will use as our reference. Let us now demonstrate how reverse-mode AD can determine exact values for these derivatives. Given a set of values for (x, y, z), we can break down Eq. (2.76) into trivial sub-operations which are performed on the inputs of the function. These sub-operations (labeled here with v_0, v_1, v_2, v_3 and v_4) can be summarized in a computational graph, as shown in Fig. 2.9. First, we perform the forward pass, meaning that we calculate the intermediate values of the computation (i.e. $v_1, \ldots v_4$). These values are stored in memory. Furthermore, we can also at this point decide to store the derivatives of each intermediate node with respect to its inputs. Since all of the performed sub-operations are trivial, these derivatives are previously known (e.g. $\partial v_2/\partial z = \partial(\sin(z))/\partial z = \cos(z)$). Note that we store the numerical values for these derivatives and not the analytic expression. For example if $z = 0$, we store $v_2 = \sin(0) = 0$



x
y
z
$\frac{\partial v_1}{\partial x}$
$\frac{\partial v_1}{\partial y}$
$\frac{\partial v_2}{\partial z}$
v_1
v_2
$\frac{\partial v_4}{\partial x}$
$\frac{\partial v_3}{\partial v_1}$
$\frac{\partial v_3}{\partial v_2}$
v_3
v_4
g

Figure 2.9.: Schematic graph of computation for Eq. (2.76).

and $\partial v_2/\partial z = \cos(0) = 1$. The different variables that are stored in each step of the computation during the forward pass, are shown in Alg. 3.

Once the forward pass is completed, and values for all the intermediate sub-operation v_i are stored, we perform the backward pass. Starting from the function output g, we calculate the so-called *adjoints* for each sub-operation, denoted as $\bar{v}_i$. The adjoints of a node v_i are defined as $\partial g/\partial v_i$ and can be calculated using the chain rule:

$$\bar{v}_i := \frac{\partial g}{\partial v_i} = \sum_{j \in Q(i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}, \tag{2.78}$$

where Q_i denotes all the child-nodes of the node i (all operations that node i has been used for in the forward pass). The adjoints for each v_i are calculated starting from the bottom of the graph and populating each adjoint by stepping back up towards the inputs, as shown in Alg. 3, hence giving rise to the name *reverse-mode* or *backpropagation* for NNs. Note that the calculation in the backward pass requires the previously stored values for the partial derivatives in the forward pass as well as the node values v_i themselves. Alternatively, since the computational graph and the values of v_i are known, the partial derivatives can also be calculated at this point to reduce memory requirements. Finally, the adjoints of the input variables $\bar{x}, \bar{y}, \bar{z}$ can be calculated, corresponding to the components of the derivative vector ∇g.

Indeed, the comparison to Eq. (2.77) shows the correct result. This is not surprising, since the backward steps correspond to the well known chain-rule of derivatives, that we also used to derive the expressions in Eq. (2.77). However, by storing intermediate values v_i and the partial derivatives of the trivial functions, the explicit derivation of the full derivative expressions can be avoided. As shown in Sec. 2.3, a NN also consists of many trivial operations and the same procedure can therefore be applied for the calculation of loss function gradients. The storage of the intermediate values is comparatively memory intensive. The overall computational of calculating a column of the Jacobian (Eq. (2.75)) is however only on the order of the cost of the function evaluation itself, as shown in [24]. This explains why back-propagation is so effective for training NNs with large numbers of parameters.

Similarly to the parameter gradients, AD can also be used to calculate the derivatives with respect to the input vector of the network $\boldsymbol{r}$. This is used to calculate derivatives in the residual loss terms of PINNs, which we will see in Sec. 2.4.1.

Algorithm 3 Example for reverse-mode AD

```
  $g(x, y, z) = (x + y)\sin(z) + x$
  Forward Pass:
  1. $v_1 \leftarrow x + y, \frac{\partial v_1}{\partial x} \leftarrow 1, \frac{\partial v_1}{\partial y} \leftarrow 1$
  2. $v_2 \leftarrow \sin(z), \frac{\partial v_2}{\partial z} \leftarrow \cos(z)$
  3. $v_3 \leftarrow v_1 \cdot v_2, \frac{\partial v_3}{\partial v_1} \leftarrow v_2, \frac{\partial v_3}{\partial v_2} \leftarrow v_1$
  4. $v_4 \leftarrow v_3 + x, \frac{\partial v_4}{\partial v_3} \leftarrow 1, \frac{\partial v_4}{\partial x} \leftarrow 1$
  Backward Pass:
  1. $\bar{v}_4 = \frac{\partial g}{\partial v_4} \leftarrow 1$
  2. $\bar{v}_3 = \frac{\partial g}{\partial v_3} \leftarrow \bar{v}_4 \cdot \frac{\partial v_4}{\partial v_3} = 1 \cdot 1 = 1$
  3. $\bar{v}_2 = \frac{\partial g}{\partial v_2} \leftarrow \bar{v}_3 \cdot \frac{\partial v_3}{\partial v_2} = 1 \cdot v_1 = v_1$
  4. $\bar{v}_1 = \frac{\partial g}{\partial v_1} \leftarrow \bar{v}_3 \cdot \frac{\partial v_3}{\partial v_1} = 1 \cdot v_2 = v_2$
  5. $\bar{v}_x = \frac{\partial g}{\partial x} \leftarrow \bar{v}_1 \cdot \frac{\partial v_1}{\partial x} + \bar{v}_4 \cdot \frac{\partial v_4}{\partial x} = v_2 \cdot 1 + 1 \cdot 1 = \sin(z) + 1$
  6. $\bar{v}_y = \frac{\partial g}{\partial y} \leftarrow \bar{v}_1 \cdot \frac{\partial v_1}{\partial y} = v_2 \cdot 1 = \sin(z)$
  7. $\bar{v}_z = \frac{\partial g}{\partial z} \leftarrow \bar{v}_2 \cdot \frac{\partial v_2}{\partial z} = v_1 \cdot \cos z = (x + y) \cdot \cos(z)$
```

Modifications to the Classical Architecture

The classical fully connected feed-forward architecture, presented in Sec. 2.3, has proven to be a solid baseline for many NN applications. Over the years, many minor alterations of this architecture have been proposed. In this work two modifications are used which have shown to be a robust extension, the *weight normalization* and the *Fourier feature embedding* methodologies.

Weight Normalization One methodology that has generally shown to improve the trainability of NNs is the so-called *weight normalization*. The basic idea is to replace each vector in the weighting matrix of each layer $w^k \in$ by a normalized vector $\boldsymbol{z}/\|z\|$ and a scalar magnitude ϱ:

$$\boldsymbol{w} \longrightarrow \varrho \frac{\boldsymbol{z}}{\|\boldsymbol{z}\|}. \tag{2.79}$$

This improves the conditioning of the optimization problem. It has consistently shown to outperform the classical fully connected architecture [103]. The factorization adds a number of additional parameters ϱ on the order of the number of neurons. Therefore, the increase in computational effort per epoch is again comparatively small since the majority of parameters are still contained in the weight matrices.

Spectral Bias and Fourier Feature Embedding The trainability of deep NNs generally suffers from a phenomenon called *spectral bias*. During the training of a NN, the low frequencies of the target function are generally learned first. The higher frequency modes are only learned later during training [104, 105]. In practice, this means that multi-scale problems are hard to solve for NNs and convergence is generally poor for these problems. The random Fourier-feature embedding [63, 64] aims to overcome these limitations. An additional non-trainable layer is introduced which encodes the input vector of the NN using a set of sinusoidal functions with randomly sampled frequencies:

$$\begin{gathered}
\hat{\boldsymbol{u}}_{\mathrm{fe}}(\boldsymbol{r}) : \mathbb{R}^{D_0} \longrightarrow \mathbb{R}^{2D_{\mathrm{fe}}}, \\
\begin{pmatrix} \hat{u}_{\mathrm{fe},i} \\ \hat{u}_{\mathrm{fe},i+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \sum_j^{D_0} \sin\big(\phi_{i,j} r_j\big) \\ \sum_j^{D_0} \cos\big(\phi_{i,j} r_j\big) \\ \vdots \end{pmatrix}, \\
i = 2k-1, \quad k = 1 \dots D_{\mathrm{fe}}.
\end{gathered} \tag{2.80}$$

The frequencies $\phi_{i,j}$ are typically sampled from a normal distribution $\mathcal{N}(0, \sigma_\phi^2)$. The standard deviation σ_ϕ of the distribution is a hyperparameter of the resulting model.

2.4. Physics-Informed Neural Networks

In the following section the physics-informed neural network methodology, as popularized by Raissi et al [29], is introduced in detail. Furthermore, various alterations of the method that have been proposed in literature are discussed.

2.4.1. Architecture

Physics-informed NNs are deep NNs, which are employed to solve problems governed by partial differential equations. In this work, approximation of solutions of boundary value problems are considered. PINNs can however also be used to solve inverse problems. To demonstrate how the PINN approach can be used to approximate PDE solutions, let us consider a general initial-boundary value problem for the unknown solution $\boldsymbol{u}$ on the spatial domain $\Omega \subset \mathbb{R}^d$ and in the time interval $(0, T) \subset \mathbb{R}$:

$$\begin{aligned} \mathcal{R}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t) &= 0 \qquad (\boldsymbol{x}, t) \in (\Omega \times (0, T)) \\ \mathcal{B}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t) &= 0 \qquad (\boldsymbol{x}, t) \in (\partial\Omega \times (0, T)) \\ \mathcal{I}(\boldsymbol{u}(\boldsymbol{x}, 0), \boldsymbol{x}) &= 0 \qquad \boldsymbol{x} \in \Omega, \end{aligned} \tag{2.81}$$

where $\mathcal{R}$ is a general differential operator and $\mathcal{I}$ and $\mathcal{B}$ are the initial and boundary condition, respectively. The operator $\mathcal{R}$ may include multiple nonlinear differential terms of different order. When dealing with steady-state problems no initial condition is required. Fundamentally, we try to find an approximation of the unknown $\boldsymbol{u}(\boldsymbol{x}, t)$. Instead of discretizing the domain itself, as is the case in many classical solution algorithms like finite difference, finite element or finite volume methods, we choose a global ansatz function. A fully connected feed-forward NN $\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t)$ is used to approximate the unknown solution $\boldsymbol{u}(\boldsymbol{x}, t)$. As shown in Sec. 2.3, the $\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t)$ can be adapted by changing the weight matrices w^k, bias vectors b^k and other free parameters of the network. As before, all of these parameters are contained in the parameter vector $\boldsymbol{\theta}$. As for a regression task (see Secs. 2.3-2.3), we solve an optimization problem to find optimal values for $\boldsymbol{\theta}$. Since we aim to solve the boundary value problem, the optimum values of $\boldsymbol{\theta}$ should correspond to the situation where $\boldsymbol{u} = \hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t) \in (\Omega \times (0, T))$. However, in comparison to the regression problem, we generally can not assume the availability of training samples $\{((x_i, t_i), y_i), \quad i = 0 \dots N-1 \,|\, y_i = \boldsymbol{u}(x_i, t_i)\}$, which allows for the straightforward construction of a loss functional measuring some norm of the difference between the network prediction $\boldsymbol{u}(x_i, t_i)$ and $y_i(x_i, t_i)$. Hence, PINNs aim to utilize Eqs. (2.81) directly.

We define the functional

$$\begin{aligned} \mathcal{L}(\boldsymbol{u}) &= \lambda_{\mathrm{R}} \|\mathcal{R}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t)\|^2_{L^2(\Omega \times (0,T))} + \lambda_{\mathrm{I}} \|\mathcal{I}((\boldsymbol{u}(\boldsymbol{x}, 0), \boldsymbol{x})\|^2_{L^2(\Omega)} \\ &+ \lambda_{\mathrm{B}} \|\mathcal{B}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t)\|^2_{L^2(\partial\Omega)} \\ &= \lambda_R \int_0^T \int_\Omega \mathcal{R}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t)^2 \;\, \mathrm{d}\boldsymbol{x} \; \mathrm{d}t \; + \lambda_{\mathrm{I}} \int_\Omega \mathcal{I}(\boldsymbol{u}(\boldsymbol{x}, 0), \boldsymbol{x})^2 d\boldsymbol{x} \\ &+ \lambda_{\mathrm{B}} \int_0^T \int_{\partial\Omega} \mathcal{B}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{x}, t)^2 \;\, \mathrm{d}s(\boldsymbol{x}) \; \mathrm{d}t. \end{aligned}$$

By construction, we have

$$\mathcal{L} = 0 \tag{2.82}$$

as the integrands vanish on the entire domain $x \in \Omega$ and for all times $t \in (0, T)$. When approximating u with a NN $\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t)$, we have

$$\begin{aligned} \mathcal{L}(\hat{\boldsymbol{u}}) \;\; &= \;\; \lambda_R \int_0^T \int_\Omega \mathcal{R}(\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t), \boldsymbol{x}, t)^2 \;\, \mathrm{d}\boldsymbol{x} \; \mathrm{d}t \; + \lambda_{\mathrm{I}} \int_\Omega \mathcal{I}(\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, 0), \boldsymbol{x})^2 d\boldsymbol{x} \\ &+ \;\; \lambda_{\mathrm{B}} \int_0^T \int_{\partial\Omega} \mathcal{B}(\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t), \boldsymbol{x}, t)^2 \;\, \mathrm{d}s(\boldsymbol{x}) \; \mathrm{d}t \geq 0, \end{aligned} \tag{2.83}$$

which takes on a minimal value of 0 if and only if $\hat{\boldsymbol{u}}_\theta(\boldsymbol{x}, t)$ is a solution of Eqs. (2.81). Note that Eq. (2.83) essentially reformulates the Cauchy problem (2.81) as a variational principle, where the quantity $\mathcal{L}$ needs to be minimized to calculate the solution. The coefficients $\lambda_\mathrm{I}, \lambda_\mathrm{B}, \lambda_\mathrm{R} \in \mathbb{R}^+$ can be used to rescale the magnitude of the individual loss terms. Clearly, Eq. (2.83) is still 0 and thus minimal for arbitrary $\lambda_\mathrm{I}, \lambda_\mathrm{B}, \lambda_\mathrm{R} \in \mathbb{R}^+$ if and only if $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ is a solution to Eqs. (2.81). The coefficients can however be used to control the magnitude of gradients of the individual loss terms with respect to $\boldsymbol{\theta}$. This is relevant for the optimization and further discussed in Sec. 2.4.4. When dealing with steady-state problems, the initial condition term is not required.

In practice Eq. (2.83) is not utilized directly because the integrals need to be approximated numerically. The numerical quadrature, used in PINNs is typically a Monte Carlo integration [106]. The integrands of Eq. (2.83) are only evaluated point wise and the integrals are replaced by the sum over a representative point distribution

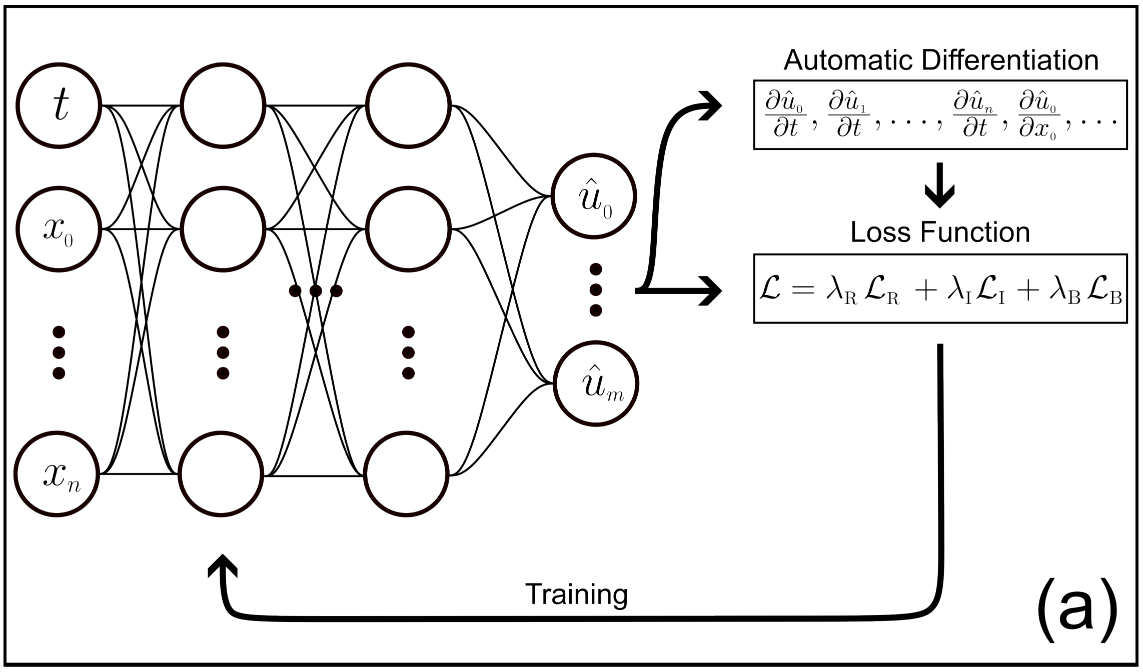
$$
\begin{aligned}
\mathcal{L}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}, t)) &= \mathcal{L}_\mathrm{R} + \mathcal{L}_\mathrm{I} + \mathcal{L}_\mathrm{B}, \\
\mathcal{L}_\mathrm{R} &= \lambda_\mathrm{R} \frac{1}{N} \sum_{i=1}^{N_\mathrm{R}} \mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i))^2, \qquad \boldsymbol{x}_i \in \Omega\ ; t_i \in (0, T), \\
\mathcal{L}_\mathrm{I} &= \lambda_\mathrm{I} \frac{1}{N_\mathrm{I}} \sum_{i=1}^{N_\mathrm{I}} \mathcal{I}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, 0))^2, \qquad \boldsymbol{x}_i \in \Omega, \\
\mathcal{L}_\mathrm{B} &= \lambda_\mathrm{B} \frac{1}{N_\mathrm{B}} \sum_{i=1}^{N_\mathrm{B}} \mathcal{B}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i))^2, \qquad \boldsymbol{x}_i \in \partial\Omega\ ; t_i \in (0, T),
\end{aligned}
\tag{2.84}
$$

where N_R, N_I and N_B are the number of points that evaluate the residual, the initial condition and the boundary condition, respectively. The selection of these training or collocation points is discussed in detail in Sec. 2.4.5. The typical normalization by the volume for the Monte-Carlo integration that would be performed to calculate the exact integral, can be absorbed into the scaling factors λ.

The partial derivatives in $\mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i))$ are calculated, using AD, as discussed in Sec. 2.3. However, in this case the derivatives are not calculated with respect to the trainable parameters $\boldsymbol{\theta}$ but with respect to the network inputs $(\boldsymbol{x}_i, t_i)$.

The network parameters $\boldsymbol{\theta}$ are tuned using an iterative optimization algorithms as discussed in Sec.2.3, where the gradients $\boldsymbol{\nabla}_{\boldsymbol{\theta}}\mathcal{L}$ are calculated using the backpropagation algorithm. This is also possible in a mini-batching approach, where only a subset of all $(\boldsymbol{x}_i, t_i)$ is evaluated in each iteration, as discussed in Sec. 2.3. However, since the evaluation of $\mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i))$ already involves the calculation of derivatives, the evaluation of the PINN loss function generally requires the calculation of higher order derivatives. An overview of the basic PINN architecture is shown in Fig. 2.10 (a). The PINN loss function is the distinguishing feature of the PINN architecture because it directly incorporates the information, given by the boundary value problem (2.81) and thus the underlying physics.

Generally, it has a composite structure, meaning that it consists of multiple terms which represent different objectives of the optimization. Hence, a more general view of the loss function is to simply enumerate the different loss terms and write the composite loss


t
x_0
x_n
$\hat{u}_0$
$\hat{u}_m$
Automatic Differentiation
$\frac{\partial \hat{u}_0}{\partial t}, \frac{\partial \hat{u}_1}{\partial t}, \dots, \frac{\partial \hat{u}_n}{\partial t}, \frac{\partial \hat{u}_0}{\partial x_0}, \dots$
Loss Function
$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_I \mathcal{L}_I + \lambda_B \mathcal{L}_B$
Training
(a)

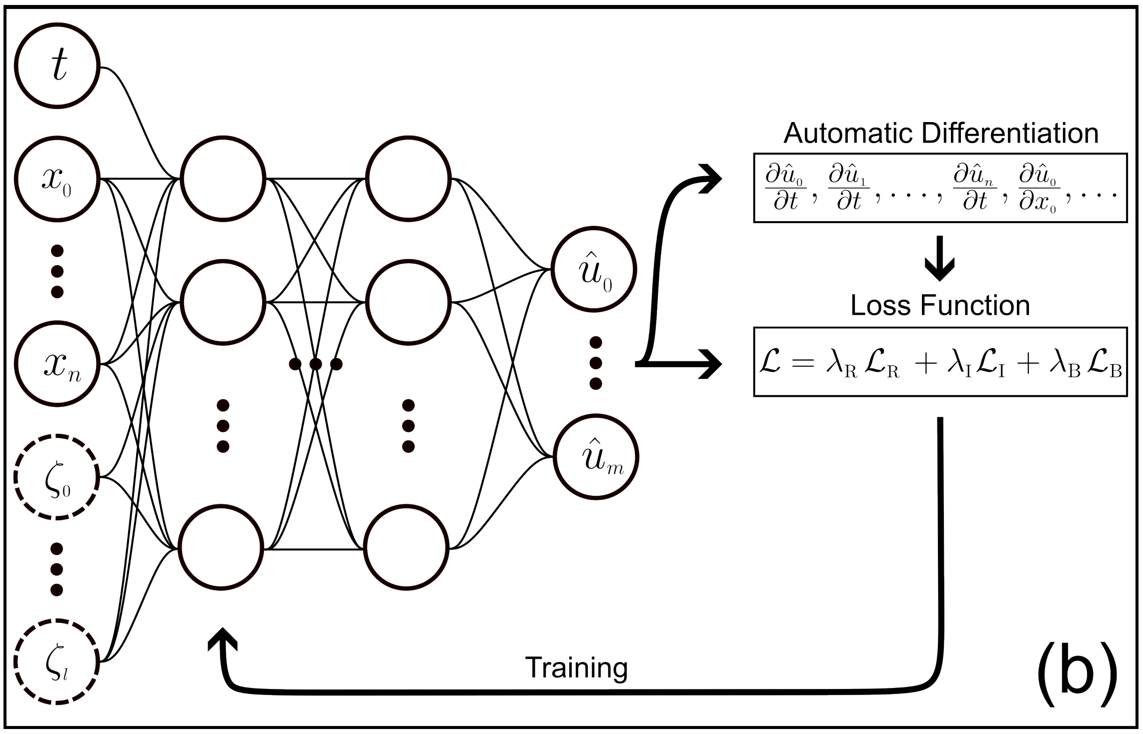

t
x_0
x_n
ζ_0
ζ_l
$\hat{u}_0$
$\hat{u}_m$
Automatic Differentiation
$\frac{\partial \hat{u}_0}{\partial t}, \frac{\partial \hat{u}_1}{\partial t}, \dots, \frac{\partial \hat{u}_n}{\partial t}, \frac{\partial \hat{u}_0}{\partial x_0}, \dots$
Loss Function
$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_I \mathcal{L}_I + \lambda_B \mathcal{L}_B$
Training
(b)

Figure 2.10.: Schematic overview of PINN architecture. Fig (a) shows the structure for a general initial-boundary value problem. Fig. (b) shows the structure for a general parametric problem with the parameter vector $(\zeta_0, \dots, \zeta_l)$.

function as follows:

$$\mathcal{L}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x},t)) = \lambda_k \sum_{k=1}^{N_{\text{Loss}}} \mathcal{L}_k(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x},t)), \tag{2.85}$$

where each loss term $\mathcal{L}_k$ corresponds to a different objective, such as the boundary condition fulfillment, the initial condition fulfillment or the vanishing of the residual. N_{Loss} is the toal number of loss terms and λ_k is the corresponding scaling factor for each loss term. Assuming that a solution to Eqs. (2.81) exists and that the NN can approximate the solution to a satisfactory degree, we expect that these objectives are not conflicting and that all of the different loss terms can be reduced to zero simultaneously.

2.4.2. Boundary and Initial Conditions

In the following, different types of boundary conditions are considered and it is discussed how to implement them in terms of a discrete loss functional $\mathcal{L}_{\text{B}}$. Initially, basic types of boundary conditions for scalar PDEs are introduced.

Dirichlet Boundary Condition and Initial Condition

For a Dirichlet-type boundary condition:

$$\mathcal{R}(u(\boldsymbol{x}),\boldsymbol{x},t) = u_{\text{B}}(\boldsymbol{x}) - u(\boldsymbol{x},t) = 0, \quad (\boldsymbol{x},t) \in \partial\Omega \times (0,T), \tag{2.86}$$

with a known function u_{B} on $\partial\Omega$ the boundary loss is given by the the mean squared error between the PINN prediction $\hat{u}(x,t)$ and the boundary function $u_{\text{B}}(x,t)$:

$$\begin{aligned} \mathcal{L}_{\text{B}} &= \lambda_{\text{B}} \frac{1}{N_{\text{B}}} \sum_{i=1}^{N_{\text{B}}} \mathcal{B}(\hat{u}(\boldsymbol{x}_i,t_i))^2 \\ &= \lambda_{\text{B}} \frac{1}{N_{\text{B}}} \sum_{i=1}^{N_{\text{B}}} (\hat{u}(\boldsymbol{x}_i,t_i) - u_{\text{B}}(\boldsymbol{x}_i,t_i))^2, \quad \boldsymbol{x}_i \in \partial\Omega\ ; t_i \in (0,T). \end{aligned} \tag{2.87}$$

Similarly for an initial condition $\boldsymbol{u}_0(\boldsymbol{x})$, we have:

$$\begin{aligned} \mathcal{L}_{\text{I}} &= \lambda_{\text{I}} \frac{1}{N_{\text{I}}} \sum_{i=1}^{N_{\text{I}}} \mathcal{I}(\hat{u}(\boldsymbol{x}_i,0))^2 \\ &= \lambda_{\text{I}} \frac{1}{N_{\text{I}}} \sum_{i=1}^{N_{\text{I}}} (\hat{u}(\boldsymbol{x}_i,0) - \boldsymbol{u}_0(\boldsymbol{x}_i))^2, \quad \boldsymbol{x}_i \in \Omega. \end{aligned} \tag{2.88}$$

Neumann Boundary Condition

A Neumann-type boundary condition is given by:

$$\mathcal{R}(u(\boldsymbol{x}),\boldsymbol{x},t) = \frac{\partial u(\boldsymbol{x},t)}{\partial \boldsymbol{n}} - \frac{\partial u_B(\boldsymbol{x},t)}{\partial \boldsymbol{n}} = 0, \quad (\boldsymbol{x},t) \in \partial\Omega \times (0,T), \tag{2.89}$$

where $\partial f / \partial \boldsymbol{n} = \nabla f \cdot \boldsymbol{n}$ denotes the normal derivative of the boundary. To construct the loss term for the NN, we require the normals $\boldsymbol{n}_i$ at the boundary points. Furthermore, the loss requires the spatial gradient $\nabla \hat{u}(\boldsymbol{x})$. This gradient is obtained using AD, as described in Sec. 2.3. The loss term is then given by:

$$\begin{aligned}\mathcal{L}_\mathrm{B} &= \lambda_\mathrm{B} \frac{1}{N_\mathrm{B}} \sum_{i=1}^{N_\mathrm{B}} \mathcal{B}(\hat{u}(\boldsymbol{x}_i, t_i))^2 \\ &= \lambda_\mathrm{B} \frac{1}{N_\mathrm{B}} \sum_{i=1}^{N_\mathrm{B}} \left(\boldsymbol{n} \cdot \nabla \hat{u}(\boldsymbol{x}_i, t_i) - \frac{\partial u_\mathrm{B}(\boldsymbol{x}_i, t_i)}{\partial \boldsymbol{n}} \right)^2, \quad \boldsymbol{x}_i \in \partial\Omega\,, t_i \in (0, T).\end{aligned} \tag{2.90}$$

Hard Boundary Conditions

In Sec. 2.4.1, we introduced the basic form of the PINN methodology using explicit loss terms penalizing disagreements with the boundary conditions. At this point, let us also mention an alternative approach, which does not require such explicit loss terms for the boundary condition. The hard-boundary condition method modifies the ansatz function such that boundary conditions are always fulfilled. Let us consider again the Dirichlet-type boundary condition Eq. (2.86). We construct a new ansatz function $\tilde{u}(\boldsymbol{x}, t)$ to approximate the solution $u(\boldsymbol{x}, t)$:

$$\tilde{u}(\boldsymbol{x}, t) = \hat{u}(\boldsymbol{x}, t) \cdot \varphi + b(\boldsymbol{x}, t), \tag{2.91}$$

where $\hat{u}(\boldsymbol{x}, t)$ is the NN and φ is a so-called *approximate distance function* which vanishes only on the boundary. The function $b : \Omega \times (0, T) \longrightarrow \mathbb{R}$ is a continous differentiable function that equals the boundary condition on the boundary:

$$b(\boldsymbol{x}, t) = u_\mathrm{B}(x, t) \quad \forall (x, t) \in \partial\Omega \times (0, T). \tag{2.92}$$

Then the ansatz $\tilde{u}$ always fulfills Eq. (2.86). This approach can also be extended to the initial conditions. In this case the NN is only optimized to minimize the residual loss and the second and third term in Eq. (2.84) can be disregarded. The usage of hard boundary conditions has been shown to simplify the resulting optimization problem for many simple test cases [55, 56, 107] and can also be extended to other types of boundary conditions, as shown in [55]. However, this requires adequate expressions for φ and b. While these can be easily constructed for one dimensional problems, it can be a challenging task in itself for higher dimensions and non-trivial geometries. This is due to the fact that φ also has to be sufficiently smooth and ideally monotonically increasing in the boundary normal direction with a gradient of $\partial\varphi/\partial\boldsymbol{n} = 1$. Sukumar and Srivastava [55] showed different methods to construct such functions for higher dimensional problems. They have, however, not been applied widely to PDEs with Neumann-type boundaries in higher dimensions. In these scenarios an inadequate approximate distance function can impair the convergence behavior instead of improving it. As we will discuss in Sec. 2.4.4, there are other ways to improve the convergence during training, even with multiple loss terms.

Boundary Conditions for the Euler equations

A central system of PDEs in aerodynamics are the compressible Euler equations (see Sec. 2.1.3). When dealing with problems in aerodynamics we typically impose two types of boundary conditions for these equations, the far-field condition and the slip-boundary condition. In the following these typical boundary conditions for the Euler equations are discussed and it is shown how they can be applied when solving Cauchy problems (2.81) with PINNs. In addition, periodic boundary conditions are introduced, which are required when mesh-transformations (Sec. 2.4.6) are used.

Far-Field Boundary Condition On the outer boundary of the domain Ω_∞, we typically impose far-field boundary conditions which are a Dirichlet-type boundary condition. This means that far away from aerodynamic objects, the primitive flow variables take on constant values $(\rho_\infty, u_\infty, v_\infty, E_\infty) = \boldsymbol{w}_\infty$ which correspond to the upstream flow conditions. As explained in Sec. 2.1.3, one can replace E_∞ by p_∞, using Eq. (2.42). In this work we solve the non-dimensionalized equations. As, for example, shown in [6, ch. 2.3], the far-field condition is fully specified by the far field Mach number M_∞ and the angle of attack α. We choose the far-field density and pressure as reference states, which is commonly done in classical CFD solvers, giving $\rho_\infty = 1$ and $p_\infty = 1$. For an ideal gas, the speed of sound c is given by:

$$c = \sqrt{\kappa p/\rho}. \tag{2.93}$$

Therefore, we have:

$$\begin{aligned} u_\infty &= c_\infty M_\infty \cos(\alpha) = \sqrt{\kappa \frac{p_\infty}{\rho_\infty}} M_\infty \cos(\alpha) = \sqrt{\kappa} M_\infty \cos(\alpha) \\ v_\infty &= \sqrt{\kappa \frac{p_\infty}{\rho_\infty}} M_\infty \sin(\alpha) = \sqrt{\kappa} M_\infty \sin(\alpha) \\ E_\infty &= \frac{1}{\kappa - 1} \cdot \frac{p_\infty}{\rho_\infty} + \frac{(u_\infty^2 + v_\infty^2)}{2} = \frac{1}{1-\kappa} + \frac{\kappa M_\infty^2}{2}, \end{aligned} \tag{2.94}$$

for the velocities and the energy. In the following let $\hat{\boldsymbol{w}} = (\hat\rho, \hat u, \hat v, \hat E)$ be the NN prediction of the primitive variable vector. We have the far field boundary condition:

$$\hat{\boldsymbol{w}}(x, y) \equiv \boldsymbol{w}_\infty \quad (x, y) \in \partial\Omega_\infty, \tag{2.95}$$

with the upper boundary of the computational domain $\partial\Sigma_\infty$. The far field boundary loss term $\mathcal{L}_\infty$ is then given by the squared L_2 norm of the error:

$$\mathcal{L}_\infty = \frac{1}{N_\infty} \sum_i \|\hat{\boldsymbol{w}}(x_{\infty,i}, y_{\infty,i}) - \boldsymbol{w}_\infty\|_2{}^2, \qquad (x_{\infty,i}, y_{\infty,i}) \in \partial\Omega_\infty, \qquad i = 1, \ldots, N_\infty, \tag{2.96}$$

for N_∞ points, located on the far field boundary.

Slip Boundary Condition For an deflecting object, such as a wall, a plate or an airfoil, we impose a slip boundary condition. This means that the flow velocity $\boldsymbol{q} = (u, v)$ in wall normal direction $\boldsymbol{n}$ should vanish on the boundary of aerodynamic object $\partial\Omega_{\text{ob}}$:

$$\boldsymbol{q}(x, y) \cdot \boldsymbol{n}(x, y) \equiv 0, \qquad (x, y) \in \partial\Omega_{\text{ob}}. \tag{2.97}$$

The resulting loss term is given by:

$$\begin{gathered}\mathcal{L}_{\text{ob}} = \frac{1}{N_{\text{ob}}} \sum_i \left(\boldsymbol{q}(x_{\text{ob},i}, y_{\text{ob},i}) \cdot \boldsymbol{n}(x_{\text{ob},i}, y_{\text{ob},i})\right)^2, \\ (x_{\text{ob},i}, y_{\text{ob},i}) \in \partial\Omega_{\text{ob}}, \qquad i = 1, \dots, N_{\text{ob}},\end{gathered} \tag{2.98}$$

where N_{ob} is the number of points on the surface of the aerodynamic body.

Periodic Boundary Condition When utilizing the mesh-transformation methodology (introduced later in Sec. 2.4.6), we require a third type of boundary condition for the Euler equations. In the mesh transformation, we make use of a computational domain, featuring a left boundary and a right boundary, for which we have to impose periodic boundary conditions. This is due to the fact that the computational domain is obtained from a mapping from the physical domain, wrapped around an aerodynamic object. In the physical domain, we define the two boundaries $\partial\Omega_{\text{l}}$ and right boundary $\partial\Omega_{\text{r}}$, which can be seen as the left an right limit towards the same line. The periodic boundary condition is then given by:

$$\begin{gathered}\hat{\boldsymbol{w}}(x_{\text{l}}, y_{\text{l}}) \equiv \hat{\boldsymbol{w}}(x_{\text{r}}, y_{\text{r}}), \\ (x_{\text{l}}, y_{\text{l}}) \in \partial\Omega_{\text{l}}, \quad (x_{\text{r}}, y_{\text{r}}) \in \partial\Omega_{\text{r}},\end{gathered} \tag{2.99}$$

resulting in the loss term:

$$\mathcal{L}_{\text{per}} = \frac{1}{N_{\text{per}}} \sum_j \left(\hat{\boldsymbol{w}}(x_{\text{l},j}, y_j) - \hat{\boldsymbol{w}}(x_{\text{r},j}, y_j)\right)^2, \qquad i = 1, \dots, N_{\text{per}}, \tag{2.100}$$

for N_{per} points on the periodic boundary.

Mixed Boundary Condition For a problem with mixed boundary conditions, we simply take the sum over all boundary loss terms. For example:

$$\mathcal{L}_{\text{bdr}} = \mathcal{L}_{\infty} + \mathcal{L}_{\text{ob}} + \mathcal{L}_{\text{per}}. \tag{2.101}$$

2.4.3. Parametric Problems

One prospect of physics-informed NNs is the possibility to solve parametric problems. Let ζ be a general parameter of the initial and boundary value problem. A PINN approximation $\hat{\boldsymbol{u}}$ of the unknown solution u simply receives ζ as an additional input to the NN alongside $\boldsymbol{x}$ and t. The parameter adds an additional dimension to the input space and thus the training points. The training points are now sampled in a $d + 2$-dimensional domain $(\boldsymbol{x}, t, \zeta) \in \Omega \times (0, T) \times (\zeta_{\min}, \zeta_{\max})$. The parameter ζ can then be incorporated into the calculation of any of the loss terms in Eq. (2.84). Similarly, this approach can be extended to more than one parameter. A schematic overview of the parametric PINN is shown in Fig. 2.10 (b) for a parameter vector $\boldsymbol{\zeta} = (\zeta_0, \zeta_1, \dots, \zeta_l)$.

2.4.4. Loss Term Imbalances and Counter Measures

As shown in Sec. 2.4.1 the typical NN loss function consists of multiple terms associated with different tasks. The boundary loss term $\mathcal{L}_\mathrm{B}$ and the initial condition loss $\mathcal{L}_\mathrm{I}$ aim to alter the NN prediction such that it fulfills the boundary and initial condition, respectively. The residual loss $\mathcal{L}_\mathrm{R}$ aims to adjust the NN prediction such that the PDE is fulfilled. If a solution to the PDE exists and a sufficiently expressive NN is used, we expect that all three loss terms can be minimized simultaneously, since for an arbitrarily close approximation of the solution, we expect that all three terms approach 0. However, for practical applications, the mere existence of such a global minimum of the loss function is not sufficient. The training algorithm also has to be able to find the minimum. For non-convex optimization problems this is in general NP-hard [108]. For PINNs in particular, we are facing the issue that imbalances between loss terms and their respective gradients $\nabla_\theta \mathcal{L}_k$. When the gradients of one particular loss term are significantly smaller than for the other loss terms, the optimizer may disregard the corresponding sub-task, for example the fitting of the boundary conditions, completely. This may lead to a scenario, where the optimizer converges to a local minimum, where for example only the residual loss is minimized and an incorrect approximation of the solution is obtained. This phenomena was first analyzed by Wang et al. [41]. Let us consider their original example, namely the solution of the Helmholtz equations with a harmonic boundary condition using a PINN. The problem is described in more detail in Appendix C. Since there is no time dependence, the loss function for this case only contains a boundary loss $\mathcal{L}_\mathrm{B}$ and a residual loss $\mathcal{L}_\mathrm{R}$. For uniform loss term weights, λ_B and λ_R, we see inaccurate results. When analyzing the histograms of the gradients with respect to the trainable parameters (Fig. 2.11) in certain NN layers, we observe a clear imbalance and the overall values of the residual loss are much more pronounced. Especially the maximum values should be considered here as they will determine the main descend direction of the gradient based optimizer.

To overcome these limitations, we can consider the loss term weights λ_R, λ_B and λ_I in Eq. (2.84). These weights allow for the manual scaling of the respective contributions of each loss term to the total loss $\mathcal{L}$ and hence also to the gradient magnitudes. To demonstrate this, it can be shown that an increase of $\lambda_\mathrm{B} = 10$ leads to a highly improved prediction accuracy (see Appendix C). Note that no systematic parameter study of λ_B is performed at this point and that this is merely an example to highlight the effect of the loss term weight. While such manual tuning is sufficient in many cases, the gradient magnitude might also change during training. Furthermore, the selection of these parameters, based on empirical experiments can become impractical, when they have to be chosen alongside many other hyperparameters. Therefore, Wang et al. [41] proposed an adaptive weighting algorithm that automatically changes the weighting parameters based on the gradient of the individual loss terms. The structure of adaptive weighting algorithms is shown in Alg. 4. Note that it is an extension to the classical SGD algorithm 1 but can be applied to other optimization algorithms such as Alg. 2 analogously. Every l_w-th iteration, the weights of all non-residual loss terms (see. Eq. (2.84)) are updated. For recommended values for l_w, see [41]. The update is performed using a running average, to avoid extreme noise affecting weights. The updating rate β_w determines the exponential decrease of past weights on the current weight, similar to the moment rates β_1 and β_2

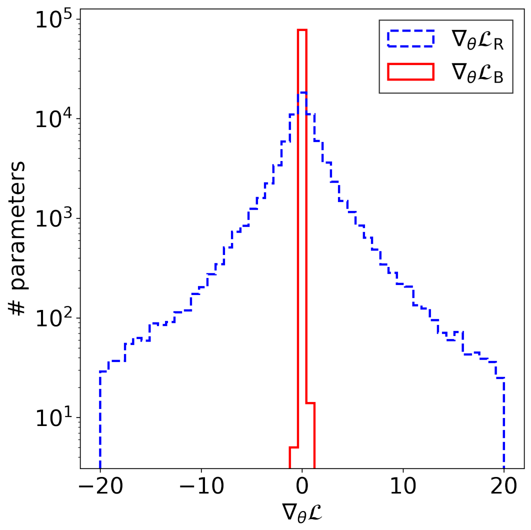


Figure 2.11.: Histogram of gradients of the boundary and the initial loss term for Helmholtz Equation. For a detailed description of the problem, see Appendix C.

in ADAM (Alg. 2). The update value $\hat{\lambda}_k$ is determined, based on the gradients of the individual loss terms $\boldsymbol{g}_{i,k}$. Originally, the authors proposed to balance the maximum of the residual loss gradient, with the average of the other loss term gradients. They argue that generally, the residual loss term is dominating the gradients of other loss terms. Shortly after, Jin et al. [52] proposed to treat all loss terms equally, using the average gradients. Maddu et al. [53] proposed to use the variance of gradients instead. Recently, Wang et al. [54] showed an updated version of their initial method, using the L_2-norm of the gradients. A summary of the listed weighting update methods is shown in Tab. 2.1.

In practice, the performance of the different variants is problem dependent. Generally, the version proposed in [54] strikes a good balance between accuracy and robustness but all variants are able to equalize imbalances to some degree. The additional computational cost of Alg. 4 is limited, since the gradient update only needs to be performed every few hundred iterations. Depending on the experienced training noise, a higher or lower update rate β_w should be selected.

A drawback of Alg. 4 is that it often leads to incorrect results, when being employed at the start of the training because the NN is generally initialized with weights and biases close to zero. Hence, the prediction is also close to zero in the entirety of $\Omega \times (0, T)$. For many PDEs, without source terms, this prediction is a trivial solution to the differential equation but does not fulfill the boundary conditions. In this case, the adaptive weighting algorithms detects, that the boundary loss term gradients are much larger than the residual loss ones. Hence, λ_R is increased drastically and the boundary condition is not learned properly anymore. Therefore, it makes sense to only activate the adaptive weighting algorithm after an initial training phase with constant weighting factors. In many cases constant weighting factors, if chosen adequately, can achieve similar accuracies as the

Algorithm 4 SGD with adaptive loss term weighting originally proposed by [41]

```
Require: $\tau$: Step size/learning rate
Require: $\tilde{\mathcal{L}}^{(i)}(\theta)$ stochastic loss function at iter. $i$ consisting of terms $\tilde{\mathcal{L}}_k^{(i)}(\theta)$ (see Eq. (2.85))
Require: $\theta_0$: initial parameter
Require: $\lambda_k^{(0)} \forall k$ : initial loss term weights        ▷ $k$ is indexing the different loss terms
  $i \leftarrow 0$
  $\lambda_k \leftarrow \lambda_k^{(0)} \forall k$
  while $\theta^{(i)}$ not converged do
      $i \leftarrow i + 1$
      $\theta^{(i)} \leftarrow \theta^{(i-1)} - \tau \nabla_\theta \tilde{\mathcal{L}}^{(i)}(\theta^{(i)})$        ▷ Perform step into negative gradient direction
      if $(i \bmod l_\mathrm{w}) = 0$ then        ▷ Perform weight update every $l_\mathrm{w}$-th iteration.
          $g_k^{(i)} \leftarrow \nabla_\theta \tilde{\mathcal{L}}_k^{(i)}(\theta) \forall k$
          $\hat{\lambda}_k \leftarrow \Lambda_\mathrm{dw}(k, g_0^{(i)}, g_1^{(i)}, \dots)$        ▷ Determine update value based on loss term
  gradients.
          $\lambda_k \leftarrow (1 - \beta_\mathrm{w})\lambda_k + \beta_\mathrm{w}\hat{\lambda}_k \forall k$        ▷ Update weights with running average
      end if
  end while
```

adaptive weighting algorithms while avoiding possible training instabilities.

2.4.5. Collocation Point Generation and Distribution

For a well defined problem for which the bounds of the domain are known, the generation of the training/collocation points can be achieved with quasi-random low discrepancy sequences such as Sobol [109] or Halton [110] or with other methods like Latin Hypercube sampling [111] at little additional cost. As highlighted in various publications [112, 113, 71], a non uniform distribution of training points or an adaptive training point selection, based on the local residual may accelerate training and improve the final accuracy for certain problems. However, at this point there is no generally accepted approach for determining optimal point locations. In this work we generally follow three different strategies:

1. Select a number of training point for the boundary N_B and for the residual losses N_R. Use a uniform quasi-random sampling technique that is space filling to obtain the designated number of training points for the domain Ω and the boundary $\partial\Omega$.
2. Select a number of training point for the boundary N_B and for the residual losses N_R. Each set of training points is split into two subsets. The first subset is randomly sampled on the entire domain Ω and boundary $\partial\Omega$ as before. The second subset of points is uniformly sampled in a subset $V \subset \Omega$ of the domain or its boundaries that is a-priori known to require higher point densities.
3. Generate an elliptic structured CFD mesh of designated dimensions as described in [2, pp. 194-200]. All of the mesh nodes are used as residual training points and the

Table 2.1.: Overview on different loss weight update formulas in 4.
The average over the vector entries is indicated with $\langle\cdot\rangle$.
The standard deviation over the vector entries is indicated with $\mathrm{std}(\cdot)$.

Reference	$\Lambda_{\mathrm{dw}}(k, \boldsymbol{g}_0^{(i)}, \boldsymbol{g}_1^{(i)}, \dots)$
Wang et al. [41]	$\dfrac{\max_{\theta^{(i)}}\{\boldsymbol{g}_0^{(i)}\}}{\langle\lvert\boldsymbol{g}_k^{(i)}\rvert\rangle}$
Jin et al. [52]	$\dfrac{\langle\lvert\boldsymbol{g}_0^{(i)}\rvert\rangle}{\langle\lvert\boldsymbol{g}_k^{(i)}\rvert\rangle}$
Maddu et al. [53]	$\dfrac{\max_l\{\mathrm{std}(\boldsymbol{g}_l^{(i)})\}}{\mathrm{std}(\boldsymbol{g}_k^{(i)})}$
Wang et al. [54]	$\dfrac{\sum_l \lVert\boldsymbol{g}_l^{(i)}\rVert_2}{\lVert\boldsymbol{g}_k^{(i)}\rVert_2}$

boundary points are used as the boundary training points.

