On the development of data-driven numerical schemes in Computational Fluid Dynamics

Luciano Drozda Dantas Martins

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Dynamique des Fluides

Présentée et soutenue par :

M. LUCIANO DROZDA DANTAS MARTINS

le mardi 12 septembre 2023

Titre :

Sur le développement de schémas numériques guidés par les données en Mécanique des Fluides Numérique

---

Ecole doctorale :

Mécanique, Energétique, Génie civil, Procédés (MEGeP)

Unité de recherche :

Centre Européen de Recherche et Formation Avancées en Calcul Scientifique (CERFACS)

Directeur(s) de Thèse :

M. THIERRY POINSOT

M. CORENTIN LAPEYRE

Rapporteurs :

M. TAPAN KUMAR SENGUPTA, IIT KANPUR

Membre(s) du jury :

M. JENS MUELLER, QUEEN MARY UNIVERSITY OF LONDON, Président

M. AMIR ADLER, BCE KARMIEL, Invité(e)

M. CORENTIN LAPEYRE, CERFACS, Membre

M. JOSÉ IGNACIO CARDESA, ONERA TOULOUSE, Membre

MME ANDREA BECK, UNIVERSITAT STUTTGART, Membre

M. THIERRY POINSOT, TOULOUSE INP, Membre

Abstract The numerical simulation of fluid flows has become an essential part of the virtual prototyping in the aerospace industry. It is made possible through Computational Fluid Dynamics (CFD) tools. These tools rely on algebraic relations between the values stored at mesh points called numerical schemes, which must fulfill specific criteria to lead to reliable simulations. Namely, numerical schemes must be accurate without sacrificing stability (among other conditions). Traditionally, accuracy and stability conditions have been derived for problems governed by linear equations on regular meshes. However, fluid motion equations are non-linear, and meshes used in industrial CFD can be irregular (especially when dealing with complex geometries). This thesis proposes a novel framework to analyze the accuracy and stability of numerical schemes for non-linear systems of equations and irregular meshes. This framework also establishes conditions for optimizing numerical schemes locally in time and space. It is named Local Transfer function Analysis (LTA). Under the light of LTA, each mesh element acts as an impedance block that resists to the solution propagation over time and space. The optimization of numerical schemes becomes an impedance-matching problem. It is solved by minimizing the value of an objective function. The LTA objective function measures the distance between the dynamics predicted by the numerical scheme and some reference dynamics. Using reference data in this optimization process turns LTA into a tool that generates accurate and stable data-driven numerical schemes. More recently, Machine Learning (ML) has emerged as a field dedicated to extracting knowledge from data. This thesis then proposes employing ML architectures to find optimal values for the parameters of a numerical scheme in the sense of the LTA objective function. The method is applied first on 1D problems modeled by the Convection/Burgers' equations and finally extended to 2D applications governed by the Convection/Euler equations.

Keywords Computational Fluid Dynamics, Local Transfer function Analysis, Data-driven numerical schemes, Machine Learning

Contents

Nomenclature **6**

List of Figures **9**

List of Tables **18**

Introduction **19**

1 Introducing machine learning **25**
1.1 Machine learning: from checkers to scientific computing 25
1.1.1 Multilayer perceptrons 31
1.1.2 Convolutional neural networks 33
1.1.3 Graph neural networks 36
1.1.4 The supervised learning paradigm 39
1.2 An introduction to Algorithmic Differentiation (AD) 44
1.2.1 Forward mode of AD 47
1.2.2 Reverse mode of AD 50
1.3 An overview of AD frameworks 53
1.3.1 TensorFlow & PyTorch (Python/C++) 53
1.3.2 JAX (Python) 53
1.3.3 Tapenade (Fortran/C) 53
1.3.4 Zygote (Julia) 54
1.3.5 Enzyme (LLVM Intermediate Representation) 54

2 Introducing data-driven numerical schemes **55**
2.1 Generalities about the numerical approximation of hyperbolic conservation laws 56
2.2 Numerical analysis of schemes 61

2.2.1 First Order Upwind (FOU) and Lax-Wendroff (LW) . . 61
2.2.2 Monotonic Upstream-centered Schemes for Conservation Laws (MUSCL) 65
2.2.3 Weighted Essentially Non-Oscillatory (WENO) 67
2.3 Data-driven spatial discretizations on 2D cartesian grids . . . 69
2.3.1 Neural network model 72
2.3.2 Numerical experiments 76