Overall, the generation of the collocation points is task-dependent. For many simple problems, uniform point distributions may be sufficient to obtain reasonable accuracies (strategy 1). For certain problems, such as the flow around an aerodynamic object, it is natural to assume that nonlinear effects are strongest near that object. Hence, the number of training point in that area should be increased to properly resolve these non-linearities (strategy 2). For highly non-uniform point distributions, one may however have to apply different weights to each individual training point, to avoid training biases towards high density areas. Such a weighting strategy is presented in Sec. 2.4.7. Strategy 3 closely resembles classical numerical methods, which make use of curvilinear grids. When such a grid is available we can make use of the structure of the grid lines to precondition the optimization problem using mesh transformations. This idea is further discussed in the following section. As highlighted in Sec. 2.3, the computational cost of the gradient calculation does typically not increase with the number of points, as the calculation can be vectorized for an entire row of the Jacobian. However, the memory of the device might limit the number of points that are used during training. How the accuracy scales with the number of points is another interesting point of discussion. Currently, for many practical applications, the main limitation of the accuracy of PINNs is the training convergence. The residual loss can typically not be reduced to machine level precision. Thus, one can hardly evaluate the influence of the collocation point resolution on the accuracy, as this discretization error is typically smaller than the convergence error.

2.4.6. Mesh Transformation

Mesh transformation (MT) is a classical method used in CFD. Fundamentally, the idea is to use a curvilinear grid to accurately discretize a non-trivial geometry in the *physical*

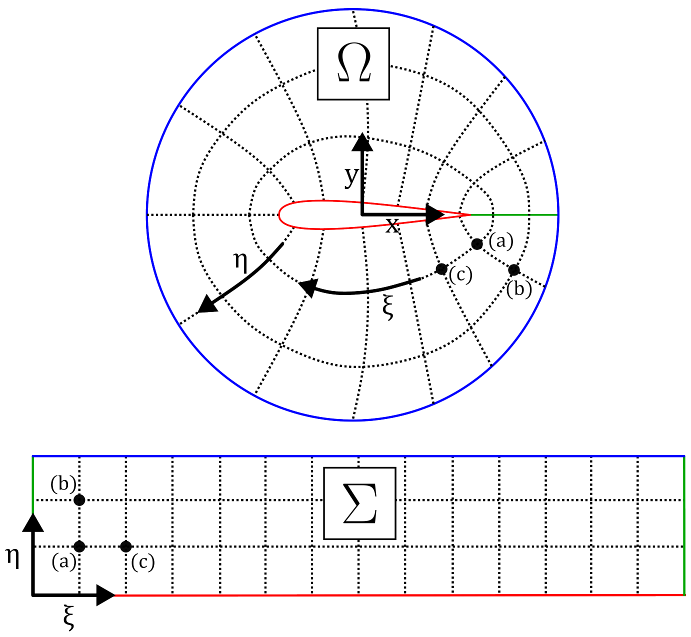


Figure 2.12.: Schematic representation of curvilinear grid (top) and corresponding computational grid (bottom). The curvilinear grid wraps around the airfoil. The origin is located at the center. The Cartesian coordinates are x and y. The curvilinear coordinates ore oriented in tangential and normal direction of the airfoil surface. For the computational grid, the origin is at the lower left. The Cartesian coordinates are ξ and η. All grid points of the curvilinear (physical) grid, can be identified with points in the computational grid (e.g. (a), (b), (c)). The outer (blue) and inner (red) domain boundary become the top and bottom boundary in the computational grid. The trailing edge line (green) gets split into the left and right boundary.

domain Ω. However, certain solution methods favor or even require a Cartesian grid. Therefore, the curvilinear grid is transformed into a Cartesian grid in a *computational domain* Σ. Afterwards, the computations are carried out in this computational domain. In this work, we consider O-type grids, where a single aerodynamic object, such as and airfoil, is located at the center of a circular domain Ω. Fig. 2.12 shows a schematic curvilinear grid around an airfoil in Ω and its transformed analog in Σ. For PINNs, the MT approach was first adapted by Cao et al. [86]. Similarly to the classical CFD variant, the idea is to train the NN in the computational domain Σ. The mesh transformation can simplify the optimization problem for the NN, as highly nonlinear regions of the flow field are stretched, while linear regions are compressed. The resulting function in the computational domain can be learned more easily, improving the convergence of the model and the prediction accuracy in critical regions such as the surface of the object.

In mathematical terms, we define the transformation as an invertable and differentiable mapping

$$\mathcal{F} : \Omega \to \Sigma, \qquad (x, y) \mapsto (\xi, \eta). \tag{2.102}$$

In our context the curvilinear grid is denoted by $\Omega_\mathrm{h} \subset \Omega$,

$$\Omega_\mathrm{h} = \left\{(x_{i,j}, y_{i,j}) \in \Omega : i = 1, \ldots, N_\xi, \quad j = 1, \ldots, N_\eta\right\}. \tag{2.103}$$

N_ξ and N_η are the number of points on the grid lines tangential and normal to the airfoil's surface, respectively. The grid can be constructed as for example illustrated in [2, pp. 194-200]. The Cartesian grid $\Sigma_\mathrm{h} \subset \Sigma$ in the computational domain is

$$\Sigma_\mathrm{h} = \left\{\mathcal{F}(x_{ij}, y_{ij}) \in \Sigma : (x_{i,j}, y_{i,j}) \in \Omega, \quad i = 1, \ldots, N_\xi,\ j = 1, \ldots, N_\eta\right\}. \tag{2.104}$$

If no analytical expression for $\mathcal{F}$ is available, the mapping is constructed in a discrete sense such that

$$\mathcal{F}(x_{ij}, y_{ij}) = (\xi_{ij}, \eta_{ij}), \qquad \xi_{ij} = i\Delta\xi, \quad \eta_{ij} = j\Delta\eta, \tag{2.105}$$

holds. The NN $\hat{u}$ receives the grid points $(\xi_{i,j}, \eta_{i,j}) \in \Sigma_\mathrm{h}$ as inputs and predicts the solution as before. Predictions in Ω_h are obtained, using $\hat{\boldsymbol{u}}\left(\mathcal{F}(x_{ij}, y_{ij})\right)$, with $(x_{ij}, y_{ij}) \in \Omega_\mathrm{h}$. Contrary to classical PINNs we do not obtain predictions at arbitrary coordinates. Similarly to classical CFD solvers, the state flow field is only exactly predicted at the mesh nodes. In general, this is not critical. For example, interpolation can be used to obtain predictions at arbitrary points $(x, y) \in \Omega$. In practice, linear interpolation is often sufficient and is used throughout this work, when the solution is evaluated on points $(x, y) \notin \Omega_\mathrm{h}$.

The calculation of the loss function (see Sec. 2.4.1) requires derivatives of the NN outputs with respect to (x, y). The MT is, however, only defined in terms of the discrete mapping of Eq. (2.105) and we have, in general, no analytical expression at hand. Therefore, we cannot directly use AD to calculate the derivatives. Instead, using the differentiability of $\mathcal{F}$, the derivatives can be expressed in terms of the derivatives in Σ. The relations for the reconstruction of the first and second order derivatives are derived in [2, 114] and listed in A.1. For this reconstruction, so-called inverse metrics are required. These are the derivatives of the physical coordinates (x, y) with respect to the curvilinear coordinates (ξ, η) and can be approximated in Σ_h using finite differences.

2.4.7. Volume Weighting

Song et al. [115] proposed a volume weighting strategy to improve the conditioning of the loss function (2.84) and the corresponding optimization problem, when the training point density is strongly non-uniform. Let $\mathrm{vol}(i)$ be the volume that is occupied by a training point $(\boldsymbol{x}_i, t_i)$. Then, for the modified residual loss, each sample is weighted by the squared, normalized volume:

$$\mathcal{L}_\mathrm{R} = \lambda_\mathrm{R} \frac{1}{N_\mathrm{R}} \sum_{i=1}^{N_\mathrm{R}} \mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i))^2 \cdot \frac{\mathrm{vol}(i)^2}{\frac{1}{N_\mathrm{R}} \sum_{j=1}^{N_\mathrm{R}} \mathrm{vol}(j)^2} = \lambda_\mathrm{R} \frac{\sum_{i=1}^{N_\mathrm{R}} (\mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, t_i)) \cdot \mathrm{vol}(i))^2}{\sum_{j=1}^{N_\mathrm{R}} \mathrm{vol}(j)^2}$$
$$\boldsymbol{x}_i \in \Omega\ ;\ t_i \in (0, T). \tag{2.106}$$

When a curvilinear grid is used, the training point volume can be identified with the determinant of the Jacobian of the MT J (see Appendix A.1).

2.5. Error Metrics

To evaluate the performance of the models, introduced in this work, we make use of different error metrics. The goal of these metric is to provide a concise, quantitative overview of the model performance with a single number. However, it is typically reasonable to consider different metrics, as they have different strengths and weaknesses.

Firstly, we consider the mean absolute error

$$MAE = \frac{1}{n}\sum_{i=0}^{n} |\hat{u}_i - u_i|. \tag{2.107}$$

As before, $\hat{u}$ is the model prediction and u is the analytical/reference solution. The index i counts over all analyzed points with $u_i = u(\boldsymbol{x}_i, t_i)$, $\hat{u}_i = u(\boldsymbol{x}_i, t_i)$ and $(\boldsymbol{x}_i, t) \in \Omega \times (0, T)$. Note that we assume a uniform distribution of these points. The mean absolute error gives an estimate of the mean error in the units of the quantity analyzed. For example when considering the MAE of a C_p distribution, the resulting value gives a directly interpretable estimate of the mean absolute deviation of the prediction from the dataset. On the other hand, the relative mean absolute error

$$RMAE = \frac{MAE}{\max\limits_i(u_i) - \min\limits_i(u_i)}, \tag{2.108}$$

is normalized with the range of predictions to give a relative estimate on how much the prediction deviates on average from the reference in comparison to the full value range of the reference. Note that we do not use the standard mean absolute percentage error (MAPE), which is not well suited when dealing with values close to 0. Lastly we consider the coefficient of determination (or R_2 score)

$$R_2 = 1 - \frac{\sum_{i=0}^{n}(\hat{u} - u_i)^2}{\sum_{i=0}^{n}(\langle u_i\rangle - u_i)^2}, \tag{2.109}$$

as a squared, relative error metric. The operator $\langle\cdot\rangle$ indicates the mean over all i points. Since the difference in the numerator is squared, the R_2-score emphasize outliers more. Furthermore, it is limited to $R_2 \in (-\infty, 1]$, where 1 corresponds to a prefect prediction $\hat{u}_i = u_i$, making it easily interpretable.

2.6. Summary

This chapter initially lays out the theoretical foundations of scalar conservation laws, which form the basis for understanding many physical phenomena in fluid dynamics and aerodynamics. The general form of these equations and the concept of weak solutions are introduced, highlighting that these solutions are not unique. To address this non-uniqueness, the concept of entropy solutions is presented, which is closely linked to the idea of artificial viscosity (AV) through the vanishing viscosity limit of the modified PDE with a viscous term. Following this, the chapter introduces the Euler equations, a set of nonlinear hyperbolic conservation laws that play a crucial role in aerodynamics. It then delves into finite volume methods, a classical approach for solving conservation laws. This includes an illustration of how a simple upwinding schemes is equivalent to a central scheme with artificial viscosity, and it presents a classical central difference scheme with scalar and matrix-valued artificial viscosity as exemplary methods for solving the Euler equations. The focus is then shifted to the basics of neural networks, covering their architecture, the training process, and the concept of AD. Modifications to the classical architectures, aiming to improve the convergence properties of the networks during training, are also covered. In the next step, the physics-informed neural network (PINN) methodology is introduced. The chapter explains how PINNs can be used to solve PDEs, detailing the construction of the loss function and the enforcement of boundary conditions. It addresses how PINNs can tackle parametric problems and discusses strategies for balancing loss terms to improve convergence. Additionally, methods for generating training/collocation points and how MTs can be utilized to enhance the accuracy of predictions are discussed. Overall, the chapter bridges the gap between traditional numerical methods for solving conservation laws and the emerging field of neural network-based solvers, providing a comprehensive overview of both the theoretical underpinnings and practical applications of these methods.

3. Artificial Viscosity for Physics-Informed Neural Networks

The failure of PINNs to predict shocks, without additional modifications, has been rigorously confirmed in numerous experiments [74, 80, 81, 73, 86, 61] and was recently also studied theoretically [85]. At the start of this chapter, we revisit this argument to better understand this failure mode of standard PINNs. To overcome this limitation, this chapter introduces novel artificial viscosity (AV) variants for stabilizing the training process of PINNs, enabling the application of neural network-based solution algorithms to problems with aerodynamics relevance. Based on the inviscid Burgers equation, two advanced methods for localizing the required AV magnitudes and distributions in the computational domain are proposed. After this initial introduction, it is shown how these methodologies can be extended to, not only, solve scalar conversation laws, but also the Euler equations, a significantly more complex system of conversation laws. These newly introduced methods for solving the Euler equations represent the main contribution of this work and advance the state-of-the art for solving complex flow problems in aerodynamics with PINNs. In the following chapters, they are applied to solve a number of challenging test cases for different geometries, flow conditions and even parametric problems. Note that parts of this chapter are based on [95]

3.1. Why do PINNs Fail to Predict Shock Waves?

In the following, it is illustrated why PINNs fail to predict shock waves. This argument has been derived by Chaumet and Giesselmann [85]. We are interested in a solution u to a scalar conservation law (2.17) in the compact interval $\Omega \subset \mathbb{R}$ and in time $[0, T)$. The residual loss of the baseline PINN (2.84) approximates the L^2-norm of the residual:

$$\mathcal{L}_R \approx \|\mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}, t))\|_{L^2(\Omega\times(0,T))}{}^2 = \int_0^T \int_\Omega |\mathcal{R}(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}, t))|^2 dx dt \tag{3.1}$$

It will be shown that the L^2 norm of the residual is not reducing, when a continuous approximation of a shock is improving. Hence, the optimizer is not able to converge to the correct solution featuring a shock if the classical PINN formulation is employed. We consider the Riemann problem:

$$u_0 = \begin{cases} 1, & x < 0, \\ -1, & x \geq 0, \end{cases} \tag{3.2}$$

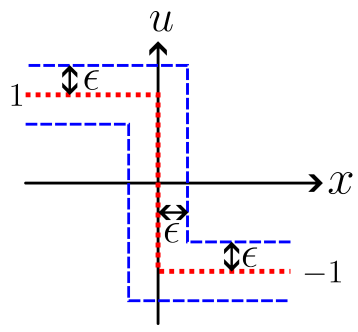


Figure 3.1.: Schematic illustration of the shock solution Eq. (3.2) (in red) and the construction of the approximated solution candidate Eq. (3.3) (in blue).

which has a steady state shock as the entropy solution. Let $\hat{u}$ be a smooth and time-independent approximation of this solution

$$\begin{cases} \hat{u}(x) \in [1-\varepsilon, 1+\varepsilon], x < -\varepsilon \\ \hat{u}(x) \in [-1-\varepsilon, -1+\varepsilon], x > \varepsilon \\ \hat{u}(x) \in [-1-\varepsilon, 1+\varepsilon], -\varepsilon \le x \le \varepsilon, \end{cases} \tag{3.3}$$

for a small scalar $\varepsilon > 0$ and $(-\varepsilon, \varepsilon) \subset \Omega$. A schematic illustration of $\hat{u}$ is shown in Fig. 3.1. It converges to u in $L^2(\Omega \times [0,T))$ for $\varepsilon \longrightarrow 0$. We approximate the L_2 norm over the region around the shock. This norm is less than the global norm

$$\|\mathcal{R}(\hat{u})\|_{L^2(\Omega\times[0,T))} \ge \|\mathcal{R}(\hat{u})\|_{L^2((-\varepsilon,\varepsilon)\times[0,T))}. \tag{3.4}$$

By means of Hölder's inequality, one can estimate the upper limit of the L_2 norm over this region as:

$$\|\mathcal{R}(\hat{u})\|_{L^1(\Omega\times(0,T))} \le \sqrt{2\varepsilon T}\|\mathcal{R}(\hat{u})\|_{L^2(\Omega\times(0,T))}. \tag{3.5}$$

Since $\hat{u}$ is a continuous approximation of u, it has at least one zero at location $-\varepsilon < \bar{x} < \varepsilon$. Hence, one can estimate an upper limit of the L_1 norm as

$$\begin{aligned} \|\mathcal{R}(\hat{u})\|_{L^1((-\varepsilon,\varepsilon)\times(0,T))} &= \int_0^T \int_{-\varepsilon}^{\varepsilon} |f(\hat{u}(x,t))_x| dxdt \\ &\ge \int_0^T \int_{-\varepsilon}^{\bar{x}} |\mathcal{R}(\hat{u}(x,t))| dx + \int_{\bar{x}}^{\varepsilon} |\mathcal{R}(\hat{u}(x,t))| dxdt \\ &\ge \int_0^T \left| \int_{-\varepsilon}^{\bar{x}} f(\hat{u}(x,t))_x dx \right| + \left| \int_{\bar{x}}^{\varepsilon} f(\hat{u}(x,t))_x dx \right| dt \\ &= \int_0^T \left| \frac{1}{2}\hat{u}^2(-\varepsilon) \right| + \left| \frac{1}{2}\hat{u}^2(\varepsilon) \right| dt \\ &\ge (1+\varepsilon)T, \end{aligned} \tag{3.6}$$

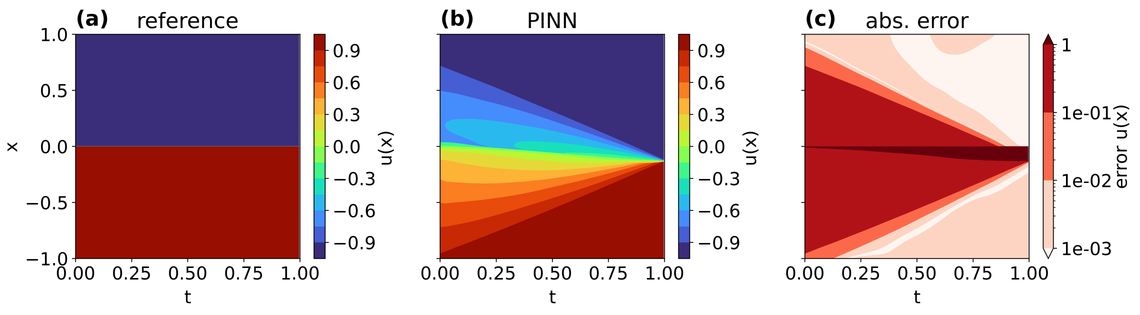


Figure 3.2.: Standard PINN prediction, in comparison to reference analytical solution for the Riemann problem, given by Eq. (3.2). Fig. (a) shows the reference, (b) the PINN and (c) the absolute difference.

using the Burgers equation flux function $f(u) = (1/2)u^2$. In combination with Eq. (3.4), this gives

$$\|\mathcal{R}(\hat{u})\|_{L^2((-\varepsilon,\varepsilon)\times(0,T))} \geq \frac{\sqrt{T}(1+\varepsilon)}{\sqrt{2\varepsilon}} \geq \sqrt{\frac{T}{2\varepsilon}}. \tag{3.7}$$

Hence, for $\varepsilon \longrightarrow 0$ the L_2 norm of the residual around the shock diverges. A shock solution is therefore not a minimum of the L_2 norm based residual loss that is typically employed in PINNs. Note that this is just a one-dimensional argument for one specific conservation law. However, it provides some insight into the underlying issues that arise when shocks solutions are being approximated with PINNs.

In practice, when naively applying PINNs to the Riemann problem (3.2), this becomes obvious, as shown in Fig. 3.2. The predicted shock is completely smoothed out and the optimizer fails to find a better approximation of the shock, because it does not lead to a reduction in the residual loss. To utilize PINNs to find a reasonable approximation of the shock, various approaches have been considered. Overall, the modifications can be divided into three categories:

1. Modifications of the ansatz function
2. Modifications of the loss functional
3. Modifications of the underlying PDEs.

The three different classes of approaches and their advantages and possible drawbacks are discussed in more detail in the following.

Firstly, since NNs are a continuous ansatz function, standard PINNs can only provide a continous approximation of the shock. A **modified ansatz function** could exactly approximate discontinuities and therefore alleviate the issue. This could for example be achieved, by modifying the activation function. Such an approach was pursued by Jin et al. [61] who introduced general exponential activation functions for the approximation of discontinuities with PINNs. It should be noted, that the derivatives are no longer well defined, possibly leading to further instabilities during training. The authors do however report improved prediction accuracies overall, when using the newly designed activation function. Along the same lines, Lorin and Novruzi [83] decomposed the domain into subdomains on either side of the shock. Each subdomain is approximated by a separate NN and the shock can be approximated in a non-dissipative manner, by enforcing the Runkine-Huginot conditions at the interface, using an additional loss function. Hence, the modified ansatz function is in this case constructed via domain decomposition, where jumps according to the Rankine-Hugiot conditions are allowed between the subdomains. This approach has shown a lot of promise for simple examples, providing accurate, non-dissipative and stable predictions. One disadvantage is however that the number of required NNs is dictated by the number of shocks. For complicated problems, the number of shocks might not be known, limiting the applicability of the method in such scenarios.

A second approach would be to **modify the loss function** such that the entropy solution is indeed a minimum. This can for example be achieved by altering the norm applied to the residual. It is well known that the L_2 norm of the point-wise residual might not be a suitable norm for the approximation of discontinuous solutions. De Ryck [116] have introduced weak PINNs, which weakly impose the Kruzhkov entropy conditions. This approach is mathematically sound. Chaumet and Giesselmann explicitly derived the previously shown argument on why PINNs fail to predict shocks and motivate the use of dual norms instead of the point-wise L2 approximation in standard PINNs. Furthermore they make several modifications to the original weak PINN algorithm, leading to higher accuracies and improved efficiency. However, the calculation of the weak PINN loss is still significantly more complex and involves the solution of a min-max problem. Furthermore, even when using weak PINNs, the resulting solutions are still somewhat dissipative, so the practical application to more complex problems is currently limited. Besides the use of different norms, one could also add additional loss terms which regularize the original optimization problem such that the entropy solution is a minimum of the loss. Patel et al. [80] have for example explicitly added an entropy condition loss. Furthermore, oscillations around the shock are penalized with an additional loss, reminiscent of the TVD schemes [3][Ch. 3.1.5] in classical methods.

Lastly, the **PDEs themselves can be adjusted**, such that the modified equations are better suited for the approximation of the solution with a continuous NN. More specifically, the addition of AV smooths out discontinuities, as shown in Fig. (2.5). When adding a small dissipative term to the PDE, the resulting solution is a continuous approximation of the entropy solution. This is due to the fact that the entropy solution of a conservation law itself is defined as the limit of vanishing viscosity of the regularized PDE (2.35). At first, it seems counter-intuitive to modify the equations themselves. Afterall, the goal of the method is to solve the inviscid equations, including the sharp transitions. Hence, additional dissipation will lead to a reduced accuracy which is generally not desirable. However, if the level of dissipation of the approximated solution is acceptable and the resulting algorithm benefits significantly, in-terms of robustness, it is generally regarded as a sensible compromise for practical applications.

It is also worth noting that many classical CFD methods also introduce upwinding using AV. For the compressible Euler equations and other conservation laws some form of dissipation is used to obtain a stable discretization scheme. The development of both accurate and simultaneously robust central difference schemes with AV has been an

Table 3.1.: Different test cases for Burgers' equation.

Case I	Case II	Case III
$u_0(x) = \begin{cases} 1, & x < 0, \\ -1, & x > 0, \end{cases}$	$u_0(x) = \begin{cases} 0, & x < 0, \\ 1/4, & x > 0 \end{cases}$	$u_0(x) = -\sin(\pi x)$
$u(x,t) = u_0(x)$	$u(x,t) = \begin{cases} 0, & x \leq 0 \\ \frac{x}{t}, & 0 < x < t/4 \\ (1/4), & x > t/4 \end{cases}$	numerical reference

active field of research for decades [76, 77, 78, 75, 79]. In terms of final accuracy, a well designed, AV based algorithm can be competitive with upwinding schemes [96].

The addition of dissipation to the system is easily implemented, as only the residual loss term $\mathcal{L}_\mathrm{R}$ has to be modified and the general architecture (Fig. 2.10) is preserved. No additional modifications to the ansatz function or loss function are required. The main challenge for this approach lies however in determining adequate viscosity levels which are sufficiently large to stabilize PINN convergence but not too large such that the inviscid solution is not approximated well anymore.

In the following, different methodologies for determining adequate viscosity levels will are introduced and it will be explored how the system of PDEs can be modified to achieve a PINN algorithms that converges to an accurate approximation of the inviscid entropy solution. This chapter summarizes various works, which have been published as part of this research activity [117, 118, 72, 95, 119]. Independently, similar approaches have been proposed by Coutinhou et al. [81] and Waagenar [73]. In the following, various algorithms for scalar conservation laws are first derived and showcased via application to Burgers' equation. Subsequently, these methods are extended for systems of conservation laws, such as the Euler equations.

3.2. Artificial Viscosity Concepts for Scalar Conservation Laws

To stabilize the training process of PINNs, the PDEs themselves are modified with a small dissipative term, as previously discussed. In the most trivial case, the general structure of the NN (as in Fig.2.10 (a)) is kept and only the residual operator $\mathcal{R}$ that is used for the calculation of the residual loss $\mathcal{L}_\mathrm{R}$ in Eq. (2.84) is modified. The modified system with artificial viscosity for a scalar conservation law is given by:

$$\mathcal{R}(u(x,t)) = \frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} - \mu \frac{\partial^2 u}{\partial x^2}, \tag{3.8}$$

with a variable $\mu = \mu(x,t)$, called the **artificial viscosity** which is to be determined such that the resulting model approximates the inviscid solution. For Burgers' equation the flux is given by $F(u) = u^2/2$.

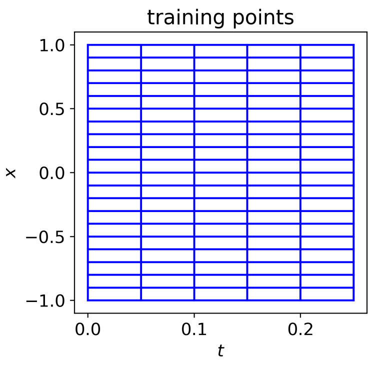


Figure 3.3.: Uniform grid for Burgers test cases. The grid nodes are used as training points. For better visibility, only every 10th grid line is shown.

3.2.1. Test Cases

To analyze the efficacy of different viscosity formulations, three different Cauchy problems for Burgers' equation, given in Tab 3.1, are considered. Case I is the previously analyzed Riemann problem with a stationary shock solution. It acts as a sanity check for the methods capability to approximate a simple shock. Case II is another Riemann problem. As shown in Sec. 2.1.2, the entropy solution to this problem is given by a rarefaction wave. Since the solution is continuous, case II will be used to confirm that all methods perform well, even when no shock is present. For Case III, another problem with a shock solution is considered. The initial conditions are smooth for this case and the shock establishes after an initial buildup phase. Hence, this case is of higher complexity than the previous problems. For all experiments, the domain is given by $(x, t) \in (-1, 1) \times (0, 1) \subset \mathbb{R} \times \mathbb{R}^+$ and Dirichlet boundary conditions are used, with $u(-1, t) = u_0(-1)$ and $u(1, t) = u_0(1)$. For all three cases, the shock/rarefaction are localized around the $x = 0$ position.

Initially the performance on a uniform training point distribution with a Cartesian grid of $N_x \times N_t = (200 \times 50)$ points is investigated. The grid is shown in Fig. 3.3. The boundary and initial conditions are enforced using Eqs. (2.87)-(2.88) which are evaluated at the nodes on the corresponding boundaries of the grid.

3.2.2. Global Artificial Viscosity

First, the global application of artificial viscosity in the entire domain is considered. In this case $\mu(x, t)$ simplifies to a constant $\nu = \mu(x, t)$. The magnitude of the AV needs to be determined empirically. For Burgers' equation, experiments show reasonable results for $\nu \in (10^{-3}, 10^{-2})$. Fig. 3.4 shows the predicted results when applying a global viscosity of $\nu = 3 \cdot 10^{-3}$. Accurate results are obtained for Case I besides some smoothing of the shock. This is due to the fact that the solution is constant in the entire domain besides the shock. Hence, the second derivatives vanish in the entire domain except at the smoothed out shock. The additional dissipation only influences the regions of the domain where it is actually desired. However, for the rarefaction wave in Case II, the accuracy is significantly reduced. The global dissipation smooths out the kink in the solution at $x = 0$, widens

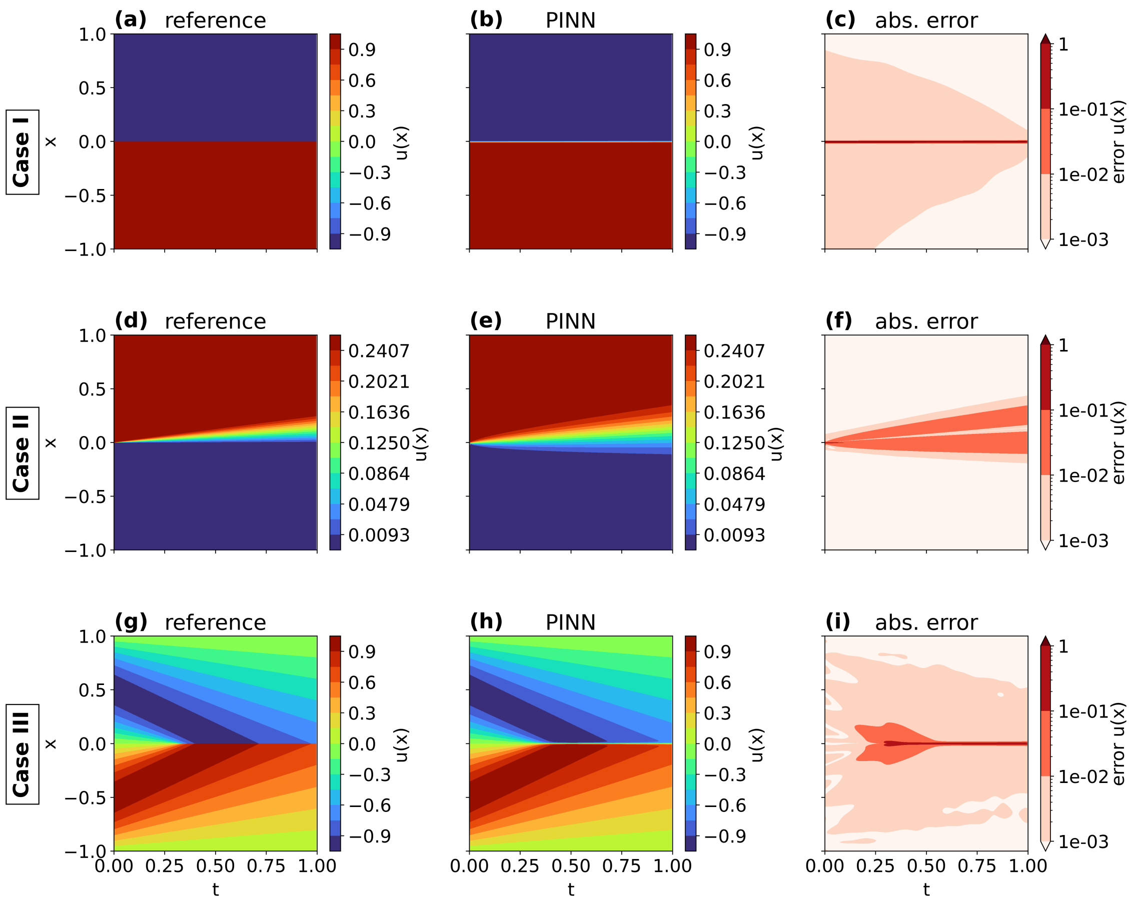


Figure 3.4.: Prediction of PINN with global viscosity $\nu = 3 \cdot 10^{-3}$ for case I (a)-(c), case II (d)-(f) and case III (g)-(i) with a uniform mesh (Fig. 3.3).

the wave and bends the characteristics. Evidently, global viscosity can deteriorate the solution in continuous regions of the flow, when they are not smooth. For case III, we again see an accurate approximation. In this case the viscosity does not significantly affect the smooth regions of the flow and is able to stabilize the shock.

3.2.3. Adaptive Artificial Viscosity

To avoid a global deterioration of accuracy due to dissipation, a viscosity distribution in the domain needs to be found, where the AV is only applied in regions where it is required for stability reasons. The first approach we follow is to let the NN itself predict a viscosity distribution. Here, the idea is to let the NN predict $\mu(x,t)$ as a second variable besides the solution $u(x,t)$. In the following, this methodology is called *adaptive AV*. This approach has been proposed independently by Wassing et al. [120, 72] and Waagenar [73], with slight variations. The positivity of the predicted viscosity can be ensured by using a positivity enforcing activation functions such as the exponential $\sigma^D(x) = \exp(x)$ or the square $\sigma^D(x) = x^2$. Since the residual loss can not be reduced around the shock without AV, the NN should be able to localize this region and automatically identify the need

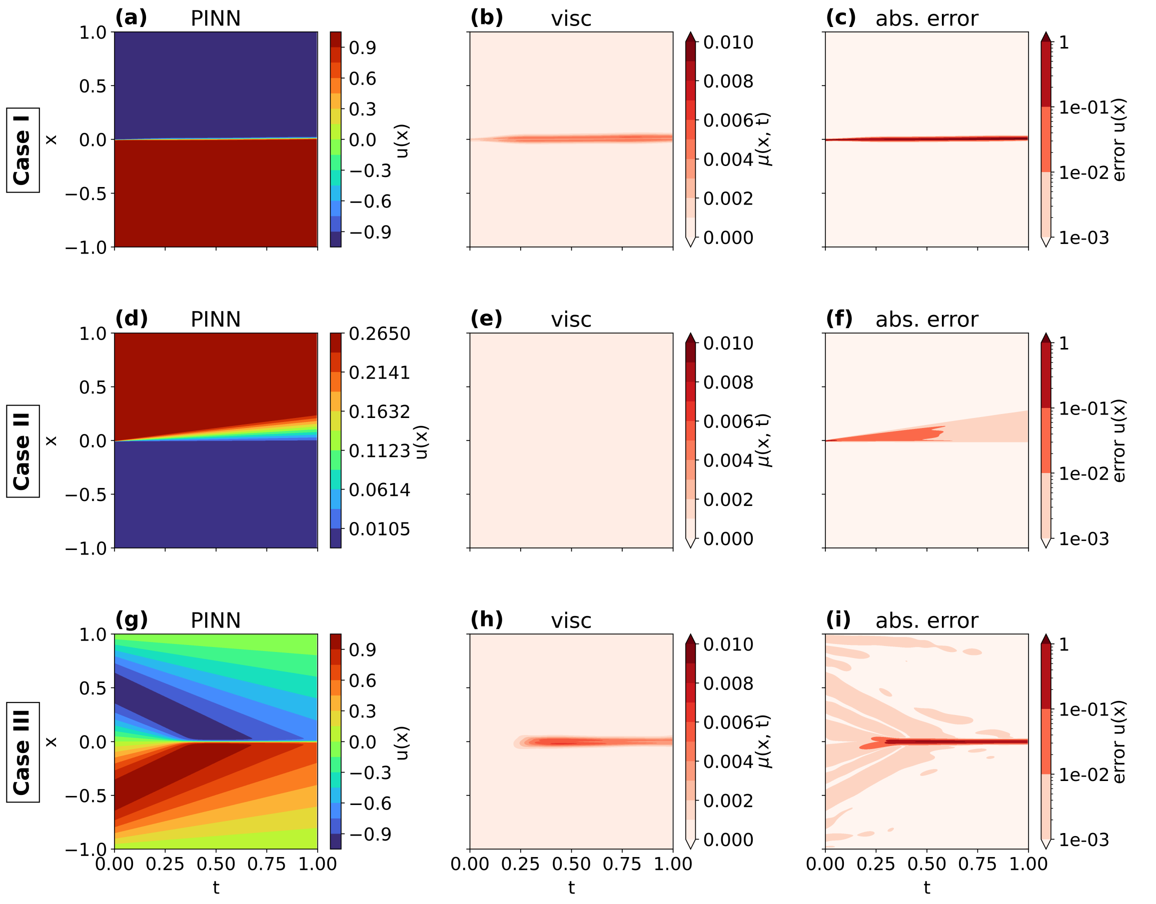


Figure 3.5.: Prediction of PINN with adaptive AV for case I (a)-(c), case II (d)-(f) and case III (g)-(i) with a uniform mesh (Fig. 3.3).

for additional dissipation in this regions. However, to avoid that the NN aggressively increases the AV in the entire domain, the increase of μ needs to be penalized with an additional loss term $\mathcal{L}_\mu$. Since the positivity is already enforced, one can simply use the average viscosity:

$$\mathcal{L}_\mu = \lambda_\mu \frac{1}{N_\mu} \sum_{i=1}^{N_\mu} \mu(x_i, t_i). \tag{3.9}$$

While in this case, no viscosity value needs to be chosen explicitly, the loss term weighting factor λ_μ is added as an additional hyperparameter to the problem. For the Burgers equation, based on empirical investigations, $\lambda_\mu \in (1, 10)$ is a reasonable range. Fig. 3.5 shows the PINN prediction in the left column, the predicted viscosity in the center column and the absolute error in the right column. The shown model uses a weighting factor of $\lambda_\mu = 2$. Similarly to the sensor-controlled AV, the adaptive AV is able to correctly identify the shocks in case I and case III and applies a reasonable amount of viscosity to stabilize the training. For case II, no additional viscosity is applied. Compared to the sensor controlled AV (see below) and the global AV, the shocks are more smoothed out.

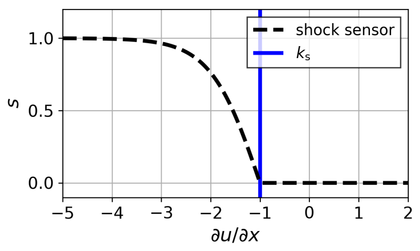


Figure 3.6.: Illustration of shock sensor for scalar conservation laws with $k_s = 1$.

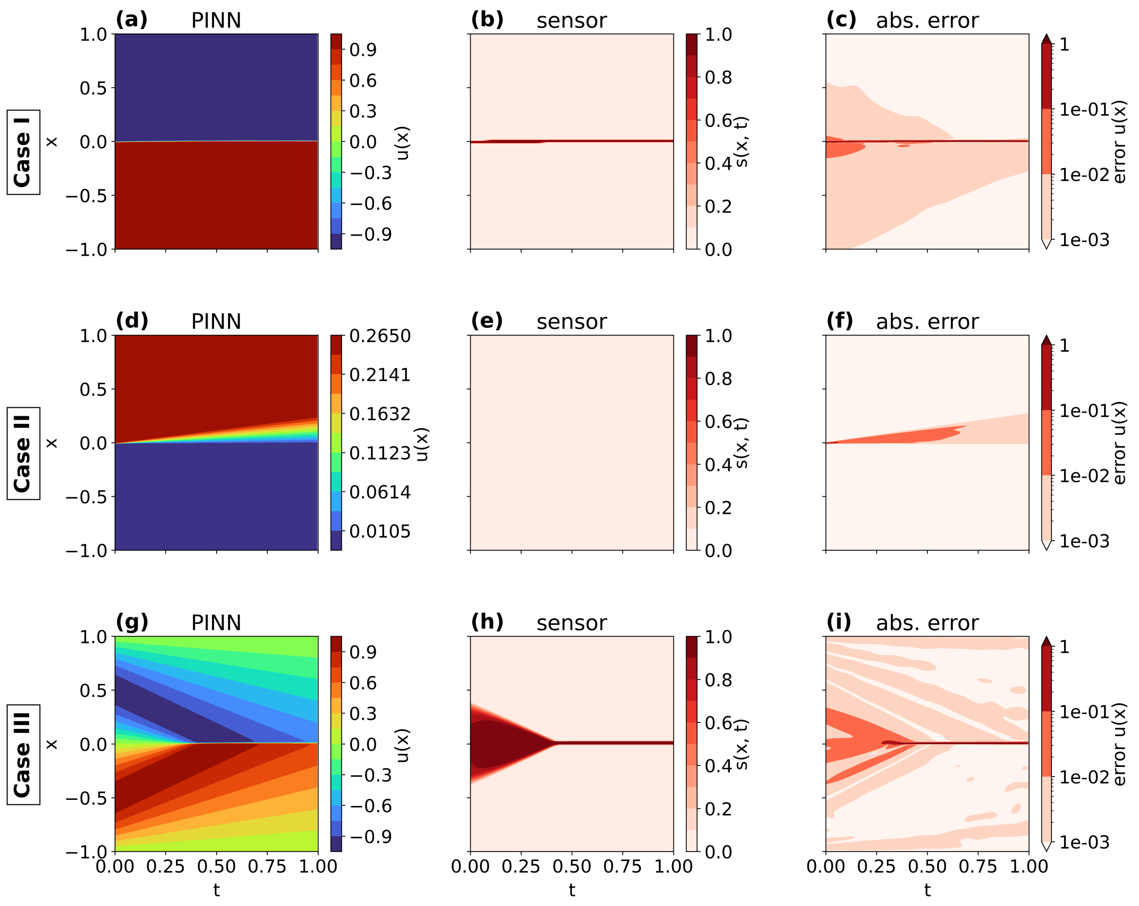


Figure 3.7.: Prediction of PINN with sensor-controlled viscosity $\nu = 3 \cdot 10^{-3}$ for case I (a)-(c), case II (d)-(f) and case III (g)-(i) with a uniform mesh (Fig. 3.3).

3.2.4. Sensor Controlled Artificial Viscosity

Another approach to avoid a global deterioration of accuracy due to dissipation, is to construct an analytical function, which identifies the regions in the domain where viscosity needs to be applied. For now, let us assume that this is only around the shock wave. Hence, the sensor function $s(u(x,t))$ is introduced, which localizes the shock, based on the spatial derivative:

$$s(u(x,t)) = \tanh\left(\max\left(0, -k_s - \frac{\partial u}{\partial x}\right)\right), \tag{3.10}$$

Here k_s is the activation threshold for the derivative. The total AV is then given by:

$$\mu(x,t) = s(x,t) \cdot \nu, \tag{3.11}$$

where ν is a constant scalar to be selected empirically and determines the magnitude of the dissipation. The NN is a continuously differentiable ansatz function. Therefore, when approximating a discontinuity, the spacial gradients of the solution exist and will increase in the vicinity of the shock. The sensor is constructed such that it is only active when negative derivatives are encountered, since decompression shocks are not possible. For rarefaction waves, it is hence not active. The outer hyperbolic tangent ensures that the sensor is normalized to values $0 \leq s(u(x,t) < 1$. The sensor values for different $\partial u/\partial x$ are shown in Fig. 3.6. The PINN predictions for a viscosity factor $\nu = 2 \cdot 10^{-3}$ are presented in Fig. 3.7. The middle column shows the sensor function. For case I and case III the PINN with sensor controlled AV achieves similar accuracies as the global AV. The sensor plots reveal how viscosity is only applied in a small region around the shock and for the build-up of the shock in Case III. For case II, the accuracy is improved dramatically, since the sensor is not active for rarefaction waves. Hence, the kink in the solution is not smoothed out and the PINN achieves higher accuracies. Therefore, case II clearly indicates that the locations where viscosity is applied need to be chosen carefully.

3.2.5. Mesh Refinement and Transformation

Overall, we have seen that with the previously described AV methodologies, we can obtain relatively accurate approximations of the solution for case I-III. However, due to the smoothing effect of the AV, shocks are less sharp compared to the reference solution. In Ch. 2.4.6, the mesh transformation (MT) methodology for PINNs has been proposed, to improve the prediction accuracy of PINNs in nonlinear regions, by transforming the problem from a physical domain into a computational domain. By constructing a mesh, which better resolves more complex regions of the domain, the optimization problem can be simplified. To make use of the MT (see Sec. 2.4), we explicitly define the transformation:

$$\begin{aligned}
&\mathcal{F}^{-1} : (-1,1) \times (0,1) \to \Sigma,\\
&\mathcal{F}^{-1}((\xi,\tau)) = (x(\xi), t(\tau)),\\
x(\xi) &= a \cdot \left(\frac{2\xi}{N_\xi - 1} - 1\right)^3 + (1-a) \cdot \left(\frac{2\xi}{N_\xi - 1} - 1\right),\\
t(\tau) &= \frac{\tau}{N_\tau - 1}.
\end{aligned} \tag{3.12}$$

Dealing with one spatial dimension and time t, the second computational coordinate, which is proportional to t, has been renamed to τ. For stepsizes $\Delta\xi = \Delta\tau = 1$, the physical grid is then given by (3.12) and the computational coordinates:

$$(\xi_{i,j}, \tau_{i,j}) \in \Sigma, \quad \xi_{i,j} = i \cdot \Delta\xi, \quad \tau_{i,j} = i \cdot \Delta\tau, i = 0, \ldots, N_\xi - 1, \quad j = 0, \ldots, N_\tau - 1. \tag{3.13}$$

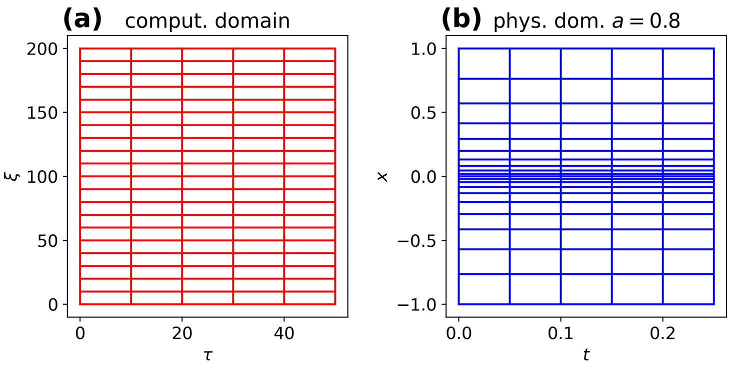


Figure 3.8.: Grids in computational domain (a) and physical domain (b). For better visibility, only every 10th grid line is shown. The parameter a can be used to control the mesh density around the shock. For $a = 0$ the physical grid is uniform in x direction (see Fig. 3.3). For $a = 0.8$, the mesh density around $x = 0$ is increased.

The parameter a stretches the grid around $x = 0$. For $a = 0$ we have uniformly distributed grid points identical to the previously used grid (Fig. 3.3), and for $a = 1$, the grid is compressed around $x = 0$ at maximum. A comparison of the computational grid with coordinates (ξ, τ) and the physical grid with coordinates (x, t) is shown in Fig. 3.8 for $a = 0$ ad $a = 0.8$. Note that for the uniform grid with $a = 0$, the MT becomes trivial since the computational and physical grid are identical besides a linear scaling and translation. For the analyzed test cases, the grid can simply be modified by changing the mesh stretch factor a. A refined mesh with $a = 0.8$ is selected, as shown in Fig. 3.8 (b), which significantly increases the spacial resolution in the neighborhood of $x = 0$. In this neighborhood, we observe the largest errors (see Fig. 3.4-3.5) because for all three problems the shock or rarefaction originates from $x = 0$. Hence, improved accuracies and possibly more sharply resolved shocks can be expected, when using the MT. We repeat all previous experiment with the refined grid with $a = 0.8$ and with the mesh transformation methodology. To analyzed whether possible improvements, compared to the previous experiments, can be mainly attributed to the increased point density or to the MT we also perform additional experiments where a standard PINN model without MT is trained on the new grid points with $a = 0.8$.

To quantitatively compare the different PINN models, the error metrics Eq. (2.107)-(2.109) are considered. As before, $\hat{u}$ is the model prediction and u is the analytical/reference solution. The metrics are evaluated on 5000×250 uniformly distributed points (x_i, t_i). The error bounds for all metrics are given by the standard deviation over four different training runs. For each run, the quasi-random initialization of the NN parameters $\boldsymbol{\theta}$ uses a different seed. For a fair comparison, ν or λ_μ, in case of the adaptive viscosity, are chosen based on a grid search. For each of the 10 model, the ν or λ_μ is selected which altogether performs best on all three cases. A summary of all metrics is shown in Appendix B.4.1 in Tabs. B.2-B.4, for case I-III, respectively. For all three cases, the prediction accuracy is generally improved when refining the mesh around $x = 0$. We also see that the MT performs similarly well to the model without MT on the refined grid with $a = 0.8$. When using the refined mesh, all three AV methods deliver satisfactory results on a similar order of magnitude with the exception of the global viscosity, which performs significantly worse on case II. As expected, the global AV still smooths out the rarefaction even when using the refined mesh. The sensor and adaptive AV models are able to identify that for this case, no AV is required and hence perform similar to the model without AV. As an example for the improved predictions with $a = 0.8$, the model with MT and adaptive viscosity is shown in Fig. 3.9. Note that the errors are overall reduced and that the shocks are more sharply resolved for case I and III. The predicted viscosity is almost an order of magnitude lower around the shocks.