3 Proposing a novel spectral analysis 90
3.1 The Taylor-Galerkin family of schemes 91
3.2 Stability, dissipation and dispersion analysis 94
3.2.1 Global Spectral Analysis (GSA) 95
3.2.2 Application of GSA to Taylor-Galerkin schemes 98
3.2.3 Locally tunable TTGC-γ scheme (TTGC-γ_L) 104
3.2.4 Local Transfer function Analysis (LTA) 109
3.2.5 LTA of the inviscid Burgers' equation 120

4 Optimizing TTGC-γ_L and Machine-Learned TTGC (ML-TTGC) 127
4.1 The optimization problem resulting from LTA 128
4.2 Numerical implementation and verification of the optimization 130
4.2.1 Multi-objective Pareto optimality 134
4.3 Neural network model 139
4.4 Application of ML-TTGC 143
4.4.1 Linear convection equation 143
4.4.2 Inviscid Burgers' equation 146

5 Extending ML-TTGC to unstructured meshes 151
5.1 TTGC in higher dimensions 152
5.2 ML-TTGC for 2D Convection/Euler 157

Conclusions and perspectives 165

Bibliography 168

A Example of Tapenade-generated adjoint program 190

B The mutation problem 192
B.1 Presentation of the problem 192
B.2 Analysis of a discretization operator 195

B.3 Analysis of a CFD kernel 205

C Local wavenumber analysis and shock sensor of Jameson 210

D Entropy condition and impedance matching 213

Nomenclature

Acronyms

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
AD	Algorithmic Differentiation
AGI	Artificial General Intelligence
AI	Artificial Intelligence
AVBP	DNS/LES code developed by CERFACS
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy condition
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNS	Direct Numerical Simulation
DRP	Dispersion Relation Preserving
DSL	Domain Specific Language
FFT	Fast Fourier Transform
FOU	First Order Upwind

FV	Finite Volume
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GVP	Group Velocity Preserving
LES	Large-Eddy Simulation
LW	Lax-Wendroff
ML	Machine Learning
MLP	Multilayer Perceptron
MUSCL	Monotonic Upstream-centered Schemes for Conservation Laws
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PINN	Physics Informed Neural Network
SUPG	Streamline Upwind/Petrov-Galerkin
TG	Taylor-Galerkin
TTG	Two-step Taylor-Galerkin
WENO	Weighted Essentially Non-Oscillatory
WP	Wave Packet

Number sets

$\mathbb{N}$	Natural numbers
$\mathbb{R}$	Real numbers
$\mathbb{Z}$	Integers

Operators

$\mathbf{div}\underline{\boldsymbol{F}}$	Divergence of a matrix-valued function $\underline{\boldsymbol{F}}$

$\mathbf{grad}_x F$ Gradient of a scalar-valued function F with respect to x

Symbols

$\boldsymbol{x}$ Tensor of order 1 (i.e., a vector)

$\underline{\boldsymbol{x}}$ Tensor of order greater than unity

List of Figures

1 Manuscript's structure. 24

1.1 Arthur Samuel and IBM 700 (1956). 26
1.2 Schematic representation of the original (photo-)perceptron. . 28
1.3 (a) Camera device capturing an image of a letter **C** drawn on a wall (1962). From Rosenblatt (1962). (b) Frank Rosenblatt manipulating wired connections of the Mark I Perceptron machine (1960). From dnvdk (2022). 29
1.4 Example of linearly separable sets. 29
1.5 Schematic representation of a 3-layer MLP. 32
1.6 Examples from the MNIST dataset. 33
1.7 Schematic comparison between a dense layer (top) and a convolutional layer (bottom). 35
1.8 Schematic representation of operations performed in a graph network layer. 37
1.9 Performance metrics for training and validation sets at the end of each epoch during the training procedure. 42
1.10 Absolute error of the ND-evaluated derivative of the function $f(x) := \exp(3x)$ at $x = 0.2$ using the approximation (1.12). . . 45
1.11 Computational graph and forward primal evaluation trace for a program evaluating $f(x_1, x_2) := \sin(x_1 + x_2)$, where $x_1, x_2 \in \mathbb{R}$. 47
1.12 (Top) Schematic representation of the sliding ladder problem. (Middle and bottom) Computational graph, forward primal, and forward tangent evaluation traces for a program evaluating the angle y_1 (θ_{sep} in Eq. (1.17)) at which the falling ladder loses contact with the vertical wall and the speed y_2 (s_{f} in Eq. (1.18)) at which it hits the floor. 49