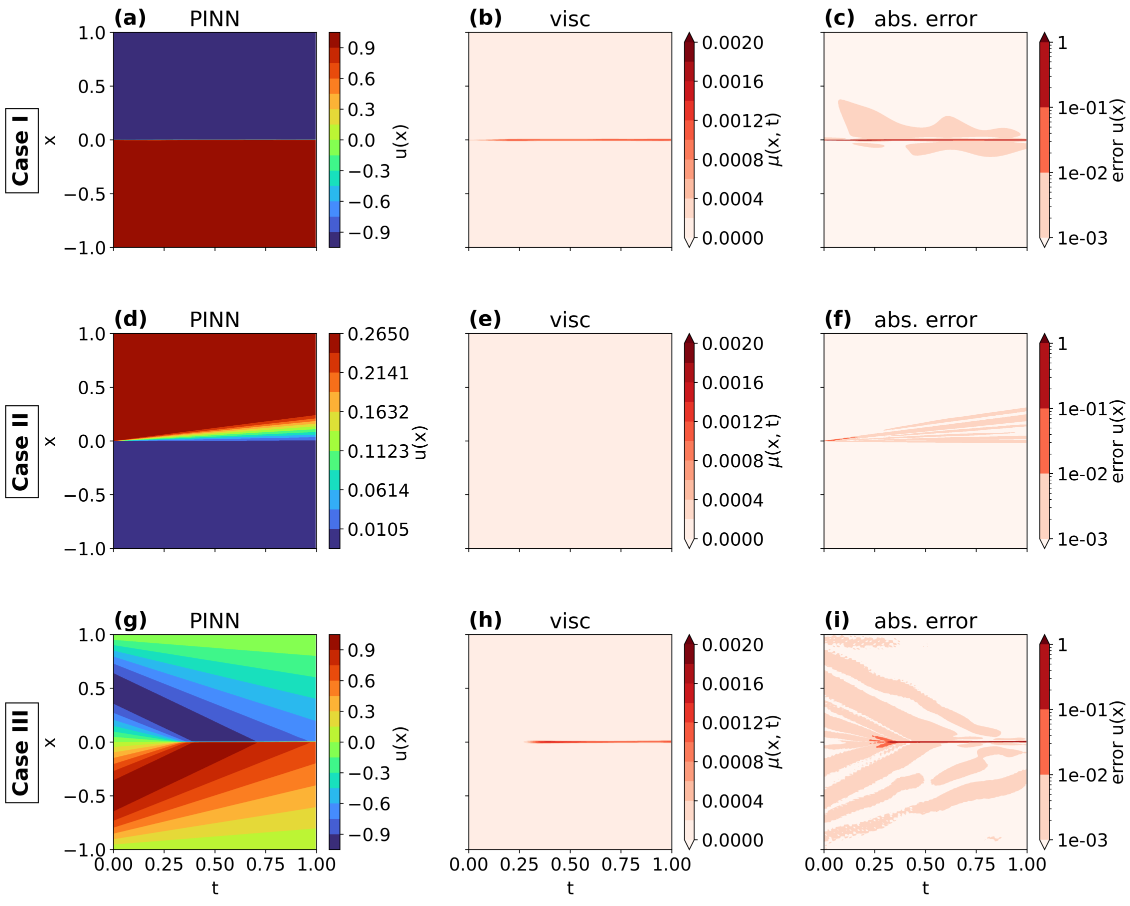


Figure 3.9.: Prediction of PINN with adaptive AV for case I (a)-(c), case II (d)-(f) and case III (g)-(i) with a refined mesh 3.8 (b) and MT.

Overall, it was confirmed that the adaptive and the sensor controlled AV are both viable options for identifying shocks and deliver accurate predictions on all analyzed cases. While increasing the point density around shocks can indeed help to improve the accuracy of the models, the MT methodology has not shown significant improvements. It should however be noted that, while the analyzed test cases show some local non-linearities, no complex domain shapes were considered, yet. When dealing with more curved geometries, the ability of the MT to transform the curved domain into a rectangular one might be more beneficial for improving the PINN convergence.

In the following section, the adaptive AV and the sensor controlled AV are further extended for solving the Euler equations.

3.3. Artificial Viscosity Concepts for the Euler Equations

As shown in the previous section, artificial viscosity can be used to approximate solutions to the inviscid Burgers equation. In the following, we shall build upon the previously identified algorithms and modify them so that they are suitable for the Euler equations. As before the residual operator of the Euler equations (2.41) is extended by a dissipative term. Here, the two dimensional Euler equations are considered, but the approach can easily be extended to higher dimensions. This work focuses one steady-state-solutions and one can hence assume $\partial \boldsymbol{W}/\partial t = 0$. The modified $\mathcal{R}$ for a steady state reads:

$$\mathcal{R}(\boldsymbol{W}(x,y)) = \frac{\partial \boldsymbol{F}_x}{\partial x} + \frac{\partial \boldsymbol{F}_y}{\partial y} - \boldsymbol{D}(\boldsymbol{W}(x,y)), \tag{3.14}$$

with the flux vectors $\boldsymbol{F}_x$ and $\boldsymbol{F}_y$, the vector of conserved variables $\boldsymbol{W}$ as defined in Eq. (2.41), and the additional dissipative term $\boldsymbol{D}(\boldsymbol{W})$. In general this term is defined as:

$$\boldsymbol{D}(\boldsymbol{W}(x,y)) = \left(\mu_x(x,y)\frac{\partial^2}{\partial x^2} + \mu_y(x,y)\frac{\partial^2}{\partial y^2}\right)\boldsymbol{W}(x,y). \tag{3.15}$$

As previously seen for Burgers' equation, the accuracy and stability of the approach depend significantly on the construction of μ_x and μ_y. On the one hand, $\boldsymbol{D}(\boldsymbol{W})$ needs to be sufficiently large to stabilize the training, in particular near discontinuities. On the other hand $\boldsymbol{D}(\boldsymbol{W})$ will smooth out shocks and potentially obstruct convergence when it is too large.

The task at hand is to model the quantities μ_x and μ_y to close the system of equations. We are dealing with a system of four equations and μ_x and μ_y can, in the most general case, be 4×4 matrices. Essentially, we have to decide how to determine the magnitude of viscosity for each equation. For the previously analyzed scalar conservation laws, this step was trivial, as we were only dealing with a single equation. In that case, only the magnitude of a scalar viscosity in the domain where it is active had to be determined.

In the following, we distinguish between these two tasks. Firstly the *AV type* is modeled, meaning the way in which one decides how viscosity is applied to each equation. Secondly the *AV localization* os modeled, meaning the methodology that is used to determine the spatial distribution of viscosity in the domain.

Hereafter, three different **AV types** are first introduced. Since the orientation of Cartesian coordinate system is arbitrary, it is reasonable to presume isotropy. Hence, μ_x and μ_y are constituted in a similar manner. Still, this leaves twelve entries to be modeled. In the most simple scenario, we use a scalar viscosity. In this case, we assume that the required AV for each equation is the same. Inspired by the classical, scalar AV methods [76] described in Sec. 2.2.2, the **scalar AV** is defined as:

$$\mu_x(x,y) = \mu_y(x,y) = \nu(x,y)\ (c(x,y) + \|\boldsymbol{q}(x,y)\|). \tag{3.16}$$

Here, $c + \|\boldsymbol{q}\|$ is the spectral radius of the flux Jacobians $A = \partial \boldsymbol{F}_x/\partial \boldsymbol{W}$ and $\boldsymbol{B} = \partial \boldsymbol{F}_y/\partial W$ [6], and it is weighted by a scalar factor $\nu(x,y)$. Note how the scaling with the spectral radius is following the general idea of (2.59) of adjusting the AV based on the local maximal wave speed. However, compared to the CFD method, fourth derivatives are not utilized. In classical CFD methods, fourth differences are used to avoid oscillation in smooth parts of the domain. PINNs are generally not prone to such oscillations and hence the fourth derivatives are not required.

In Eq. (3.16), the viscosity factor $\nu(x, y)$ still needs to be modeled to close the system of equations. It determines the general magnitude of the dissipative term as well as the location where in the domain viscosity is applied. In Secs. 3.3.1-3.3.2, we discuss two possibilities to model $\nu(x, y)$.

Eq. (3.16), applies the same viscosity magnitude to all four equations. However, in general the momentum variables ρu and ρv are continuous across shocks. Hence, when dealing with shock waves, the momentum equations do not need to be smoothed for the PINN to approximate ρu and ρv exactly. Consequently, the AV can be applied to the density and energy equations only, reducing dissipativeness in the momentum equations. We define the **density and energy AV**

$$\mu_x(x, y) = \mu_y(x, y) = \nu(x, y)\ I_{\rho,\rho E}\ (c(x, y) + \|\boldsymbol{q}(x, y)\|), \tag{3.17}$$

with $I_{\rho,\rho E} = \mathrm{diag}(1, 0, 0, 1)$. To further reduce dissipation, the propagation direction of waves needs to be taken into account. In classical CFD methods, this is achieved with so-called matrix-valued AV [78, 96], where the spectral radii in the dissipative terms are replaced with matrices. For finite volume schemes, the matrix-valued AV was introduced in Eq. (2.64). For PINNs, we define the **matrix valued AV**:

$$\begin{aligned} \mu_x(x, y) &= \nu(x, y)\ I_{\rho,\rho E}|A| \\ \mu_y(x, y) &= \nu(x, y)\ I_{\rho,\rho E}|B|, \end{aligned} \tag{3.18}$$

with $|A| = T|\Lambda_x|T^{-1}$ for the eigendecomposition $A = T\Lambda_x T^{-1}$ and $|B|$ accordingly for the y-direction. The matrix Λ_x essentially represents the diagonal matrix of the eigenvalues of A. The Matrices $|A|$ and $|B|$ are constructed as discussed in [96]. However, since we evaluate Eq. (3.14) in Cartesian coordinates, the formulation simplifies with $\xi = x, \eta = y, J = 1$. Hence, the eigenvalues for A are given by:

$$\begin{aligned} \lambda_1 &= u + c \\ \lambda_2 &= u - c \\ \lambda_3 &= u \\ \lambda_4 &= u. \end{aligned} \tag{3.19}$$

However, the eigenvalues are limited to avoid instabilities at sonic-lines and stagnation points. The limited values are given by:

$$\begin{aligned} |\tilde{\lambda}_1| &= \max(|\lambda_1|, \varepsilon) \\ |\tilde{\lambda}_2| &= \max(|\lambda_2|, \varepsilon) \\ |\tilde{\lambda}_3| &= \max(|\lambda_3|, \varepsilon), \\ |\tilde{\lambda}_4| &= \max(|\lambda_4|, \varepsilon), \end{aligned} \tag{3.20}$$

with $\varepsilon = 0.25$. For $|B|$, u is replaced by v.

3.3.1. Adaptive Artificial Viscosity

For Burgers' equation, we have seen that when using the adaptive AV method, the PINN model is able to automatically localize regions of the domain which require AV. In a similar manner, this method can be adapted for the Euler equations. As before, the viscosity factor $\nu(x, y)$ is predicted as an additional output of the neural network alongside the primitive variables $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx (\rho(\boldsymbol{x}), u(\boldsymbol{x}), v(\boldsymbol{x}), E(\boldsymbol{x}), \nu(\boldsymbol{x}))$. The linear scaling of the viscosity μ with the maximal wave speed $a + |\boldsymbol{q}|$ results in higher viscosity values in regions with higher flow speeds. One might think that the factor could be dropped in Eqs. (3.16)-(3.18), since the NN can always locally adapt ν to compensate for the scaling. However, scaling can positively affect the convergence of the model during training. It can effectively alleviate differences in the magnitudes of predicted AVs at different flow speeds, since it scales with the local velocity. Therefore, the NN prediction $\nu(x, y)$ itself can be more uniform which improves convergence. Furthermore, when considering parametric problems with variable inflow velocities, the viscosity is adjusted according to the resulting variable wave speed. Generally, this method is combined with a scalar AV Eq. (3.16), assuming that the neural network can find a reasonable distribution of $\nu(x, y)$, which is not too dissipative even with just a scalar AV.

To enforce positivity of the viscosity and to allow for a flexible prediction of values close to 0, the exponential function is used as an activation for ν in the last layer of the network. As in Eq. (3.9), to avoid large viscosity values, an additional loss term $\mathcal{L}_\mu$ is used to control the AV levels. The loss term penalizes high viscosity values

$$\mathcal{L}_\mu = \lambda_n u |\nu - \tilde{\nu}|. \tag{3.21}$$

Note that, compared to Eq. (3.9), the prescribed viscosity value $\tilde{\nu}$ has been introduced. This is a scalar reference viscosity used to control the strength of the dissipative term during training, while still allowing the network to locally choose $\nu(\boldsymbol{x}) \neq \tilde{\nu}$ when necessary. The modulus is used to enforce positivity of the term instead of the square because local outliers should not be penalized. To limit the smoothing effect of the AV while simultaneously promoting convergence towards the entropy solution, the viscosity is adjusted during the training process, to limit the dissipative effect on the final solution. The following 3 phase training routine is used

Phase 1: Train with ADAM at a constant viscosity $\tilde{\nu} = \nu_0$ until no significant changes in the residual loss are observed.

Phase 2: Train with ADAM and reduce the prescribed viscosity $\tilde{\nu}$ until $\tilde{\nu} = 0$. Continue at $\tilde{\nu} = 0$ until no significant changes in the residual loss are observed.

Phase 3: Train with L-BFGS and $\tilde{\nu} = 0$ until convergence.

The idea of this procedure is to guide the network towards a physical solution (the entropy solution) during the initial training phase. Once the network has converged to a state that resembles a physical but viscous solution, the viscosity can be reduced, since, from this point on, the network should be in a state near the entropy solution. In phase 2, $\tilde{\nu}$ is reduced to 0, to decrease the dissipative effect on the prediction. However, the network can still predict viscosity factors $\nu(\boldsymbol{x}) > 0$ where necessary, keeping the penalty term in balance with the residual and other loss terms, similarly to the procedure used in Sec. 3.2.3. The prescribed viscosity factor $\tilde{\nu}$ is reduced as follows:

$$\tilde{\nu}_i = \begin{cases} \nu_0\left(1 - \left(\frac{i}{M_{\text{red}}}\right)^k\right) & \text{if } 0 \leq i \leq M_{\text{red}} \\ 0 & \text{if } M_{\text{red}} < i, \end{cases} \tag{3.22}$$

where $i \in \mathbb{N}$ is the epoch counter starting at phase 2 and M_{red} is the epoch at which $\tilde{\nu} = 0$ is reached. The exponent $k \in \mathbb{N}^+$ can be used to modify the shape of the reduction curve. For $k = 1$ the reduction is linear and for $k > 1$ it is accelerating. For the first two phases, Adam [100] is used as the optimizer. A final training period with the quasi-Newton L-BFGS optimizer [31] has shown to be effective for convergence of PINNs in general and has been crucial to achieve high levels of accuracy with the proposed training procedure.

3.3.2. Sensor-Controlled Artificial Viscosity

When dealing with complex geometries or scenarios with more complex shocks, the adaptive AV method might be unable to reliably converge to an adequate distribution of $\nu(x,t)$. In these cases, a sensor controlled AV is more suitable, where the distribution of viscosity in the domain is modeled directly, based on analytical expressions. In this case, we have

$$\nu(x,y) = \nu\, s(x,y) \tag{3.23}$$

where the viscosity factor ν becomes a global constant, determining the overall magnitude of the viscous term, and the sensor $s(x,y) \in [0,1])$ identifies the region of the domain where the viscosity is applied. Here we focus on developing a sensor for normal shock waves, which are for example frequently encountered in transonic flows. Across normal shocks, an abrupt change in the flow field in flow direction is observed. The NN is a continuous ansatz function. It approximates a discontinuity by a continuous, possibly very sharp, transition window with high gradients of the flow variables. In general, the pressure over normal shock waves increases in flow direction. Hence, the dot product of the normalized flow velocity $\boldsymbol{q}/\|\boldsymbol{q}\|$ and the pressure gradient ∇p, will be positive in the region of a compression wave. This is exploited to construct the shock sensor

$$s_{\text{shock}}(x,y) = \tanh\left(\max\left(0, k^{(1)}_{\text{shock}}\left(\frac{\boldsymbol{q}(x,y)}{|\boldsymbol{q}|} \cdot (\boldsymbol{\nabla} p(x,y)) - k^{(0)}_{\text{shock}}\right)\right)\right). \tag{3.24}$$

The hyperbolic tangent and the inner max-function limit the sensor to $s(x,y)_{\text{shock}} \in (1,0]\text{for}(x,y) \in \Sigma$. The second parameter $k^{(1)}_{\text{shock}}$ controls the steepness of $s(x,y)$ for increasing gradients. Overall, in regions of high positive pressure gradients in flow direction the sensor becomes $s \approx 1$. Everywhere else, the sensor is turned off ($s = 0$).
While the sensor is designed to identify normal shock waves, strong pressure gradients in flow direction also occur at stagnation points, for example at the leading edge an airfoil. Our experiments show that convergence is obstructed when the sensor is active at stagnation points. Therefore, we extend the sensor by a second factor which removes the stagnation points. Here, we use the fact that the momentum in x- and y-direction is conserved across normal shock waves. Hence, we can assume that the gradients of ρu and

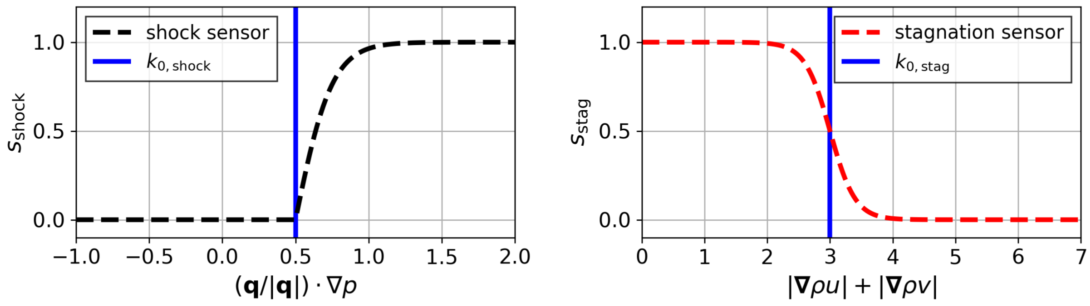


Figure 3.10.: Schematic illustration of s_{shock} (left) for $k^{(1)}_{\text{shock}} = 4$, $k^{(0)}_{\text{shock}} = 0.5$ and s_{stag} (right) for $k^{(1)}_{\text{stag}} = 5$, $k^{(0)}_{\text{stag}} = 3$.

ρv are small in the region of the shock. The momentum will however experience strong gradients near stagnation points. We therefore propose the stagnation point sensor

$$s_{\text{stag}}(x, y) = \text{sig}(k^{(1)}_{\text{stag}}(k^{(0)}_{\text{stag}} - (|\nabla\rho u| + |\nabla\rho v|)). \tag{3.25}$$

Here, the sigmoid function $\text{sig}(x) = 1/(1 + \exp(-x))$ ensures that $s_{\text{stag}}(x, y) \in (0, 1)$. Again, the parameters $k^{(1)}_{\text{stag}}$ and $k^{(0)}_{\text{stag}}$ can be used to adapt the steepness of the switch and the threshold. Overall, near the stagnation point, the sensor becomes $s_{\text{stag}} \approx 0$ and everywhere else it will be $s_{\text{stag}} \approx 1$. The complete sensor function $s(x, y)$ is the product of the shock and the stagnation-point sensor

$$s(x, y) = s_{\text{shock}}(x, y)s_{\text{stag}}(x, y). \tag{3.26}$$

Fig. 3.10 illustrates the two sensor components. In Sec. 6.2 we also show $s(x, y)$ in a practical example. The recommended values for $k^{(0)}_{\text{shock}}, k^{(1)}_{\text{shock}}, k^{(0)}_{\text{stag}}$ and $k^{(1)}_{\text{shock}}$ are shown in Appendix A.2 and the same values are used for all experiments.

3.4. Summary

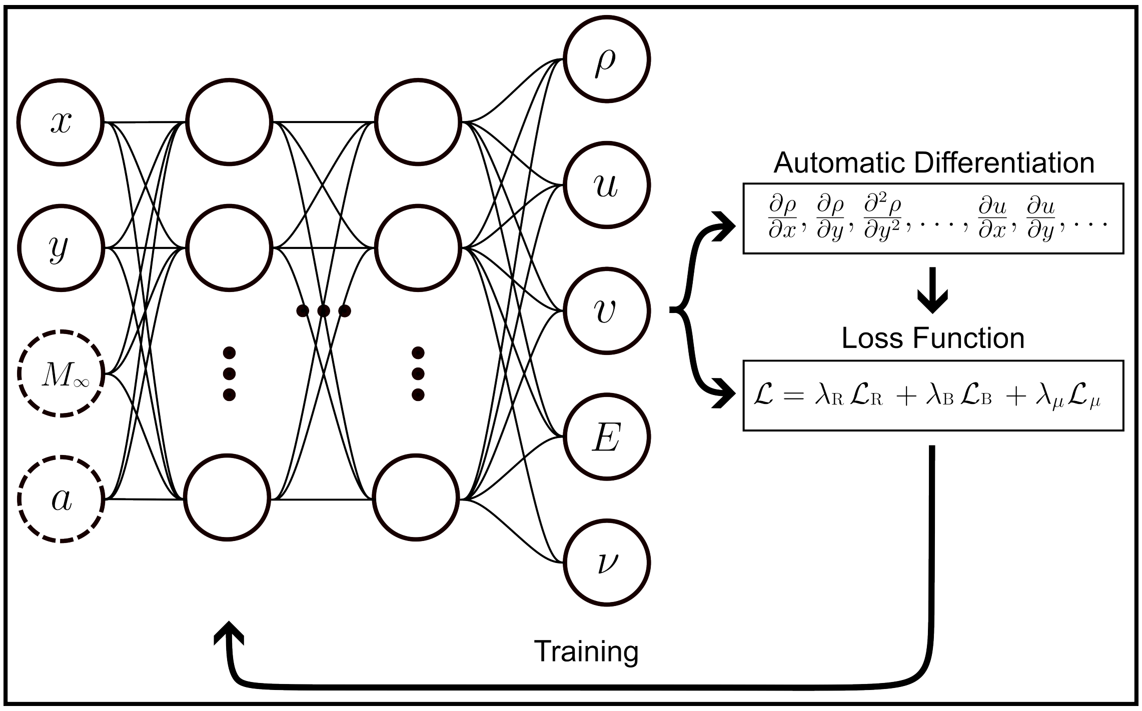
In this chapter, the limitation of Physics-Informed Neural Networks (PINNs) in predicting shock waves, which stems from the fact that a solution featuring a shock is not a minimum of the L2-norm based residual loss, is addressed. To overcome this issue, possible modifications to the PINN method are discussed, including changes to the ansatz function, the loss function, and PDE formulation using AV. This work focuses on the use of AV which smooths out discontinuities to enable accurate approximations by PINNs. Using Burgers' equation as a test case, the effectiveness of AV for this scenario is demonstrated and various AV localization methods are introduced, including global AV, adaptive AV, and sensor-controlled AV. The comparison of the performance of these methods on different Cauchy problems for Burgers' equation, highlights their advantages and disadvantages, and identifies sensor-controlled AV and adaptive AV as promising variants. Additionally, the conceptual approach is extended to the Euler equations, introducing three different types of AV, scalar AV, density and energy AV and matrix-valued AV. Furthermore, the adaptive AV and sensor-controlled AV localization methods are adapted for the use on the compressible Euler equations. The results provide a foundation for improving the accuracy and robustness of PINNs in simulating complex fluid dynamics and aerodynamics problems governed by hyperbolic conservation laws. In the following three chapters, these new approaches are utilized to solve various test cases for the compressible Euler equations, involving non-trivial geometries, shock waves and parametric problems.

4. Predicting Subsonic Flows around Cylinders

In the previous chapter, the use of artificial viscosity in PINNs was already demonstrated on the inviscid Burgers equation and different AV methods for the compressible Euler equations were introduced. In the following, the performance of a first PINN model, which makes use of these methods, is analyzed. The selected test case is a subsonic compressible flow around a cylinder. This problem can be regarded as a prototypical benchmark for problems in aerodynamics. The cylinder acts as an aerodynamic object, positioned at the center of a comparatively large domain. To obtain an accurate solution, the domain needs to be significantly larger than the object to guarantee that the far-field boundary conditions can be accurately imposed. However, near the object, the changes in the flow field becomes non-linear, leading to a combination of long and short-scale effects and flow phenomena which are challenging for the global ansatz function that is the NN.

The artificial viscosity methods in Ch. 3 have been motivated mainly to stabilize shock waves. Since the analyzed problem in this chapter is subsonic, no shock waves are expected. However, in classical CFD methods, AV is also required for subsonic problems as shown in Sec. 2.2.2. It remains to be seen if it is also relevant for PINNs in these cases. The sensor-controlled AV localization in Sec. 3.3.2 only applies AV in the vicinity of shock waves. Therefore, the adaptive AV localization (Sec. 3.3.1) is utilized in this chapter. This methodology is advantageous for the analyzed problem as the NN can adaptively decide if and where AV is required for this problem. If no AV is required for the subsonic problem, the NN should ultimately choose $\nu = 0$ in the entire domain. The used PINN model will also use the scalar AV (3.16) type, as the adaptive AV localization should be able to reduce the AV locally to limit the dissipativeness of the model. Therefore, the more complicated density and energy AV and matrix-controlled AV are not required to reduce dissipativeness.

A schematic depiction of the used NN model is shown in Fig. 4.1. The model receives, as inputs, the Cartesian coordinates (x, y). It predicts the primitive flow quantities (ρ, u, v, E). In addition, to adaptively determine the AV distribution, the viscosity factor $\nu(x, y)$ is added to the output space of the NN. High viscosity values are penalized using the additional loss term $\mathcal{L}_\mu$ (3.21), added to the standard residual and boundary loss terms in (2.84). As a soft control mechanism for the applied viscosity, the prescribed viscosity factor $\tilde{\nu}$, was introduced. To guide the network towards the inviscid solution during training, a three phase training routine, introduced in Sec. 3.3.1, is used. The model is initially trained with a higher $\tilde{\nu}$ to induce convergence towards a viscous solution. Then,



x
y
M_∞
a
ρ
u
v
E
ν
Automatic Differentiation
$\frac{\partial \rho}{\partial x}, \frac{\partial \rho}{\partial y}, \frac{\partial^2 \rho}{\partial y^2}, \dots, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \dots$
Loss Function
$\mathcal{L} = \lambda_\mathrm{R}\mathcal{L}_\mathrm{R} + \lambda_\mathrm{B}\mathcal{L}_\mathrm{B} + \lambda_\mu\mathcal{L}_\mu$
Training

Figure 4.1.: Schematic depiction of NN model model with adaptive AV for the prediction of sub and supersonic flows. The dashed inputs to the NN are parameters of the PDE and are only used for the parametric models.

in the second training phase, the prescribed viscosity is reduced, such that the model adaptively identifies the regions of the domain, where AV needs to be maintained to avoid an increase in the residual loss and reduced elsewhere. Finally, the model is fine-tuned with the L-BFGS optimizer in the third training phase.

In the second part of the chapter, the analysis is extended to a parametric version of the problem. For the parametric problem we have a variable Mach number M_∞ and the parameter a for the geometry, which are both added to the input space of the NN, as also shown in Fig. 4.1. In aerodynamic applications, variations of the geometry are often of interest. Therefore, the parametric problem also acts as a proof of concept for more complicated parametrizations of the geometry. A final aspect of the investigation in this chapter are the activation functions. As for example shown in [62], the hyperbolic tangent activation is generally a reasonable choice for PINNs. However, it remains to be seen whether the adaptive variant of the hyperbolic tangent, as introduced in Sec. 2.3 is a good choice for PINN model. Since the parametric problem can generally be considered to be more challenging, the performance of adaptive and non-adaptive hyperbolic tangent activations is compared on this task. In the following, we initially focus on the non-parametric problem and the training setup, including the used point distributions is discussed in more details. Note that parts of this chapters are based on [117, 72].

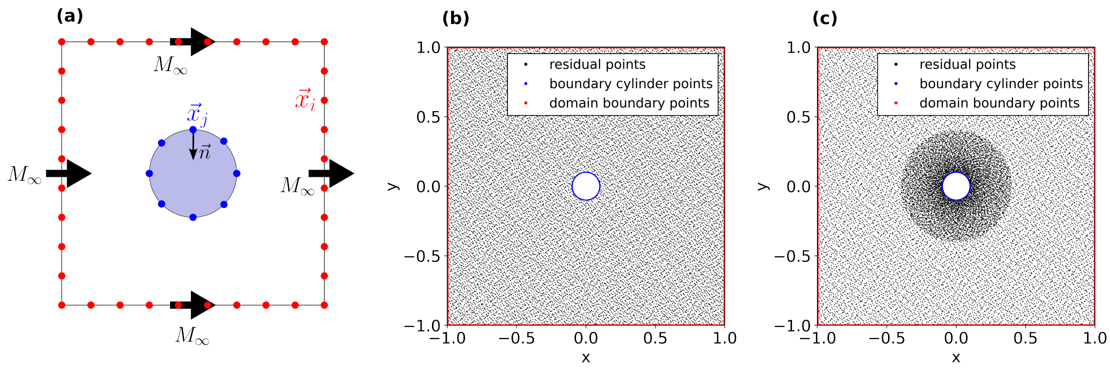


Figure 4.2.: Fig. (a) shows a schematic depiction of the test case. A cylinder is placed at the center of Ω. The uniformly distributed points on the domain boundary (blue), and the cylinder (red) are used to enforce the boundary conditions. Figs. (b)-(c) show the collocation points for evaluating the residual and the boundary conditions on the cylinder's surface

4.1. Forward Problem

As the first test-case for the Euler equations, a two-dimensional subsonic flow around a cylinder is considered. A schematic overview of the problem is shown in Fig. 4.2 (a). The cylinder is positioned at the center of the domain $\Omega = [-1, 1] \times [-1, 1]$. The cylinder radius is set to 0.1 so that the outer domain boundaries are at least ten cylinder radii away from its surface. For the distribution of collocation points, the uniform sampling strategy 1 (see Sec. 2.4.5) is initially used for generating the residual training points. The Halton sequence [110] is used as the space-filling sampling method. The boundary points for the far field boundary condition and the slip boundary condition on the cylinder surface are also uniformly distributed. The distribution of points is shown in Fig. 4.2 (b). Initially, we analyze the flow at a Mach number of $M_\infty = 0.2$. Since the cylinder is rotationally symmetrical, the flow is only analyzed at an angle of attack of $\alpha = 0^\circ$ without loss of generality. The boundary conditions are then enforced using Eq. (2.101), as explained in Sec. 2.4.2.

As introduced in Sec. 3.3.1, we make use of a the three phase training routine. For the first two phases, a total of 20000 collocation points is used for the residual loss. Since the last training phase uses the memory intensive L-BFGS [31] optimizer, we reduce the number of collocation points to 5000 in phase 3. An initial prescribed artificial viscosity factor of $\tilde{\nu} = 7.5 \cdot 10^{-4}$ is used based on the previous investigations. We have observed that this value is typically a reasonable value and can be used both for subsonic and supersonic problems. We choose a fully connected neural network of constant layer width with tanh as activation functions. The loss weighting factors are set to $\lambda_{\mathrm{B}} = 1$, $\lambda_{\mathrm{R}} = 1$ and $\lambda_\mu = 5$. As discussed in Sec. 2.4.4, the (dynamic) weighting of loss terms is an effective technique to accelerate the convergence and improve the accuracy because it can compensate imbalances in the gradients between the different loss terms. However, we have observed that for certain problems, adaptive loss term weighting may lead to instabilities during the early stages of the training. For the presented problems we do not see significant imbalances during the training and are able to achieve satisfying results with a constant weighting factor. Therefore, dynamic loss term weighting is not considered for these models. All hyperparameters of the used model are summarized in Appendix A.2. As a reference solution, we use finite volume results, as further described in Appendix D.

The reference solution is plotted in Figs. 4.3-(a)-(c) for the pressure coefficient C_p, the local Mach number M and the density field ρ. Initially, a PINN model without AV is analyzed, meaning that the viscosity factor is set to $\nu(x, y) = 0$ from the start of the training and that no additional dissipation is applied. Multiple simulation runs with different random initializations of the trainable parameters $\boldsymbol{\theta}$ are performed. Interestingly, an inconsistency can be observed, where one out of the 13 runs performs significantly worse than the others. This resulting prediction of this run is shown in Figs 4.3-(d)-(f). Especially for the velocity field in Fig. (f), we see that, the flow is detached from the cylinder as indicated by the regions of low velocity up- and downstream of the cylinder. Furthermore, in the pressure field, the peaks in the reference (a) are basically not observed in the prediction. Similarly for the density field (d), the prediction is also far off at these locations. The absolute difference between the reference simulation and the prediction of the failed training run is shown in Figs. 4.3 (a)-(c), clearly highlighting the significant errors. However, all other runs without AV converged to more reasonable predictions. Another exemplary well-converged simulation run without AV is shown in Figs. 4.3-(g)-(i) and the corresponding absolute errors are presented in Figs. 4.4-(d)-(f). The predicted solution clearly resembles the reference. The error plots do however reveal slight asymmetries in the flow field between the upper and lower side. In the pressure and density field, we also see that larger errors are observed near the cylinder surface.

The initial results without AV show that, while certain inconsistencies are observed, AV does not seem to be strictly required to obtain reasonable solutions for a majority of the runs. In a next step, we analyze, whether the PINN with adaptive AV localization can obtain similar accuracies. For the training of the model with AV, the same training parameters (e.g. learning rate, batch sizes, …) as for the model without AV are used (see Sec. A.2). The results of an exemplary simulation run are presented in Fig. 4.3 (j)-(l) with the corresponding absolute errors in Figs. 4.4 (g)-(i). Comparing the training runs with and without AV, we see a similar quality of results. The error plots reveal that the model with AV seems to provide slightly better predictions for M, with no significant asymmetries between the upper and lower side of the cylinder, while the prediction of the pressure and density field are slightly more inaccurate. Notably, all of the runs with AV converged to similarly accurate results and no training inconsistencies as for the models without AV were observed. For a more meaningful comparison of the convergence behavior, the prediction accuracy for the density field is considered during training. Fig. 4.5 (a) shows the average *RMAE* for ρ during the training. For the model with and without AV 12 runs with different random $\boldsymbol{\theta}$ initializations are performed. The failed run for the model without AV is excluded. The plotted line indicates the average *RMAE* and the shaded areas indicate the minimal and maximal accuracy of the 12 respective runs (i.e. the prediction spread). For the entire training, we see that the average PINN model with AV performs slightly better than the model without AV. For both variants, we see that

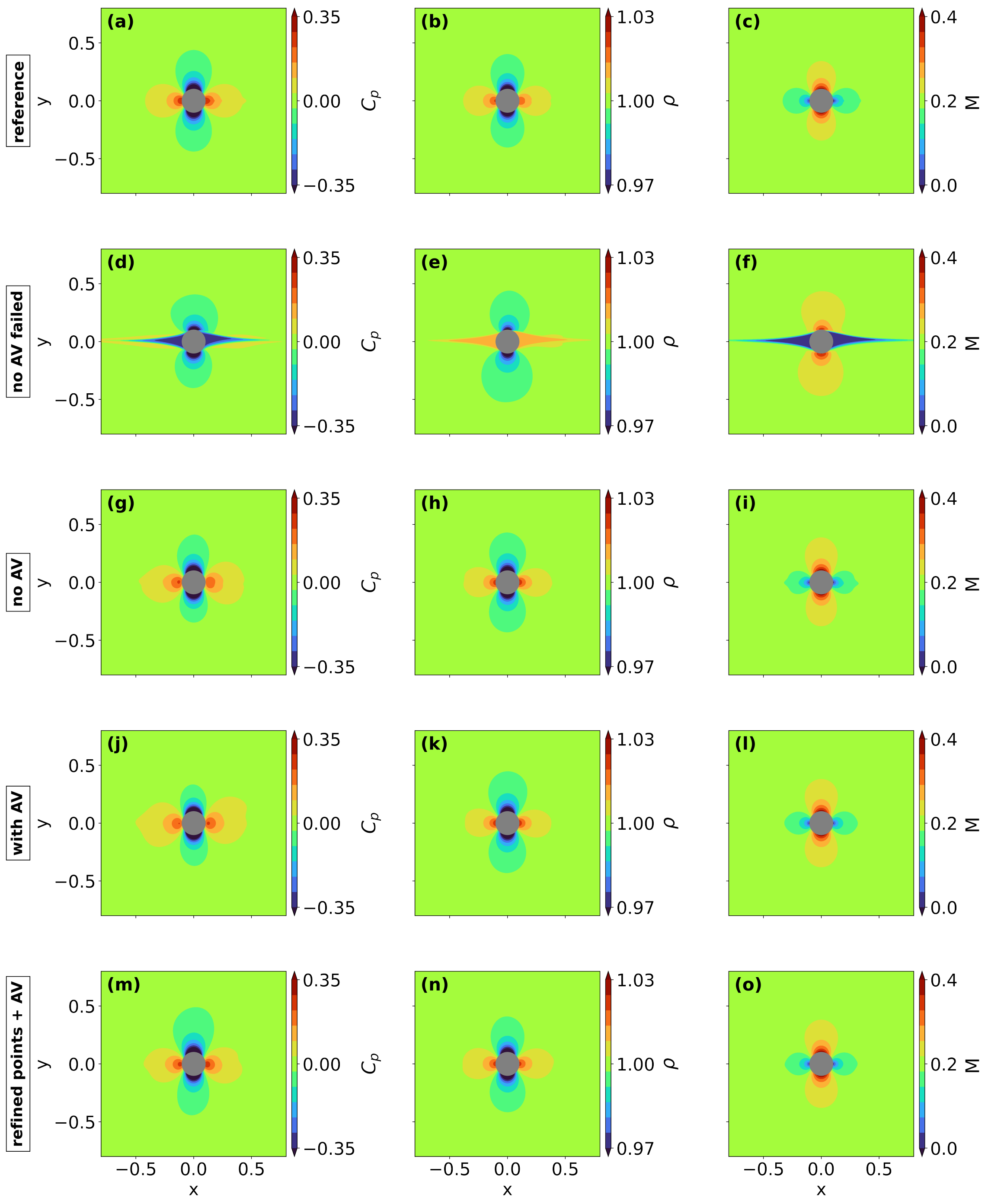


Figure 4.3.: Comparison of reference solution and different PINN prediction for C_p (left column), ρ (center column) and M (right column). Figs. (a)-(c) show the reference finite volume solution. Figs. (d)-(f) shows the failed run without AV. Figs. (h)-(i) and Figs. (j)-(l) show exemplary runs with and without AV and the uniform point distribution (see Fig. 4.2 (b)). Figs. (m)-(o) depict an exemplary result with AV and the refined point distribution with increased point densities near the cylinder surface (see Fig. 4.2 (c)).

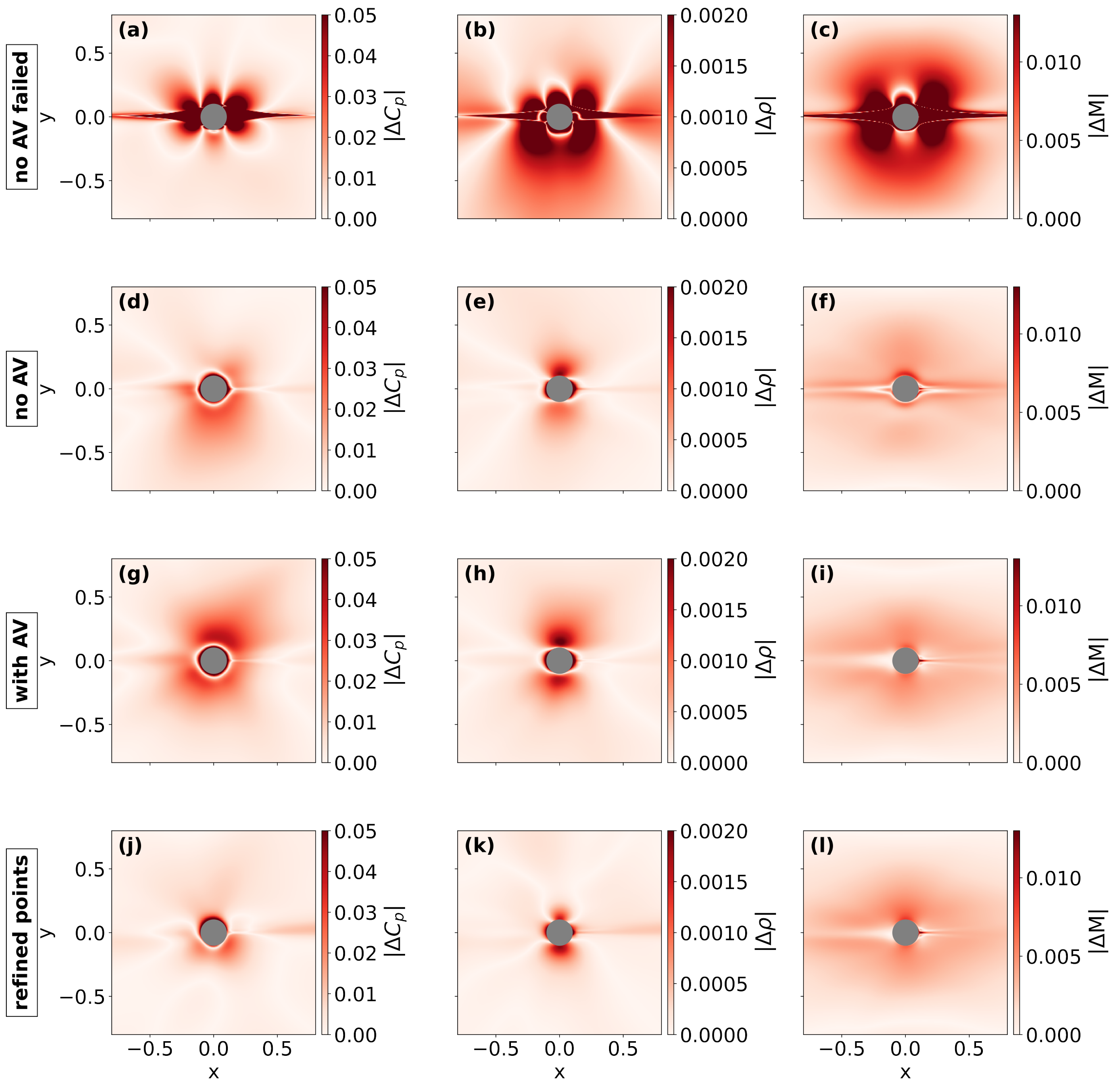


Figure 4.4.: Comparison of of absolute errors of different PINN prediction in comparison to reference solution for C_p (left column), ρ (center column) and M (right column). See also the corresponding PINN predictions in Fig. 4.3. Figs. (a)-(c) show the reference finite volume solution. Figs. (d)-(f) shows the failed run without AV. Figs. (h)-(i) and Figs. (j)-(l) show exemplary runs with and without AV and the uniform point distribution (see Fig. 4.2 (b)). Figs. (m)-(o) depict an exemplary result with AV and the refined point distribution with increased point densities near the cylinder surface (see Fig. 4.2 (c)).

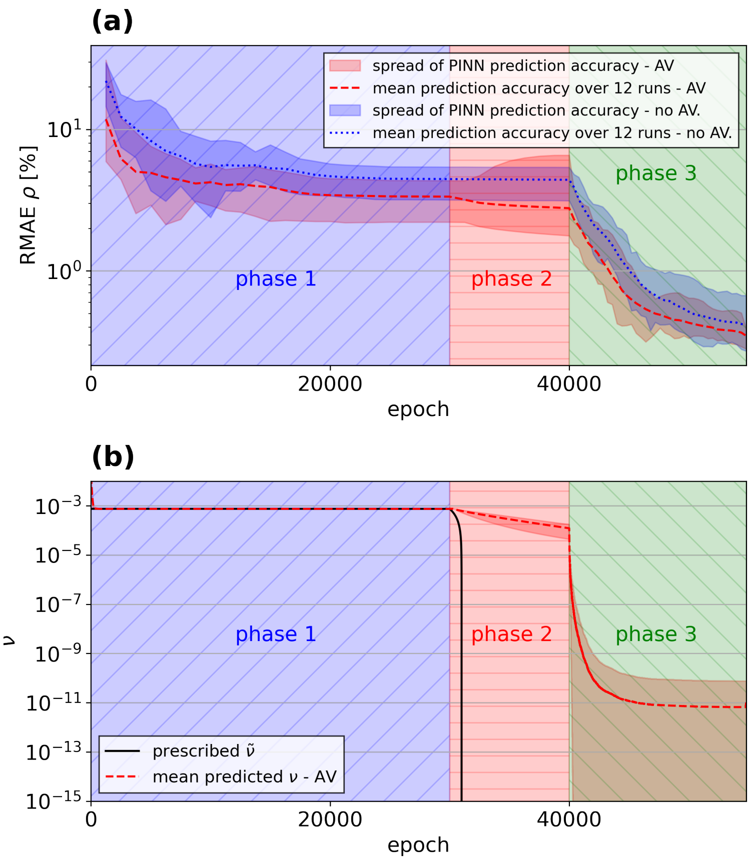


Figure 4.5.: Fig (a) shows the average *RMAE* for ρ for 12 models with and without AV. The three training phases (see Sec. 3.3.1 are highlighted in the background. Fig. (b) shows the average predicted viscosity factor $\nu(x, y)$) in comparison to the prescribed viscosity factor $\tilde{\nu}$.

the third training phase with the L-BFGS optimizer is crucial to obtain reasonable results as it improves the accuracy by more than one order of magnitude.

Another interesting aspect to consider is the artificial viscosity that is predicted during training by the PINN model with adaptive AV. The average AV levels are shown in Fig. 4.5 (b) After the first training stage, the predicted AV is on a uniform level of $\nu = 7.5 \cdot 10^-3$. This exactly agrees with the prescribed viscosity factor $\tilde{\nu}$. After the second training stage, where $\tilde{\nu}$ is reduced to 0, the PINN has also reduced the AV values significantly, resulting in a uniform AV distribution of $\nu \approx 1 \cdot 10^{-4}$. The AV is then further reduced by the L-BFGS optimizer in the third training phase, resulting in $\nu(x, y) \lesssim 10^{-10}$. Evidently the adaptive AV model finds that for the final state, no AV for stabilization is required. This confirms the previous observation that the final training state of the model does not require artificial dissipation. However, the convergence suggest that AV is still beneficial to enhance the convergence of the model.

To further improve the accuracy of the model with AV near the cylinder surface, the density of collocation points in the vicinity of the cylinder is increased, as shown in

Table 4.1.: Error metrics for non-parametric flow around cylinder.