1.13 (Top) Schematic representation of a dense layer. (Middle and bottom) Computational graph, forward primal and adjoint evaluation traces for a program evaluating the (scalar) output of the dense layer. 52

2.1 Representations of the (a) 1D Riemann problem (Def. 4) and (b) 1D periodic Riemann problem (Def. 5). 58
2.2 Representations of the analytical solutions for the 1D periodic Riemann problem (see Def. 5) following (a) the Convection equation (2.4) and (b) the Burgers' equation (2.5). In (b), $c := (u_{\max} + u_{\min}) \,/\, 2$. 58
2.3 Representation of MUSCL numerical fluxes $f^n_{i\pm1/2}$ using the extrapolations $u^{R,\,n}_{i\pm1/2}$ and $u^{L,\,n}_{i\pm1/2}$ of the state variable at the boundaries of the cell $[x_{i-1/2}, x_{i+1/2}]$, where $x_{i\pm1/2} := (x_i + x_{i\pm1})/2$. 66
2.4 (a) One box turn of a 1D periodic Riemann problem (see Def. 5) following the Convection equation (2.4) for all schemes depicted in Sec. 2.2. (b) Numerical solution of a 1D periodic Riemann problem following the Burgers' equation (2.5) at time $t = 150\Delta t$ (where Δt is the simulation time step) for all schemes depicted in Sec. 2.2. 70
2.5 (a) Grid convergence error plots for all schemes depicted in Sec. 2.2. (b) Smooth initial condition $u^0 : x \in [0, 1] \mapsto \cos(x)$ used for grid convergence studies. 71
2.6 Schematic representation of the CNN-PAL neural network architecture for the data-driven spatial discretization of the Euler equations on 2D cartesian grids. The variables $F^{\star,x}$ and $F^{\star,y}$ correspond to the components of the flux of the field $\star$ along the x and y directions, respectively. 73
2.7 Outcomes of experiment n.1 performed with two independent CNN-PAL models trained to predict (a) $N_{\text{step}} = 4$ time steps; (b) $N_{\text{step}} = 16$ time steps of the evolution of the vortex depicted in Tbl. 2.2. 81

2.8 Discretization coefficients along the x-direction generated by a CNN-PAL model for the vortex convection analyzed in experiment n.1 (depicted in Tbl. 2.2). The "UNTRAINED" columns refer to the coefficients returned by the CNN-PAL model before training, while the "TRAINED" columns relate to the coefficients returned by the same CNN-PAL model after training. The coefficients are assigned to the neighborhood $\{(i-2,j),(i-1,j),(i,j),(i+1,j),(i+2,j)\}$ around each mesh point (i,j), composing a centered stencil of size $N = 5$ (see Tbl. 2.1). 82

2.9 Evolution of training and validation loss values across epochs for independent CNN-PAL models trained to predict $N_{\text{step}} = 16$ time steps of the convection of vortices from training datasets composed of (a) $N_s = 32$ samples; (b) $N_s = 128$ samples. Results concern experiment n.2 (see Tbl. 2.2). 84

2.10 Grid convergence error plots for the WENO5 scheme implementation for the Euler equations considering either (a) mean L_2 error or (b) mean L_∞ error of density field over one box turn of the baseline vortex depicted in Tbl. 2.2. The variable h stands for the grid cell size along a single direction. 86

2.11 Outcomes of experiment n.3 performed with two independent CNN-PAL models trained to predict (a) $N_{\text{step}} = 16$ time steps; (b) $N_{\text{step}} = 32$ time steps of the evolution of the double shear layer initial condition depicted in Tbl. 2.2. The training proceure employs the loss function (2.50). (Top) Evolution of training loss values across epochs. (Center) Evolution of L_2 error of density field over time. (Bottom) Absolute error of density field predicted by the trained CNN-PAL models after 40 simulation time steps. 88

2.12 Outcomes of experiment n.3 performed with a CNN-PAL model trained to predict $N_{\text{step}} = 32$ time steps of the evolution of the double shear layer initial condition depicted in Tbl. 2.2. The training procedure employs the loss function (2.51). (a) Evolution of training loss values across epochs. (b) Evolution of L_2 error of density field over time. (c) Evolution of the sum of L_2 errors of density, components of momentum along the x and y directions, and energy fields over time. 89

2.13 Evolution of energy values in the simulation of the double shear layer from experiment n.3 (see Tbl. 2.2). The growth in energy values for the trained CNN-PAL models after a certain time demonstrate numerically unstable dynamics. 89

3.1 1D linear finite-element shape function (hat function) with nodes and elements enumerated. 92
3.2 Comparison of column-wise (i) $|G_{\text{num}}|/|G_{\text{phy}}|$; grey area shows unstable region $|G| > 1$, (ii) c_N/c; grey area shows region where $0.95 \leq c_N/c \leq 1.05$ and dotted line is $p = 1.5$ and (iii) v_{gn}/v_g; grey area shows GVP region $0.95 \leq v_{gn}/v_g \leq 1.05$ and dotted line is $p = 1$ for the indicated methods (row-wise (a-e)) tabulated in tbl. 3.1. 100
3.3 Inverse mass-matrix spectral plot ($\widehat{A} = \widehat{M}^{-1}$) compared with the dissipation $\widehat{D}$ to illustrate its anti-diffusive character and the stabilization effect of β_c. 102
3.4 Dissipation (solid contour) and GVP region (blue contour) for various β_c values in TG-LW scheme; wavy regions are numerically unstable. 103
3.5 $|G_{\text{num}}|/|G_{\text{phy}}|$ contours for indicated values of parameter γ in the TTGC-γ scheme; (i) grey shaded area indicates numerically unstable regions ($|G_{\text{num}}| > 1$) and (ii) blue shaded region indicates the group velocity preserving zone ($0.95 \leq v_{gn}/v_g \leq 1.05$). 103
3.6 Global CFL number versus maximum stable $\gamma = \gamma_{max}$; stable region obtained from GSA indicated in grey shaded area. . . . 104
3.7 Convection of WP on mesh with discontinuity at junction (⧫ is the junction) showing upstream reflected waves indicated by ⟵ ● and actual wave propagation direction indicated by ⟶ c; ● are the mesh nodes. 110
3.8 Cell and nodal impedance matching problem using local-transfer function; the dash-pot models externally added artificial dissipation. 110

3.9 $|G|$ contours of LTA nodal impedance (a) $|G_{i-1}|$ (left node), (b) $|G_i|$ (junction node) and and (c) $|G_{i+1}|$ (right); solid shaded region is numerical phase speed within $\pm 5\%$ error of exact phase speed (DRP region), vertical striped region indicates the spectral spread of the WP in terms of local Nyquist frequency p; contours of local instability shown in red (thicker) contour lines $(1, 1.1)$. 112
3.10 Contour of reflection ratio magnitude ($|\rho|$) due to nodal impedance mismatch between (a) $(i-1)$-(i) and (b) (i)-$(i+1)$ for the junction problem for varying CFL ($0 < N_c < 1$) and frequency ($0 < p < \pi/2$). 114
3.11 Space-wavenumber idealization and local neighborhood of nodal and elemental impedance to the continuous initial condition $u(x, t^n)$ leading to the solution at the next step $u(x, t^{n+1})$. . . . 115
3.12 LTA model at a Burgers shock (a) initial condition (solid line) and appearance of solution over-/undershoots (dotted line) created by the reflected waves from impedance mismatch and the individual nodal impedance (b-d) from elemental ones. . . 121
3.13 Dissipation $|G|$ and phase angle ϕ of the local nodal transfer function G_{i+1}, G_i, and G_{i-1}; grey shaded region are numerically unstable ($|G| > 1$). Note that the phase angle mismatch between G_{i-1} and G_i is much more severe than G_i and G_{i+1}. . 122
3.14 Time evolution of numerical solution zoomed near the Burgers' shock for the first (left) and second (right) time step from the initial condition. 123
3.15 $|G|$ contours of local transfer function at shock location i for the TTGC-γ scheme for Burgers' equation (a) for variant (i) with Pe = 0.5, (b) variant (ii) with Pe = 0.25 and (c) variant (ii) with Pe = 0.375. 124
3.16 Time step $t = 16\Delta t$ for the evolution of a square wave initial condition under Burgers' equation, where Δt is the simulation time step. (a) for variant (i) with Pe = 0.5, (b) variant (ii) with Pe = 0.25 and (c) variant (ii) with Pe = 0.375. Red lines delimit region near shock front where artificial viscosity is added. 124