	$\boldsymbol{RMAE}$ [%]		$\boldsymbol{MAE}$		$\boldsymbol{R_2}$	
no AV	C_p	1.1 ± 0.4	C_p	$(1.2 \pm 0.4) \cdot 10^{-2}$	C_p	0.949 ± 0.028
	M	0.797 ± 0.022	M	$(3.17) \pm 0.1) \cdot 10^{-3}$	M	0.9827 ± 0.0014
	ρ	0.42 ± 0.14	ρ	$(3.5 \pm 1.1) \cdot 10^{-4}$	ρ	0.99 ± 0.005
points in Fig. 4.2 (b)	C_p	0.85 ± 0.25	C_p	$(9.3 \pm 2.8) \cdot 10^{-3}$	C_p	0.949 ± 0.024
	M	0.81 ± 0.021	M	$(3.22 \pm 0.09) \cdot 10^{-3}$	M	0.9819 ± 0.0011
	ρ	0.37 ± 0.06	ρ	$(3.1 \pm 0.5) \cdot 10^{-4}$	ρ	0.991 ± 0.003
points in Fig. 4.2 (c)	C_p	0.55 ± 0.21	C_p	$(6.1 \pm 2.3) \cdot 10$	C_p	0.979 ± 0.019
	M	0.812 ± 0.023	M	$(3.24 \pm 0.1) \cdot 10^{-3}$	M	0.9764 ± 0.0018
	ρ	0.29 ± 0.08	ρ	$(2.4 \pm 0.6) \cdot 10^{-4}$	ρ	0.9954 ± 0.0021

Fig. 4.2 (c). This distribution follows strategy 2, outlined in Sec. 2.4.5. The total number of collocation points is retained but 25 % of all points are exclusively sampled near the cylinder. The resulting predictions, shown in Fig. 4.3 (m)-(o) with the corresponding absolute errors in Figs. 4.4 (j)-(l), are indeed improved. Especially in the pressure field, the region with large errors near the cylinder surface is visibly reduced. However, errors remain at the upper surface. Similarly, in the density field, the errors are slightly reduced. To further increase the accuracy on the surface, one could employ the mesh-transformation methodology (see Sec. 2.4.6) which would not only increase the point density near the cylinder, but also simplify the training process by stretching high point density regions through the transformation towards a computational domain. This approach is further investigated in Ch. 6.

To quantitatively evaluate the accuracy of the analyzed models, the error metrics Eqs. (2.107)-(2.109) with respect to the reference solution are calculated for a small square that contains the cylinder ($-0.5 < x < 0.5; -0.5 < y < 0.5$). The errors are averaged over twelve training runs with different NN parameter initializations. The failed run for the model without AV is excluded from this calculation to ensure a fair comparison. The standard deviation over the twelve runs is used for the error bounds. The results are presented in Tab. 4.1. Overall, we see that the use of AV can slightly improve the accuracy of the model, compared to the identical model without AV which can mainly be attributed to the improved convergence during training. In addition, it seems to be reasonable to locate more training points near the objects surface, to improve the prediction accuracy of the pressure field.

For all models with AV, the total training time on a single NVIDIA-A100 graphics card amounts to 7.5 h. Once trained, the model can be evaluated in about 0.5 s. For the model without AV, we see that the training time is reduced to 4.25 h, as no additional second derivatives need to be calculated. This significantly reduced the computational effort in the evaluation of the residual loss. However, due to the added benefit in terms of convergence and accuracy, we make use of AV in the following analysis of the parametric problem.

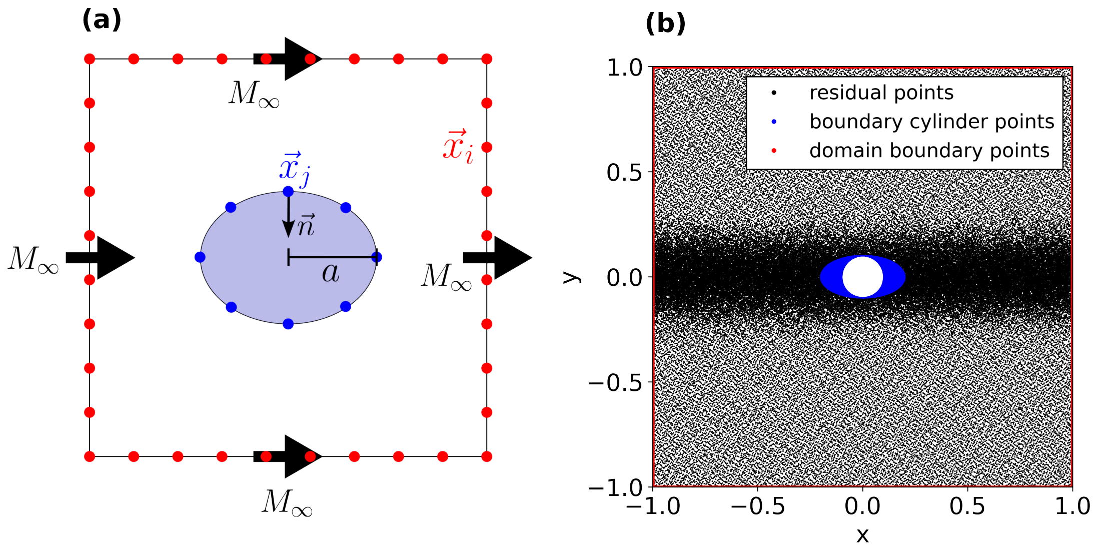


Figure 4.6.: Fig. (a) shows a schematic depiction of the parametric test case. An ellipse is placed at the center of Ω. The uniformly distributed points on the domain boundary (blue), and the cylinder (red) are used to enforce the boundary conditions. The parameter a corresponds to the major axis of the ellipse. Fig. (b) shows a projection of the actual point distribution of four-dimensional parametric space onto physical domain for subsonic ellipse problem. Note the variation in the axis a in the boundary points.

4.2. Parametric Forward Problem

As the first parametric problem, the previous test case is modified by parametrizing the inflow boundary condition with a variable Mach number $M_\infty \in [0.2, 0.4]$. Furthermore, to analyze the capability to approximate solutions around parametric geometries, the cylinder shape is parametrized. We introduce the parameter $a \in [0.1, 0.2]$ corresponding to the major axis of the resulting ellipse (see Fig. 4.6 (a)). The minor axis is constant at $b = 0.1$. The shape of an ellipse can be characterized with the eccentricity $e = \sqrt{1 - b^2/a^2}$ with $e \in [0, 1)$. At a value of $a = 0.1$, we have a cylinder of radius $r = 0.1$ which corresponds to an eccentricity of $e = \sqrt{1 - b^2/a^2} = 0$. For the maximal major axis of $a = 0.2$ the eccentricity is $e = 3/4$. The neural network receives both M_∞ and a as additional inputs. The training points are therefore sampled in a four-dimensional domain. The upper limit of the Mach number of $M_\infty = 0.4$ is slightly below the critical Mach number of the cylinder, at which the velocity locally exceeds the speed of sound. As before, the a initial prescribed artificial viscosity factor of $\nu = 7.5 \cdot 10^{-4}$ is used. Since the actual artificial viscosity in Eq. (3.16) is scaled with the local wave speed, the strength of the dissipation naturally increases at higher Mach numbers.

To further improve the model, the performance of fixed and adaptive hyperbolic tangent activation functions (Eq. (2.70)) is investigated. We observe that during the third training phase, when using the L-BFGS optimizer, the usage of adaptive activation functions leads to highly inconsistent results (i.e. in some runs convergence is similarly to tanh and in others no further convergence is possible or the optimizer diverges). Therefore, the trainable parameters ω^k for the final training phase are frozen, meaning that these parameters are no longer optimized during phase 3. In doing so, one is able to avoid the previously observed inconsistencies. The loss weighting factors are set to $\lambda_{\mathrm{B}} = 1$, $\lambda_{\mathrm{R}} = 1$ and $\lambda_{\mu} = 5$, as before A summary of all hyperparameters is shown in Appendix A.2. This includes the utilized optimizer, the learning rate lr the batch size N_{batch} and the prescribed viscosity factor $\tilde{\nu}$.

To cover the four-dimensional input space, the number of residual training points is increased compared to the previous non-parametric case ($N_{\mathrm{R}} = 100000$). However, since a mini-batch routine is used for the first two training phases, this has typically no negative effect on the training speed. Only during the last training phase 3, memory may be a limiting factor because the L-BFGS optimizer is incompatible with mini-batch training. Therefore, a reduced number of training points is used during the last training phase ($N_{\mathrm{R}} = 30000$). Since a non-uniform point distribution has improved the prediction accuracies for the non-parametric case, we again increase the density of points near the ellipse. Half of the points are distributed uniformly across the entire physical domain Ω using the Halton sequence [110]. For the other half of the points, the y-coordinate is sampled using a normal distribution with a variance of $\sigma = 0.07$ and with a uniform distribution for the x-coordinate. As before, this approach follows strategy 2 from Sec. 2.4.5. A projection of the resulting point distribution to the physical domain is shown in Fig. 4.6 (b). Compared to Fig. 4.2 (c), this strategy ensures that the density of points in x-direction is uniform. Hence, for all ellipse shapes covered by $a \in (0.1, 0.2)$, the density of points near the surface is similar. For both point sets, the same number of points is used to represent the boundary of the physical domain and the cylinder ($N_{\mathrm{R}} = N_{\infty} = N_{\mathrm{ob}}$).

The resulting predictions for the velocity field for three cases are shown in Fig. 4.7 in comparison to reference finite volume simulations. For additional information on the calculation of reference finite volume results, see Appendix D. One can see that for the cylindrical shape ($a = 0.1$) even at $M_{\infty} = 0.4$, close to the critical Mach number, the results are visually indistinguishable from the reference solution. The plot of the absolute error reveals that the inaccuracies are fairly uniform and no strong asymmetries between the upper and lower surface are observed. For the other two parameter sets with ellipsoidal shapes, a similar quality of the results can be observed. The errors are overall highest at $M_{\infty} = 0.35$ and $a = 0.2$. This test-case is located at the bounds of the parameter space and also features a comparatively high Mach number. The bottom row of Fig 4.7 depicts the artificial viscosity μ. The viscosity is relatively uniform (see the scale of the color bars) for all three Mach number. Up and downstream of the ellipse, the viscosity is however slightly reduced. We see that the remaining viscosity is larger than for the non-parametric case, hinting that the parametric problem benefits more from the AV than the more simple non-parametric variant in the previous section.

For certain problems it can be observed that PINNs can perform inconsistently, depending on the random initialization of network parameters at the start of the training. To ensure the consistency of the proposed method, the accuracy over 12 training runs with different random initialization seeds is analyzed. Fig. 4.8 (a) shows the mean error in the density field during training. The best and worst prediction accuracy over the 12 runs

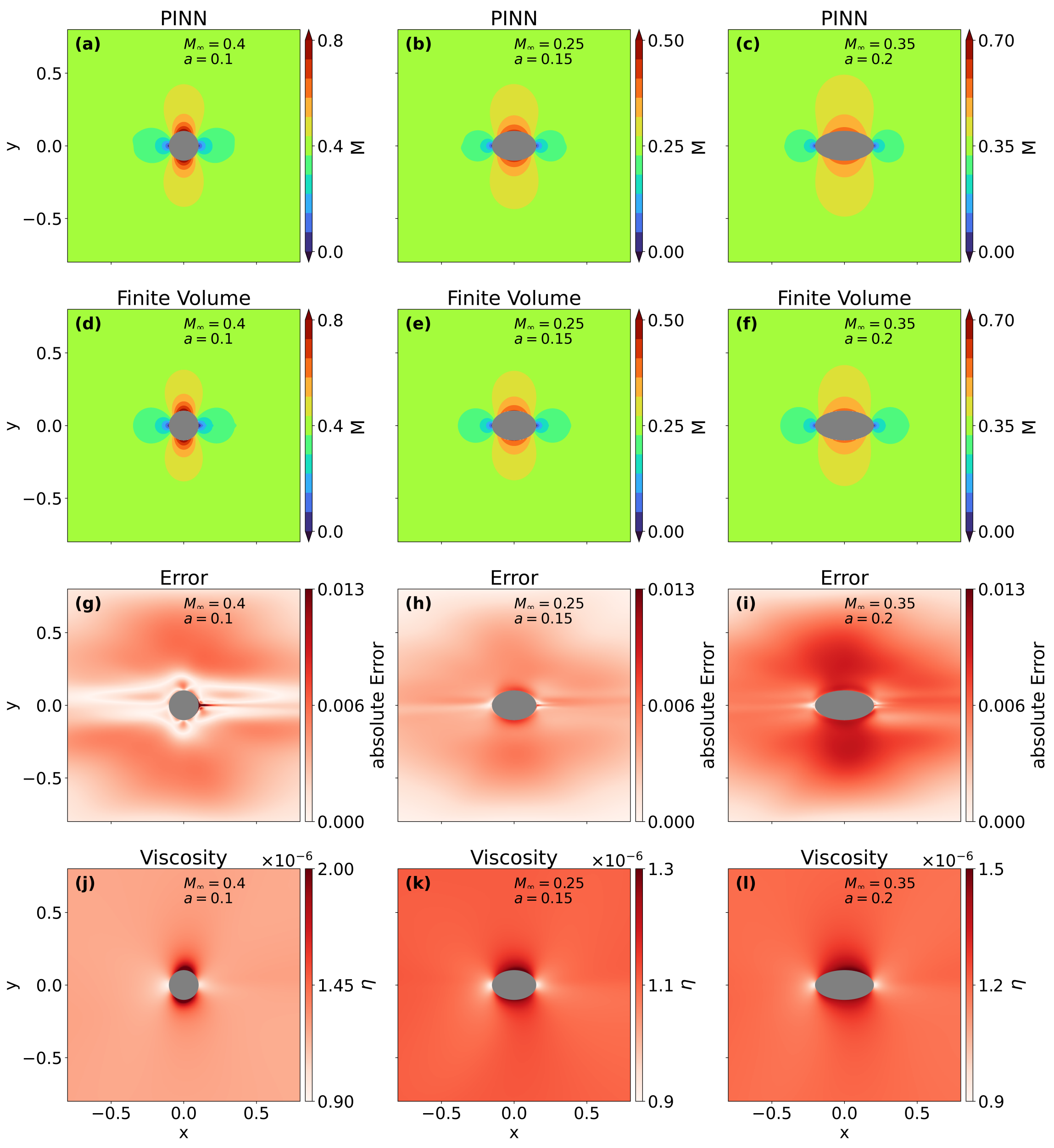


Figure 4.7.: Comparison between parametric PINN solution with adaptive activation functions and a reference finite volume result for different Mach numbers and ellipse eccentricities. The absolute errors between the reference and the PINN solution are shown in Figs. (c), (g) and (k). Figs. (d), (h) and (l), show the artificial viscosity η.

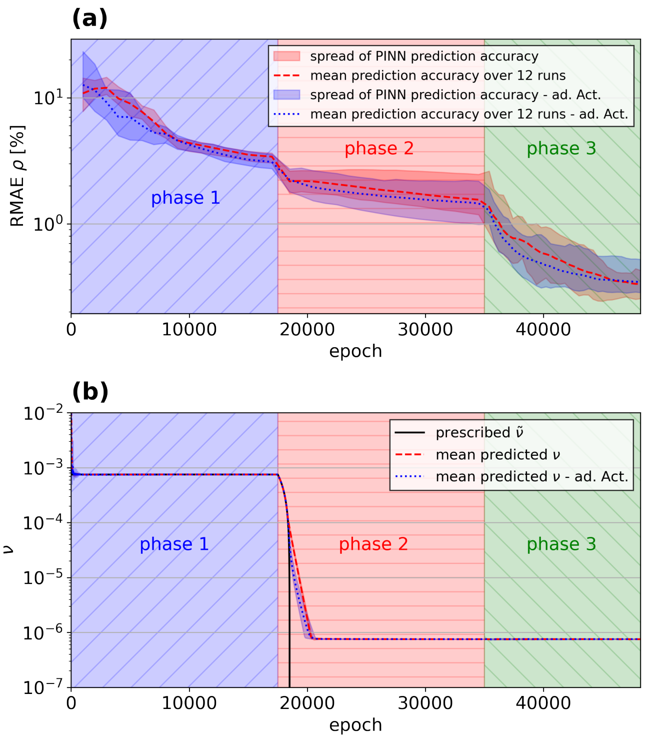


Figure 4.8.: *RMAE* for ρ in (a) and prescribed as well as predicted viscosity during training in (b) for flow around ellipse. The spread of the predictions for 12 random initializations of the network is indicated b the marked area around the lines.

(i.e. the spread of the predictions) is also indicated. While differences occur between different initializations during early training phases, all models converge to a similar final accuracy. Fig. 4.8 (b) shows the prescribed viscosity factor $\tilde{\nu}$ as well as the predicted mean viscosity $\nu(x, y)$ during training. Besides a spike in the first few epochs, the prescribed and predicted viscosity are identical during the first phase. The initially high viscosity helps to avoid convergence to an unphysical local minima of the loss. In phase 2 we see that the viscosity is comparatively quickly reduced. After about 2500 epochs, it remains at a constant value around $\nu = 10^{-6}$. This indicates that such viscosity values, significantly lower than the initial $\tilde{\nu}$, are sufficient for stabilizing the training and that ν can be reduced relatively fast during phase 2 without causing any instabilities, as no shocks are present in the solution. On average we only observe marginal improvements in accuracy and convergence speed, when using adaptive activation functions instead of the fixed hyperbolic tangent activations. For a quantitative assessment of the errors, we consider the pressure coefficient C_p, the local Mach number M and the density ρ for 100 quasi-random parameter values in the parameter space. For each parameter set, the mean absolute difference to the reference solution was calculated for a small square that contains the cylinder ($-0.5 < x < 0.5; -0.5 < y < 0.5$). Fig. 4.9 shows the resulting absolute errors for the density. These errors are then normalized with the range of values of the reference solution for each individual quantity to obtain the relative errors in Tab. 4.2 as the mean over all datasets and all 12 runs. The hyperparameters are

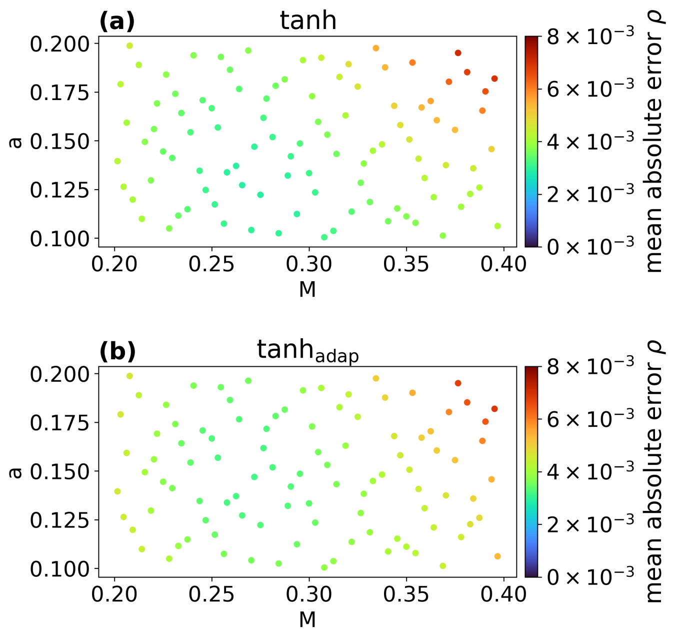


Figure 4.9.: Mean absolute errors on subsonic test case between parametric PINN and reference finite volume result for the density field, for different parameter sets. Fig. (a) shows the results without and Fig. (b) with adaptive tanh activation functions.

not optimized and higher accuracies may be possible when employing hyperparameter optimization e.g. on the network shape and learning rate.

The relative errors confirm that adaptive and non-adaptive activation functions perform very similarly for this example. We also see that the errors towards the borders of the parameter space are increased. Especially for the higher Mach numbers at $a \approx 0.2$, we see an increase in errors, as already observed in Fig. 4.7 Comparing the error metrics to the non-parametric model in Sec. 4.1, we see that especially for the pressure coefficient, the accuracy is significantly improved from $RMAE \approx 0.55\,\%$ to $RMAE \approx 0.24\,\%$ for the parametric model. Previously, we have observe that the point distribution near the aerodynamic object is highly relevant to obtain more accurate pressure fields near. Since the total number of points for the parametric model is significantly higher, this has likely contributed to the improved accuracy. For the local Mach number M we do however see a slight drop in accuracy from $RMAE = 0.81\,\%$ to $RMAE = 1\,\%$, as the Mach number prediction seems to be come more challenging at larger a (see Fig. 4.7). For all presented results, the Tensorflow backend of SMARTy [8] was used. With the Tensorflow backend the model is trained in 24 hours on a single NVIDIA A100 graphics card, which is about a four times increase compared to the non-parametric case. All the presented models were using double precision for the floating point numbers. Once trained, the evaluation of the model is fast. The prediction of the model at 300.000 points takes about 0.5 s.

Table 4.2.: Error metric for parametric model with standard and adaptive activation functions.

activation		tanh	$\tanh_{\text{adap}}$
RMAE [%]	C_p	0.24 ± 0.06	0.25 ± 0.08
	M	1.1 ± 0.1	1.0 ± 0.11
	ρ	0.33 ± 0.11	0.34 ± 0.13
MAE	C_p	$(6.9 \pm 1.3) \cdot 10^{-3}$	$(7.3 \pm 2.1) \cdot 10^{-3}$
	M	$(5.1 \pm 0.5) \cdot 10^{-3}$	$(5 \pm 0.5) \cdot 10^{-3}$
	ρ	$(4.1 \pm 0.9) \cdot 10^{-4}$	$(4.2 \pm 1) \cdot 10^{-4}$
$\boldsymbol{R_2}$	C_p	0.9982 ± 0.0006	0.9981 ± 0.0012
	M	0.9825 ± 0.0026	0.983 ± 0.004
	ρ	0.9972 ± 0.0014	0.9968 ± 0.0024

4.3. Summary

This chapter investigated the application of scalar artificial viscosity with adaptive viscosity localization for compressible flows around cylinders and ellipsoids. Initially, PINN models with and without AV were compared on a non-parametric problem, showing that even in the absence of shocks, AV can have a positive effect on the convergence of PINNs. When using AV during training, the convergence properties seem to be improved and failed simulation runs, which do not converge to reasonable approximations, are avoided. During the second and third training phase, the viscosity is reduced to negligible values, showing that no AV is required once a stable approximation of the solution is reached. This is due to the fact that in subsonic flows, no shock waves occur which need to be permanently stabilized with AV, due to the reasons outlined in Sec. 3.1. The training time for the model with AV was 7.5 hours, and the evaluation time was 0.5 seconds. We have also seen that errors are predominantly observed near the surface of the aerodynamic object. Especially for aerodynamic applications, the accuracy of the method on the surface of the aerodynamic object is crucial since the aerodynamic forces are calculated by integrating surface solutions. The errors near the surface can be alleviated to a certain point, by increasing the collocation point density near the surface, even though some non-negligible errors still remain. Therefore, in Ch. 6, the mesh transformation methodology (see Sec. 2.4.6) is adapted to improve the accuracy on defecting objects further.

In a a next step, we considered a parametrized version of the problem, with two parameters, the far-field Mach number M_∞ and the ellipse shape parameter a. The parametric model can achieve similar accuracies to the non-parametric model with training times of 24 hours and evaluation times of 0.5 seconds. The relative errors were less than 1%. Again, the errors were more pronounced near the ellipse surface and slightly increased towards higher M_∞ and a values. Models with and without adaptive hyperbolic tangent activations were also compared. The adaptive activation functions only showed marginal improvements in convergence speeds, over standard activations but the final model accuracy was on a similar level. Generally, we conclude that the parametric PINN is able to provide a good approximation of solutions for this problem, even when parametric object shapes and boundary conditions are being used, clearly showing the potential of this approach for multi-query scenarios. Since the computational time is only increased by a factor of four, compared to the non-parametric model, the use of a parametric model already becomes sensible when it is evaluated at more than four different M_∞-a-combinations.

5. Predicting Oblique Shocks

In the previous chapter, a PINN model with adaptive AV localization was used to simulate subsonic flows around a cylinder. The use of AV has initially been motivated in Ch. 3, as a methodology for facilitating the convergence of PINNs to shock solutions. For the compressible Euler equations, for smooth initial and boundary values, shocks can appear when the Mach number M exceeds the speed of sound c. When the far-field conditions are supersonic, (i.e. $M_\infty > 1$), the resulting supersonic shock waves are of crucial importance for the aerodynamic performance [1, p.29] and a numerical method needs to be able to capture these shocks precisely. In this chapter a typical benchmark problem featuring an oblique shock wave is investigated. The focus of this investigation is the model's ability to stabilize the shock and whether the proposed methodology is indeed able to provide accurate predictions, even when the solution features discontinuities. For the oblique shock, an inflow Mach number of $M_\infty = 2$ is selected. A schematic representation of the test case is shown in Fig. 5.1. The test case describes a scenario where a supersonic inflow is deflected by a wedge with deflection angle $\theta = 10°$. The resulting attached shock originates from the corner of the wedge. We define the shock angle with respect to the surface of the wedge as δ. The shock angle is a unique function of the deflection angle and the incoming Mach number $M_1 = M_\infty$ as stated by the θ-β-M relation. The field variables after the shock can be calculated analytically from M_1 and θ [1, Ch. 9.2]. This problem has also been analyzed with PINNs in [71] and [49].

The used PINN model is similar to the model, used in the previous Chapter. A schematic representation of the model is shown in Fig. 5.2. Again, a scalar AV (3.16) combined with the adaptive viscosity localization (Sec. 3.3.1) is used. As before, the network predicts the viscosity factor $\nu(x, y)$ alongside the primitive variable vector (ρ, u, v, E) and the viscosity magnitude is indirectly controlled with the penalty loss term (3.21) through the prescribed viscosity factor $\tilde{\nu}$. The sensor-controlled AV localization is not used in this chapter because the sensor is specifically designed for normal shock waves, where the shock line is directly orthogonal to the flow direction. The sensor uses the pressure gradient in flow direction to identify shocks. However, for oblique shocks, the flow direction is not orthogonal to the shock. The adaptive AV localization is more flexible in these cases and should be able to identify the need for AV regardless of the type of shock wave. As in the previous chapter, the prescribed viscosity factor is initially set to a value $\tilde{\nu} > 0$ at the start of training. In this way, the model initially approaches a viscous approximation of the inviscid solution. Subsequently $\tilde{\nu}$ is reduced to 0. The NN is then inclined to reduced the viscosity everywhere in the domain, except at locations where the loss of viscosity would lead to a rise in the other loss terms. Hence, some viscosity should in theory be maintained, at least around the shock.

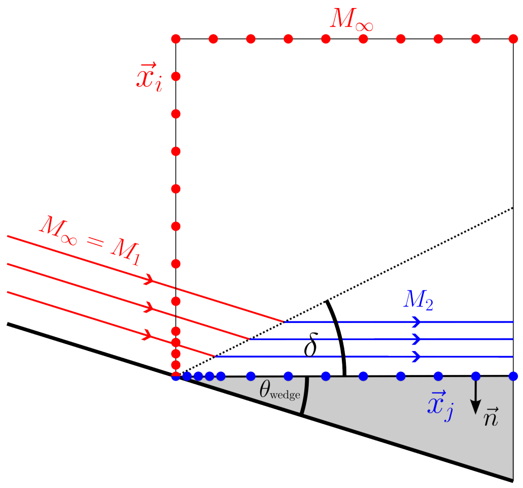


Figure 5.1.: Schematic representation of oblique shock problem. The red points are used for the Dirichlet boundaries and the blue points are used for the slip boundary.

Similarly to before, a parametric variant of the problem with a variable Mach number $M_\infty \in (2, 3)$ is also analyzed. According to the θ-β-M relation [1, Ch. 9.2], changes in M_∞ lead to a non-linear change in the shock angle. The precise localization of the shock position is crucial for aerodynamic simulation methods, because shocks significantly impact the aerodynamic performance. Hence, the analyzed parametric problem represents an important sanity check for applications of PINNs in aerodynamics.

Furthermore, adaptive activation functions are again investigated. In the previous chapter, the standard and adaptive activations delivered similar performance for subsonic problems. However, when dealing with shocks, drastic changes in the flow field are possible at short distances. Thus, it is possible that activation functions with variable slope can better adapt to these large derivatives in the flow field, by increasing the slope of the activation.

Note that parts of this chapter are based on [72].

5.1. Forward Problem

In a first step, the non-parametric oblique shock problem (see Fig. 5.1) for a Mach number of $M_\infty = 2$ is analyzed. For the collocation points for the residual loss $\mathcal{L}_\text{R}$ in Eq. (2.84), the uniform sampling strategy (see Sec. 2.4.5) is used. We generate $N_\text{R} = 5000$ uniformly distributed training points inside of the domain, using the Halton sequence [110]. Note that, in contrast to other work on this problem, no clustered training points [71] or domain decomposition [49] is used. This is advantageous, because no previous knowledge of the solution is required for the point generation or for the decomposition. Far-field boundary conditions are applied at the top and left surface. Slip wall boundary conditions are used for the bottom boundary as introduced in Sec. 2.4.2. Since the shock originates from the bottom left corner, the boundary point density for $x \in (0, 0.1)$ and $y \in (0, 0.1)$ is increased, as sketched in Fig. 5.1. A total of 1500 points are used for the boundary conditions. As before, the three phase training routine (Sec. 3.3.1) is used. For the first

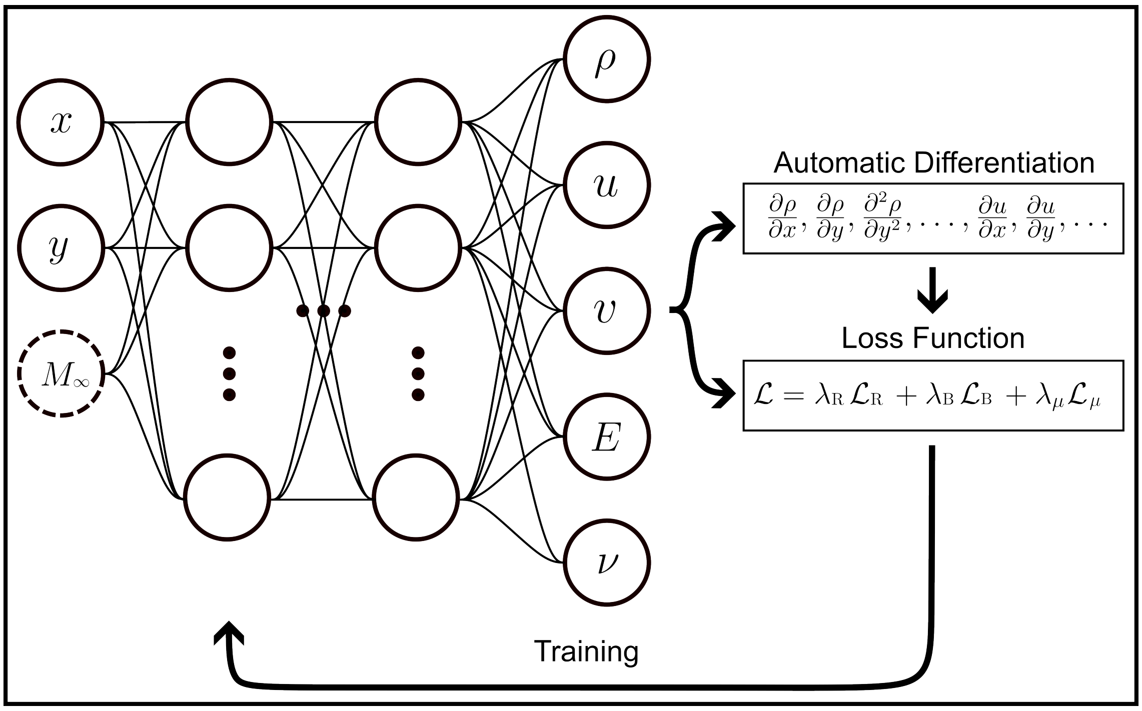

x
y
M_∞
ρ
u
v
E
ν
Automatic Differentiation
$\frac{\partial \rho}{\partial x}, \frac{\partial \rho}{\partial y}, \frac{\partial^2 \rho}{\partial y^2}, \dots, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \dots$
Loss Function
$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_B \mathcal{L}_B + \lambda_\mu \mathcal{L}_\mu$
Training

Figure 5.2.: Schmetic representation of PINN model for oblique shock problem. The input parameter M_∞ is only used for the parametric model in Sec. 5.2

and second training phase, the ADAM optimizer is used and in the last training phase an L-BFGS optimizer 2.3 is used instead. The loss term weights are $\lambda_B = 1, \lambda_R = 1$ and $\lambda_\mu = 5$. A summary of all hyperparameters is shown in A.2, including the used learning rates, network dimensions and number of training epochs.

To evaluate the efficacy of AV for supersonic problems, the PINN prediction of the model with adaptive AV to an otherwise identical model without AV is compared. The resulting predictions are presented in Fig. 5.3 in comparison to the analytical reference [1, Ch. 9.2]. The first row shows the predictions for the pressure coefficient C_p, the second row the density ρ and the last row the local Mach number M. We see that the PINN prediction with AV is visually indistinguishable from the reference in all field variables. For the PINN model without AV the shock is however smoothed out and we see some additional inconsistencies in the bottom left at the origin of the shock. As suspected, based on Sec. 3.1, the standard PINN model is unable to converge to the shock solution, as it is not a minimum of the standard L_2-based residual loss. Evidently, the PINN model with AV is indeed able to circumvent this issue and provides an accurate approximation of the shock.

Fig. 5.4 shows the absolute errors of the PINN prediction with AV in comparison to the analytical reference for the same three field variables. Generally we see slightly lower errors before an slightly higher errors behind the shock, where the new constant states of the flow variables are to be predicted, which are not identical to the Dirichlet boundary condition. The shock angle almost perfectly matches the reference and is consistent between all field variables Despite the addition of AV, the shock is predicted very sharply,

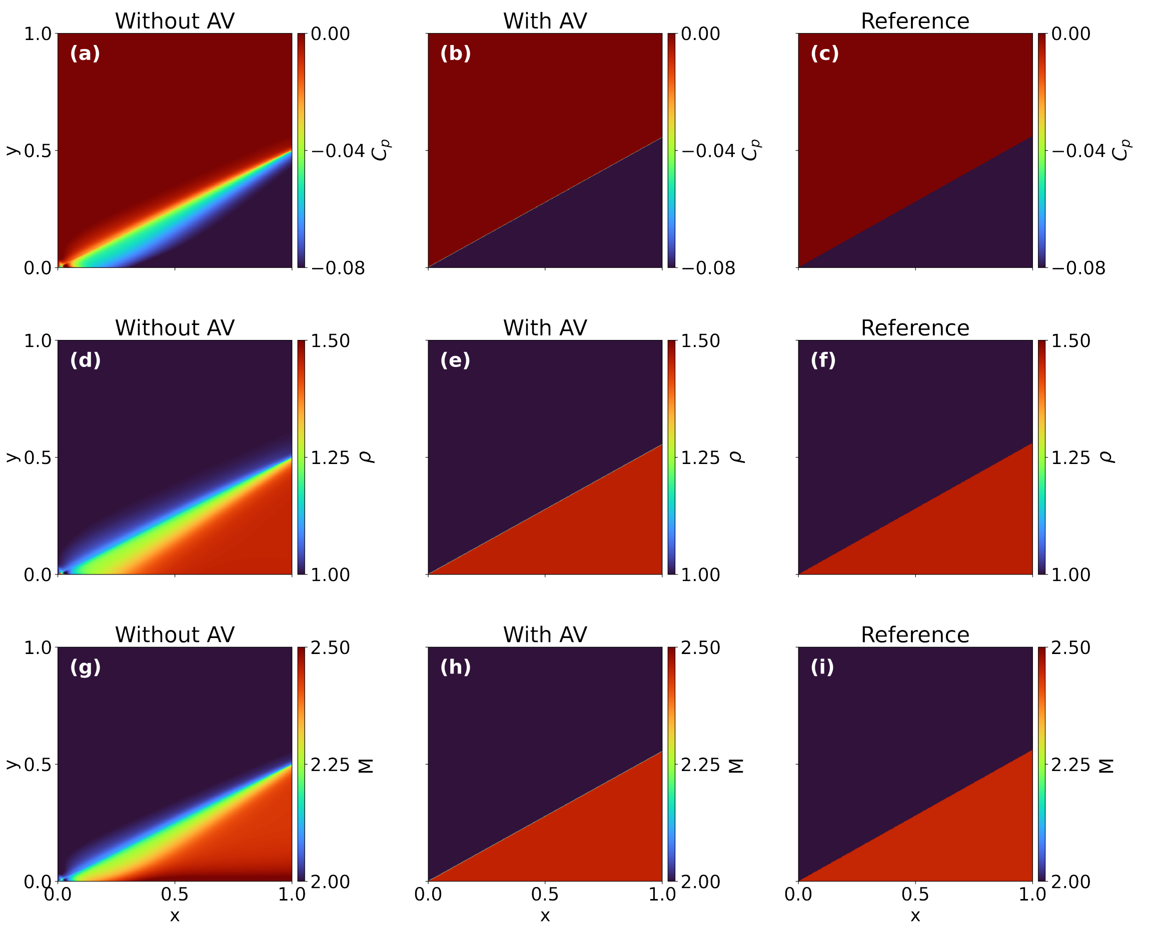


Figure 5.3.: Comparison of PINN prediction for oblique shock problem without AV (left column), with AV (center column) and the analytical reference solution (right column). The first row shows the pressure coefficient C_p, the second row the density ρ and the third row shows the local Mach number M.

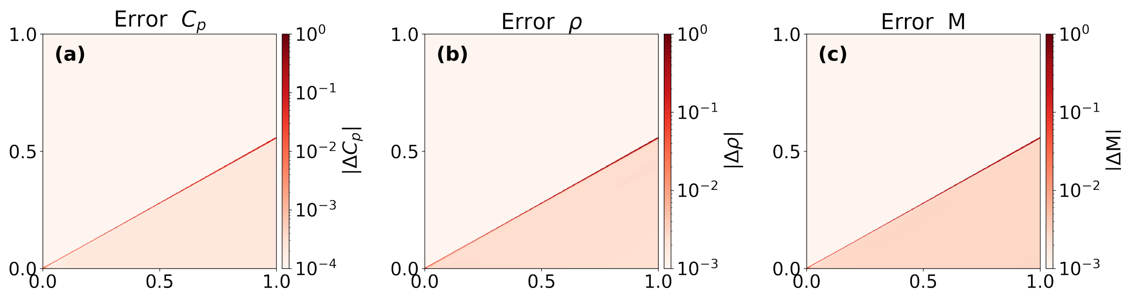


Figure 5.4.: Comparison of absolute PINN prediction errors for oblique shock problem for pressure coefficient C_p, density ρ and local Mach number M.

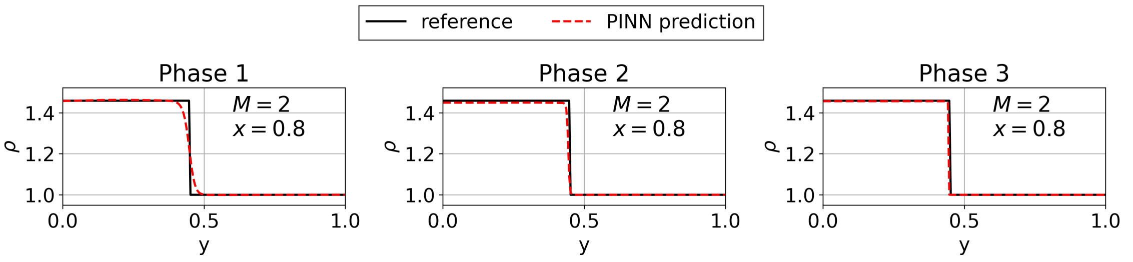


Figure 5.5.: Section through density field of non-parametric PINN prediction for the oblique shock problem at $x = 0.8$ after each of the three training phases.

which is especially obvious when considering a cross section through the flow. This is the case due to the reduction phase in the training procedure. Fig. 5.5 shows a section through the density field at $x = 0.8$ at the end of each of the three training phases. At the end of phase one, we see that the shock is still significantly smoothed, because the prescribed viscosity factor $\tilde{\nu}$ is not reduced yet. After phase two and especially after phase three, this smoothing has mostly disappeared and the shock is resolved very sharply, since the model was able to find an AV distribution which minimizes the viscous effects while keeping the shock stable during training.

For the calculation of the error metrics Eqs. (2.107)-(2.109), the field variables are evaluated at uniformly distributed points in the entire domain for four training runs. The results are shown in Tab. 5.1 where the error bounds indicate the standard deviation over the four training runs. The error metrics clearly show that the PINN with AV outperforms the variant without AV in all variables with relative errors of around 1% in all field variables. In comparison, the models without AV perform consistently worse, with absolute errors which are about an order of magnitude larger. The total training time of the model amounts to around 5 h on a single NVIDIA-A100 graphics card and the model can be evaluated in about 0.5 s. The final predicted viscosity is comparatively constant in the entire domain and reaches values of $\nu(x, y) \approx 1.5 \cdot 10^{-5}$. In comparison to the subsonic problem, the final viscosity is significantly larger as it is required to stabilize the shock.

Overall, it can be concluded that the PINN model with adaptive AV localization and a scalar AV variant is able to provide accurate predictions for the oblique shock problem. The shock angle is predicted accurately in all field variables. In addition, the shock is also resolved quite accurately and the shock is not significantly smoothed out by the remaining viscosity at the end of training. This clearly demonstrates that the three-phase training procedure with the reduction of the prescribed viscosity value $\tilde{\nu}$ can reliably stabilize the shocks at the end of training, while minimizing the smoothing effects of the added dissipative term. An identical PINN model without AV was unable to provide satisfactory predictions. This is strong evidence, that the theoretical argument for the failure of standard PINNs to predict shock waves also applies to the more complex Euler equations and that AV is an effective resolution of this issue.

Table 5.1.: Error metrics for non-parametric oblique shock model.

α [°]	$\boldsymbol{RMAE}$ [%]		$\boldsymbol{MAE}$		$\boldsymbol{R_2}$	
no AV	C_p	8.2 ± 0.8	C_p	$(6.9 \pm 0.6) \cdot 10^{-3}$	C_p	0.84 ± 0.02
	M	8.3 ± 0.7	M	$(4 \pm 0.4) \cdot 10^{-2}$	M	0.82 ± 0.02
	ρ	7.3 ± 1.4	ρ	$(3.3 \pm 0.7) \cdot 10^{-2}$	ρ	0.83 ± 0.04
with AV	C_p	0.8 ± 0.4	C_p	$(6 \pm 4) \cdot 10^{-4}$	C_p	0.97 ± 0.02
	M	0.9 ± 0.5	M	$(4 \pm 2) \cdot 10^{-3}$	M	0.97 ± 0.02
	ρ	0.7 ± 0.5	ρ	$(3.1 \pm 2.1) \cdot 10^{-3}$	ρ	0.97 ± 0.02

5.2. Parametric Forward Problem

In this section, the previous oblique shock problem is solved in a parametric fashion with a variable Mach number $M_\infty \in [2, 3]$. We use a total of 100000 points for the evaluation of the residual and the viscosity penalty loss. 80000 of these points are uniformly sampled in the three-dimensional input space $(x, y, M_\infty) \in \Omega \times [2, 3] = [0, 1] \times [0, 1] \times [2, 3]$. An additional 15.000 points are uniformly sampled on the upper $(x, y, M_\infty) \in \Omega \times \{3\}$ and lower bound $(x, y, M_\infty) \in \Omega \times \{2\}$ of the parameter space to enhance the accuracy towards the borders. The Halton sequence [110] is used for generating all three point sets. As before, Dirichlet boundary conditions are applied at the top and left boundary and slip boundary conditions are used for the bottom boundary as explained in Sec. 2.4.2. In addition, the boundary point density for $x \in (0, 0.1)$ and $y \in (0, 0.1)$ is again increased as sketched in Fig. 5.1. An initial prescribed viscosity factor of $\tilde{\nu} = 7.5 \cdot 10^{-4}$ is used. The network consists of 7 layers with layers of 30 neurons and tanh activations. Again, fixed and adaptive hyperbolic tangent activation functions are compared , but the trainable parameters of the activations in phase 3 are frozen as explained in Sec. 4.2. The loss term weights are $\lambda_\text{B} = 1, \lambda_\text{R} = 1$ and $\lambda_\mu = 5$. A detailed overview of training parameters is shown in Appendix A.2.

Fig. 5.6 provides an overview of the result for three different Mach numbers in comparison to the analytical solution. The field values before and after the shock are accurately predicted and the shock is well resolved. Slight inaccuracies in the shock angle are visible. The bottom row shows the artificial viscosity. Due to the local adaptivity, the dissipation is increased close to the shock. This shows that the proposed method is able to locally identify regions which require additional viscosity for stabilization, similarly to the analytical sensor function. Since PINNs can perform inconsistently, depending on the random initialization of network parameters at the start of the training, we analyze the accuracy over 12 training runs with different random initialization seeds. The following plots highlight the mean prediction over those 12 runs and the largest and smallest values (i.e. the spread of the predictions). As an integral indicator of prediction accuracy the shock angle δ (see Fig. 5.1) is considered. Fig. 5.9 shows an overview of predicted shock angles for the entire parameter space. Overall the error of the shock angle is lower than one degree. The errors are larger at the lower bound of the parameter space towards $M_\infty = 2$. The spread of the predictions over the 4 training runs indicates that the results are generally within a one degree neighborhood of the analytical solution both with and

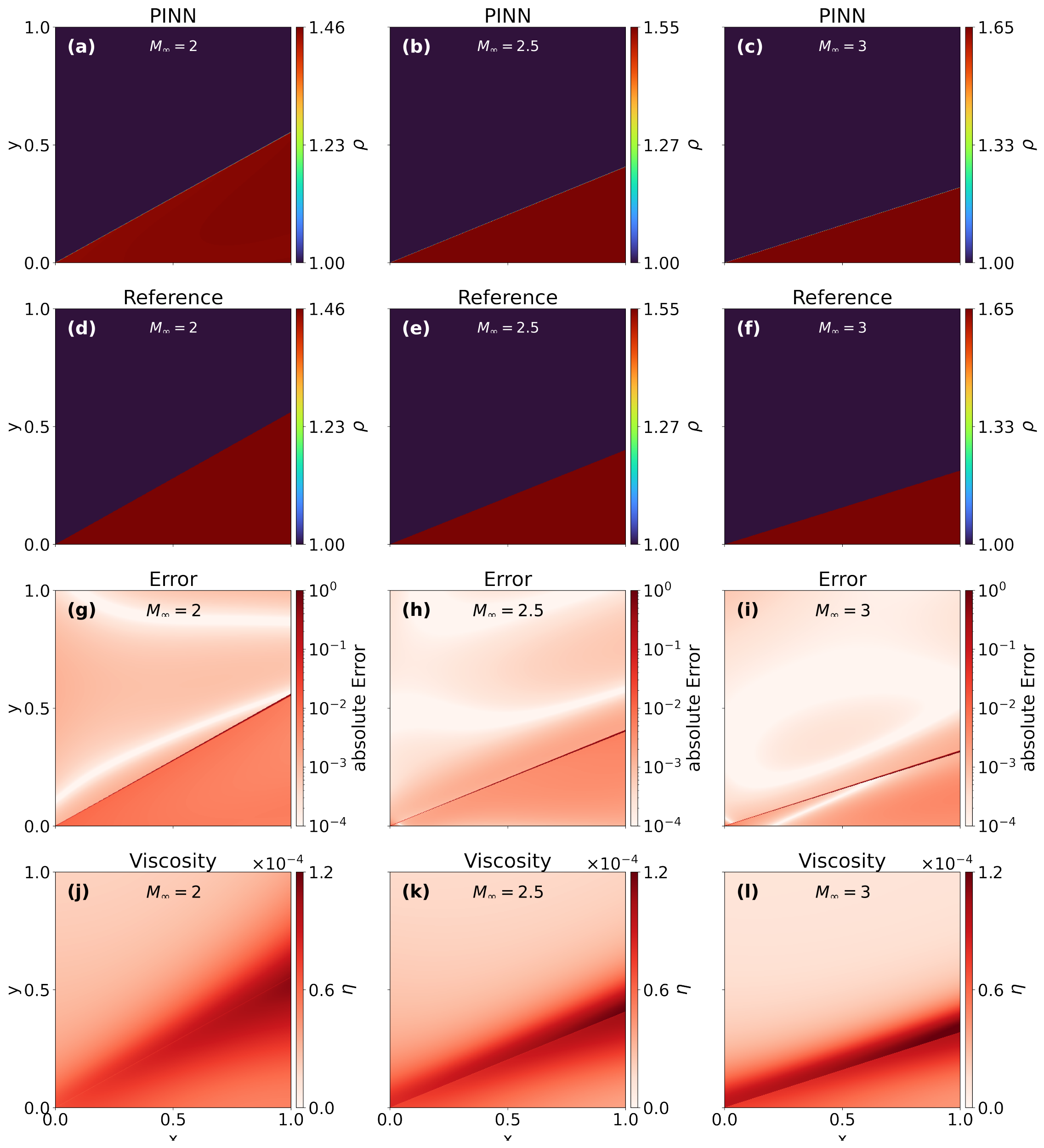


Figure 5.6.: Comparison of parametric PINN solution using adaptive activation functions for oblique shock test case with the analytical reference solution for different Mach numbers. The absolute errors between the reference and the PINN solution are shown in Figs. (c), (g) and (k). Figs. (d), (h) and (l), show the artificial viscosity η.

without adaptive activation functions. In this example, one can see that the adaptive activation functions outperform the fixed ones. Fig. 5.7 (a) shows how the error for the angle delta changes during the training.