4.1 Study n.1. Spatio-temporal evolution of numerical error in the solution to the WP convection (Eq. (4.3)); WP and its Fast Fourier Transform (FFT) along with the comparison of numerical errors of optimized and un-optimized TTGC and the optimal γ_{NLopt} obtained using NLopt are plotted for central wavenumbers (a-c) $k_0h = 0.4$, (d-f) $k_0h = 0.6$, and (g-i) $k_0h = 0.8$ respectively. 131
4.2 (a) Numerical error convergence with mesh refinement (b-e) (finest to coarsest) and the dotted line indicates the third order slope; initial condition and FFT for the mesh refinement levels containing (b) 256, (c) 128, (d) 64 and (e) 32 elements respectively. 133
4.3 Study n.2. (a) Left: Periodic initial condition over an irregular mesh of 101 nodes (or 100 finite-elements). Parameters values are $(k_0h, \alpha, x_0, c) = (0.5, 25, 0.5, 1)$. Right: Fast-Fourier Transform evaluated over the regular counterpart of the actual mesh; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively. 135
4.4 Study n.3. (a) Left: Periodic initial condition over a regular mesh of 101 nodes (or 100 finite-elements). Parameters values are $(k_0h, \alpha, x_0, c) = (0.5, 25, 1.2, 1)$. Right: Fast-Fourier Transform; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of exact solution, NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively. 136
4.5 Study n.4. (a) Initial condition and its frequency domain counterpart. Parameter $a = 1$; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of exact solution, NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively. . . 138

4.6 (a) 1D primal/dual mesh (top) and GN based on the dual mesh (bottom); (b) Unstructured primal/dual mesh in 2D (left) and GN based on the dual mesh (right). Arrows denote the GN message-passing, red boxes are the GN vertices (which are equivalent to the dual mesh nodes, indicated by red dots), blue dots are the primal mesh nodes and green dotted lines are the dual mesh edges. 140
4.7 ML-TTGC architecture. The GNN instances trained for the studies presented in Sec. 4.4 use an encoding block, two message-passing layers and a decoding block, which update the vertices and edges attributes using dense layers with a hidden feature space of size $L = 32$. Each instance has a total of $10,637$ trainable parameters. 142
4.8 (a) Low-frequency WP convection over an irregularly spaced mesh. (b) High-frequency WP convection over irregular mesh. Top: TTGC-γ (Default) and analytical/exact solution. Center: ML-TTGC and analytical/exact solution. Bottom: GNN predicted parameters for locally tunable TTGC-γ_L. Initial conditions are not part of the training set. 145
4.9 (a) High-frequency WP convection over regular mesh. Top: TTGC-γ (Default) and analytical/exact solution. Bottom: ML-TTGC and analytical/exact solution. Initial condition is not part of the training set. (b) L2 norm of the error for TTGC-γ (Default) and ML-TTGC, comparing their dissipation levels over time. 145
4.10 Numerical solution obtained at time steps (a) $t = 35\Delta t$ and (b) $t = 40\Delta t$ (TTGC-γ and ML-TTGC scheme with predicted parameters γ_{opt} and ν_{opt}) to the inviscid Burgers' equation for the square-wave initial condition, where Δt is the simulation time step. Parameter values used are (a, x_0, x_1) = (1., 0.2, 0.65) for (a) on the left and (a, x_0, x_1) = (2., 0.4, 0.75) for (b) on the right; data is not part of the training set. 149
4.11 Numerical solution obtained at time steps (a) $t = 40\Delta t$ and (b) $t = 70\Delta t$ (TTGC-γ and ML-TTGC with predicted γ_{opt} and ν_{opt}) to the inviscid Burgers' equation for the sine wave initial condition, where Δt is the simulation time step; data is not part of the training set. 150

5.1 Different finite volume schemes: (a) cell-centered, (b) cell-vertex. Inspired from Crumpton et al. (1993). 153
5.2 Illustration of cell residuals computation via trapezoidal rule (a) and of their scattering to nodes (b). 154
5.3 (a) Unstructured regular triangular grid and (b) its perturbed counterpart. 160
5.4 Surface plots of wave packet whose dynamics are governed by the 2D Convection equation. Time step n. 360. (a) Analytical solution, (b) Output from TTGC scheme, (c) Output from ML-TTGC scheme trained by pure $l2$ loss function, (d) Output from ML-TTGC scheme trained by pure TVD loss function. 161
5.5 (a, b, c) Surface plots and (d) cut along $x = 0.5$ of isentropic vortex after completing one box turn (periodic conditions apply). (a) Analytical density field, (b) Output density field from TTGC scheme, (c) Output density field from ML-TTGC scheme trained on pure $l2$ loss function. 163
5.6 Dynamics of fluid's momentum component along the x direction (ρu) for double shear layer problem. (a) TTGC time step n. 600 (just before simulation crash), (b) ML-TTGC time step n. 600, (c) ML-TTGC time step n. 2400, (d) ML-TTGC time step n. 3600. 164

B.1 Illustration of `x[0] += 1.` operation done either (a) in-place (actual mutation of `x`) or (b) by re-assignement of `x`. 196
B.2 Illustration of how `du` is computed either by (a) `du_mutation` or (b) `du_reassignment`. 198
B.3 Computational graph, *forward primal* and *adjoint* evaluation traces for the program `du_reassignment`. 199
B.4 Schematic representation of `circshift` operation on some array `w`. 201
B.5 Adjoint programs of (a) `du_mutation` (Fig. B.2(a)) and (b) `du_reassignment` (Fig. B.2(b)). Note that the arrays `ub` and `dub` respectively corresponding to the reverse derivatives of `u` and `du` are initialized to zeros beforehand and given as inputs to each function. 202

B.6 Benchmarking results for the 1D discretization operator problem regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes. 204
B.7 Benchmarking results for the 3D discretization operator problem regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes. 206
B.8 Portions of the algorithms of the (a) mutating and (b) non-mutating TTGC scheme implementations (3D Euler equations; tetrahedral meshes). 207
B.9 Benchmarking results for the TTGC scheme (3D Euler equations; tetrahedral meshes) regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes. 209

C.1 Comparison of local wavenumber analysis using Eikonal and JST sensor; (a-c) WP function and (d-f) Gaussian pulse function. 211

D.1 (a) Thermodynamic cycle of a shock wave, (b) waves overtaking each other, (c) multiple solution at shock, (d) the shock entropy solution and (e) x-t diagram of the shock problem. . . 214

List of Tables

1.1 Overview of presented neural network models under relational inductive bias framework. 37

2.1 Parameters common to all CNN-PAL models used in experiments. 76
2.2 Parameter values used in all experiments of Sec. 2.3.2. 79

3.1 Amplification factor (real and imaginary part) for indicated numerical method. 99

4.1 Sets of initial conditions for ML-TTGC applications. 143

Introduction

In the aerospace industry, engineering systems design must consider the effects of fluid motion around airfoils or inside nozzles, among other components. Nevertheless, real-world prototyping of these systems can be excessively expensive. Their numerical simulation through Computational Fluid Dynamics (CFD) has become essential to explore design spaces and speed up development (Hirsch, 2007). First frameworks such as Boeing's PAN AIR (Magnus and Epton, 1981) employed simple potential flow models, where the velocity field is assumed to be irrotational. Over the years, the increasing availability of computational power has allowed manufacturers to employ more detailed mathematical models to improve predictions. The most accurate model that describes the motion of viscous fluid substances is the set of Navier-Stokes (NS) equations. In the motion of viscous fluids, velocity and pressure fluctuations of chaotic nature may arise whenever inertial forces are strong enough compared to viscous forces. These fluctuations constitute a phenomenon called turbulence. Kolmogorov (1941) showed that turbulent flows are characterized by an energy transfer from their largest to their smallest eddies. Spatial discretization of the NS equations down to the length scale of the flow corresponding to the smallest eddies remains prohibitively expensive in industrial configurations (Sagaut, 2006). State-of-the-art simulation frameworks still consider modified forms of the underlying Partial Differential Equations (PDEs). In combustion and aero-acoustics, for instance, a filtered version of the NS equations is solved instead, such as in Large-Eddy Simulations (LES) (Poinsot and Veynante, 2011; Wagner et al., 2007). In LES, the effects of eddies of size under a given threshold are modeled using subgrid closure rules. The first LES is credited to Deardorff (1970) for the case of a simple plane Poiseuille flow. More recently, Pérez Arroyo et al. (2021) performed the first high-fidelity LES of a full aircraft engine, thanks to the evolution of computational resources.