We see that the error in the angle does not improve significantly after phase 1 and that the spread of predicted angles in fact increases. However, Fig. 5.8 shows that during the second and third training phase, the shock becomes much sharper and thus approximates the expected analytical result better. This is also confirmed by the decrease in the relative density errors (see Fig. 5.7 (b)). Note that the final training phase 3 with L-BFGS is crucial to obtain good accuracy. The error history shows that on average, the adaptive activation functions can accelerate convergence in phase 1 and 2. In phase 3 the most inaccurate runs with and without adaptive activations are very similar, while the most accurate runs are improved for the adaptive activations. Fig. 5.7 (c) shows the prescribed viscosity value $\tilde{\nu}$ and the mean (over the domain) predicted viscosity ν during the training. Contrary to the subsonic test case we can clearly see the adaptivity of viscosity and how it is only loosely coupled to the prescribed value. Early in the training an increase beyond the prescribed value is accompanied by a fast convergence during the first few thousand epochs. Then, during phase 2, the predicted viscosity is again reduced less steeply than the prescribed value which indicates that more viscosity is necessary to stabilize the training during the reduction phase. During phase 3 the predicted viscosity decreases more steeply. The final values are between $\nu = 2 \cdot 10^{-5}$ and $\nu = 2 \cdot 10^{-6}$ and thus higher than for the subsonic test case. The viscosity for the adaptive activations is on average slightly increased. The necessity for more viscosity during the training is to be expected since we are dealing with a supersonic flow that involves a shock. The additional AV is required to stabilize the shock since the residual loss can otherwise not be minimized around the shock, as explained in Sec. 3.1.

For a quantitative comparison of relative errors with the subsonic problem, the pressure coefficient C_p, the local Mach number M and the density ρ for 20 uniformly distributed Mach number values with $M \in [2, 3]$ are considered. For each parameter, the mean absolute difference to the reference solution was calculated for the entire domain. This difference was then normalized with the range of values of the reference solution for each individual quantity. For the calculation of the error metrics (Tab. 5.2), the prediction for 12 different runs is compared to the analytical reference for uniformly distributed points in the entire domain Ω. The uncertainty bounds take both the local variance in errors in the domain as well as the variance between the different runs into account.

The hyperparameters are not optimized and higher accuracies may be possible when employing hyperparameter optimization e.g. on the network shape and learning rate. The use of adaptive activations generally leads to improved absolute accuracies, especially for the Mach number M. Compared to the subsonic problem 4.2, we see slightly increased errors within the same order of magnitude. The bounds of the error are higher, mainly due to the fact that accuracies are worse close to the lower Mach number $M = 2$, while the errors of the subsonic problem are consistently low for the entire parameter space.

Again, the models are implemented with SMARTy [8] using the tensorflow backend and the total training time for one run on a NVIDIA A100 graphics card is about 23 hours.

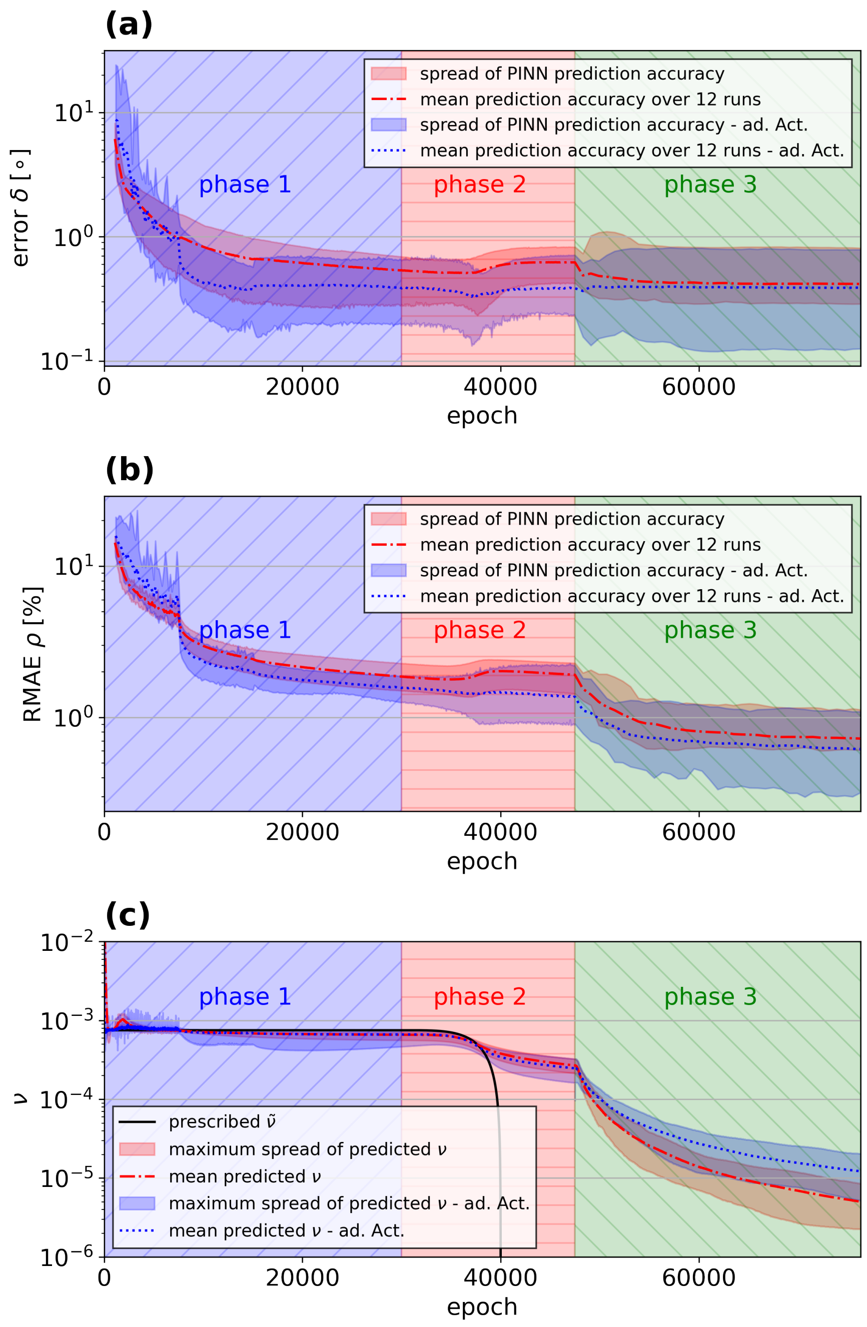


Figure 5.7.: Error history and artificial viscosity factor ν during training of PINN for parametric oblique shock problem. The three phases of the applied training procedure are highlighted.

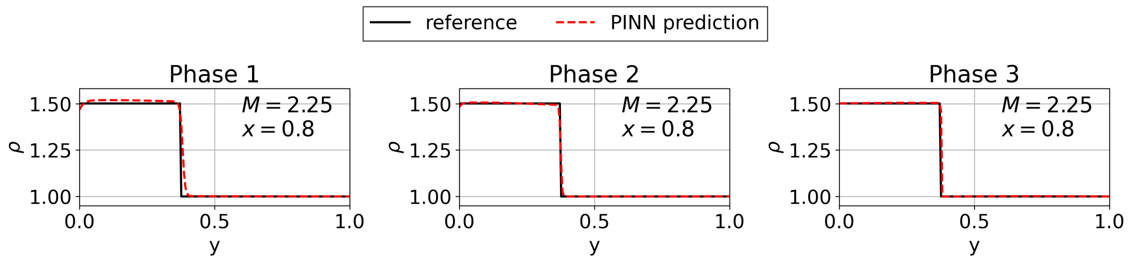


Figure 5.8.: Exemplary cross-section through density field at $M_\infty = 2.25$ shows how shock becomes less dissipative after reducing the viscosity during the training. Results were obtained using the adaptive activation functions.

Table 5.2.: Error metric for parametric model with standard and adaptive activation functions.

activation		tanh	$\tanh_{\text{adap}}$
***RMAE* [%]**	C_p	(0.8 ± 0.7)	(0.6 ± 0.7)
	M	(1.3 ± 0.9)	(0.8 ± 0.7)
	ρ	(0.7 ± 0.7)	(0.6 ± 0.6)
MAE	C_p	$(1 \pm 0.8) \cdot 10^{-3}$	$(0.72 \pm 0.9) \cdot 10^{-3}$
	M	$(6 \pm 4) \cdot 10^{-3}$	$(3 \pm 2.9) \cdot 10^{-3}$
	ρ	$(4 \pm 4) \cdot 10^{-3}$	$(3 \pm 4) \cdot 10^{-3}$
$\boldsymbol{R_2}$	C_p	0.977 ± 0.011	0.976 ± 0.011
	M	0.976 ± 0.011	0.976 ± 0.011
	ρ	0.977 ± 0.011	0.976 ± 0.011

The prediction at 300000 points takes about 0.5 s. Overall it can be conclude that the parametric PINN model can again provide similarly accurate predictions as the non-parametric model. It provides a good approximation of the oblique shock and the entire flow field for a wide range of Mach numbers at a moderate increase of computational effort, compared to the non-parametric model.

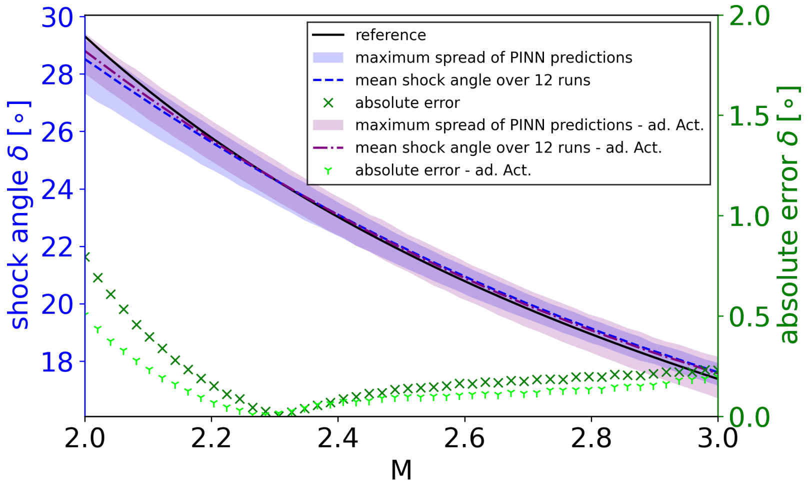


Figure 5.9.: Mean and spread of predicted shock angles. The blue curves show the angle δ (for the definition see Fig. 5.1). The green curve shows the absolute error which is the absolute difference between the two blue curves.

5.3. Summary

This chapter investigated the application of scalar artificial viscosity with adaptive viscosity localization on the oblique shock test problems. The PINN only provided reasonable predictions with the introduction of the adaptive AV while without AV, the model fails to correctly approximate the shock at the corner of the wedge where the shock originates. During training, it can be observed that the adaptive AV methodology adjust the viscosity magnitude. Especially during the second and third training phase, one can see how the reduction in AV leads to a better localization and a more sharply resolved shock. At the end of training, AV was maintained, as it was required to stabilize the shocks. The training time was 7 hours, and the evaluation time was 0.5 seconds. A parametric version of the oblique shock problem was also investigated with a variable far-field Mach number, achieving similar accuracies as the non-parametric model. Notably, the neural network was able to localize the shock location and apply AV in its vicinity. The training time for this case was 23 hours, and the evaluation time was 0.5 seconds. Overall, this chapter demonstrated the effectiveness of adaptive AV in improving the accuracy and robustness of PINNs for simulating supersonic flows governed by the compressible Euler equations.

6. Predicting Flows around Airfoils

In the previous chapter, a PINN model with scalar adaptive AV was used to successfully approximate a supersonic oblique shock. In this chapter, more advanced problems are considered, where the approaching flow is deflected by an airfoil. We already saw for the flow around the cylinder that errors in the prediction are mostly observed directly on the surface of the aerodynamic object. This problem becomes more severe when dealing with geometries such as airfoils. Airfoils typically feature a high curvature at the leading edge and a sharp trailing edge. To better approximate the flow around such a geometry, we adapt the mesh transformation (MT) methodology 2.4.6, which can effectively improve the prediction accuracy of PINNs for non-trivial geometries and near such surfaces, as shown in [86]. In the first part of this chapter, the mesh-transformation methodology is analyzed for subsonic flow problems, confirming its efficacy. Another point of interest is the influence of AV for these slower velocities. In the second part of the chapter flows at higher velocities, only slightly below the speed of sound, are considered. When the flow is then deflected by an object, the velocity can locally exceed the speed of sound. In this case one speaks of transonic flows. Transonic flows often feature normal shock waves that are characterized by sudden changes in the field variables in flow direction. These types of shocks are highly relevant in aerodynamics. Today's transport aircraft typically fly at transonic speeds, where normal shock waves form for example in the low pressure regions on the suction side of the airfoil. Hence, these shocks significantly impact the forces and moments generated by the flow. A suitable simulation method needs to be able to capture the location and magnitude of the shocks accurately. While the adaptive AV method has shown to be effective for simple problems, for transonic flows, the shock waves are only present near aerodynamic objects and the application of AV in other parts of the domain could negatively influence the convergence and accuracy of the model. Hence, for the following problem, the sensor-controlled AV is utilized. Furthermore, the energy and density AV and the matrix-valued AV, introduced in Sec. 3.3, are analyzed to confirm whether they can help to further improve the consistency and accuracy of the PINN model.

A schematic overview of the model architecture is shown in Fig. 6.1. To solve the compressible Euler equations (see Sec. 2.1.3), the model is trained at the grid points (x, y) in the physical domain. This corresponds to strategy 3 in Sec. 2.4.5. Each point can be identified with a corresponding point in the computational domain (ξ, η) using the MT methodology (see Sec. 2.4.6). The points (ξ, η) are passed through a random Fourier-feature embedding layer (Sec. 2.3). In Appendix B.1, the efficacy of the Fourier embedding is further analyzed. The output is passed through multiple fully connected, trainable layers. Since we have seen favorable performance using adaptive activations (Sec. 2.3) in the previous chapter, they are also used for this model. In addition the weight normaliza-

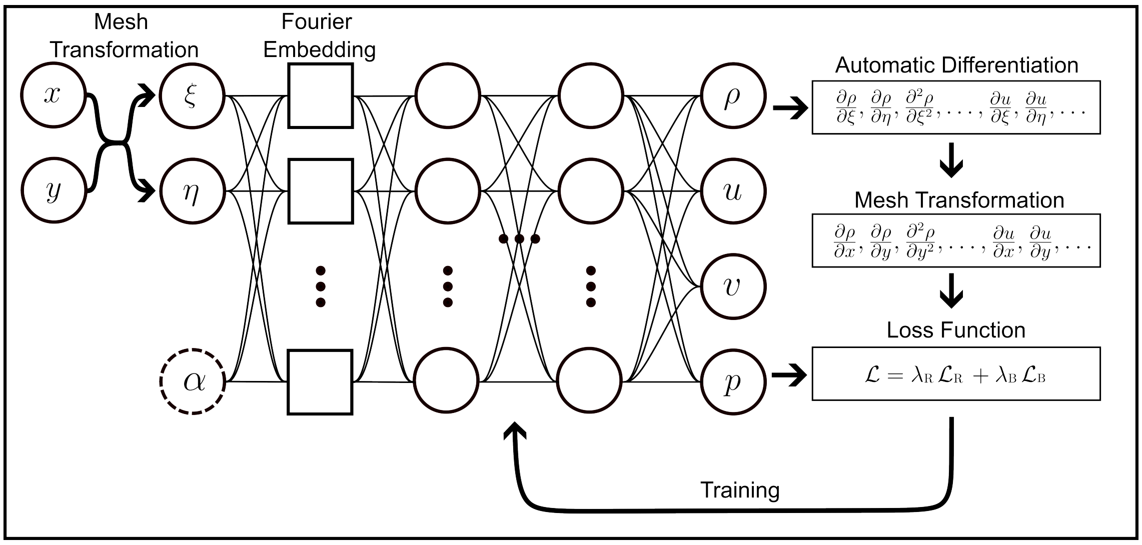

Mesh Transformation
Fourier Embedding
x
y
ξ
η
α
ρ
u
v
p
Automatic Differentiation
$\frac{\partial\rho}{\partial\xi}, \frac{\partial\rho}{\partial\eta}, \frac{\partial^2\rho}{\partial\xi^2}, \dots, \frac{\partial u}{\partial\xi}, \frac{\partial u}{\partial\eta}, \dots$
Mesh Transformation
$\frac{\partial\rho}{\partial x}, \frac{\partial\rho}{\partial y}, \frac{\partial^2\rho}{\partial y^2}, \dots, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \dots$
Loss Function
$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_B \mathcal{L}_B$
Training

Figure 6.1.: Schematic description of NN model for the prediction of transonic flows a

tion (Sec. 2.3) is used to further enhance the convergence properties of the model. The final output of the network is the vector of the predicted primitive variables (ρ, u, v, p). Next, one can use automatic differentiation to calculate the derivatives of these predicted variables with respect to the network inputs. Since no analytic expression for the MT is available, one can only calculate the derivatives with respect to the coordinates of the computational domain ξ and η. However, as explained in Sec. 2.4.6, the derivatives with respect to the physical coordinates can be reconstructed. These derivatives and the predicted variables themselves are used to calculate the loss function, defined in Sec. 2.4.1. As explained in Sec. 3.3 the Euler equations are modified by adding an additional viscous term (3.15). The term is only active in certain regions of the flow, determined by the sensor function, as described in Sec. 3.3.2. This sensor is crucial for accurately predicting the flow in the transonic regime. Finally, one can use the backpropagation algorithm to optimize the parameters in the trainable layers by minimizing the loss function, thus training the NN to predict the solution of the PDE. Additional parameters of the PDE can be added to the input space of the NN alongside (ξ, η). The PINN is then trained in a range of parameter values. The trained model can almost instantaneously predict solutions in the analyzed parameter range.

In this case, the PyTorch [22] backend of SMARTy [8] is used, which provides crucial parts of the model and the training algorithms, as well as a straightforward calculation of the derivatives with automatic differentiation. Furthermore, the PyTorch implementation of the low memory Broyden-Fletcher-Goldfarb–Shanno algorithm (L-BFGS), with a maximum number of 1000 inner iterations, is selected as the optimizer. It is used in a mini-batch fashion. However, since the PyTorch implementation of L-BFGS is not natively suited for stochastic loss functions, the optimizer is reset after each outer iteration to avoid complications with the approximation of the Hessian as the underlying data changes for each batch. Note that the training time can vary between runs, since the inner iterations are stopped when the convergence criteria are reached. All models use single floating point precision. No significant changes were observed when using double precision.

The implementation of the MT is heavily inspired by the code provided by Cao et al. [86]. Similarly to them, the elliptical grids are generated, using a finite difference solver for the grid point locations as described in [2, pp. 192-200]. For the PINN, we use an O-type grid with 400×200 points. The grid is shown in Fig. 6.2. Compared to [86], where a

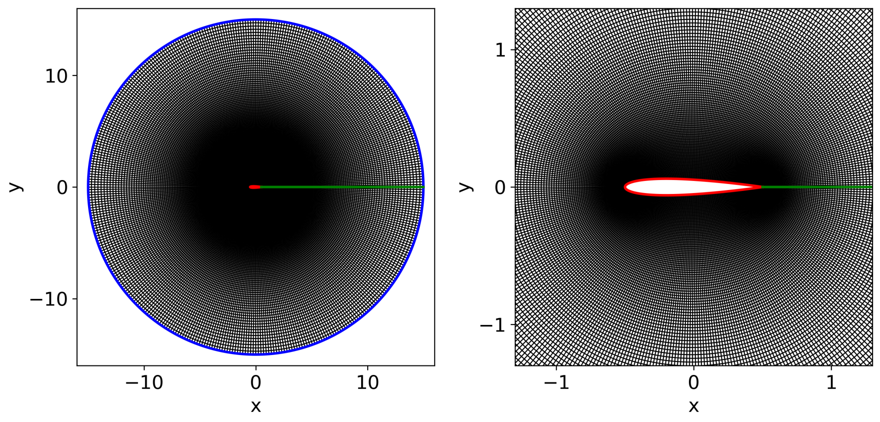


Figure 6.2.: The used curvilinear grid. The left image shows the entire grid. The right image shows a close-up of the airfoil. The boundaries of the computational grid are color coded, similarly to Fig. 2.12

200×100 point grid is used, we have seen that when shocks are being approximated the method performs more consistently, when a higher resolution grid is used. The inverse metric terms (Sec. A.1) are calculated using higher order finite differences [121]. The second order mesh metrics were calculated based on the relations derived in [114]. The calculation of the second order derivatives is significantly more costly than for the first order derivatives. The second order derivatives are however only required in regions where viscosity is applied i.e. where $s > 0$. Therefore, all all first order derivatives required for the loss function are calculated first for all grid points. This includes the pressure gradients which are required to calculate the sensor function. Once the sensor function (Eq. (3.26)) is calculated, one can check at which grid points the sensor exceeds a threshold of 10^{-4}. The viscous terms are only calculated for these points and set to 0 for all other grid points, thus reducing the overall computational cost per training iteration. Note also that, the sensor function $s(x, y)$ is detached from the computational graph, meaning that the sensor function does not influence $\nabla_\theta \mathcal{L}$. This is due to the fact that the PINN should learn the solution to the inviscid PDEs and not minimize for the best sensor position to lower the loss function. The sensor position should be solely determined by the location of the shock according to the PDEs themselves. The effect of not detaching the sensor is further investigated in Appendix B.1.

For the calculation of the loss function, the weighting of the residuals based on the cell volume is employed, as proposed by Song et al. [122]. To do that, one can modify the loss function as explained in Sec. 2.4.7, multiplying each term in the sum with a weighting factor, which increases the weight of bigger cells on the loss to avoid a bias towards high point density areas in the domain. The combination of the MT with the volume weighting leads to a residual loss term which is typically orders of magnitude lower than the boundary loss term. Therefore, a large constant scaling of $\lambda_{\mathrm{R}} = 2 \cdot 10^4$ (see Eq. (2.84)) is applied. The selection of this large weighting factor is further illustrated in Appendix B.2. In Appendix B.1 an empirical ablation study was performed, where the model performance with and without dynamic loss term weighting is compared, showing no significant benefits of the weighting algorithm for this particular test case. Hence, in the following, no dynamic weighting is used. In the following the performance of the described model under subsonic conditions is investigated first. The focus is shifted onto transonic conditions, in the following section. Note that this chapter is partially reproduced from [119], with the permission of AIP Publishing.

6.1. Subsonic Conditions

The efficacy of the mesh-transformation-based PINN methodology for subsonic problem has already been confirmed by Cao et al. [86]. Under subsonic flow conditions, the authors have observed satisfactory results without using any artificial viscosity in their model.

Initially, we shall analyze the significance of the MT. Two PINN models without any AV are compared. Model (i) is only trained on the nodes of the grid in Ω without using the mesh-transformation. Model (ii) used the mesh-transformation and is therefore trained in the computational domain Σ. In Ch. 3, AV methods for stabilizing shock waves were introduced. While no shock waves occur under subsonic conditions, it needs to be ensured that the AV methodologies do not negatively influence the PINN performance. Hence, PINN models with enabled AV are also investigated. Model (iii) uses a scalar AV combined with the shock sensor s_{shock} for the viscosity localization but with a disabled stagnation point sensor ($s_{\mathrm{stag}} = 1$). Model (iv) uses the full sensor function with both the shock sensor and the stagnation point sensor enabled. Similarly, model (v) and model (vi) make use of the full shock sensor, but it is combined with the density and energy and matrix-valued AV, respectively. For all models with AV, a grid search is preformed to obtain an optimal value for ν. The used values are shown in Tab. 6.1 and are in a similar range of $\nu \in [1.8, 2.1] \cdot 10^{-3}$. Generally, for all ν values in this range, the accuracy of the different models does not change significantly. However, since the produced viscosity differs between the models, it is reasonable to choose an optimal values of ν to ensure a fair comparison. For practical use, a value of $\nu = 1.9 \cdot 10^{-3}$ is a sensible starting point as it has shown to be similarly effective for all cases analyzed in this work. A summary of the different model setups is shown in Tab. 6.1. Four different subsonic flow conditions are considered, covering a wide range of Mach numbers and angles of attack, as summarized in Tab. 6.2.

As the first problem, consider case (a) with a comparatively high angle of attack of $\alpha = 5^\circ$ and a Mach number of $M_\infty = 0.3$. The pressure coefficient C_p for different models, is shown Fig. 6.3. The pressure field for the reference solution, obtained with a finite volume solver (see Appendix D), is shown in Fig. 6.3 (a). In Fig. 6.3 (b), we see the absolute difference $|\Delta C_p|$ between the reference pressure and model (i) which does not use the MT. It is evident that the model is unable to even remotely follow the trend of the reference. During the training for this model, we observe that the optimizer is unable to find a

Table 6.1.: Overview of analyzed models for subsonic problems.

model	MT	s_{stag}	s_{shock}	scalar AV	ρ-E AV	matrix AV	$\nu \cdot 10^{-3}$
(i)							0
(ii)	✓						0
(iii)	✓		✓	✓			2.1
(iv)	✓	✓	✓	✓			2.1
(v)	✓	✓	✓		✓		2.0
(vi)	✓	✓	✓			✓	1.9

Table 6.2.: Parameter configuration for different subsonic test cases.

	case (a)	case (b)	case (c)	case (d)
Mach number M_∞	0.3	0.6	0.4	0.35
angle of attack α $[^\circ]$	5	1	3	4

reasonable descend direction for the loss and gets stuck early on in a local minimum, leading to a completely uniform flow field without any noticeable deflection near the airfoil. Comparing this result with the prediction for model (ii) which uses the MT, we observe how the model provides an accurate prediction of the pressure field. Note that the range of the colorbar has been scaled down by an order of magnitude because the errors are so drastically reduced. Slight deviations can still be observed mainly at the leading and trailing edge, where the pressure peaks are located. To evaluate whether AV might negatively affect the accuracy of the model for subsonic flow conditions, model (iii), which uses a scalar AV with an active shock sensor but no stagnation point sensor, is considered first. Here, we see a similarly accurate prediction as for model (ii) with a very slight increase in errors on the upper side of the airfoil near the trailing edge. However, the deterioration in accuracy is rather insignificant and needs to be analyzed quantitatively to make a definite statement whether this deterioration is even statistically significant. It should however be mentioned here that additional simulations beyond the scope of this work have shown that the $\rho - E$-AV and matrix-valued AV are more likely to not converge, when the stagnation sensor is not active. When the stagnation sensor is enabled, models (iv)-(vi) (Tab. 6.2) all perform very similarly, as can be seen for example in Figs. 6.3 (e)-(f).

In a next step, we compare the surface pressure distributions for all four subsonic test cases, as shown in Fig. 6.4. As already seen in the pressure distribution for model (i), the surface pressure distribution is overall accurately predicted by all models except for model (i) which does not use the MT. Visually, the predictions of models (ii)-(v) are indistinguishable and minor deviations from the reference can only be seen at the pressure minimum. Zooming into this point, we see that the PINN models predict a less strong peak and that model (v) predicts the peak best out of all PINN models. For the three other test cases, shown in Figs. 6.4 (b)-(d), a similar behavior is observed. Model (ii) does not provide any sensible predictions since the convergence during training is impeded when no MT is used. The predicted pressure fields are completely flat, confirming that

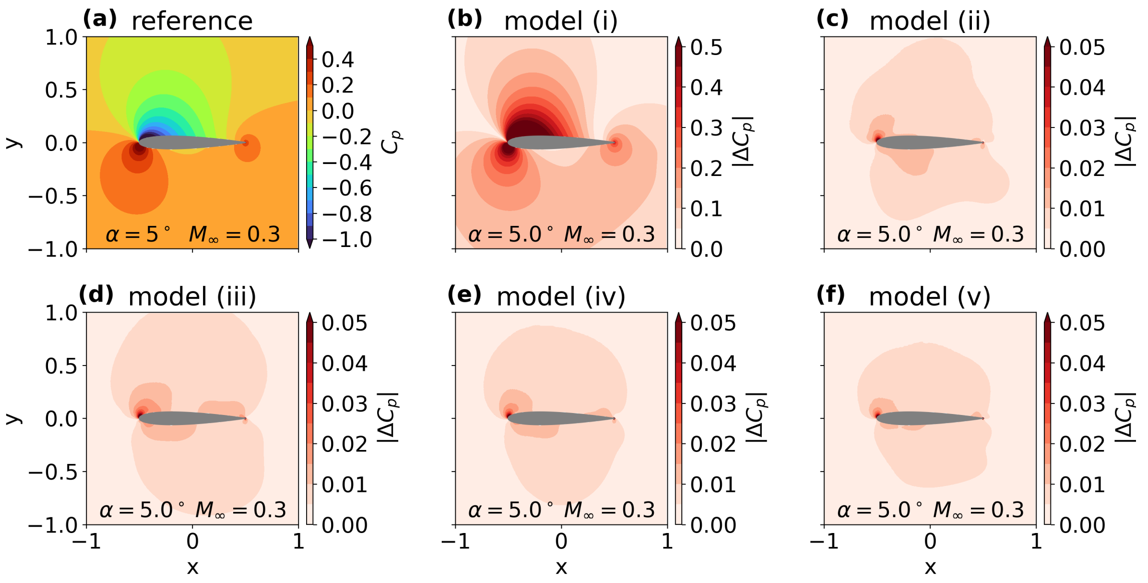


Figure 6.3.: Plot (a) shows the reference pressure coefficient C_p for test case (a). Plots (b)-(f) show the absolute difference of the pressure coefficient to the reference $|\Delta C_p|$ for models (i)-(v)

the deflection of the incoming flow by the airfoil is not properly captured. Evidently the optimizer gets stuck in state with a local minimum that corresponds to the free steam conditions for the whole domain. Models (iii)-(v) provide accurate predictions where slight deviations to the reference can only be observed at the leading and trailing edge. For example, at the pressure maximum in Fig. 6.4 (c), we see that the PINN models produce a more distinct pressure maximum than the finite volume simulation. For case (b) with $\alpha = 1^\circ$ the prediction is agrees best with the reference, even at the leading edge.

Lastly we perform a quantitative comparison of all subsonic models for all test cases. We evaluate the accuracy for the local Mach number M and the pressure coefficient C_p. The error is measured in terms of the *MAE*, *RMAE*, and R_2-score (Eqs. (2.107)-(2.109)) in comparison to the reference simulation. Each metric is calculated as the average over four different runs with different random initialization of the trainable network parameters (c.f. 2.3). For each run we calculate the error metrics over a box $(x, y) \in [-1, 1] \times [-1, 1]$. The points for this evaluation are uniformly distributed in the physical domain while points inside the airfoil are excluded. The error bounds are given by the standard deviation over the four training runs. Tab. 6.3 shows the results for the *RMAE* and the *MAE* and R_2-score are shown in the Appendix in Tab. B.5 and B.6. Italic values represent cases, where significant outliers (i.e. single runs that did not converge) negatively affected the metrics. Bold values show the best performing model. The quantitative analysis confirms that the model without MT is significantly outperformed by al analyzed models with MT. For example for the RMAE, we see that model (i) has a relative error of $RMAE \in (6, 7)\,\%$ and all models with MT consistently reach errors of $RMAE < 0.3\,\%$. We also see that the models with AV perform similarly to the model without AV, as the error metrics are generally within the first or second 1-σ- 2-σ interval of each other. Overall, we have seen that the MT is a crucial tool when dealing with non-trivial geometries such as airfoils.

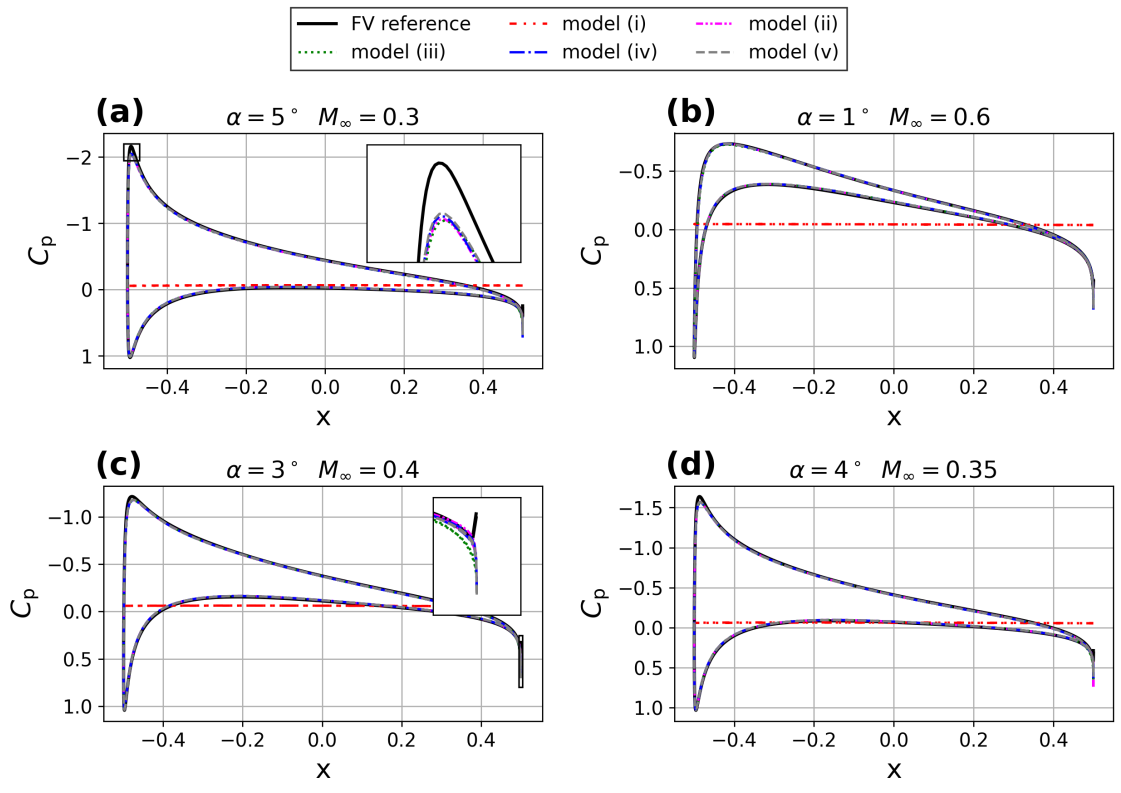


Figure 6.4.: Surface pressure coefficient C_p for case (a)-(d). The models (i)-(v) are compared to the FV reference.

Compared to Ch. 4, which considered a subsonic flow around an aerodynamic object, we can see that we do not required additional AV to facilitate convergence, when using the MT methodology. We have however also seen that the sensor controlled AV localization does not negatively influence the performance of the model.

In terms of runtimes, the models without AV outperform the models with AV. This is due to the fact that no second order derivatives need to be calculated when no AV is used, leading to reduced computational effort per evaluation of the loss function. Model (ii) is trained in 600 ± 500 s. The error bounds are given by the standard deviation in training times over all 16 runs (4 runs per test case). The variance is caused by the convergence criteria in the inner iteration of the L-BFGS optimizer. Out of all the models with AV, model (vi) is trained the fastest in 1100 ± 900 s. In comparison, the other models with AV all take more than 3500 ± 1400 s to train. Evidently, the matrix-valued AV leads to a loss landscape which is easier to navigate for the optimizer in the subsonic regime. Thus the average time per iteration is lowered, even though the same derivatives need to be calculated. In Sec. 3.3, the AV was motivated as a tool for approximating shock waves. In the following, we move on to transonic flow conditions where normal-shock waves occur and the effect of AV in these cases will be investigated.

Table 6.4.: Parameter configuration for different transonic test cases.

	case A	case B	case C	case D
Mach number M_∞	0.7	0.72	0.75	0.78
angle of attack α [°]	4	3	2	3

Table 6.3.: *RMAE* in [%] for different models for subsonic flow around airfoil.

Model	Case A $M_\infty = 0.3$ $\alpha = 5°$		Case B $M_\infty = 0.6$ $\alpha = 1°$		Case C $M_\infty = 0.4$ $\alpha = 3°$		Case D $M_\infty = 0.35$ $\alpha = 4°$	
(i)	C_p	6.2 ± 1.5	C_p	6.7 ± 1.7	C_p	6 ± 2.1	C_p	6.9 ± 1.5
	M	15 ± 1	M	14 ± 5	M	14 ± 2	M	14 ± 2
(ii)	C_p	0.208 ± 0.028	C_p	0.095 ± 0.009	C_p	**0.134 ± 0.018**	C_p	0.17 ± 0.06
	M	0.18 ± 0.021	M	0.077 ± 0.007	M	**0.098 ± 0.017**	M	0.15 ± 0.07
(iii)	C_p	0.2 ± 0.04	C_p	0.13 ± 0.04	C_p	0.16 ± 0.04	C_p	0.19 ± 0.04
	M	0.18 ± 0.05	M	0.105 ± 0.025	M	0.116 ± 0.024	M	0.15 ± 0.04
(iv)	C_p	**0.158 ± 0.015**	C_p	**0.09 ± 0.04**	C_p	0.135 ± 0.017	C_p	**0.142 ± 0.01**
	M	**0.132 ± 0.001**	M	**0.071 ± 0.026**	M	0.098 ± 0.015	M	**0.109 ± 0.008**
(v)	C_p	0.161 ± 0.009	C_p	0.101 ± 0.011	C_p	0.145 ± 0.014	C_p	0.1471 ± 0.0029
	M	0.139 ± 0.011	M	0.09 ± 0.01	M	0.106 ± 0.013	M	0.112 ± 0.004
(vi)	C_p	0.22 ± 0.07	C_p	0.103 ± 0.008	C_p	0.136 ± 0.011	C_p	0.164 ± 0.015
	M	0.2 ± 0.06	M	0.085 ± 0.007	M	0.098 ± 0.008	M	0.13 ± 0.016

6.2. Transonic Conditions

A flow is called transonic, when for a subsonic free stream M_∞, the flow becomes locally supersonic field [1, p. 65]. For a particular airfoil geometry, one can compute an estimate for which $M_\infty < 1$ this will be the case [1, ch. 11.6]. For the sake of this section, four flow conditions in the transonic regime, with different far-field Mach numbers M_∞ and angles of attack α were selected (see Tab. 6.4). They represent typical combinations of M_∞ and α, encountered on commercial aircraft in the transonic regime and feature a distinct normal shock wave on the upper side of the airfoil. Considering that the sweep angle of aircraft wings reduces the effective Mach number present at a local airfoil section, these cases emulate typical effective Mach numbers and angles of attack encountered on turbo-fan aircraft. Test case D represents an edge case scenario at a comparatively high Mach number. For a sweep angle of 30° it corresponds to a flight Mach number of $M_\infty \approx 0.9$ and a high angle of attack for such speeds.

As before, the reference results are obtained, using a finite volume solver (see Appendix D). An overview of all other hyperparameters of the model is shown in Appendix A.2. A total of five models with different AV variants are analyzed and as a sanity check, additional runs without AV (i.e. $\nu = 0$) were also performed. The model variants are summarized in Tab. 6.5. For the models where the stagnation sensor or the shock sensor are not active, we fall back to global AV, simply setting $s_{\text{stag}} = 1$ or $s_{\text{shock}} = 1$, respectively. As before, a

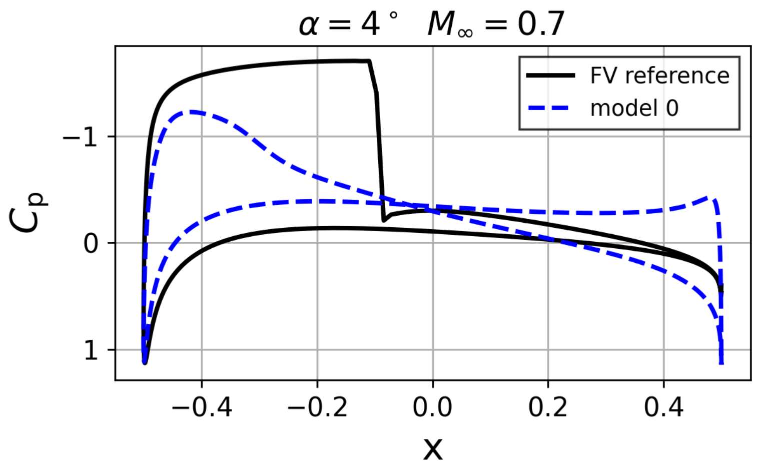


Figure 6.5.: Comparison of PINN prediction without AV for case A and reference finite volume solution.

Table 6.5.: Overview of analyzed models with different AV types.

model	s_{stag}	s_{shock}	scalar AV	ρ-E AV	matrix AV	$\nu \cdot 10^{-3}$
O						0
I			✓			2.1
II	✓		✓			1.9
III	✓			✓		1.8
IV	✓	✓		✓		2
V	✓	✓			✓	1.9

grid search is performed to obtain an optimal value for ν for all models. The used values are shown in Tab. 6.5 and are in a similar range of $\nu \in [1.8, 2.1] \cdot 10^{-3}$. Generally, for all ν values in this range, the accuracy of the different models does not change significantly. Again, a value of $\nu = 1.9 \cdot 10^{-3}$ is recommended as a sensible starting point for new investigations, as it has delivered similar accuracies for all analyzed models. The selection of ν is further illustrated in Appendix B.2

To highlight the failure of standard PINNs to capture shock waves in transonic flows, case A (see Tab. 6.4) is considered. Fig. 6.5 shows predicted surface pressure, when using model 0 with MT but with no artificial viscosity. Evidently, the model fails completely to capture the correct pressure field and no approximation of the shock is visible. As described in Sec. 3.1, the shock solution is likely not a minimum of the standard L_2-norm based residual loss for this case. Hence, the model fails to converge.

The effectiveness of different AV schemes is first tested on test case A. The reference pressure field is shown in Fig. 6.6 (a). Figs. 6.6 (b)-(f) show the absolute difference of the pressure coefficient to the reference $|\Delta C_p|$ for the different AV variants. Fig. 6.6 (b) shows the result when a global scalar AV (3.16) is used and no sensors are active. In this case, the PINN model still completely fails to approximate the shock, even though in Sec. 3.2 this was sufficient to obtain at least acceptable results. However, when the

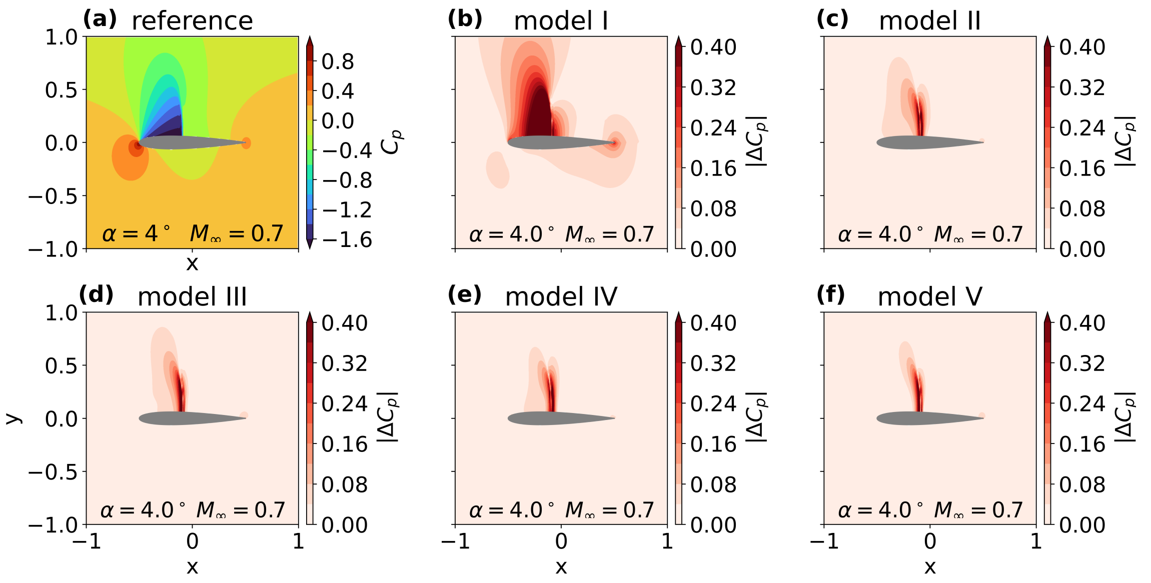


Figure 6.6.: Plot (a) shows the reference pressure coefficient C_p for test case A. Plots (b)-(f) show show the absolute difference of the pressure coefficient to the reference $|\Delta C_p|$ for models I-V.

stagnation point sensor (3.25) is activated, even though the shock sensor is still set to $s_{\text{shock}} = 1$, the resulting model is able to approximate the shock quite accurately, as shown in Fig. 6.8 (c). While the shock is still smoothed out and some inaccuracies can be seen in the low pressure region in front of the shock, the improvements due to the AV are clearly visible. Note how the stagnation sensor only became relevant in the transonic regime, as in the subsonic regime it did not have a significant impact on the model performance. Switching to the density and energy AV, the accuracy can be further increased, as shown in Fig. 6.6 (d). Evidently, dissipation in the momentum equations is not strictly required to stabilize the shocks, as postulated in Sec. 3.3.1. The removal of the dissipation in the momentum equations hence leads to a less dissipative but similarly stable model. When the shock sensor is activated in Fig. 6.6 (e), the slight inaccuracies in-front of the shock can be further reduced, since the dissipation is only activate around the shock itself. The best approximation is however obtained when using matrix-valued AV, where significant errors are only visible due to the smoothing of the shock.

To evaluate the performance of the different AV variants on the other cases, surface pressure plots for each model and all four cases are shown in Fig. 6.7. For case B-C, a similar behavior as for case A can be observed. Model I fails to provide accurate predictions for all four test cases and similarly to the models without AV and not even the slightest hint of a shock approximation is visible. Comparing models II-V, we see how all of them form the expected shocks and can accurately match the shock location of the reference solution for the cases A-C. Since similar ν-values have been applied to all models, ewe can also see that the shock are similarly sharply resolved. However, in the low pressure region upstream of the shock, model V with matrix-valued AV matches the minimum pressures best, outperforming all other model in that regard (see zoomed in window in Fig. (b)). However, for case D, we see that the shock location is not accurately

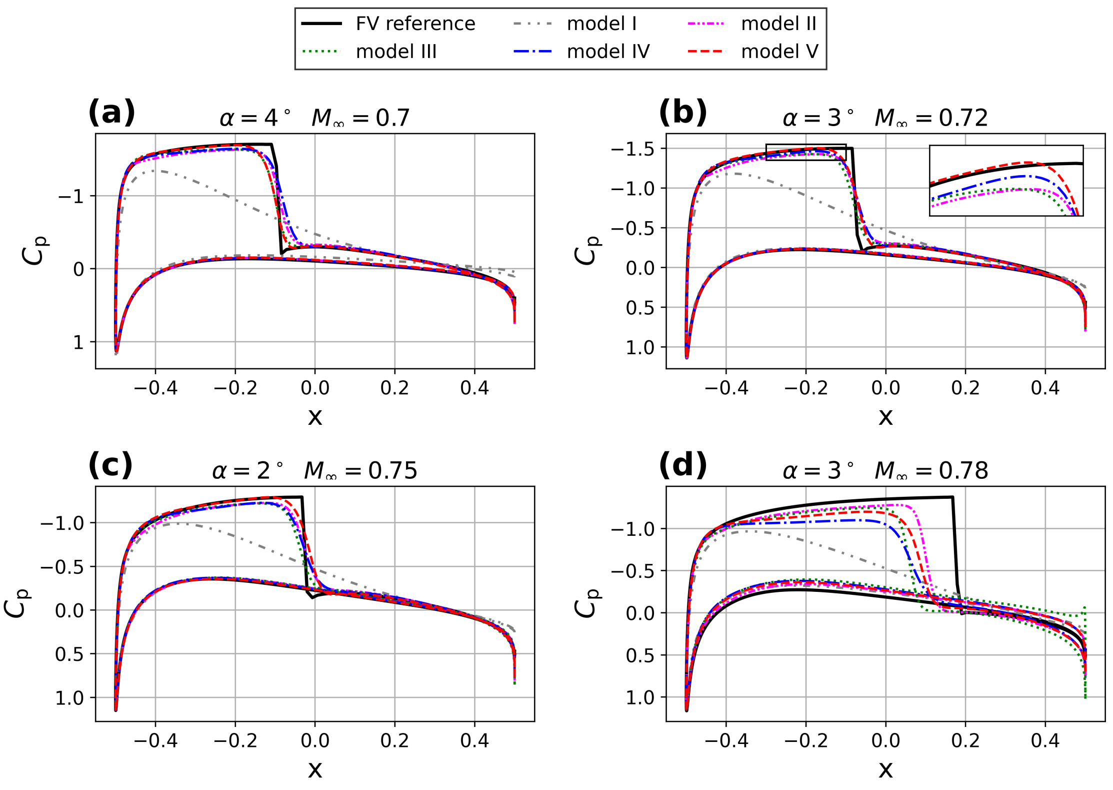


Figure 6.7.: Surface pressure coefficient C_p for case A-D. Different AV methods are compared to the finite volume reference.

predicted by all AV versions and overall the predictions deviate from the reference. Model II provides the most accurate prediction, followed by model V but both model predictions are still unsatisfactory. Interestingly, when using ADAM [100], instead of L-BFGS, as the optimizer, we observed that the shock moves too far downstream during training. Hence, it is likely that the issue is related to the choice of the optimization algorithm and an unsatisfactory convergence. In Appendix B.3, the loss history during training is shown for all four test cases, where we see that the magnitude of the losses is significantly larger for case D, supporting the claim that insufficient convergence might be responsible for the deteriorated accuracies on case D. Recently, Cao and Zhang [69] have analyzed the ill-conditioning of physics-informed neural networks during training and proposed time-stepping-oriented neural networks to alleviate the ill-conditioning of the underlying system when training PINNs. They propose a quasi-time stepping optimization procedure to alleviate the ill-conditioning. Future extensions of the presented method could make use of such problem-specific optimization procedures to improve the prediction accuracy, even for higher Mach numbers.

Fig. 6.8 shows the sensor s for model V for case A-D. In addition, the contour lines of the coefficient of pressure C_p are plotted. The sensor is mainly active ($s \approx 1$) near the shock, where the contour lines converge. One can also see active regions near the leading edge and close to the trailing edge. Upon close inspection, the sensor is however inactive at the stagnation points themselves due to the stagnation point sensor given in Eq. (3.25). The removal of the stagnation points from the sensor drastically improves the prediction accuracy, as shown in Fig. 6.6. In theory, the activation threshold $k^{(0)}_{\text{shock}}$ can be increased to further limit the regions where viscosity is applied. However, empirical trials show that this does not lead to significant improvements in accuracy and slightly reduces the consistency of the method overall.

The quantitative analysis of the errors is done as for the the subsonic conditions. The *MAE*, *RMAE*, and R_2-score for model I-V and cases A-D are analyzed (in comparison to the reference simulation). Each metric is calculated as the average over four different runs with different random initialization of the trainable network parameters (c.f. 2.3). For each run we calculate the error metrics (2.107)-(2.109) over a box $(x, y) \in (-1, 1) \times (-1, 1)$. The points for this evaluation are uniformly distributed in the physical domain while points inside the airfoil are excluded. The error bounds are given by the standard deviation over the four training runs. Tab. 6.3 shows the results for the *RMAE* and the *MAE* and R_2-score are shown in the Appendix in Tab. B.7 and B.8. Italic values represent cases, where significant outliers (i.e. single runs that did not converge) negatively affected the metrics. Bold values show the best performing model.

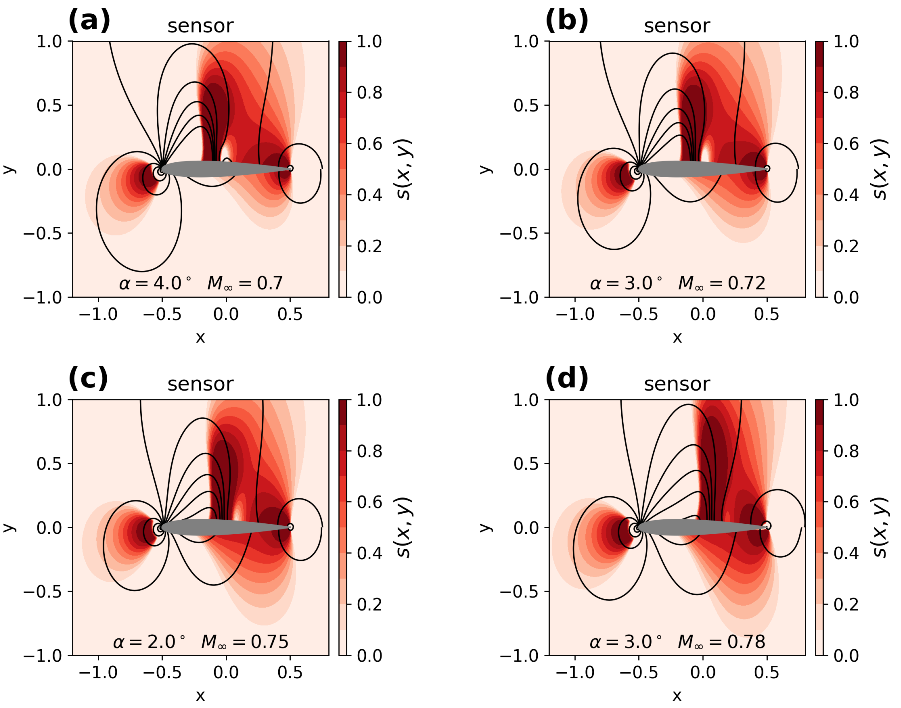


Figure 6.8.: The sensor function $s(x, y)$ and the applied AV $\mu(x, y)$ for the three transonic test cases in Fig. 6.6. The contour lines show the pressure field to indicate how the sensors are affected by the pressure gradient.

Table 6.6.: *RMAE* in [%] for different PINN models for transonic flow around airfoil.

Model	Case A $M_\infty = 0.7$ $\alpha = 4^\circ$		Case B $M_\infty = 0.72$ $\alpha = 3^\circ$		Case C $M_\infty = 0.75$ $\alpha = 2^\circ$		Case D $M_\infty = 0.78$ $\alpha = 3^\circ$	
0	C_p	4.82 ± 0.28	C_p	4.1 ± 0.28	C_p	4 ± 1	C_p	8.35 ± 0.11
	M	3.85 ± 0.19	M	3.33 ± 0.22	M	3.1 ± 0.8	M	6.76 ± 0.11
I	C_p	2.2 ± 0.4	C_p	4 ± 2.4	C_p	1.79 ± 0.19	C_p	5.2 ± 1.9
	M	1.79 ± 0.24	M	18 ± 20	M	1.59 ± 0.16	M	15 ± 22
II	C_p	0.48 ± 0.03	C_p	*6 ± 7*	C_p	1 ± 0.9	C_p	**1.85 ± 0.15**
	M	0.474 ± 0.018	M	*15 ± 18*	M	0.9 ± 0.7	M	**1.65 ± 0.13**
III	C_p	*2.1 ± 2.8*	C_p	*3 ± 4*	C_p	0.46 ± 0.04	C_p	5± 4
	M	*5 ± 8*	M	*10 ± 18*	M	0.466 ± 0.029	M	13 ± 20
IV	C_p	0.49 ± 0.14	C_p	0.34 ± 0.11	C_p	0.56 ± 0.15	C_p	3.11 ± 0.14
	M	0.48 ± 0.12	M	0.35 ± 0.09	M	0.54 ± 0.13	M	2.8 ± 0.4
V	C_p	**0.42 ± 0.04**	C_p	**0.3 ± 0.08**	C_p	**0.33 ± 0.11**	C_p	3 ± 0.4
	M	**0.427 ± 0.022**	M	**0.31 ± 0.05**	M	**0.35 ± 0.09**	M	2.7 ± 0.4

The metrics confirm that model V performs best on case A-C and that Model II performs best on case D. Compared to the reference simulations, we generally see errors of less than one percent for all analyzed test cases, when using model V. Moreover, it performs consistently well with a comparatively low standard deviation. Model IV with the density and energy AV performs only slightly worse. Models II-III without s_{shock} perform inconsistently, since some individual runs diverge or certain cases cannot be solved accurately at all. Model II, without the stagnation sensor, is unable to reach acceptable error levels for all cases.

The average training time for model V is 2100 ± 400 s. Due to the global viscosity, the training time of model II, with 3900 ± 1400 s, is for example significantly longer since the second order derivatives have to be evaluated in the entire domain. The variance in training time is caused by the convergence criteria in the inner iteration of the L-BFGS optimizer. Compared to the subsonic problems, the training time of the model with matrix-valued AV is a bit higher. For the transonic problems, the models were trained for 40 instead of 30, epochs which partially explains the increase. In addition, it can be expected that convergence takes longer to accurately approximate the nonlinear flow features at the shock.

However, overall the training time that we see for the models with MT and sensor controlled AV is significantly reduced, compared to the training routine used in Ch. 4 and Ch. 5 where training times were on the order of 7 – 8 h for a non-parametric model. This speed-up can be attributed to two main factors. On the one hand, the model architecture uses several methods to improve model convergence, such as Fourier embedding (Sec. 2.3), weight normalization (Sec. 2.3) and most importantly MTs (Sec. 2.4.6). On the other hand, the model is exclusively trained with the L-BFGS optimizer, leading to significantly reduced wall clock times due to the faster convergence rates of quasi Newton methods, compared to gradient descent methods such as ADAM (Sec. 2.3). Once trained, the model can be evaluated in about 45 ms.

Based on the error metrics, the matrix-valued AV, combined with the shock and stagnation-point sensor seems to be well suited for moderate Mach numbers and angles of attack in the transonic regime in addition to the subsonic regime. The matrix-valued AV, combined with the sensor is able to reduce dissipativeness, compared to the scalar and density and energy AV. It provides an accurate approximation of the entire flow field, except for the region around the shock which is smoothed. However, compared to the legacy finite volume method and also classical matrix-valued AV schemes [75], the approach is still significantly more dissipative. This can also be confirmed using the total pressure loss

$$p_{\text{tot}} = p\left(1 + \frac{1}{2}(\kappa - 1)M^2\right)^{\overline{\kappa - 1}^{\,\kappa}\!\!} . \tag{6.1}$$

The far field total pressure $p_{\text{tot},\infty}$ is obtained when evaluating Eq. (6.1) at the far field conditions, given by $\boldsymbol{w}_{\infty} = (\rho_{\infty}, u_{\infty}, v_{\infty}, p_{\infty})$. The total pressure loss is then given by:

$$C_{p_{\text{tot}},\text{loss}} = 1 - \frac{p_{\text{tot}}}{p_{\text{tot},\infty}}. \tag{6.2}$$

The total pressure loss on the airfoil's surface can serve as a measure of the accuracy of the numerical scheme. Fig. 6.9 shows the pressure loss for model V and the reference simulation for the four test cases in Tab. 6.4. As highlighted by the reference simulation,

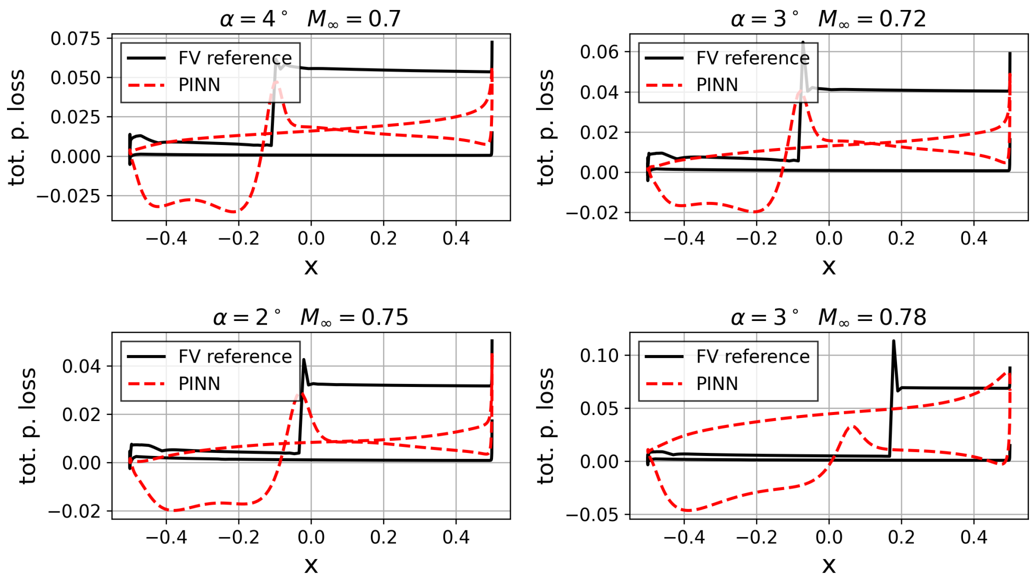


Figure 6.9.: Total pressure loss for the non-parametric model V.

we generally expect pressure losses close to 0 for subsonic flows, a steep increase around shocks and a constant, increased value behind shocks. For the PINN, the increase around the shock is on the same order of magnitude as for the reference. We do however see significant changes in the pressure loss in regions besides the shock, which indicates, that the PINN methodology is more viscous and further improvements can be made. Comparing the locations where the sensor is active (in Fig. 6.8) to the sensor functions in legacy methods (e.g. [79, p.50]), we see that the active area, is much larger. This possibly contributes to the higher pressure losses. However, empirical tests have also shown that an increase in $k^{(0)}_{\text{shock}}$ does not lead to higher accuracies and can potentially reduce the robustness. Another possibility to reduce dissipativeness, would be to perform mesh refinement around the shock location. So-called h-refinement is well known for classical CFD methods [3, p. 304] and could potentially help to obtain more accurate approximations of the shock. For the shock solutions in Burgers' equation, we have seen that a refinement of the mesh around the shock was able to significantly reduced the distinctiveness of the predicted solution (see Sec. 3.2.5), giving a good indication, that this is a feasible way to reduce dissipativeness. However, the analysis of mesh refinement procedures for PINNs are beyond the scope of this work. Another possible solution would be to also make use of an adaptive viscosity procedure, which might be able to find a better distribution of $\nu(x, y)$ than the analytical sensor function. Here, the main challenge would be to find a training procedure that can reliably produce an adequate distribution for transonic problems. The experiments have shown that with the current training procedures, the sensor function is more reliable and accurate for these kinds of problems.

6.3. Parametric Problem

The previously analyzed methodology can be adapted to parametric problems. In Secs. 4.2 and 5.2, this approach was already investigated, using a PINN model with adaptive AV localization. As shown in Fig. 6.1, a parameter of the PDE is added to the input space of the network to solve the PDE in a parametric fashion. In addition to the coordinates in the computational grid (ξ, η), the PINN is trained in a third dimension defined by the angle of attack. Here, we parameterize the angle of attack. The trained model can be used to make predictions for the entire training range of α at once. To train the parametric model we randomly sample N_α values for the angle of attack in the range $\alpha \in (0, 4)^\circ$. During a training iteration, each grid point is randomly combined with one of the sampled α values. To improve the predictions at the bounds of the parameter space, we add $N_{\alpha,\text{bound}} = 40000$ additional α values with $\alpha = 0^\circ$ and $\alpha = 4^\circ$, similarly to the sampling of the Mach numbers in Sec. 5.2. The Mach number is kept constant at $M_\infty = 0.72$. Since model V performed best on the non-parametric problem, we make use of the matrix-valued AV, combined with the sensor function for localizing the viscosity. An overview of all hyperparameters of the parametric model is shown in Appendix A.2.

The resulting pressure field in comparison to the reference solution is shown in Fig. 6.10 for five different angles of attack. At the lowest angle of attack $\alpha \approx 0$, the model approximates the fully subsonic flow accurately, even when using the AV. This observation is in agreement with Sec. 6.1 where we have seen that the matrix-valued AV does not have a noticeable negative influence on the prediction capabilities at subsonic conditions. We can also see how the model perfectly matches the symmetric pressure distribution that we expect for the symmetric NACA 0012 airfoil at $\alpha = 0^\circ$. For $\alpha = 1^\circ$, the prediction is very accurate overall, but since this shock is weak, it not captured by the PINN. The viscosity that is applied at the shock location leads to a smoothing of the discontinuity. Hence the shock is not present in the PINN prediction. However this error only affects a small part of the domain, near the shock and the prediction is otherwise accurate. At $\alpha = 2$, the PINN starts to resolve the shock and we can see a noticeable drop in pressure at $x \approx -0.15$ on the upper side of the airfoil. The stronger shocks at $\alpha \geq 3^\circ$ are all approximated well and the smoothing due to the AV becomes less significant with higher angles of attack. Overall, at these higher angles of attack the error plots confirm that difference in comparison to the reference are mainly observed around the shock and that the flow field is well approximated overall. Looking at the corresponding surface pressure distributions in Fig. 6.11, the high accuracy of the model, at different α is confirmed. For $\alpha = 1$ smoothing of the weak shock by the PINN is confirmed but it can also be seen that the pressure profile is well matched on the remaining surface. Fig. (d) also shows the prediction of the non-parametric model V for case B, matching the parametric model perfectly. This gives a first indication that the parametric model can provide similarly accurate predictions as the non-parametric model.

In Fig. 6.12, the mean coefficient of lift prediction over four randomly initialized training runs is shown. For the entire parameter range, the lift agrees well with the reference. The hatched area shows the minimum and maximum predictions over four training runs. Overall, the model performs consistently with almost identical lifts at lower α and slightly

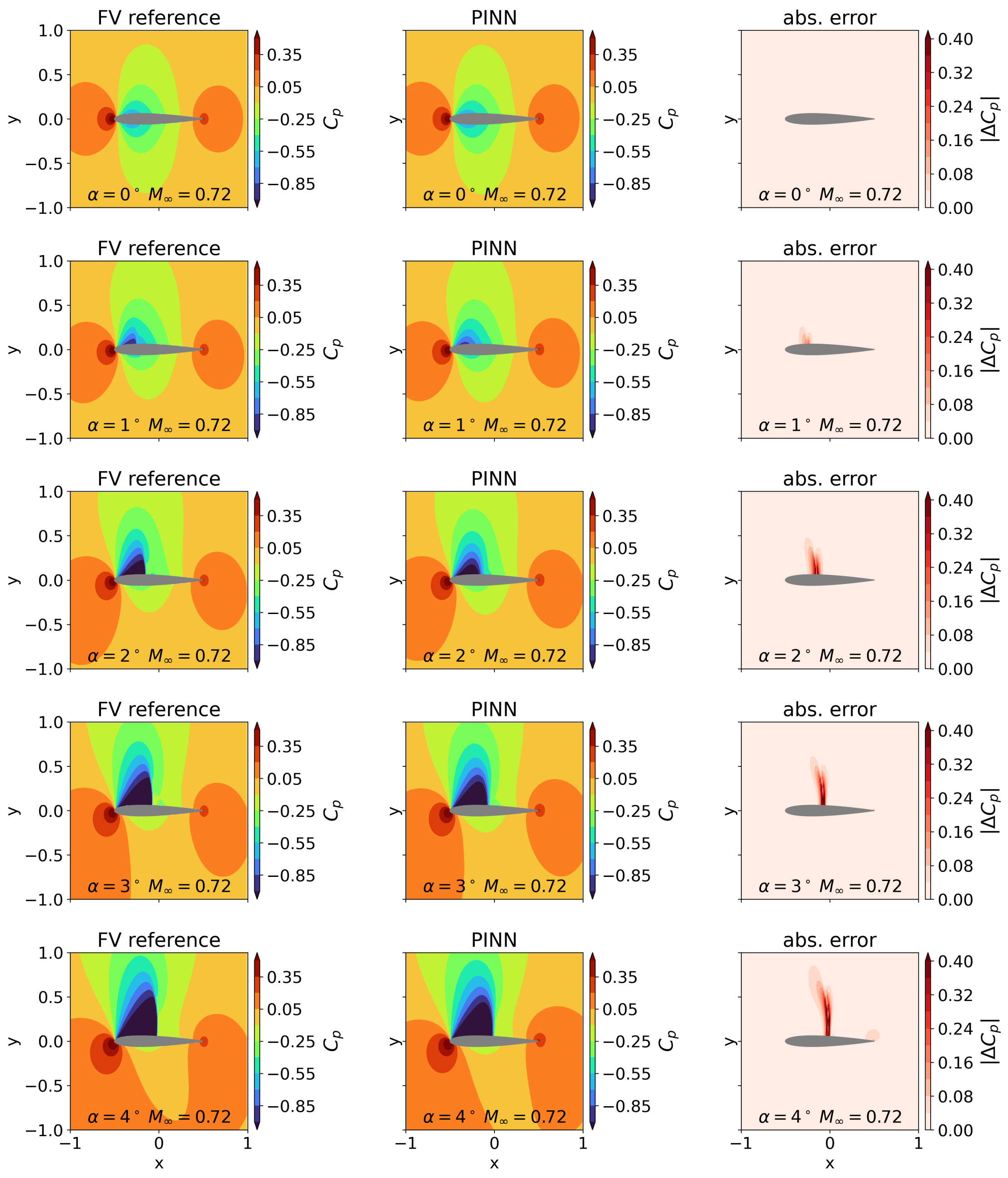


Figure 6.10.: Predicted pressure fields by a single parametric PINN in comparison to reference solutions at different α.

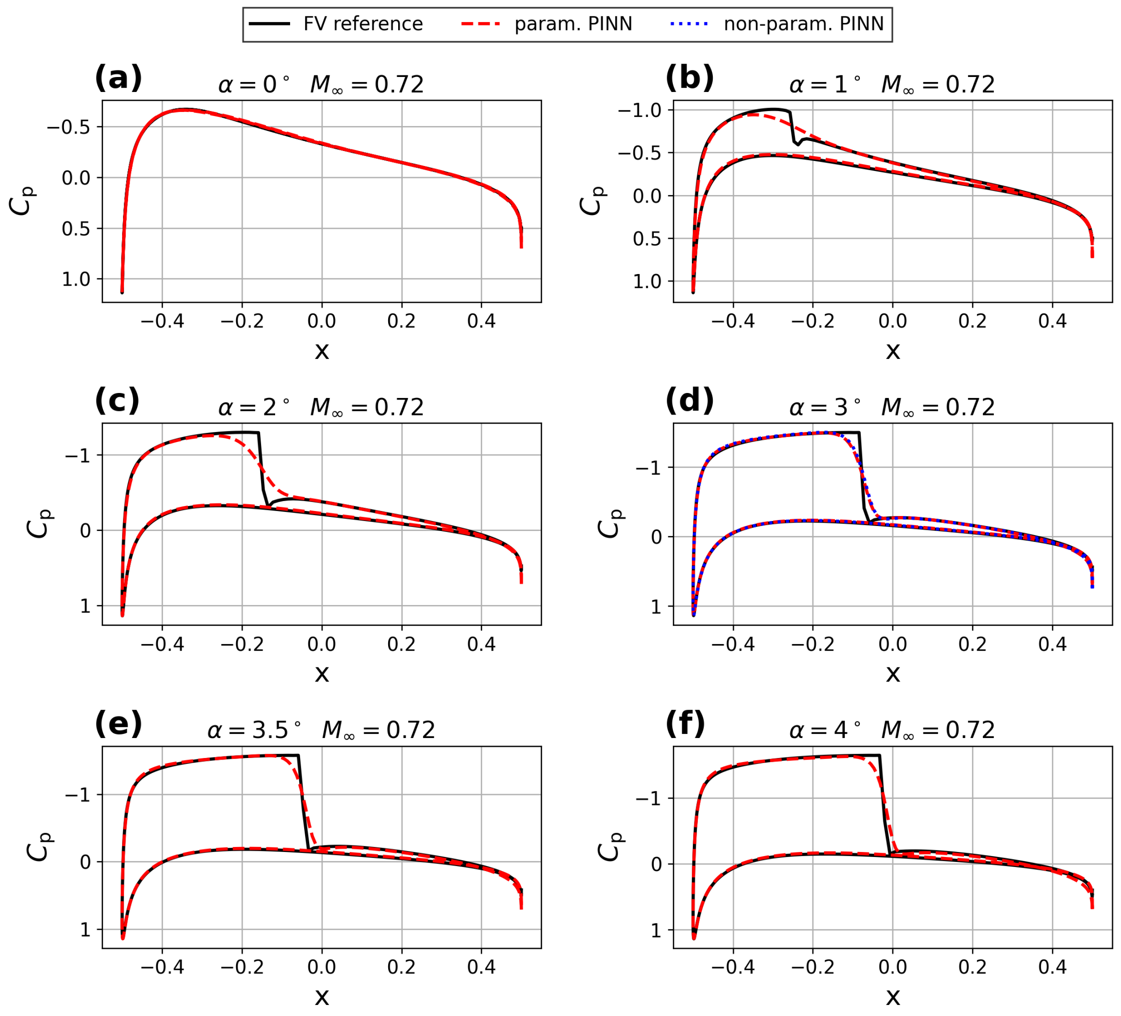


Figure 6.11.: Predicted surface pressure distribution by a single parametric PINN in comparison to reference solutions at different α. For $\alpha = 2^\circ$ the prediction of the corresponding non-parametric PINN in Fig. 6.6 is also shown.

larger spread in the transonic region. In the lower angle of attack range ($\alpha \approx 1^\circ$), we see that slightly lower C_l values are predicted. This is due to the fact that the weak shock at these conditions is smoothed out, as seen in Figs. 6.10-6.11, leading to a slight drop in C_l compared to the reference. At $\alpha > 2.5^\circ$, once the shock is fully established, the average prediction matches the reference perfectly again.

Fig. 6.13 shows the predicted pitching moment coefficient $C_{m,25}$ with respect to the 25 % cord point. For a symmetric airfoil. such as the NACA 0012, this pitching moment is expected to be close to 0, in the subsonic regime. The comparison with the reference shows that the parametric PINN follows the same general trend. However, the predicted pitching moment is generally over predicted and one can observe a significant variance in the predicted moments over the different training runs. This is due to the fact that the shock location and positioning has a major impact on the moment for this particular choice of airfoil and reference point. Since prediction errors are mostly present at the

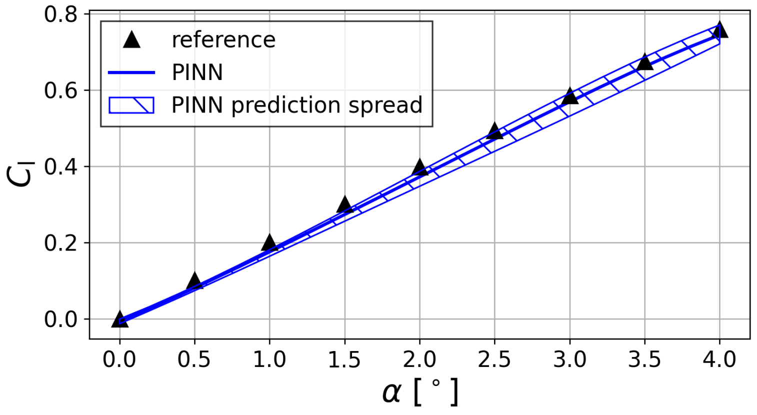


Figure 6.12.: Predicted coefficient of lift for different angles of attack.

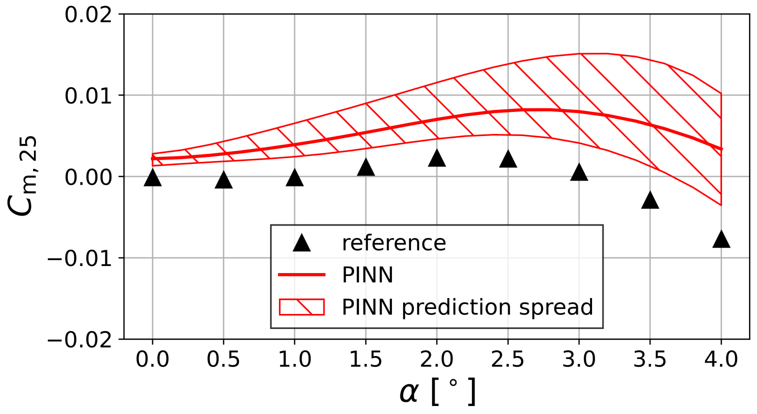


Figure 6.13.: Predicted pitching moment coefficient for different angles of attack.

shock location, due to the smoothing of the shock, the pitching moment is affected by these inaccuracies. It should however be noted that the absolute values of the moment for this particular airfoil are small, hence the variance in the moment prediction gives the impression to be very large. The quantitative accuracy of the predictions is again evaluated using the *RMAE*, *MAE* and R_2-score over four randomly initialized training runs, as explained in Sec. 6.2. The error metrics are summarized in Tab 6.7. Similarly to the non-parametric version, we see relative errors below one percent for all angles of attack. Overall, we can see that the proposed PINN methodology provides good estimations of the lifting coefficient for the entire parameter range. The results do also reveal that due to the relatively high dissipativeness, weak shocks at the lower angles might be smoothed out. The training time varies between 1.5 h and 2.75 h due to the convergence criteria in the inner L-BFGS iterations. It was however observed that the accuracy does not significantly improve anymore after the first epoch, which requires about 30 min on average. Predicting a specific solution at an angle of attack of interest after the training is possible in about 45 ms.

Table 6.7.: Mean percentage errors for parametric PINN at $M_\infty = 0.72$ for different α.

α [°]	$\boldsymbol{RMAE}$ [%]		$\boldsymbol{MAE}$		$\boldsymbol{R_2} \cdot 10$	
0	C_p	0.12 ± 0.04	C_p	$(1.8 \pm 0.6) \cdot 10^{-3}$	C_p	9.996 ± 0.003
	M	0.1 ± 0.04	M	$(7 \pm 2.5) \cdot 10^{-4}$	M	9.994 ± 0.003
1	C_p	0.3 ± 0.08	C_p	$(5.9 \pm 1.5) \cdot 10^{-3}$	C_p	9.96 ± 0.012
	M	0.27 ± 0.07	M	$(2.3 \pm 0.6) \cdot 10^{-3}$	M	9.958 ± 0.012
2	C_p	0.32 ± 0.13	C_p	$(8 \pm 4) \cdot 10^{-3}$	C_p	9.88 ± 0.04
	M	0.29 ± 0.11	M	$(3.2 \pm 1.2) \cdot 10^{-3}$	M	9.85 ± 0.07
3	C_p	0.32 ± 0.14	C_p	$(9 \pm 4) \cdot 10^{-3}$	C_p	9.88 ± 0.06
	M	0.33 ± 0.1	M	$(4.3 \pm 1.3) \cdot 10^{-3}$	M	9.84 ± 0.08
4	C_p	0.44 ± 0.06	C_p	$(1.24 \pm 0.17) \cdot 10^{-2}$	C_p	9.88 ± 0.03
	M	0.47 ± 0.04	M	$(6.73 \pm 0.5) \cdot 10^{-3}$	M	9.831 ± 0.022

6.4. Summary

In this chapter the application of sensor-controlled viscosity localization for simulating flows around a NACA0012 airfoil under various transonic flow conditions was investigated. The model employed in this study combined several techniques, including MTs, Fourier embedding, and training using the L-BFGS algorithm. A key focus of the chapter was the comparison of different artificial viscosity methods and the combination with the sensor function. The methods examined included scalar AV, density and energy AV, and matrix-valued AV. For subsonic flow conditions, we have seen that the MT methodology is essential for obtaining reasonable predictions around the airfoil. When MT is used, accurate predictions with and without AV were obtained for all analyzed test cases. For transonic flow conditions, the results underscored the critical role of the stagnation sensor in achieving accurate predictions, as the application of AV at stagnation points int th transonic conditions was found to impede convergence. This effect of the stagnation point sensor was however less noticeable for subsonic conditions. The findings indicated that the combination of the sensor function with matrix-valued AV yields the best performance, closely followed by the combination of the sensor function with density and energy AV. These models consistently demonstrated errors of less than 1% and accurate shock localization, across all test cases except for the case with the highest Mach number $M = 0.78$ and angle of attack $\alpha = 3°$. In this edge scenario, the PINN models failed to predict the correct shock position. In comparison to the finite volume reference solution, the PINN models produced more smoothed-out shocks. The computational efficiency of the PINN approach was highlighted, with training times of approximately 0.5 hours and evaluation times of about 45 ms. The chapter also explored the development of a parametric model with a variable angle of attack. The parametric model achieved similar accuracies as the non-parametric version and accurately predicted the lifting coefficient for the entire parameter range. At subsonic angles of attack, the parametric model matched the reference solution almost perfectly. However, it struggled to capture weak shocks at lower angles of attack due to noticeable smoothing. For higher angles of attack, the PINN model closely matched the reference solution, albeit still with a slight smoothing around the shock. Notably, the parametric model required only a marginal increase in training time. An accurate approximation was obtained after just one epoch, with a training time of 30 minutes, and evaluation times remained at 45 ms. Even after five epochs, the training time was less than 3 hours, demonstrating the efficiency and potential of the PINN approach for parametric problems in aerodynamics.

7. Discussion of Research Questions

This work investigated physics-informed neural networks (PINNs) for applications in fluid dynamics and aerodynamics with a focus on steady compressible flows, governed by the Euler equations. It was shown that artificial viscosity can be a vital tool for addressing convergence and stability issues when solving scalar conversation laws like Burgers' equation and the compressible Euler equations. Artificial viscosity localization techniques such as the adaptive and the sensor-controlled methodology were introduced to determine a sensible distribution of viscosity inside the domain. In addition, different ways to determine the viscosity magnitudes for each equation, such as the scalar AV, density and energy AV ,and matrix-valued AV were analyzed. All of these methodologies were applied to a number of test cases, such as the subsonic flow around a cylinder, the supersonic oblique shock problem and flows around airfoils in the subsonic and transonic regime. For all of these test cases, significant improvements in terms of convergence rates and accuracy could be observed. Lastly, the parametric models were also investigated, clearly showing the potential of the method for many query scenarios. To discuss the implications of these findings and to put them into a broader context, in this chapter, we revisit the research questions outlined at the start of this work, starting with question 1

1. Are classical numerical techniques and concepts for solving PDEs relevant for PINNs?

Recently, the PINN methodology has attracted significant attention in the field of fluid dynamics, promising to be a novel algorithm which can utilize the strengths of machine learning architectures to solve numerical problems, governed by the challenging partial differential equations that describe the motion of fluids. Moreover, its implementation can directly use existing NN software, simplifying the initial setup and the method does not require a computational grid. In classical CFD grid generation can be a complex task, in particular for complex three-dimensional geometries (e.g. the gear of an aircraft). The simplicity of the methodology has resulted in a considerable amount of scientific literature. Especially in the field of fluid dynamics, the interest has been substantial (see e.g. Sec. 1.3). In some sense, the fact that PINNs promise to avoid aspects of classical solvers often seen as undesirable can be regarded as the main driving factor for this trend.

However, throughout this work, the proposed modifications and improvements to the models have repeatedly been motivated based on classical CFD methods and even adapted them for PINNs. Furthermore, many of the most effective modifications of the standard PINN method in literature that have been utilized in this work are also tied to classical CFD concepts. These modifications include the incorporation of artificial viscosity to facilitate the reliable approximation of shock waves. Moreover, similarly to the JST-scheme (Sec. 2.2.2), this artificial viscosity was scaled with the spectral radius of the flux Jacobians. To further reduce the dissipativeness of the shock approximation, matrix-valued artificial viscosity was introduced in analogy to the corresponding FV scheme. Additionally, the localization of artificial viscosity is achieved through the use of a sensor function based on pressure gradients, allowing for more precise control over where the viscosity is applied. The training process is further improved by applying MTs, which enhance the conditioning of the underlying optimization problem. To use the mesh-transformation, a classical grid is even reintroduced. The weighting of point-wise residuals based on cell volume is another critical modification, ensuring that the contribution of each cell to the overall residual is appropriately scaled.

Based on this non-exhaustive list, it is apparent that many classical concepts, well known from decades of CFD research, still seem to be relevant for PINNs. The core contribution of this work is the introduction of the various AV methodologies which can be seen as a fundamental CFD concept. However, it is not surprising that these methodologies are effective because the use of AV can even be motivated at a fundamental mathematical level. As outlined in Sec. 2.1.2, the notion of an entropy solution is fundamentally linked to the concept of AV. The reason why FV methods are so well suited for the solution of conservation laws is because they reformulate the continuous problem in to a finite number of Riemann problems for the cell faces through the discretization into cell volumes with piece-wise constant ansatz functions. Due to the mathematical understanding of the Riemann problem, researchers were able to design effective solution algorithms, such as upwinding or central difference schemes, which rely on the knowledge of the direction of wave speeds. Thus, it makes only sense that PINN algorithms that emulate classical CFD methods are better suited for the solution of these types of equations, as they indirectly make use of the underlying mathematical theories that are already incorporated into the legacy methods. With the development and extension of more fundamental theories for the solution of conservation laws with PINNs, further progress with regards to their capabilities is to be expected.

However, it should also be mentioned that the fundamental differences between PINNs and established methods warrant certain non-obvious modifications when trying to adapt classical techniques for PINNs. For example, as explained in Sec. 2.2.2, we have seen that in the finite volume method, central difference schemes apply fourth differences in smooth regions of the flow to avoid unwanted oscillations or so-called odd-even decoupling, where the solution at one particular node becomes independent from its neighbors. This phenomenon is only possible because the derivatives/fluxes are approximated with difference operators. Since this is not the case for PINNs, we generally do not observe such phenomena and hence do not need to apply AV in smooth regions of the flow. The additional AV would only deteriorate the solution accuracy and significantly increase the training cost due to the expensive calculation of higher order derivatives with AD. This clearly shows that researchers need to pay close attention to where and how classical methodologies can be transferred to PINNs.

Initially, a main selling point for investigating PINNs was the promise to be free of grid generation. While this is technically true, PINNs are still trained on discrete points. In general, a random sampling of these points only works well for rather simple problems and leads to inadequate results when dealing with multi-scale or highly heterogeneous problems. Unsurprisingly, the grid generation task is replaced by an equally challenging task of generating adequate collocation point distributions. Various adaptive point distribution techniques have been proposed in literature (see e.g. [112]), but there seems to be no consensus yet about a universally applicable and robust approach. In this work, a MT methodology is used, reintroducing a classical grid. As we have seen, it may even be advantageous to not abandon the knowledge about the subject of grid generation that has been attained over decades. It is well known that a well or poorly designed grid can make or break a CFD simulation. In particular, structured grids encode essential information about the geometry, based on the orientation and density of the grid points. It turns out that this information is also relevant for PINNs and potentially even more than for classical methods.

Overall it can be concluded that many classical concepts, such as the mesh generation and artificial dissipation, need to be considered for PINNs to successfully solve problems in fluid dynamics and aerodynamics. However, the unique properties of PINNs also lead to some differences. Hence, researchers need to carefully consider which classical concepts are relevant and how existing knowledge can potentially be used to build better neural network-based solvers.

2. What can be the role of PINNs in compressible aerodynamics?

(a) To which extent can PINNs solve forward problems and how do they compare to classical CFD solvers?

(b) Can PINNs solve complex parametric problems?

Moving on to question 2.(a), in this work, PINNs have been used to successfully approximate solutions to various forward problems for the inviscid Burgers equation (Ch. 3), and for the compressible Euler equations for a subsonic flow around a cylinder (Ch. 4), a supersonic oblique shock (Ch. 5) and subsonic and transonic flows around an airfoil (Ch. 6).

It was repeatedly observed that standard PINNs, meaning PINNs without specialized adaptations for conservation laws or non-trivial geometries, fail to approximate solutions successfully. As explained in Sec. 3.1, this can partially be explained by a failure to approximate shock wave solutions, since these solutions generally do not correspond to minima of the commonly used L_2-norm-based residual loss. However, using the newly developed AV methodologies, presented in this work, reasonable approximations of shock wave solutions can indeed be obtained. Through the use of AV, the discontinuities are smoothed out, enabling the finding of smooth approximations of the discontinuous solutions. Even for problems with a subsonic, continuous entropy solution (Sec. 4.1) we have seen that the use of AV can be beneficial during training. However, compared to finite volume simulations, which have been used as a reference solution method for most problem analyzed in this work, we generally see that the PINN predictions are unable to match the accuracy of a CFD methods with a sufficiently well-resolved grid. For instance, for transonic flows around airfoils, we see that shocks are generally more smoothed out than in the finite volume solution, even though the grid, used for the PINN, is actually more finely resolved (400×200 points) than the CFD grid (400×91 points). In addition, we generally observe that the PINN training is more expensive than classical CFD simulations. Of course, a direct comparison is difficult, as PINNs are trained on graphic processing units and the used CFD solvers use conventional processing units. However, to get a general idea, we can consider the training wall clock time of the most efficient PINN model in this work (model V in Ch. 6) which lies in the range of 2100 s when using a single NVIDIA A 100 graphics card. In comparison, the same problem can be solved using CODA (see Appendix D) on a single Intel i7-9700 CPU in about 180 s. This amounts to an entire order of magnitude difference in wall clock time. Combined with the fact that the finite volume method is more accurate, we conclude that at the current point of time, non-parametric PINNs can not compete with classical method for solving forward problems for the compressible Euler equations. This is also the general sentiment in literature for other applications in aerodynamics. Only in a few cases, PINNs have been compared favorably to classical numerical methods. As recently highlighted in [123], such comparisons are also oftentimes biased and compare the performance to weak baselines. In addition, when designing a new PINN model, a lot of additional simulations are typically required for hyperparameter tuning. Changes to parameters like the learning rate, the batch size and the network dimensions may have a significant impact on the accuracy. In comparison to classical solvers, these parameters may often not have a reasonable physical interpretation. Hence, in practice, hyperparameter tuning may increase the effective time until a solution is obtained, by orders of magnitude. Due to the number of hyperparameters, automatic hyperparameter optimization is infeasible when dealing with more complex scenarios. Moreover, for complex cases ground truth or reference solutions might no longer be available making the search for a feasible set of hyperparameters even more challenging. Furthermore, it should be mentioned that at this point in time, applications of PINNs in fluid dynamics are mostly limited to comparatively simple problems, such as two dimensional flows and simple geometries. While methodologies such as the MT have been promising, they must be extended to unstructured grid to enable the simulation of flows around complex three-dimensional geometries. On the other hand, it is also worth considering that the pace at which progress was made in recent years is substantial. In literature, PINNs are being applied to more and more complex problems [87, 69], even for compressible flows. Furthermore, initial theoretical advancements for hyperbolic problems are being made [84, 85], an area that was previously lacking behind. Thus it is possible that with future improvements of the algorithms and more efficient software, PINNs will be able to match or outperform classical methods in certain tasks.

This perspective becomes even more relevant when we consider the possible application of PINNs as parametric solvers, bringing us to question 2 (b). In chapters 4-6, parametric variants of each of the analyzed forward problems were also investigated. For the subsonic flow around the cylinder, the geometry and the Mach number were parametrized. For the oblique shock, the Mach number was parametrized and for the transonic flow around the airfoil the angle of attack was parametrized. For all of these problems, we can obtain similarly accurate predictions for the parametric problems, as for the non-parametric problems. For both models in chapters 4 and 5 the training of the parametric models is about four times as expensive as the non-parametric problems, due to the increase in the number of training points and network dimensions. For Ch. 6, we even see that with a similar wall-clock time as for the non-parametric problem, we can train the parametric model. The potential of parametric PINN models in fluid dynamics has also been explored in other work [87, 69, 124, 125], confirming that parametric PINN variants can successfully learn solutions in a continuous parameter space and that the increase in training cost can be modest, even for problems involving up to 14 parameters. For highly parametric problems, a single PINN could thus potentially replace multiple separate runs of a classical solver, leading to an overall decrease in computational effort. If we consider, for example, the parametric model in Sec. 6.3 with a training time of 1 h, we would reach a break-even point when it would substitute about 30 evaluations of a CFD solver which take 180 s each. It is to be expected that the break-even point for higher dimensional parameter space can be reached even earlier. Moreover, once the PINN is trained, it can be evaluated at any parameter combination in fractions of a second, making the application to time critical scenarios, for example control problems a possibility. It needs to be considered however, that the method is currently limited to comparatively simple geometries and that the parametric PINN solutions are not as accurate as the results that can be obtained with an evaluation of the CFD solver for a single parameter combination. As a long term perspective and with future improvements of algorithms and software, parametric PINNs could become an important tool for many query applications, such as design exploration and optimization or real time applications. In that sense, they could serve as a surrogate which benefits from additional information provided by the partial differential equations.

8. Outlook

Throughout this work, answers to some of the open questions concerning the viability of PINNs for applications in aerodynamics were found. In the following, future directions and ideas are discussed, which could potentially build upon this work and address the remaining challenges. Furthermore, more broadly defined research directions are considered, which are, in the author's opinion, crucial for the future progress of PINN research. The following discussion starts with the near term future of PINNs and moves on to progressively more long-term potentials of the method.

Adaptive and Sensor-Controlled Artificial Viscosity Hybrids

The main contribution of this work are the newly developed AV methods which enable the solution of the compressible Euler equations under various flow conditions. When using adaptive viscosity for the oblique shock problem we have seen that the model can resolve the oblique shock more sharply. This is due to the fact that the adaptive AV localization is able to find the minimal amount of AV that is required to stabilize the training without excessive smoothing of the shock. However, the adaptive AV localization tends to perform too inconsistently when dealing with more complicated problems, such as transonic flows around airfoil. The sensor-controlled AV performs more robustly. However, since we manually select the viscosity magnitude with the factor ν, the resulting solution can be relatively dissipative. In the future, combinations of adaptive AV and sensor controlled AV could help to reduce dissipativeness of the method, where an optimal distribution of ν is learned only in the active range of the sensor.

Neural Networks as suitable Ansatz Functions for Conservation Laws

A point that has perhaps received little attention so far is that, until now, the PINN algorithm has been viewed in its entirety. In contrast, it is of course also possible to focus on individual components of the method and pick out the most promising ones. And it is natural to understand the NNs as ansatz functions for the discretization of the partial differential equations. In this sense, based on the observations we made, these functions seem promising candidates to be used in classical methods. It seems that these functions, for an adequate choice of parameters, have the potential to approximate shocks and other characteristics of solutions of transport-type PDEs. Integrating these into classical methods, as previously seen with polynomial approaches for discontinuous Galerkin methods [3, pp. 38-39], could open up new possibilities to precisely determine shocks or other properties of the solutions by selecting certain parameters in the ansatz functions. Future work may also concentrate on this topic to extend existing CFD software.

Training Augmentation with Data

Besides the solution of boundary value problems, PINNs are often used for inverse problems. Some form of solution data (experimental or synthetic), is used in an additional supervised loss term, which is added to the other loss terms and some input information of the forward problem can then be reconstructed. The possibility of handling such inverse problems is a unique capability that PINNs have compared to classical solvers. In our opinion, when solution data is included in the loss function, some works fail to point out the difference to the strict forward problem. Depending on the amount of data that is used for the training, the resulting model may mainly interpolate between this data instead of learning from physics. Hence, such models should always be compared to purely data-driven approaches. In addition, it is highly likely that the provided data is not a solution to the investigated PDE itself, leading to a training conflict in which the network can either learn behavior according to data, the PDE or an unknown compromise between these two. However, when used correctly, the integration of data into the loss offers an interesting opportunity. For example, when training a parametric PINN model for, say the flow around an airfoil, the model could potentially be enhanced with more accurate solution data coming from a CFD solver. The additional data can improve the accuracy of the model at critical parameter values or be used to enhance the convergence of the model. Similarly to a surrogate based model, the local residual of the PINN at certain parameter configurations could also provide an indication for where in the parameter space additional higher fidelity data is required to improve the model. As previously mentioned, one needs to ensure that the used data is in agreement with the underlying PDE to avoid training conflicts.

Domain Decomposition for Highly Parametric Models

For highly parametric problems or more complex geometries, a global ansatz function, such as the NN, might be unable to approximate the solution or at least require very large networks leading to high memory requirements. Domain decomposition approaches for PINNs such as finite basis PINNs [48], could potentially address these limitations. By splitting the problem into smaller subdomains, each covered by a separate NN, the complexity of the solution for each subdomain can be reduced. When utilizing parallel training routines for each subdomain, the domain decomposition can also help to decrease wall-clock times. When extending PINNs to more complex problems in the future, domain decomposition might become indispensable.

Acceleration Potential with Improved Optimization Algorithms

One of the main limitations of PINNs are the long training times, caused by the slow convergence rates during training. By design, the loss function consists of multiple terms which themselves include higher order derivatives, leading to a loss landscape which, in many cases, is difficult to navigate for optimizers. This results in long wall-clock times as previously discussed. Not only do PINNs often converge to local minima but the minima that they converge to are also inconsistent. This means that for a different initial state of the NN weights and biases, we may obtain results which are orders of magnitude more or less accurate, even though the exact same hyperparameters are used. This is a crucial issue that is often ignored when results are reported. Often, only single training runs are shown, which gives little insight in the actual average performance of an algorithm. Average accuracies and variances should be reported instead.

It is likely that new optimization algorithms, specifically designed for PINNs and even specific types of PDEs, are required to reliably enhance convergence rates and consistency. At the moment, PINNs are still predominantly trained with stochastic gradient descent algorithms, developed for standard deep learning tasks. Quasi Newton algorithms such as L-BFGS [31] have been more effective for PINNs but are often less consistent and also inadequate for stochastic training. Recent works report significant benefits of using more advanced optimization algorithms [126, 127, 69] which can potentially address the convergence problems.

Acceleration Potential with Quantum Computers

An additional future point of interest is the modification of the algorithm towards using quantum computers. Here, the neural network is replaced by a variational quantum circuit which is optimized, instead of the neural network [128, 129]. This approach was labeled a physics-informed quantum circuit (PIQC). Initial experiments on one-dimensional transient problems show promising result. However, due to the limited availability of quantum hardware, it is unclear whether this approach offers significant acceleration potential. Because of the differences in the underlying function spaces, covered by the variational quantum circuit, compared to the classical neural network, it should also be further investigated if the quantum circuit is able to approximate discontinuities well. Siegl et al. [129] showed initial evidence that problems with discontinuous solutions might be more difficult for the quantum computing based approach.

Towards Applications of Industrial Complexity

As discussed in the introduction, for today's aerodynamics solvers in the aviation industry, RANS models are still predominantly used, since simulations with higher fidelity than RANS are still computationally too expensive. For PINNs to reach a similar problem complexity, their use has to be extended to the compressible RANS equations first. In literature, initial use cases of PINNs for solving the incompressible RANS equations have been shown [70, 89]. Potentially, these approaches need to be combined with shock capturing capabilities, as presented in this work, to take the next step towards the compressible RANS equations. Secondly, the application to complex geometries presents a significant challenge. The use of mesh-transformation has been essential for obtaining reasonable predictions around airfoil geometries, so far. However, for complex, three-dimensional geometries, the generation of curvilinear grids becomes a major challenge. For future applications of PINNs, it is possible that block structured grid may be required, where separate coordinate transforms are defined for each block. However, it should be noted that the mesh transformation for PINNs is merely a tool to precondition the underlying optimization problem that is solved during training. With the development of more powerful optimizers or other preconditioning techniques, the mesh-transformation might not be required anymore. Alternatively, in analogy to a hybrid-mesh approach, mesh-transformations might only be used in structured grid blocks, for instance in boundary layers, while randomly distributed points or unstructured grid are combined with other preconditioning techniques in the remaining blocks. The previously discussed domain decomposition approaches could be combined with such a multi-block hybrid mesh to also train the different blocks in parallel on multiple GPUs. This might also be more important for complex geometries as the memory of single graphics cards might become a limiting factor. Furthermore, general acceleration techniques may be required to train PINNs more efficiently for industrial applications. Several options to accelerate PINN training were already discussed in the previous sections. Lastly, more efficient software frameworks that rely on just-in-time compilation, for instance the recently popularized JAX library [130], can be use to further optimize the efficiency of PINNs, significantly reducing the training time.

Conclusively, for applications in aerodynamics, the potential of parametric PINN solvers appears to be just as significant as the open challenges that must be overcome before industrially relevant problems can be solved. Considering that the original paper by Raissi et al. [29] is still increasingly cited, even five year after the original publication, it can be inferred that the development of PINNs will not slow down in the near future. Therefore, it is not unlikely that the methods will become an important tool for aerodynamics in the industrial context in the next decade, especially when applied to multi-query scenarios. Significant potential also lies in the combination of existing numerical frameworks with PINNs. Future PINN software needs to be integrated into the established modeling pipelines so that they can be combined with computer-aided design software, enabling the application to complex geometries. The advancements in efficiency required for complex applications will most likely be only possible with an improved theoretical understanding of the method. For application in high speed aerodynamics, this implies that the algorithm needs to be adapted to the mathematical intricacies of conservation laws, similarly to finite volume methods in the past. Hopefully, the presented work and other recent developments can help to forge a way towards more efficient and accurate neural network based solution methods for partial differential equations in aerodynamics and fluids mechanics.

References

[1] John David Anderson. *Fundamentals of aerodynamics*. 5. ed. in SI Units. McGraw-Hill series in aeronautical and aerospace engineering. New York, NY: McGraw-Hill, 2011. ISBN: 978-1-259-01028-6.

[2] John David Anderson. *Computational fluid dynamics: The basics with applications*. McGraw-Hill series in aeronautical and aerospace engineering. New York: McGraw-Hill, 1995. ISBN: 9780070016859.

[3] Jiri Blazek. *Computational fluid dynamics: Principles and applications*. Third edition. Amsterdam, Boston, and Heidelberg: Butterworth-Heinemann an imprint of Elsevier, 2015. ISBN: 9780080999951.

[4] Merle C. Potter and Bassem H. Ramadan. *An Introduction to Fluid Mechanics*. Cham: Springer International Publishing, 2025. ISBN: 978-3-031-64819-9. DOI: 10.1007/978-3-031-64820-5.

[5] Charles L. Fefferman. “Existence and Smoothness of the Navier–Stokes Equation”. In: *The Millennium Prize Problems*, pp. 51–61.

[6] Stefan Langer. *Preconditioned Newton methods to approximate solutions of the Reynolds averaged Navier-Stokes equations*. 2018. URL: https://elib.dlr.de/121515/.

[7] Lawrence C. Evans. *Partial differential equations*. Reprinted with corr. Vol. 19. Graduate studies in mathematics. Providence, RI: American Mathematical Society, 2008. ISBN: 0821807722.

[8] Philipp Bekemeyer et al. “Data-Driven Aerodynamic Modeling Using the DLR SMARTy Toolbox”. In: *AIAA AVIATION 2022 Forum*. Reston, Virginia: American Institute of Aeronautics and Astronautics, 2022. ISBN: 978-1-62410-635-4. DOI: 10.2514/6.2022-3899.

[9] Philipp Bekemeyer et al. “Introduction of Applied Aerodynamics Surrogate Modeling Benchmark Cases”. In: *AIAA SCITECH 2025 Forum*. Reston, Virginia: American Institute of Aeronautics and Astronautics, 2025. ISBN: 978-1-62410-723-8. DOI: 10.2514/6.2025-0036.

[10] Robert K. Remple and Mark B. Tischler. *Aircraft and Rotorcraft System Identification*. Reston ,VA: American Institute of Aeronautics and Astronautics, 2006. ISBN: 978-1-56347-837-6. DOI: 10.2514/4.861352.

[11] M. J. D. Powell. “Radial basis functions for multivariable interpolation: a review”. In: 1987. URL: https://api.semanticscholar.org/CorpusID:118224933.

[12] T. W. Simpson et al. “Metamodels for Computer-based Engineering Design: Survey and recommendations”. In: *Engineering with Computers* 17.2 (2001), pp. 129–150. ISSN: 0177-0667. DOI: 10.1007/PL00007198.

[13] David J. Lucia, Philip S. Beran, and Walter A. Silva. “Reduced-order modeling: new approaches for computational physics”. In: *Progress in Aerospace Sciences* 40.1-2 (2004), pp. 51–117. ISSN: 03760421. DOI: 10.1016/j.paerosci.2003.12.001.

[14] Ralf Zimmermann and Stefan Görtz. “Non-linear reduced order models for steady aerodynamics”. In: *Procedia Computer Science* 1.1 (2010), pp. 165–174. ISSN: 18770509. DOI: 10.1016/j.procs.2010.04.019.

[15] P. Bekemeyer et al. “Nonlinear Unsteady Reduced-Order Modeling for Gust-Load Predictions”. In: *AIAA Journal* 57.5 (2019), pp. 1839–1850. ISSN: 0001-1452. DOI: 10.2514/1.J057804.

[16] Emre Yilmaz and Brian German. “A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance”. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Reston, Virginia: American Institute of Aeronautics and Astronautics, 2017. ISBN: 978-1-62410-507-4. DOI: 10.2514/6.2017-3660.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

[21] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Red Hook, NY, USA: Curran Associates Inc, 2017, pp. 6000–6010. ISBN: 9781510860964.

[22] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. URL: http://arxiv.org/pdf/1912.01703v1.

[23] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[24] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Second. Society for Industrial and Applied Mathematics, 2008. DOI: 10.1137/1.9780898717761.

[25] Derrick Hines and Philipp Bekemeyer. “Graph neural networks for the prediction of aircraft surface pressure distributions”. In: *Aerospace Science and Technology* 137 (2023), p. 108268. ISSN: 12709638. DOI: 10.1016/j.ast.2023.108268.

[26] Abhijith Moni, Weigang Yao, and Hossein Malekmohamadi. “Data-driven reduced-order modeling for nonlinear aerodynamics using an autoencoder neural network”. In: *Physics of Fluids* 36.1 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0177577.

[27] Mateus Dias Ribeiro, Mario Stradtner, and Philipp Bekemeyer. “Unsteady reduced order model with neural networks and flight-physics-based regularization for aerodynamic applications”. In: *Computers & Fluids* 264 (2023), p. 105949. ISSN: 00457930. DOI: 10.1016/j.compfluid.2023.105949.

[28] George Em Karniadakis et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[29] M. Raissi, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 00219991. DOI: 10.1016/j.jcp.2018.10.045.

[30] M. W. M. G. Dissanayake and N. Phan-Thien. “Neural-network-based approximations for solving partial differential equations”. In: *Communications in Numerical Methods in Engineering* 10.3 (1994), pp. 195–201. ISSN: 10698299. DOI: 10.1002/cnm.1640100303.

[31] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical Programming* 45.1-3 (1989), pp. 503–528. ISSN: 0025-5610. DOI: 10.1007/BF01589116.

[32] I. E. Lagaris, A. Likas, and D. I. Fotiadis. “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. ISSN: 10459227. DOI: 10.1109/72.712178.

[33] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Second edition. Springer series in operation research and financial engineering. New York, NY: Springer, 2006. ISBN: 978-0-387-30303-1. URL: http://www.loc.gov/catdir/enhancements/fy0818/2006923897-d.html.

[34] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. “Neural-network methods for boundary value problems with irregular boundaries”. In: *IEEE Transactions on Neural Networks* 11.5 (2000), pp. 1041–1049. ISSN: 10459227. DOI: 10.1109/72.870037.

[35] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017. DOI: 10.48550/arXiv.1711.10561.

[36] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. DOI: 10.48550/arXiv.1711.10566.

[37] Salvatore Cuomo et al. “Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What’s Next”. In: *Journal of Scientific Computing* 92.3 (2022). DOI: 10.1007/s10915-022-01939-z.

[38] Juan Diego Toscano et al. “From PINNs to PIKANs: recent advances in physics-informed machine learning”. In: *Machine Learning for Computational Science and Engineering* 1.1 (2025). ISSN: 3005-1428. DOI: 10.1007/s44379-025-00015-1.

[39] Umair bin Waheed et al. “PINNeik: Eikonal solution using physics-informed neural networks”. In: *Computers & Geosciences* 155 (2021), p. 104833. ISSN: 00983004. DOI: 10.1016/j.cageo.2021.104833.

[40] Yanghua Wang. *Seismic Inversion.* Wiley, 2016. ISBN: 9781119257981. DOI: 10.1002/9781119258032.

[41] Sifan Wang, Yujun Teng, and Paris Perdikaris. "Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks". In: *SIAM Journal on Scientific Computing* 43.5 (2021), A3055–A3081. ISSN: 1064-8275. DOI: 10.1137/20M1318043.

[42] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks". In: *Proceedings. Mathematical, physical, and engineering sciences* 476.2239 (2020), p. 20200334. ISSN: 1364-5021. DOI: 10.1098/rspa.2020.0334.

[43] Donatien Hainaut and Alex Casas. "Option pricing in the Heston model with physics inspired neural networks". In: *Annals of Finance* 20.3 (2024), pp. 353–376. ISSN: 1614-2446. DOI: 10.1007/s10436-024-00452-7.

[44] Baltasar Rüchardt. *Reconstructing the electrical dynamics on the heart.* 2022. DOI: 10.53846/goediss-9545.

[45] Joakim Sundnes. *Computing the Electrical Activity in the Heart.* Vol. v.1. Monographs in Computational Science and Engineering Ser. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2006. ISBN: 9783540334378. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=371600.

[46] Carlos Ruiz Herrera et al. "Physics-informed neural networks to learn cardiac fiber orientation from multiple electroanatomical maps". In: *Engineering with Computers* 38.5 (2022), pp. 3957–3973. ISSN: 0177-0667. DOI: 10.1007/s00366-022-01709-3.

[47] Federico Antonello, Jacopo Buongiorno, and Enrico Zio. "Physics informed neural networks for surrogate modeling of accidental scenarios in nuclear power plants". In: *Nuclear Engineering and Technology* 55.9 (2023), pp. 3409–3416. ISSN: 17385733. DOI: 10.1016/j.net.2023.06.027.

[48] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. "Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations". In: *Advances in Computational Mathematics* 49.4 (2023). ISSN: 1019-7168. DOI: 10.1007/s10444-023-10065-9.

[49] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems". In: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), p. 113028. ISSN: 00457825. DOI: 10.1016/j.cma.2020.113028.

[50] Ameya D. Jagtap and George Em Karniadakis. "Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations". In: *Communications in Computational Physics* 28.5 (2020), pp. 2002–2041. ISSN: 1815-2406. DOI: 10.4208/cicp.OA-2020-0164.

[51] Axel Klawonn, Martin Lanser, and Janine Weber. “Machine learning and domain decomposition methods - a survey”. In: *Computational Science and Engineering* 1.1 (2024). DOI: 10.1007/s44207-024-00003-y.

[52] Xiaowei Jin et al. “NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 426 (2021), p. 109951. ISSN: 00219991. DOI: 10.1016/j.jcp.2020.109951.

[53] Suryanarayana Maddu et al. “Inverse Dirichlet weighting enables reliable training of physics informed neural networks”. In: *Machine Learning: Science and Technology* 3.1 (2022), p. 015026. DOI: 10.1088/2632-2153/ac3712.

[54] Sifan Wang et al. *An Expert’s Guide to Training Physics-informed Neural Networks.* 2023. DOI: 10.48550/arXiv.2308.08468.

[55] N. Sukumar and Ankit Srivastava. “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333. ISSN: 00457825. DOI: 10.1016/j.cma.2021.114333.

[56] Luning Sun et al. “Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data”. In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112732. ISSN: 00457825. DOI: 10.1016/j.cma.2019.112732.

[57] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks”. In: *Journal of Computational Physics* 404 (2020), p. 109136. ISSN: 00219991. DOI: 10.1016/j.jcp.2019.109136.

[58] Ameya D. Jagtap et al. “Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions”. In: *Neurocomputing* 468 (2022), pp. 165–180. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.10.036.

[59] Vincent Sitzmann et al. “Implicit Neural Representations with Periodic Activation Functions”. In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc, 2020, pp. 7462–7473. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/53c04118df112c13a8c34b38343b9c10-Paper.pdf.

[60] Michael Mommert et al. “Periodically activated physics-informed neural networks for assimilation tasks for three-dimensional Rayleigh–Bénard convection”. In: *Computers & Fluids* 283 (2024), p. 106419. ISSN: 00457930. DOI: 10.1016/j.compfluid.2024.106419.

[61] Ge Jin et al. “Improve neural representations with general exponential activation function for high-speed flows”. In: *Physics of Fluids* 36.12 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0239889.

[62] Ameya D. Jagtap and George Em Karniadakis. “HOW IMPORTANT ARE ACTIVATION FUNCTIONS IN REGRESSION AND CLASSIFICATION? A SURVEY, PERFORMANCE COMPARISON, AND FUTURE DIRECTIONS”. In: *Journal of Machine Learning for Modeling and Computing* 4.1 (2023), pp. 21–75. ISSN: 2689-3967. DOI: 10.1615/JMachLearnModelComput.2023047367.

[63] Matthew Tancik et al. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. DOI: 10.48550/arXiv.2006.10739.

[64] Sifan Wang, Hanwen Wang, and Paris Perdikaris. “On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 384 (2021), p. 113938. ISSN: 00457825. DOI: 10.1016/j.cma.2021.113938.

[65] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science (New York, N.Y.)* 367.6481 (2020), pp. 1026–1030. DOI: 10.1126/science.aaw4741.

[66] Shengze Cai et al. “Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks”. In: *Journal of Fluid Mechanics* 915 (2021), A102 <div></div>. ISSN: 0022-1120. DOI: 10.1017/jfm.2021.135.

[67] B. Steinfurth and J. Weiss. “Assimilating experimental data of a mean three-dimensional separated flow using physics-informed neural networks”. In: *Physics of Fluids* 36.1 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0183463.

[68] Wenbo Cao and Weiwei Zhang. *TSONN: Time-stepping-oriented neural network for solving partial differential equations*. 2023. DOI: 10.48550/arXiv.2310.16491.

[69] Wenbo Cao and Weiwei Zhang. “An analysis and solution of ill-conditioning in physics-informed neural networks”. In: *Journal of Computational Physics* 520 (2025), p. 113494. ISSN: 00219991. DOI: 10.1016/j.jcp.2024.113494.

[70] Hamidreza Eivazi et al. “Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations”. In: *Physics of Fluids* 34.7 (2022), p. 075117. ISSN: 1070-6631. DOI: 10.1063/5.0095270.

[71] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. “Physics-informed neural networks for high-speed flows”. In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112789. ISSN: 00457825. DOI: 10.1016/j.cma.2019.112789.

[72] Simon Wassing, Stefan Langer, and Philipp Bekemeyer. “Physics-informed neural networks for parametric compressible Euler equations”. In: *Computers & Fluids* 270 (2024), p. 106164. ISSN: 00457930. DOI: 10.1016/j.compfluid.2023.106164.

[73] Thomas Wagenaar. “Physics-informed neural networks for highly compressible flows: assessing and enhancing shock-capturing capabilities”. PhD thesis. Delft University of Technology, 2023.

[74] Olga Fuks and Hamdi A. Tchelepi. “LIMITATIONS OF PHYSICS INFORMED MACHINE LEARNING FOR NONLINEAR TWO-PHASE TRANSPORT IN POROUS MEDIA”. In: *Journal of Machine Learning for Modeling and Computing* 1.1 (2020), pp. 19–37. ISSN: 2689-3967. DOI: 10.1615/JMachLearnModelComput.2020033905.

[75] R. Swanson, R. Radespiel, and E. Turkel. “Comparison of several dissipation algorithms for central difference schemes”. In: *13th Computational Fluid Dynamics Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics, 1997. DOI: 10.2514/6.1997-1945.

[76] A. Jameson, W. Schmidt, and E. Turkel. “Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes”. In: *14th Fluid and Plasma Dynamics Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics, 1981. DOI: 10.2514/6.1981-1259.

[77] R. C. Swanson and E. Turkel. “Artificial dissipation and central difference schemes for the Euler and Navier-Stokes equations”. In: *AIAA Paper, AIAA-1987-1107* (1987).

[78] E. Turkel. “Improving the accuracy of central difference schemes”. In: *11th International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics*. Ed. by D. L. Dwoyer and M. Y. Hussaini. Vol. 323. Springer-Verlag, 1989.

[79] Stefan Langer and Axel Schwöppe. *TAU Hyperflex Report*. Ed. by DLR. 2022. URL: https://elib.dlr.de/188909/.

[80] Ravi G. Patel et al. “Thermodynamically consistent physics-informed neural networks for hyperbolic systems”. In: *Journal of Computational Physics* 449 (2022), p. 110754. ISSN: 00219991. DOI: 10.1016/j.jcp.2021.110754.

[81] Emilio Jose Rocha Coutinho et al. “Physics-informed neural networks with adaptive localized artificial viscosity”. In: *Journal of Computational Physics* 489 (2023), p. 112265. ISSN: 00219991. DOI: 10.1016/j.jcp.2023.112265.

[82] Li Liu et al. “Discontinuity Computing Using Physics-Informed Neural Networks”. In: *Journal of Scientific Computing* 98.1 (2024). ISSN: 0885-7474. DOI: 10.1007/s10915-023-02412-1.

[83] Emmanuel Lorin and Arian Novruzi. “Non-diffusive neural network method for hyperbolic conservation laws”. In: *Journal of Computational Physics* 513 (2024), p. 113161. ISSN: 00219991. DOI: 10.1016/j.jcp.2024.113161.

[84] Tim de Ryck, Siddhartha Mishra, and Roberto Molinaro. “wPINNs: Weak Physics Informed Neural Networks for Approximating Entropy Solutions of Hyperbolic Conservation Laws”. In: *SIAM Journal on Numerical Analysis* 62.2 (2024), pp. 811–841. ISSN: 0036-1429. DOI: 10.1137/22M1522504.

[85] Aidan Chaumet and Jan Giesselmann. “Improving Weak PINNs for Hyperbolic Conservation Laws: Dual Norm Computation, Boundary Conditions and Systems”. In: *The SMAI Journal of computational mathematics* 10 (2024), pp. 373–401. DOI: 10.5802/smai-jcm.116.

[86] Wenbo Cao, Jiahao Song, and Weiwei Zhang. "A solver for subsonic flow around airfoils based on physics-informed neural networks and mesh transformation". In: *Physics of Fluids* 36.2 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0188665.

[87] Wenbo Cao, Jiahao Song, and Weiwei Zhang. "Solving high-dimensional parametric engineering problems for inviscid flow around airfoils based on physics-informed neural networks". In: *Journal of Computational Physics* 516 (2024), p. 113285. ISSN: 00219991. DOI: 10.1016/j.jcp.2024.113285.

[88] Wenbo Cao et al. *Solving all laminar flows around airfoils all-at-once using a parametric neural network solver.* URL: http://arxiv.org/pdf/2501.01165v1.

[89] Wenbo Cao et al. "Solving parametric high-Reynolds-number wall-bounded turbulence around airfoils governed by Reynolds-averaged Navier–Stokes equations using time-stepping-oriented neural network". In: *Physics of Fluids* 37.1 (2025). ISSN: 1070-6631. DOI: 10.1063/5.0245918.

[90] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: a review". In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738. ISSN: 0567-7718. DOI: 10.1007/s10409-021-01148-1.

[91] Chi Zhao et al. "A comprehensive review of advances in physics-informed neural networks and their applications in complex fluid dynamics". In: *Physics of Fluids* 36.10 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0226562.

[92] Siddhartha Mishra. *Numerical methods for conservation laws and related equations.* URL: https://web.iitd.ac.in/~vvksrini/Oldhomepage/smishra.pdf.

[93] Stefan Langer, Axel Schwöppe, and Norbert Kroll. "The DLR Flow Solver TAU - Status and Recent Algorithmic Developments". In: *Proc. 52nd Aerospace Sciences Meeting, January 2014.* Conference Proceeding Series. AIAA, 2014. DOI: 10.2514/6.2014-0080.

[94] Stefan Langer, Axel Schwöppe, and Tobias Leicht. "Comparison and Unification of Finite-Volume Discretization Strategies for the Unstructured Node-Centered and Cell-Centered Grid Metric in TAU and CODA". In: *New Results in Numerical and Experimental Fluid Mechanics XIV*. Ed. by Andreas Dillmann et al. Vol. 154. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Cham: Springer Nature Switzerland, 2024, pp. 262–272. ISBN: 978-3-031-40481-8. DOI: 10.1007/978-3-031-40482-5{\textunderscore}25.

[95] Simon Wassing, Stefan Langer, and Philipp Bekemeyer. "Adopting Computational Fluid Dynamics Concepts for Physics-Informed Neural Networks". In: *AIAA SCITECH 2025 Forum.* Reston, Virginia: American Institute of Aeronautics and Astronautics, 2025. ISBN: 978-1-62410-723-8. DOI: 10.2514/6.2025-0269.

[96] R.C Swanson and Eli Turkel. "On central-difference and upwind schemes". In: *Journal of Computational Physics* 101.2 (1992), pp. 292–306. ISSN: 00219991. DOI: 10.1016/0021-9991(92)90007-L.

[97] P. L. Roe. "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes". In: *Journal of Computational Physics* 135.2 (1997), pp. 250–258. ISSN: 00219991. DOI: 10.1006/jcph.1997.5705.

[98] Zhou Lu et al. “The Expressive Power of Neural Networks: A View from the Width”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf.

[99] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. *An Alternative View: When Does SGD Escape Local Minima?* 2018. DOI: 10.48550/arXiv.1802.06175.

[100] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. URL: http://arxiv.org/pdf/1412.6980v9.

[101] Albert S. Berahas, Jorge Nocedal, and Martin Takac. “A Multi-Batch L-BFGS Method for Machine Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc, 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/8ebda540cbcc4d7336496819a46a1b68-Paper.pdf.

[102] Xiao Wang et al. “Stochastic Quasi-Newton Methods for Nonconvex Stochastic Optimization”. In: *SIAM Journal on Optimization* 27.2 (2017), pp. 927–956. ISSN: 1052-6234. DOI: 10.1137/15M1053141.

[103] Tim Salimans and Diederik P. Kingma. *Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks*. 2016. DOI: 10.48550/arXiv.1602.07868.

[104] Nasim Rahaman et al. “On the Spectral Bias of Neural Networks”. In: (2018). DOI: 10.48550/arXiv.1806.08734.

[105] Zhi-Qin John Xu, Yaoyu Zhang, and Tao Luo. *Overview frequency principle/spectral bias in deep learning*. 2022. DOI: 10.48550/arXiv.2201.07395.

[106] Russel E. Caflisch. “Monte Carlo and quasi-Monte Carlo methods”. In: *Acta Numerica* 7 (1998), pp. 1–49. ISSN: 0962-4929. DOI: 10.1017/S0962492900002804.

[107] Sokratis J. Anagnostopoulos et al. “Residual-based attention in physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 421 (2024), p. 116805. ISSN: 00457825. DOI: 10.1016/j.cma.2024.116805.

[108] Marina Danilova et al. “Recent Theoretical Advances in Non-Convex Optimization”. In: *High-Dimensional Optimization and Probability*. Ed. by Ashkan Nikeghbali et al. Vol. 191. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2022, pp. 79–163. ISBN: 978-3-031-00831-3. DOI: 10.1007/978-3-031-00832-0{\textunderscore}3.

[109] I. M. Sobol’. “On the distribution of points in a cube and the approximate evaluation of integrals”. In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967), pp. 86–112. ISSN: 0041-5553. DOI: 10.1016/0041-5553(67)90144-9.

[110] J. H. Halton. “On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals”. In: *Numerische Mathematik* 2.1 (1960), pp. 84–90. ISSN: 0945-3245. DOI: 10.1007/BF01386213.

[111] M. D. McKay, R. J. Beckman, and W. J. Conover. “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In: *Technometrics* 21.2 (1979), p. 239. ISSN: 00401706. DOI: 10.2307/1268522.

[112] Lu Lu et al. “DeepXDE: A Deep Learning Library for Solving Differential Equations”. In: *SIAM Review* 63.1 (2021), pp. 208–228. DOI: 10.1137/19M1274067.

[113] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. “Efficient training of physics–informed neural networks via importance sampling”. In: *Computer-Aided Civil and Infrastructure Engineering* 36.8 (2021), pp. 962–977. ISSN: 1093-9687. DOI: 10.1111/mice.12685.

[114] Matthew McCoy. “A comparison of finite difference methods for solving Laplace’s equation on curvilinear coordinate systems. M.S. Thesis”. In: vol. NASA-CR-163236. 1980. URL: https://ntrs.nasa.gov/citations/19800017591.

[115] Jiahao Song et al. “VW-PINNs: A volume weighting method for PDE residuals in physics-informed neural networks”. In: *Acta Mechanica Sinica* 41.3 (2025). ISSN: 0567-7718. DOI: 10.1007/s10409-024-24140-x.

[116] Tim de Ryck, Siddhartha Mishra, and Roberto Molinaro. *wPINNs: Weak Physics informed neural networks for approximating entropy solutions of hyperbolic conservation laws*. 2022. DOI: 10.48550/arXiv.2207.08483.

[117] S. Wassing, S. Langer, and P. Bekemeyer. “Parametric Compressible Flow Predictions using Physics-Informed Neural Networks”. In: *8th European Congress on Computational Methods in Applied Sciences and Engineering*. CIMNE, 5th - 9th Jun 2022. DOI: 10.23967/eccomas.2022.217.

[118] Simon Wassing, Stefan Langer, and Philipp Bekemeyer. “Artificial Viscosity in Physics-Informed Neural Networks for Parametric Compressible Flows”. In: *SSRN Electronic Journal* (2023). DOI: 10.2139/ssrn.4353534.

[119] S. Wassing, S. Langer, and P. Bekemeyer. “Physics-informed neural networks for inviscid transonic flows around an airfoil”. In: *Physics of Fluids* 37.8 (2025). ISSN: 1070-6631. DOI: 10.1063/5.0276518.

[120] Simon Wassing, Stefan Langer, and Philipp Bekemeyer. *Physics-Informed Neural Networks for Parametric Compressible Euler Equations*. 2023. DOI: 10.48550/arXiv.2307.14045.

[121] Bengt Fornberg. “Generation of finite difference formulas on arbitrarily spaced grids”. In: *Mathematics of Computation* 51.184 (1988), pp. 699–706. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-1988-0935077-0.

[122] Jiahao Song et al. *VW-PINNs: A volume weighting method for PDE residuals in physics-informed neural networks*. 2024. DOI: 10.48550/arXiv.2401.06196.

[123] Nick McGreivy and Ammar Hakim. “Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations”. In: *Nature Machine Intelligence* 6.10 (2024), pp. 1256–1269. DOI: 10.1038/s42256-024-00897-5.

[124] F. Lorenzen, A. Zargaran, and U. Janoske. “Potential of physics-informed neural networks for solving fluid flow problems with parametric boundary conditions”. In: *Physics of Fluids* 36.3 (2024). ISSN: 1070-6631. DOI: 10.1063/5.0193952.

[125] Woojin Cho et al. *Parameterized Physics-informed Neural Networks for Parameterized PDEs.* 2024. DOI: 10.48550/arXiv.2408.09446.

[126] Sifan Wang et al. *Gradient Alignment in Physics-informed Neural Networks: A Second-Order Optimization Perspective.* URL: http://arxiv.org/pdf/2502.00604v1.

[127] Elham Kiyani et al. *Which Optimizer Works Best for Physics-Informed Neural Networks and Kolmogorov-Arnold Networks?* URL: http://arxiv.org/pdf/2501.16371v3.

[128] Oleksandr Kyriienko, Annie E. Paine, and Vincent E. Elfving. "Solving nonlinear differential equations with differentiable quantum circuits". In: *Physical Review A* 103.5 (2021). ISSN: 2469-9926. DOI: 10.1103/PhysRevA.103.052416.

[129] Pia Siegl et al. "Solving transport equations on quantum computers—potential and limitations of physics-informed quantum circuits". In: *CEAS Aeronautical Journal* (2024). ISSN: 1869-5582. DOI: 10.1007/s13272-024-00774-2.

[130] Roy Frostig, Matthew Johnson, and Chris Leary. "Compiling machine learning programs via high-level tracing". In: 2018. URL: https://mlsys.org/Conferences/doc/2018/146.pdf.

[131] Philipp Bekemeyer et al. "Recent Advances in Data-Driven Modeling for Aerodynamic Applications using DLR's SMARTy Toolbox". In: *AIAA SCITECH 2024 Forum.* Reston, Virginia: American Institute of Aeronautics and Astronautics, 2024. ISBN: 978-1-62410-711-5. DOI: 10.2514/6.2024-0010.

[132] Levi D. McClenny and Ulisses M. Braga-Neto. "Self-adaptive physics-informed neural networks". In: *Journal of Computational Physics* 474 (2023), p. 111722. ISSN: 00219991. DOI: 10.1016/j.jcp.2022.111722.

A. PINN Implementation Details

In the following, additional insights into the model setup are given, including the explicit formulas of the derivative reconstruction in the mesh-transformation, a detailed list of all used hyperparameters, and additional information on the general software framework.

A.1. Reconstruction of Derivatives in Mesh Transformation

The inverse metric terms are the derivatives of x and y with respect to the computational coordinates ξ and η. Given the discrete mapping in Eq. (2.102), they can be approximated with finite differences. Using the inverse mesh metric terms, the first derivatives of a function f with respect to the physical coordinates x and y can then be reconstructed as:

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{1}{J}\left[\frac{\partial f}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial f}{\partial \eta}\frac{\partial y}{\partial \xi}\right],\\ \frac{\partial f}{\partial y} &= \frac{1}{J}\left[\frac{\partial f}{\partial \eta}\frac{\partial x}{\partial \xi} - \frac{\partial f}{\partial \xi}\frac{\partial x}{\partial \eta}\right],\end{aligned}$$

with the Jacobian of the MT

$$J = \det\begin{pmatrix}\frac{\partial x}{\partial \xi}, & \frac{\partial y}{\partial \xi}\\ \frac{\partial x}{\partial \eta}, & \frac{\partial y}{\partial \eta}\end{pmatrix}.$$

The second derivatives are reconstructed as

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2} = \frac{1}{J^2}&\left[\frac{\partial^2 f}{\partial \xi^2}\left(\frac{\partial y}{\partial \eta}\right)^2 - 2\frac{\partial^2 f}{\partial \xi \partial \eta}\frac{\partial y}{\partial \xi}\frac{\partial y}{\partial \eta} + \frac{\partial^2 f}{\partial \eta^2}\left(\frac{\partial y}{\partial \xi}\right)^2\right]\\ &+\left[\left(\frac{\partial y}{\partial \eta}\right)^2\frac{\partial^2 y}{\partial \xi^2} - 2\frac{\partial y}{\partial \xi}\frac{\partial y}{\partial \eta}\frac{\partial^2 y}{\partial \xi \partial \eta} + \left(\frac{\partial y}{\partial \xi}\right)^2\frac{\partial^2 y}{\partial \eta^2}\right]\\ &\left[\frac{\partial f}{\partial \xi}\frac{\partial x}{\partial \eta} - \frac{\partial f}{\partial \eta}\frac{\partial x}{\partial \xi}\right]\\ &+\frac{1}{J^3}\left[\left(\frac{\partial y}{\partial \eta}\right)^2\frac{\partial^2 x}{\partial \xi^2} - 2\frac{\partial y}{\partial \xi}\frac{\partial y}{\partial \eta}\frac{\partial^2 x}{\partial \xi \partial \eta} + \left(\frac{\partial y}{\partial \xi}\right)^2\frac{\partial^2 x}{\partial \eta^2}\right]\\ &\left[\frac{\partial f}{\partial \eta}\frac{\partial y}{\partial \xi} - \frac{\partial f}{\partial \xi}\frac{\partial y}{\partial \eta}\right],\end{aligned}$$

for the x-coordinate and

$$\begin{aligned}\frac{\partial^2 f}{\partial y^2} = \frac{1}{J^2} & \left[\frac{\partial^2 f}{\partial \xi^2}\left(\frac{\partial x}{\partial \eta}\right)^2 - 2\frac{\partial^2 f}{\partial \xi \partial \eta}\frac{\partial x}{\partial \xi}\frac{\partial x}{\partial \eta} + \frac{\partial^2 f}{\partial \eta^2}\left(\frac{\partial x}{\partial \xi}\right)^2\right] \\ & + \left[\left(\frac{\partial x}{\partial \eta}\right)^2 \frac{\partial^2 y}{\partial \xi^2} - 2\frac{\partial x}{\partial \xi}\frac{\partial x}{\partial \eta}\frac{\partial^2 y}{\partial \xi \partial \eta} + \left(\frac{\partial x}{\partial \xi}\right)^2 \frac{\partial^2 y}{\partial \eta^2}\right] \\ & \left[\frac{\partial f}{\partial \xi}\frac{\partial x}{\partial \eta} - \frac{\partial f}{\partial \eta}\frac{\partial x}{\partial \xi}\right] \\ & + \frac{1}{J^3}\left[\left(\frac{\partial x}{\partial \eta}\right)^2 \frac{\partial^2 x}{\partial \xi^2} - 2\frac{\partial x}{\partial \xi}\frac{\partial x}{\partial \eta}\frac{\partial^2 x}{\partial \xi \partial \eta} + \left(\frac{\partial x}{\partial \xi}\right)^2 \frac{\partial^2 x}{\partial \eta^2}\right] \\ & \left[\frac{\partial f}{\partial \eta}\frac{\partial y}{\partial \xi} - \frac{\partial f}{\partial \xi}\frac{\partial y}{\partial \eta}\right],\end{aligned}$$

for the y-coordinate. For a derivation of these relations, the interested reader is referred to [2, pp. 178–183] [114].

A.2. Hyperparameters

A.2.1. Helmholtz Equation

- tensor backend: Tensorflow
- number of hidden layers $D = 4$
- number of neurons per hidden layer $N_k = 50$
- activation function $\sigma^k = \tanh$
- batch size 1024
- number of training epochs 10000
- floating point precision: `float32`
- loss term weight $\lambda_\mathrm{R} = 1$
- loss term weight $\lambda_\mathrm{B} \in \{1, 10\}$
- optimizer settings Adam (Alg. 2):
  - `lr`$= 10^{-3}$
  - exponential learning rate decay
  - final `lr`$= 10^{-6}$
- training/collocation points $(N_\mathrm{R}, N_\mathrm{B}) = (10^4, 10^4)$

A.2.2. Burgers' Equation

- tensor backend: PyTorch

- number of hidden layers $D = 6$
- number of neurons per hidden layer $N_k = 50$
- activation function $\sigma^k = \tanh^k_{\text{adap}}$
- batch size 2500
- number of training epochs 8000
- floating point precision: `float32`
- number of Fourier embedding feature $D_{\text{fe}} = 40$
- standard deviation Fourier embedding sampling $\sigma_\phi = 2$
- loss term weight $\lambda_{\text{R}} = 1$
- loss term weight $\lambda_{\text{B}} = 1$
- loss term weight $\lambda_{\text{I}} = 1$
- mesh dimensions $(N_\xi, N_\tau) = (200, 50)$
- optimizer settings Adam (Alg. 2):
  - `lr`$= 10^{-3}$
  - exponential learning rate decay
  - final `lr`$= 10^{-6}$

A.2.3. Flow around Cylinder

Non-Parametric Model

- tensor backend: Tensorflow
- number of hidden layers $D = 8$
- number of neurons per hidden layer $N_k = 20$
- activation function $\sigma^k = \tanh$
- floating point precision: `float64`
- loss term weight $\lambda_{\text{R}} = 1$
- loss term weight $\lambda_{\text{B}} = 1$
- loss term weight $\lambda_\nu = 5$
- training settings - phase 1:
  - optimizer: Adam (Alg. 2)
  - `lr`$= 10^{-3}$
  - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_\infty) = (2 \cdot 10^4, 2 \cdot 10^4, 2 \cdot 10^4)$
  - batch size 2500
  - number of training epochs 30000

  - multi step learning rate decay
  - 15000 epochs at `lr`$= 10^{-3}$
  - 5000 epochs at `lr`$= 10^{-4}$
  - 5000 epochs at `lr`$= 10^{-5}$
  - 5000 epochs at `lr`$= 10^{-6}$
  - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
- training settings - phase 2:
  - optimizer: Adam (Alg. 2)
  - `lr`$= 10^{-3}$
  - training/collocation points $(N_{\mathrm{R}}, N_{\mathrm{obj}}, N_{\infty}) = (2 \cdot 10^4, 2 \cdot 10^4, 2 \cdot 10^4)$
  - batch size 2500
  - number of training epochs 10000
  - multi step learning rate decay
  - 5000 epochs at `lr`$= 10^{-4}$
  - 2000 epochs at `lr`$= 10^{-5}$
  - 3000 epochs at `lr`$= 10^{-6}$
  - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
  - reduce $\tilde{\nu} = 7.5 \cdot 10^{-4}$ to $\nu_0 = 0$
  - reduction epochs $M_{\mathrm{red}} = 2500$
  - reduction exponent $k = 4$
- training settings - phase 3:
  - optimizer: L-BFGS [31]
  - training/collocation points $(N_{\mathrm{R}}, N_{\mathrm{obj}}, N_{\infty}) = (5 \cdot 10^3, 5 \cdot 10^3, 5 \cdot 10^3)$
  - number of training epochs 20000
  - `tolerance`$= -1$
  - `tolerance_x`$= -1$
  - `f_relative_tolerance`$= -1$
  - `num_correction_pairs`$= 20$
  - `max_line_search_iterations`$= 100$
  - $\tilde{\nu} = 0$

Parametric Model

- tensor backend: Tensorflow
- number of hidden layers $D = 8$
- number of neurons per hidden layer $N_k = 20$
- activation function $\sigma^k = \tanh^k_{\text{adap}}$
- floating point precision: `float64`
- loss term weight $\lambda_{\text{R}} = 1$
- loss term weight $\lambda_{\text{B}} = 1$
- loss term weight $\lambda_{\nu} = 5$
- training settings - phase 1:
    - optimizer: Adam (Alg. 2)
    - `lr`$= 10^{-3}$
    - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (10^5, 10^5, 10^5)$
    - batch size 10000
    - number of training epochs 17500
    - multi step learning rate decay
    - 7500 epochs at `lr`$= 10^{-3}$
    - 7500 epochs at `lr`$= 10^{-4}$
    - 2500 epochs at `lr`$= 10^{-5}$
    - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
- training settings - phase 2:
    - optimizer: Adam (Alg. 2)
    - `lr`$= 10^{-5}$
    - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (10^5, 10^5, 10^5)$
    - batch size 10000
    - number of training epochs 17500
    - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
    - reduce $\tilde{\nu} = 7.5 \cdot 10^{-4}$ to $\nu_0 = 0$
    - reduction epochs $M_{\text{red}} = 2500$
    - reduction exponent $k = 1$
- training settings - phase 3:
    - optimizer: L-BFGS [31]
    - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (3 \cdot 10^4, 3 \cdot 10^4, 3 \cdot 10^4)$

  - number of training epochs 30000
  - `tolerance`$= -1$
  - `tolerance_x`$= -1$
  - `f_relative_tolerance`$= -1$
  - `num_correction_pairs`$= 20$
  - `max_line_search_iterations`$= 100$
  - $\tilde{\nu} = 0$

A.2.4. Oblique Shock

Non-Parametric Model

- tensor backend: Tensorflow
- number of hidden layers $D = 7$
- number of neurons per hidden layer $N_k = 20$
- activation function $\sigma^k = \tanh^k_{\text{adap}}$
- floating point precision: `float64`
- loss term weight $\lambda_{\text{R}} = 1$
- loss term weight $\lambda_{\text{B}} = 1$
- loss term weight $\lambda_{\nu} = 5$
- training settings - phase 1:
  - optimizer: Adam (Alg. 2)
  - `lr`$= 10^{-3}$
  - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (5 \cdot 10^3, 5 \cdot 10^2, 10^3)$
  - batch size 5000 (full batch)
  - number of training epochs 30000
  - multi step learning rate decay
  - 12000 epochs at `lr`$= 10^{-3}$
  - 18000 epochs at `lr`$= 10^{-4}$
  - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
- training settings - phase 2:
  - optimizer: Adam (Alg. 2)
  - `lr`$= 5 \cdot 10^{-4}$
  - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (5 \cdot 10^3, 5 \cdot 10^2, 10^3)$
  - batch size 5000 (full batch)

- number of training epochs 7500
- multi step learning rate decay
- 15000 epochs at `lr`$= 5 \cdot 10^{-4}$
- 5000 epochs at `lr`$= 1 \cdot 10^{-4}$
- $\tilde{\nu} = 7.5 \cdot 10^{-4}$
- reduce $\tilde{\nu} = 7.5 \cdot 10^{-4}$ to $\nu_0 = 0$
- reduction epochs $M_{\text{red}} = 5000$
- reduction exponent $k = 1$

- training settings - phase 3:
  - optimizer: L-BFGS [31]
  - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (5 \cdot 10^3, 5 \cdot 10^2, 10^3)$
  - number of training epochs 4000
  - `tolerance`$= -1$
  - `tolerance_x`$= -1$
  - `f_relative_tolerance`$= -1$
  - `num_correction_pairs`$= 20$
  - `max_line_search_iterations`$= 100$
  - $\tilde{\nu} = 0$

Parametric Model

- tensor backend: Tensorflow
- number of hidden layers $D = 7$
- number of neurons per hidden layer $N_k = 30$
- activation function $\sigma^k = \tanh^k_{\text{adap}}$
- floating point precision: `float64`
- loss term weight $\lambda_{\text{R}} = 1$
- loss term weight $\lambda_{\text{B}} = 1$
- loss term weight $\lambda_{\nu} = 5$
- training settings - phase 1:
  - optimizer: Adam (Alg. 2)
  - `lr`$= 5 \cdot 10^{-4}$
  - training/collocation points $(N_{\text{R}}, N_{\text{obj}}, N_{\infty}) = (10^5, 5 \cdot 10^3, 10^4)$
  - batch size 10000

    - number of training epochs 30000
    - multi step learning rate decay
    - 7500 epochs at `lr`$= 5 \cdot 10^{-4}$
    - 7500 epochs at `lr`$= 10^{-4}$
    - 15000 epochs at `lr`$= 2^{-5}$
    - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
- training settings - phase 2:
    - optimizer: Adam (Alg. 2)
    - `lr`$= 2 \cdot 10^{-5}$
    - training/collocation points $(N_\text{R}, N_\text{obj}, N_\infty) = (10^5, 5 \cdot 10^3, 10^4)$
    - batch size 10000
    - number of training epochs 17500
    - $\tilde{\nu} = 7.5 \cdot 10^{-4}$
    - reduce $\tilde{\nu} = 7.5 \cdot 10^{-4}$ to $\nu_0 = 0$
    - reduction epochs $M_\text{red} = 10000$
    - reduction exponent $k = 4$
- training settings - phase 3:
    - optimizer: L-BFGS [31]
    - training/collocation points $(N_\text{R}, N_\text{obj}, N_\infty) = (3 \cdot 10^4, 2.5 \cdot 10^3, 5 \cdot 10^4)$
    - number of training epochs 30000
    - `tolerance`$= -1$
    - `tolerance_x`$= -1$
    - `f_relative_tolerance`$= -1$
    - `num_correction_pairs`$= 20$
    - `max_line_search_iterations`$= 100$
    - $\tilde{\nu} = 0$

A.2.5. Flows around Airfoils

Subsonic Conditions

- tensor backend: PyTorch
- number of hidden layers $D = 5$
- number of neurons per hidden layer $N_k = 128$
- activation function $\sigma^k = \tanh^k_\text{adap}$

- batch size 64000
- number of training epochs 40
- floating point precision: `float32`
- number of Fourier embedding feature $D_{\text{fe}} = 128$
- standard deviation Fourier embedding sampling $\sigma_\phi = 1$
- loss term weight $\lambda_\text{R} = 2 \cdot 10^4$
- loss term weight $\lambda_\text{B} = 1$
- stag. sensor activation threshold $k^{(0)}_{\text{stag}} = 3$
- stag. sensor activation steepness $k^{(1)}_{\text{stag}} = 5$
- shock sensor activation threshold $k^{(0)}_{\text{shock}} = 4$
- shock sensor activation steepness$k^{(1)}_{\text{shock}} = 0.03$
- mesh dimensions $(N_\xi, N_\eta) = (400, 200)$
- optimizer settings L-BFGS:
  - `max_iter`$= 1000$
  - `max_iter`$= 1000$
  - `max_eval`$= 1250$
  - `tolerance_grad`$= 10^{-6}$
  - `tolerance_change`$= 10^{-10}$
  - `history_size`$= 1000$
  - `line_search_fn="strong_wolfe"`

Transonic Conditions

- tensor backend: PyTorch
- number of hidden layers $D = 5$
- number of neurons per hidden layer $N_k = 128$
- activation function $\sigma^k = \tanh^k_{\text{adap}}$
- batch size 64000
- number of training epochs 40
- floating point precision: `float32`
- number of Fourier embedding feature $D_{\text{fe}} = 128$
- standard deviation Fourier embedding sampling $\sigma_\phi = 1$
- loss term weight $\lambda_\text{R} = 2 \cdot 10^4$

- loss term weight $\lambda_{\mathrm{B}} = 1$
- stag. sensor activation threshold $k^{(0)}_{\mathrm{stag}} = 3$
- stag. sensor activation steepness $k^{(1)}_{\mathrm{stag}} = 5$
- shock sensor activation threshold $k^{(0)}_{\mathrm{shock}} = 4$
- shock sensor activation steepness$k^{(1)}_{\mathrm{shock}} = 0.03$
- mesh dimensions $(N_\xi, N_\eta) = (400, 200)$
- optimizer settings L-BFGS:
  - `max_iter`= 1000
  - `max_iter`= 1000
  - `max_eval`= 1250
  - `tolerance_grad`= 10^{-6}
  - `tolerance_change`= 10^{-10}
  - `history_size`= 1000
  - `line_search_fn="strong_wolfe"`

Parametric Problem

- tensor backend: PyTorch
- number of hidden layers $D = 5$
- number of neurons per hidden layer $N_k = 128$
- activation function $\sigma^k = \tanh^k_{\mathrm{adap}}$
- batch size 80000
- number of training epochs 5
- floating point precision: `float32`
- number of Fourier embedding feature $D_{\mathrm{fe}} = 128$
- standard deviation Fourier embedding sampling $\sigma_\phi = 1$
- loss term weight $\lambda_{\mathrm{R}} = 2 \cdot 10^4$
- loss term weight $\lambda_{\mathrm{B}} = 1$
- stag. sensor activation threshold $k^{(0)}_{\mathrm{stag}} = 3$
- stag. sensor activation steepness $k^{(1)}_{\mathrm{stag}} = 5$
- shock sensor activation threshold $k^{(0)}_{\mathrm{shock}} = 4$
- shock sensor activation steepness$k^{(1)}_{\mathrm{shock}} = 0.03$
- mesh dimensions $(N_\xi, N_\eta) = (400, 200)$

- number of alpha samples $N_\alpha = 56 \cdot 10^4$
- number of alpha samples on bounds $N_{\alpha,\mathrm{bound}} = 4 \cdot 10^4$
- optimizer settingsL-BFGS:
  - `max_iter`= 1000
  - `max_iter`= 1000
  - `max_eval`= 1250
  - `tolerance_grad`= 10^{-6}
  - `tolerance_change`= 10^{-10}
  - `history_size`= 1000
  - `line_search_fn="strong_wolfe"`

A.3. Software Details

As before the *Surrogate Modeling for Aero Data Toolbox in Python* (SMARTy) [8] is used for the implementation of the NN. Besides deep learning, SMARTy can be used for many other data-driven tasks (see e.g. [131]). SMARTy's neural network module allows for a straightforward creation of deep learning models and with a focus on applications in aerodynamics. The module comes with a PyTorch [22] and a Tensorflow [23] backend, which can be used interchangeably in most cases. These backends allow for the training on graphics cards, enabling the efficient vectorization of the vector and matrix multiplication involved in the backpropagation algorithm. In this work, both backends were used for different chapters of the thesis.

B. Supplementary Studies and Results

In the following sections, additional studies of the models, presented in Ch. 6 are shown, aiming to further illustrate the selection of certain hyperparameters and the use of certain techniques.

B.1. Complementary Ablation Study

The baseline performance (i.e. the full model) used for this ablation study, is model V, as introduced in Tab. 6.5, because it performed best on a variety of test cases in Sec. 6.2. We then perform additional simulations, where certain parts of this model are removed/exchanged.

The Fourier embedding methodology has been introduced in Sec. 2.3, as an improvement to the basic PINN architecture, aiming to avoid the spectral bias phenomenon during training. For this ablation study, an otherwise identical model without the Fourier embedding layer (see Fig. 6.1) is trained using the same Hyperparameters (see Appendix A.2). Furthermore, as discussed in Ch. 6, the sensor $s(x, y)$ in the baseline model is detached from the computational graph. To test the effect of this, the performance of an otherwise identical model where the sensor is not detached is also analyzed.

Dynamic loss term weighting [41, 53, 54], has shown significant improvements when imbalances in the gradients of different loss terms occur. For the sake of Ch. 6, no dynamic loss term weighting was used, since it could be observed that constant loss term weights λ_R and λ_B perform similarly well. To demonstrate this, an additional model is trained using the L_2-norm based weighting algorithm, presented by Wang et al. [54] (See Tab. 2.1). The loss term weights are initialized as before, with $\lambda_\mathrm{R} = 2 \cdot 10^4$ and $\lambda_\mathrm{B} = 1$. The loss term weights are updated after every outer iteration of the L-BFGS optimizer with a running average with an update factor of 0.5.

To analyze the results of the ablation study, we again consider *RMAE* for M and C_p for the four cases A-D (see Tab. 6.4). The resulting errors are shown in Tab. B.1. As before, the errors represent the mean over four training runs with different random initializations of the trainable parameters of the model and the error bounds are given by the standard deviation over these four runs.

The model without Fourier embedding performs significantly worse on cases A-C and about similar on case D. On cases A-C we observe an increase in error by a factor of 3-4, confirming the improvements that we see for PINNs in the literature (e.g. [54]), when Fourier embedding is used. Therefore, this method has been used for all models presented in Ch. 6. When the sensor is not excluded from the gradient calculation of

Table B.1.: *RMAE* [%] for ablation study.

Model		Case A $M_\infty = 0.7$ $\alpha = 4^\circ$		Case B $M_\infty = 0.72$ $\alpha = 3^\circ$		Case C $M_\infty = 0.75$ $\alpha = 2^\circ$		Case D $M_\infty = 0.78$ $\alpha = 3^\circ$
Model V	C_p	0.42 ± 0.04	C_p	$\mathbf{0.3 \pm 0.08}$	C_p	$\mathbf{0.33 \pm 0.11}$	C_p	3 ± 0.4
(baseline)	M	0.427 ± 0.022	M	$\mathbf{0.31 \pm 0.05}$	M	$\mathbf{0.35 \pm 0.09}$	M	2.7 ± 0.4
No Fourier	C_p	1.4 ± 0.5	C_p	1.4 ± 0.5	C_p	1.2 ± 0.7	C_p	3.8 ± 0.3
embedding	M	1.2 ± 0.4	M	1.2 ± 0.4	M	1.1 ± 0.5	M	3.3 ± 0.2
No sensor	C_p	1.2 ± 0.8	C_p	1 ± 0.9	C_p	0.8 ± 0.3	C_p	4.3 ± 1.3
detach	M	1 ± 0.7	M	0.9 ± 0.7	M	0.7 ± 0.3	M	4 ± 1
With d.	C_p	$\mathbf{0.36 \pm 0.09}$	C_p	0.39 ± 0.22	C_p	0.39 ± 0.28	C_p	$\mathbf{2.8 \pm 0.7}$
weighting	M	$\mathbf{0.39 \pm 0.07}$	M	0.39 ± 0.18	M	0.4 ± 0.24	M	$\mathbf{2.5 \pm 0.6}$

the loss function, the performance of the model also significantly deteriorates on all four cases. As stated in Ch. 6, the sensor should be detached. Otherwise, the sensor location will be changed such that the residual loss is minimized, leading to incorrect shock locations. When using the adaptive weighting algorithm, the average accuracy of the model is similar to the baseline model on all four cases. The baseline model performs slightly better on cases B and C while the model with dynamic weighting performs slightly better on cases A and D. But the accuracies generally lie in the error bounds of each other. During training, we observe that the magnitude of the loss term weights stays similar to the initial values, confirming that dynamic changes in the loss weights are not required. Furthermore, we see that the variance in accuracy is generally larger, when using dynamic weighting. The changes in the weights lead to more noise in the loss function during training, which might negatively affect the consistency. Hence, no dynamic weighting was used in Ch. 6. Note however, that the initial choice of the weighting parameter in the first epoch does have a significant impact on the performance of the model, as discussed in Appendix B.2.

B.2. Complementary Parameter Study

To support the choice of the selected hyperparameters, we show additional parameter studies for the loss term weight λ_{res} and the viscosity ν. As described in Secs. 6.1-6.2, separate grid searches for ν for models I-V in Tab. 6.5 were performed. As an example for this, the parameter study for model V is shown, as it is the best performing model in Sec. 6.2. We consider the *RMAE* values for otherwise identical models with different ν values. All models use the hyperparameters shown in Appendix A.2. Fig. B.1 shows the average results over four runs with different random initializations of the trainable model parameters. The error bars represent the standard deviation of the error over these four runs. It is evident that the model performs best in the range of $\nu \in (1.8 - 2.0) \cdot 10^{-3}$, where the average accuracies generally lie within the error bounds of each other. While slight performance differences can be seen depending on the test-case, we see that the finally selected viscosity $\nu = 1.9 \cdot 10^{-3}$ generally represents a good compromise for all analyzed test cases with consistently low errors for all analyzed cases.

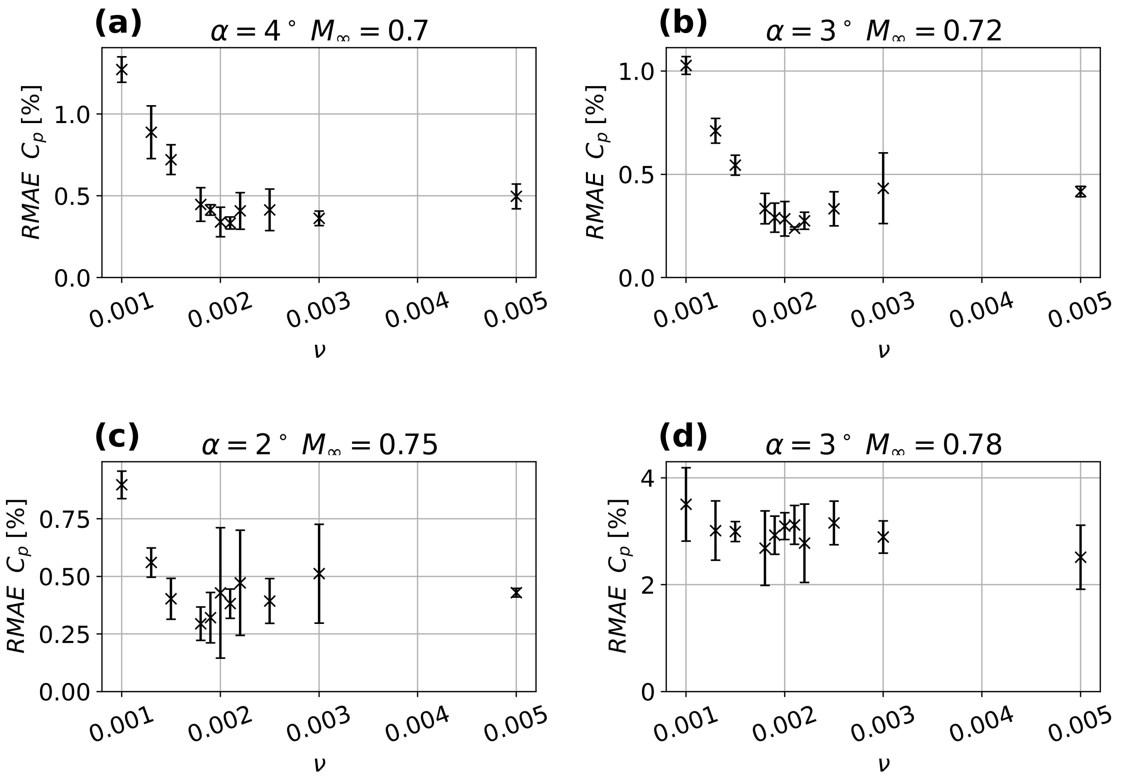


Figure B.1.: Figs. (a)-(d) show a comparison of the *RMAE* for model V with different ν values for cases A-D, respectively.

Next the loss term weighting parameter λ_{res} is analyzed. Again we take model V as the example and analyze the performance of otherwise identical models, trained with different λ_{res}. We consider the *RMAE* after training. Fig. B.2 shows an overview for test cases A-D (see Tab. 6.4). We see that the choice of the weighting parameter has a crucial impact on the performance of the model. At $\lambda_{\text{res}} = 2 \cdot 10^4$, the model performs consistently well on all cases. Hence, this weighting parameter was used for all analyzed models in Ch. 6.

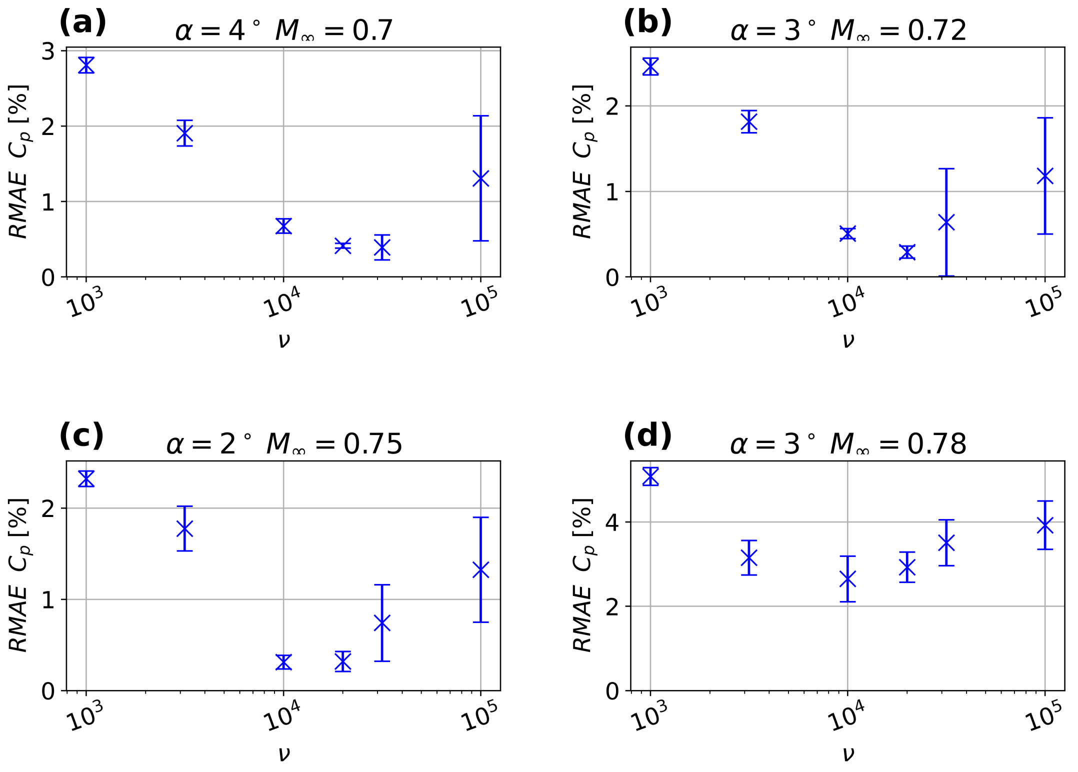


Figure B.2.: Figs. (a)-(d) show a comparison of the *RMAE* for model V with different λ_{res} values for cases A-D, respectively.

B.3. Loss History

To illustrate the convergence, we analyze the magnitude of the different loss terms in Eq. (2.84) during training. As an exemplary model, model V is selected, as it delivers the highest accuracy out of all models analyzed in Sec. 6.2. Fig. B.3 shows the loss history over the 40 training epochs for all four analyzed test cases. Figs. B.3 (a) shows the total loss while Fig. B.3 (b)-(c) show two loss terms, separately. The plotted line shows the average loss value over four randomly initialized training runs, while the shaded area highlights the maximum/minimum loss over the four runs. We see that the residual loss is generally around four orders of magnitude lower than the boundary loss but due to the large loss term weight of $\lambda_{\text{res}} = 2 \cdot 10^4$, both terms contribute about equally to the total loss $\mathcal{L}$. We also observe that the general loss magnitude for case D is significantly larger than for cases A-D. This observation leads to the conjecture that insufficient convergence is responsible for the degraded performance that is observed on case D. Specialized optimization strategies for PINN based approaches could potentially address these limitations and ensure sufficient convergence levels, even on higher Mach number cases.

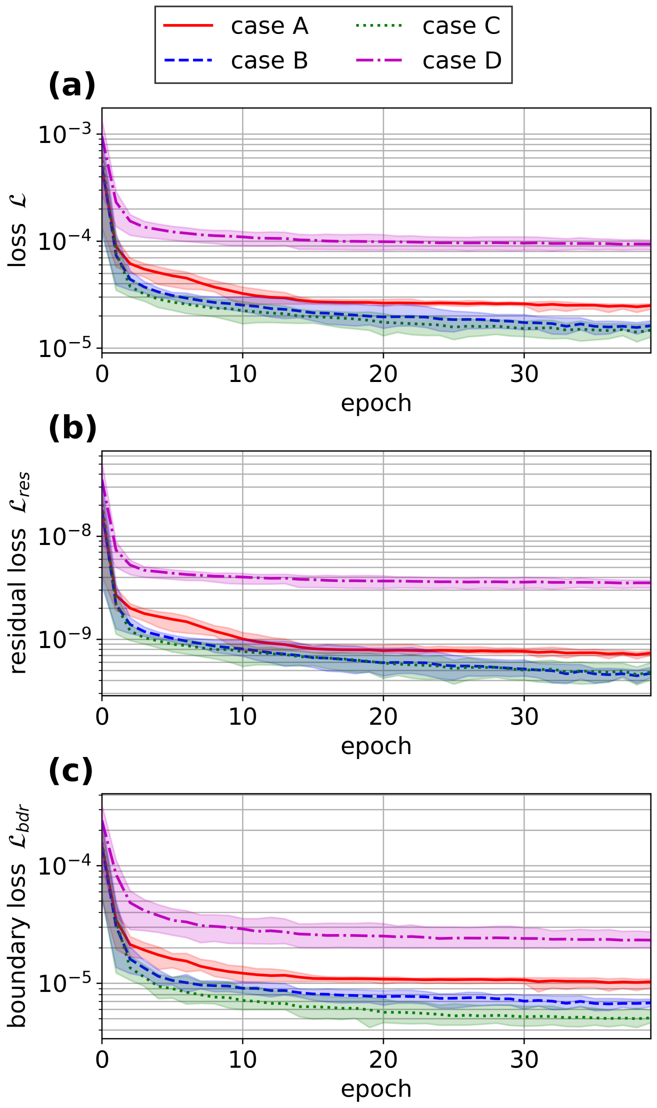


Figure B.3.: Loss history for model V on cases A-D. Fig. (a) shows the total loss $\mathcal{L}$ and Figs. (b) and (c) show the individual loss terms. Each line indicates the average over four randomly initialized training runs, while the shaded area highlights the maximum and minimum over the four training runs.

B.4. Error Metrics

B.4.1. Burgers' Equation

Table B.2.: Error metrics for different AV method for Burgers' equation for case I

AV Type	**a**	MT	ν	λ_μ	*MAE*	*RMAE*[%]	R_2
global	0	No	0	/	$(2 \pm 0.2) \cdot 10^{-1}$	(9.8 ± 0.9)	(0.86 ± 0.03)
global	0	No	$2 \cdot 10^{-3}$	/	$(5.7 \pm 1.1) \cdot 10^{-3}$	(0.28 ± 0.6)	(0.995 ± 0.003)
global	0.8	No	$4/3 \cdot 10^{-3}$	/	$\mathbf{(2.5 \pm 0.2) \cdot 10^{-3}}$	$\mathbf{(0.126 \pm 0.009)}$	(0.99881 ± 0.00017)
global	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(2.8 \pm 0.4) \cdot 10^{-3}$	(0.142 ± 0.017)	$\mathbf{(0.99888 \pm 0.00013)}$
sensor	0	No	$2 \cdot 10^{-3}$	/	$(5.2 \pm 0.9) \cdot 10^{-3}$	(0.27 ± 0.05)	(0.9957 ± 0.0015)
sensor	0.8	No	$4/3 \cdot 10^{-3}$	/	$(3.2 \pm 0.9) \cdot 10^{-3}$	(0.16 ± 0.05)	(0.9981 ± 0.0012)
sensor	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(4.4 \pm 0.7) \cdot 10^{-3}$	(0.22 ± 0.04)	(0.997 ± 0.001)
adaptive	0	No	/	2	$(7.1 \pm 1.6) \cdot 10^{-3}$	(0.36 ± 0.08)	(0.994 ± 0.003)
adaptive	0.8	No	/	8	$(4.4 \pm 1.7) \cdot 10^{-3}$	(0.22 ± 0.09)	(0.994 ± 0.004)
adaptive	0.8	Yes	/	8	$(3.6 \pm 2.1) \cdot 10^{-3}$	(0.18 ± 0.11)	(0.995 ± 0.005)

Table B.3.: Error metrics for different AV method for Burgers’ equation for case II.

AV Type	**a**	MT	ν	λ_μ	MAE	$RMAE[\%]$	R_2
global	0	No	0	/	$(6.5 \pm 0.7) \cdot 10^{-4}$	(0.26 ± 0.03)	(0.99928 ± 0.00022)
global	0	No	$2 \cdot 10^{-3}$	/	$(3.19 \pm 0.02) \cdot 10^{-3}$	(1.27 ± 0.01)	(0.99363 ± 0.00009)
global	0.8	No	$4/3 \cdot 10^{-3}$	/	$(2.45 \pm 0.01) \cdot 10^{-3}$	(0.98 ± 0.006)	(0.9959 ± 0.00012)
global	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(2.52 \pm 0.06) \cdot 10^{-3}$	(1.01 ± 0.03)	(0.99583 ± 0.00007)
sensor	0	No	$2 \cdot 10^{-3}$	/	$(6.7 \pm 1.1) \cdot 10^{-4}$	(0.27 ± 0.05)	(0.9993 ± 0.0004)
sensor	0.8	No	$4/3 \cdot 10^{-3}$	/	$(3.5 \pm 3.0) \cdot 10^{-4}$	(0.14 ± 0.12)	(0.9998 ± 0.0001)
sensor	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(2.2 \pm 0.4) \cdot 10^{-4}$	(0.09 ± 0.02)	(0.99993 ± 0.00001)
adaptive	0	No	/	2	$(6.9 \pm 1.3) \cdot 10^{-4}$	(0.28 ± 0.06)	(0.9992 ± 0.0004)
adaptive	0.8	No	/	8	$(2.4 \pm 0.5) \cdot 10^{-4}$	(0.1 ± 0.02)	(0.999931 ± 0.000013)
adaptive	0.8	Yes	/	8	$\mathbf{(2.1 \pm 0.1) \cdot 10^{-4}}$	$\mathbf{(0.082 \pm 0.004)}$	$\mathbf{(0.99994 \pm 0.00001)}$

Table B.4.: Error metrics for different AV method for Burgers' equation for case III.

AV Type	**a**	MT	ν	λ_μ	MAE	$RMAE[\%]$	R_2
global	0	No	0	/	$(1.44 \pm 0.03) \cdot 10^{-1}$	(7.16 ± 0.12)	(0.804 ± 0.008)
global	0	No	$2 \cdot 10^{-3}$	/	$(4.9 \pm 0.4) \cdot 10^{-3}$	(0.25 ± 0.02)	(0.9948 ± 0.0016)
global	0.8	No	$4/3 \cdot 10^{-3}$	/	$\mathbf{(2.587 \pm 0.021) \cdot 10^{-3}}$	$\mathbf{(0.129 \pm 0.001)}$	$\mathbf{(0.9987 \pm 0.000008)}$
global	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(2.71 \pm 0.06) \cdot 10^{-3}$	(0.136 ± 0.003)	(0.99855 ± 0.00006)
sensor	0	No	$2 \cdot 10^{-3}$	/	$(2.3 \pm 2.1) \cdot 10^{-2}$	(1.149 ± 1.004)	(0.93 ± 0.06)
sensor	0.8	No	$4/3 \cdot 10^{-3}$	/	$(4.1 \pm 0.6) \cdot 10^{-3}$	(0.21 ± 0.03)	(0.9939 ± 0.0017)
sensor	0.8	Yes	$4/3 \cdot 10^{-3}$	/	$(5.6 \pm 2.2) \cdot 10^{-3}$	(0.29 ± 0.12)	(0.989 ± 0.007)
adaptive	0	No	/	2	$(6.5 \pm 0.7) \cdot 10^{-3}$	(0.33 ± 0.04)	(0.9921 ± 0.0015)
adaptive	0.8	No	/	8	$(3.2 \pm 0.6) \cdot 10^{-3}$	(0.16 ± 0.03)	(0.995 ± 0.002)
adaptive	0.8	Yes	/	8	$(3.1 \pm 0.8) \cdot 10^{-3}$	(0.16 ± 0.05)	(0.9957 ± 0.0027)

B.4.2. Airfoil Subsonic

Table B.5.: $MAE \cdot 10^2$ for different PINN models for subsonic flow around airfoil.

Model		Case (a) $M_\infty = 0.3$ $\alpha = 5^\circ$		Case (b) $M_\infty = 0.6$ $\alpha = 1^\circ$		Case (c) $M_\infty = 0.4$ $\alpha = 3^\circ$		Case (d) $M_\infty = 0.35$ $\alpha = 4^\circ$
(i)	C_p	18 ± 5	C_p	11 ± 2.8	C_p	13 ± 5	C_p	17 ± 4
	M	6.4 ± 0.5	M	8.2 ± 2.6	M	7.2 ± 1.1	M	6.9 ± 1
(ii)	C_p	0.6 ± 0.08	C_p	0.156 ± 0.014	C_p	$\mathbf{0.29 \pm 0.04}$	C_p	0.41 ± 0.15
	M	0.09 ± 0.01	M	0.046 ± 0.004	M	$\mathbf{0.054 \pm 0.01}$	M	0.08 ± 0.04
(iii)	C_p	0.56 ± 0.11	C_p	0.21 ± 0.05	C_p	0.34 ± 0.07	C_p	0.46 ± 0.1
	M	0.079 ± 0.022	M	0.064 ± 0.015	M	0.064 ± 0.013	M	0.074 ± 0.02
(iv)	C_p	$\mathbf{0.46 \pm 0.05}$	C_p	$\mathbf{0.14 \pm 0.06}$	C_p	0.3 ± 0.04	C_p	$\mathbf{0.346 \pm 0.023}$
	M	$\mathbf{0.061 \pm 0.005}$	M	$\mathbf{0.043 \pm 0.016}$	M	0.054 ± 0.009	M	$\mathbf{0.054 \pm 0.004}$
(v)	C_p	0.461 ± 0.024	C_p	0.172 ± 0.017	C_p	0.315 ± 0.03	C_p	0.36 ± 0.007
	M	0.064 ± 0.005	M	0.053 ± 0.006	M	0.058 ± 0.007	M	0.0552 ± 0.0015
(vi)	C_p	0.616 ± 0.018	C_p	0.17 ± 0.012	C_p	0.295 ± 0.024	C_p	0.4 ± 0.04
	M	0.089 ± 0.025	M	0.051 ± 0.005	M	0.054 ± 0.005	M	0.064 ± 0.008

Table B.6.: $R_2 \cdot 10$ for different PINN models for subsonic flow around airfoil.

Model		Case A $M_\infty = 0.3$ $\alpha = 5^\circ$		Case B $M_\infty = 0.6$ $\alpha = 1^\circ$		Case C $M_\infty = 0.4$ $\alpha = 3^\circ$		Case D $M_\infty = 0.35$ $\alpha = 4^\circ$
(i)	C_p	-14 ± 13	C_p	-11 ± 11	C_p	-15 ± 17	C_p	-19 ± 12
	M	-170 ± 22	M	-140 ± 100	M	-210 ± 70	M	-190 ± 60
(ii)	C_p	9.983 ± 0.004	C_p	9.9963 ± 0.0007	C_p	9.9924 ± 0.0021	C_p	9.988 ± 0.009
	M	9.982 ± 0.004	M	9.9925 ± 0.0008	M	**9.9898 ± 0.0022**	M	9.984 ± 0.012
(iii)	C_p	9.984 ± 0.006	C_p	9.9942 ± 0.0026	C_p	9.989 ± 0.005	C_p	9.986 ± 0.006
	M	9.983 ± 0.012	M	9.993 ± 0.017	M	9.989 ± 0.004	M	9.986 ± 0.008
(iv)	C_p	**9.9903 ± 0.002**	C_p	**9.997 ± 0.0015**	C_p	**9.9925 ± 0.002**	C_p	**9.9925 ± 0.001**
	M	**9.9895 ± 0.001**	M	**9.9946 ± 0.0027**	M	9.9895 ± 0.0021	M	**9.9907 ± 0.0007**
(v)	C_p	9.99 ± 0.001	C_p	9.996 ± 0.0006	C_p	9.9912 ± 0.0017	C_p	9.992 ± 0.0004
	M	9.9887 ± 0.0015	M	9.9936 ± 0.0002	M	9.9886 ± 0.002	M	9.9905 ± 0.0005
(vi)	C_p	9.981 ± 0.011	C_p	9.9957 ± 0.0004	C_p	9.9923 ± 0.0014	C_p	9.9898 ± 0.002
	M	9.978 ± 0.01	M	9.9918 ± 0.0003	M	9.9897 ± 0.0011	M	9.9883 ± 0.0021

B.4.3. Airfoil Transonic

Table B.7.: $MAE \cdot 10^2$ for different PINN models for transonic flow around airfoil.

Model		Case A $M_\infty = 0.7$ $\alpha = 4°$		Case B $M_\infty = 0.72$ $\alpha = 3°$		Case C $M_\infty = 0.75$ $\alpha = 2°$		Case D $M_\infty = 0.78$ $\alpha = 3°$
0	C_p	13.6 ± 0.7	C_p	10.7 ± 0.8	C_p	8.8 ± 0.24	C_p	22.2 ± 0.4
	M	5.19 ± 2.6	M	4.37 ± 0.29	M	3.8 ± 1	M	9.43 ± 0.15
I	C_p	6 ± 1	C_p	11 ± 7	C_p	4.4 ± 0.5	C_p	13 ± 5
	M	2.4 ± 0.4	M	23 ± 25	M	1.92 ± 0.19	M	21 ± 30
II	C_p	1.35 ± 0.07	C_p	*15 ± 18*	C_p	2.2 ± 2	C_p	**4.7 ± 0.4**
	M	0.639 ± 0.024	M	*19 ± 24*	M	1 ± 0.9	M	**2.3 ± 0.17**
III	C_p	*6 ± 8*	C_p	*6 ± 9*	C_p	1.19 ± 0.09	C_p	12 ± 9
	M	*6 ± 11*	M	*13 ± 24*	M	0.57 ± 0.04	M	18 ± 28
IV	C_p	1.4 ± 0.4	C_p	0.91 ± 0.29	C_p	1.4 ± 0.4	C_p	7.9 ± 0.4
	M	0.66 ± 0.15	M	0.45 ± 0.12	M	0.65 ± 0.15	M	3.91 ± 0.15
V	C_p	**1.17 ± 0.09**	C_p	**0.78 ± 0.19**	C_p	**0.79 ± 0.27**	C_p	7.5 ± 1
	M	**0.58 ± 0.03**	M	**0.41 ± 0.07**	M	**0.42 ± 0.11**	M	3.7 ± 0.5

Table B.8.: $R_2 \cdot 10$ for different PINN models for transonic flow around airfoil.

Model	Case A $M_\infty = 0.7$ $\alpha = 4°$		Case B $M_\infty = 0.72$ $\alpha = 3°$		Case C $M_\infty = 0.75$ $\alpha = 2°$		Case D $M_\infty = 0.78$ $\alpha = 3°$	
0	C_p	5.2 ± 0.4	C_p	5.8 ± 0.4	C_p	6.2 ± 1.1	C_p	1 ± 0.18
	M	5.2 ± 0.3	M	5.7 ± 0.4	M	6.1 ± 1.1	M	1.34 ± 0.15
I	C_p	8 ± 0.5	C_p	4 ± 6	C_p	8.1 ± 0.4	C_p	3 ± 5
	M	7.7 ± 0.5	M	−1 ± 14	M	7.8 ± 0.5	M	−38 ± 9
II	C_p	9.85 ± 0.04	C_p	*0 ± 14*	C_p	9.3 ± 1	C_p	**8.56 ± 0.15**
	M	9.79 ± 0.06	M	*−76 ± 120*	M	9.2 ± 1	M	**8.32 ± 0.15**
III	C_p	*8 ± 4*	C_p	*7 ± 6*	C_p	9.81 ± 0.14	C_p	5 ± 5
	M	*−8 ± 34*	M	*−8 ± 16*	M	9.76 ± 0.19	M	−3 ± 8
IV	C_p	9.84 ± 0.05	C_p	9.86 ± 0.04	C_p	9.75 ± 0.1	C_p	7.27 ± 0.17
	M	9.77 ± 0.08	M	9.82 ± 0.06	M	9.67±0.12	M	6.92±0.18
V	C_p	**9.88 ± 0.01**	C_p	**9.89 ± 0.02**	C_p	**9.86 ± 0.04**	C_p	7.4 ± 0.4
	M	**9.819 ± 0.011**	M	**9.85 ± 0.03**	M	**9.83 ± 0.06**	M	7.1 ± 0.5

C. Helmholtz Problem

The Helmholtz equation is a linear elliptic PDE. It is regularly used as a benchmark for numerical methods including PINNs [41, 132, 48, 107]. In the following, a particular two-dimensional boundary value problem is analyzed, following the original work of Wang et al. [41]. For the unknown $u : \mathbb{R}^2 \to \mathbb{R}$, the boundary value problem in the domain $\Omega = (-1,1) \times (-1,1)$ is given by

$$\begin{aligned}
&\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} - k^2 u(x,y) - q(x,y) = 0 \quad (x,y) \in \Omega,\\
&q(x,y) = -(a_1\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)\\
&\qquad\quad - (a_2\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)\\
&\qquad\quad + k^2 \sin(a_1\pi x)\sin(a_2\pi y),\\
&u(x,y) = 0 \quad (x,y) \in \partial\Omega,
\end{aligned} \tag{C.1}$$

with the source term $q(x,y)$ and the wave number $k = 1$. The analytical solution to this problem is given by

$$u(x,y) = \sin(a_1\pi x)\sin(a_2\pi y). \tag{C.2}$$

The problem is analyzed for $a_1 = 1$ and $a_2 = 4$. A standard PINN model is trained to approximate the solution to Eq. (C.1). The used hyperparameters are shown in Sec. A.2. Equal loss term weights of $\lambda_B = 1$ and $\lambda_R = 1$ were selected. The resulting approximation, shown in Fig. C.1, shows significant errors, especially at the boundaries of the domain. As discussed in Sec. 2.4.4, this failure of the PINN model can be traced back to imbalances

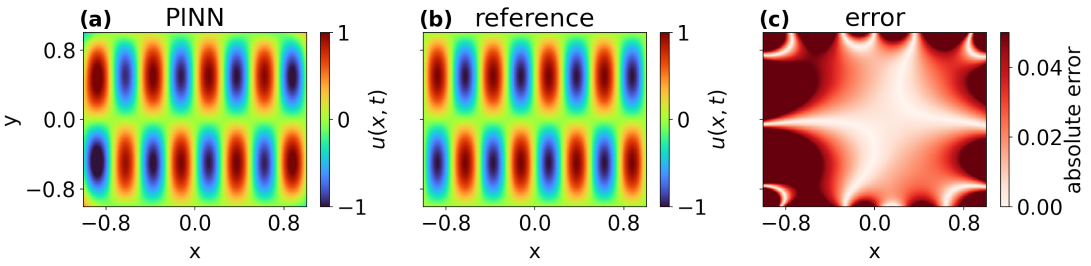


Figure C.1.: Prediction of standard PINN model for Helmholtz equation problem. Fig (a) shows the PINN prediction, Fig (b) shows the analytical solution and Fig. (c) shows the absolute error of the PINN prediction.

between the different loss terms. Fig. 2.11, shows off significant imbalances between the gradients of the loss terms $\nabla_\theta \mathcal{L}_\mathbb{R}$ and $\nabla_\theta \mathcal{L}_\mathbb{B}$. The magnitude of the boundary loss terms is comparatively low. Hence, the boundary condition is not sufficiently accurately fulfilled and the overall prediction accuracy deteriorates. These histograms were obtained by collecting the entries of the gradient vectors of all layers of the NN for 10 different training runs with identical hyperparameters but different random initializations of $\boldsymbol{\theta}$. The gradients are evaluated after only 1000 epochs, as the imbalances are especially pronounced at the start of training.

The imbalances between the loss terms can, for example be addressed using the loss term weights λ_i, as introduced in Eq. 2.85. The corresponding prediction for an identical model with $\lambda_B = 10$ and $\lambda_R = 1$ (Fig. C.2) shows significant improvements.

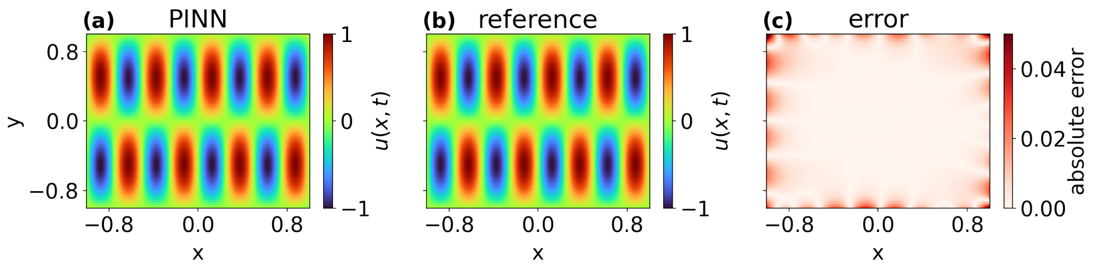


Figure C.2.: Prediction of the PINN model with an increased boundary weight of $\lambda_\mathrm{B} = 10$ for Helmholtz equation problem. Fig (a) shows the PINN prediction, Fig (b) shows the analytical solution and Fig. (c) shows the absolute error of the PINN prediction.

To also account for changes in the imbalances throughout training, one can use dynamics weighting algorithms, as introduced in Sec 2.4.4. This approach is tested, using the L_2-norm based algorithm, introduced in [54] (see Tab. 2.1). The loss terms are updated every $l_w = 500$ with an update rate of $\beta_w = 0.5$. The resulting PINN prediction is shown in Fig. C.3. For both the model the model with adaptive weighting, the prediction accuracy

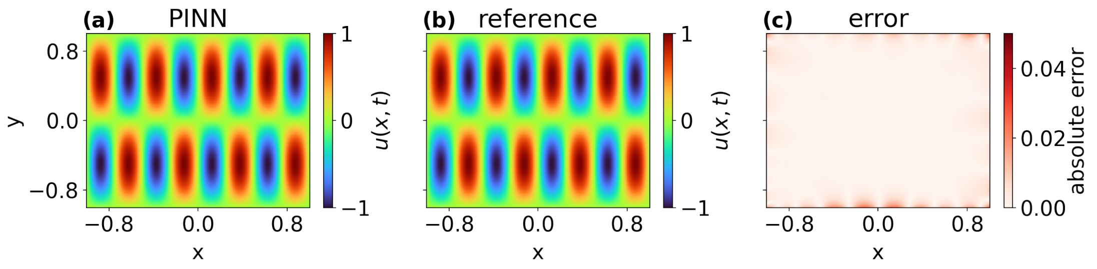


Figure C.3.: Prediction of the PINN model with dynamic loss term weighting for Helmholtz equation problem. Fig (a) shows the PINN prediction, Fig (b) shows the analytical solution and Fig. (c) shows the absolute error of the PINN prediction.

is even further improved since the dynamic loss term weighting can adaptively increase the magnitude of the gradients of the boundary loss during training. Hence the boundary condition is more accurately fulfilled and the overall accuracy is improved. Fig. C.4 compares the development of the mean absolute error throughout training for the PINN models with and without the dynamic weighting. A total of 10 training runs with random initializations were performed in both cases and the plot shows the average as well as the minimum and maximum error over the 10 runs. The models with dynamic weighting

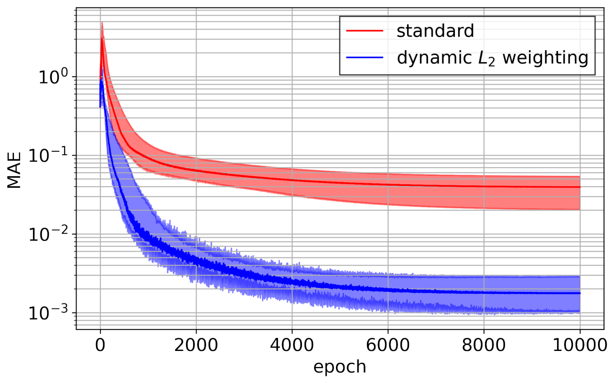


Figure C.4.: Comparison of mean absolute error (MAE) for PINN models with and without dynamic weighting for 10 randomly initialized runs for Helmholtz problem. The plotted lines indicate the average value over the 10 runs while the bounds indicate the minimum and maximum value.

significantly outperform the model without and is more than an order of magnitude more accurate.

Overall, it is evident that imbalances in the loss term gradients can have a significant, negative effect on PINN performance. We have seen that dynamic weighting algorithms can help to overcome these issues and effectively mitigate the imbalances. Whether it makes sense to employ these algorithms mostly depends on the problem at hand. When no imbalances are present, these algorithms can slightly increase the training cost without any significant benefits. Hence it typical makes sens to perform an ablation study to determine whether dynamic weighting is beneficial for a certain problem or not.

D. Reference Simulations

For all reference simulations for the Euler equations, we use CODA. CODA is the computational fluid dynamics (CFD) software being developed as part of a collaboration between the French Aerospace Lab ONERA, the German Aerospace Center (DLR), Airbus, and their European research partners. CODA is jointly owned by ONERA, DLR and Airbus. The grids are generated, using the software *Pointwise* and shown in Fig. D.1

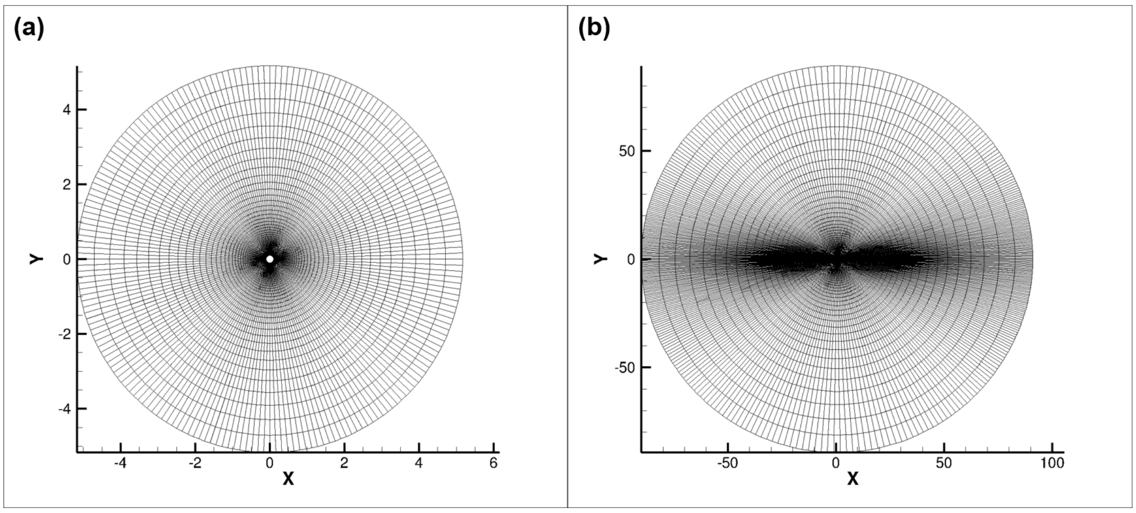


Figure D.1.: Grids used for the finite volume reference simulation for the cylinder (a) and the NACA 0012 airfoil (b) geometries.

In both cases, an o-type grid was used. For the cylinder, the grid featured 200×67 points with 200 points on the cylinders surface. The cylinder has a radius of 0.1. The outer boundary of the domain is approximately 50 cylinder diameters away from the surface. The airfoil grid featured a total of 400×96 points with 400 points on the airfoil's surface. The outer boundary of the domain is located 90 chord lengths away from the airfoil. The accuracy has been confirmed with a mesh convergence study. Note that the used airfoil grid is different to the grid used for the PINN model (see Fig. 6.2). All residuals are converged by 12 orders of magnitude. We use a second order finite-volume discretization. No far-field correction (see e.g. [3, pp.262-268]) has been used. The convective fluxes are evaluated using Roe's approximate Riemann solver [97]. For all quantitative comparisons to PINN models we interpolate the PINN prediction and the FV reference to a Cartesian grid in the physical domain. Points inside the aerodynamic objects are excluded. We then calculate the errors based on the points in this regular grid.