The increase in available computational power has also contributed to the emergence of large-scale applications in an apparently uncorrelated field: Machine Learning (ML). It aims at extracting knowledge from data to improve task automation (Jordan and Mitchell, 2015). An ML architecture is a parametrized function to be adjusted in order to execute a given task better. The process of using data to tune the parameters of an ML model is called training. The widespread availability of Graphics Processing Units (GPUs) enabled the training of ML models at a much faster pace than with previous devices (Chellapilla et al., 2006; Steinkraus et al., 2005). As a consequence, during the past two decades, ML applications in a myriad of technology areas have been developed. Some examples are the automation of speech recognition (Nassif et al., 2019), image segmentation (Minaee et al., 2021), and autonomous driving (Kiran et al., 2022). Recently, the ML-based chatbot GPT-4 exhibited human-level performance on various professional and academic benchmarks (OpenAI, 2023). The ability of an ML model to solve problems across different contexts or disciplines is called Artificial General Intelligence (AGI) (Goertzel, 2014). The capacities of GPT-4 in handling tasks that span mathematics, vision, coding, law, psychology, medicine, and more have sparked a debate on whether it could be considered an early (yet incomplete) version of an AGI system (Bubeck et al., 2023). It opens promising perspectives for using ML models in all aspects of human life.

Most of the ML applications mentioned above employ (artificial) neural networks: a class of ML architectures inspired by information processing in biological systems (Bishop, 2007). They are universal approximators of continuous functions over compact domains (Hornik et al., 1989). This fact triggered the scientific community's interest in possible ML applications in fields such as CFD, from turbulence modeling (Duraisamy et al., 2019) to the development of ML-based numerical solvers (Karniadakis et al., 2021). The former is concerned with using neural networks as surrogate models of the unresolved scales of fluid motion. The latter aims to employ neural networks to resolve the governing equations. Its goal is to improve the accuracy of the numerical predictions at cost levels similar to those of a traditional solver. This thesis will only address the development of ML-based numerical solvers. In this context, Raissi et al. (2019) proposed to approximate the fluid flow function of space and time by directly minimizing the solution residual on a sample of observation points, a popular approach known as Physics-Informed Neural Networks (PINNs). This is a powerful method for problems where multiple constraints compete or when parts of the problem are not well known

(e.g., boundary conditions in hydrology, initial conditions in weather predictions...). However, for well-defined problems, it is less accurate compared to direct resolution (Fuks and Tchelepi, 2020). In the scope of fluid mechanics, P. Sharma et al. (2023) noted that the error levels in the prediction of turbulent flows using PINNs are often orders of magnitude higher than those of laminar flows. This is due to problems associated with non-linearity, high-dimensionality, and multi-scale physics. S. Wang et al. (2022) showed that PINNs suffer from spectral bias, a well-known pathology that prevents deep fully-connected neural networks from learning high-frequency functions (Rahaman et al., 2019). Markidis (2021) assessed the potential of PINNs as linear solvers in the case of Poisson's equation (found in incompressible flow solvers, among many other applications in scientific computing). Markidis (2021) also found that PINNs learn the low-frequency components of the solution field more easily than the high-frequency ones, showing that accuracy is a limiting factor for using PINN linear solvers alone, and eventually recommended applying hybrid strategies where PINNs are integrated into traditional linear solvers. By doing this, one could develop ML-based solvers that are competitive with state-of-the-art ones, both performance- and accuracy-wise.

Combining traditional numerical methods with ML to propose a new class of ML-based numerical schemes for CFD is the main focus of this thesis. This project does not aim to rely on an ML model alone to unveil some system dynamics but to use the traditional methods as inductive biases. More precisely, ML models will be used to tune the parameters of numerical schemes. Tuning discretization parameters has been essential to practical numerical algorithms to solve PDEs. They aim to ensure that the numerical solution satisfies certain desired but critical characteristics of the physical problem. These characteristics relate mainly to stability, dispersion, and dissipation numerical errors. Rajpoot et al. (2010) and T. K. Sengupta, Rajpoot, and Bhumkar (2011) proposed optimized compact finite-difference schemes that accurately capture the dispersion, dissipation, and group velocity characteristics of the physical problem through a comprehensive approach considering both temporal and spatial discretizations along with the governing equation. Lele (1992) performed parameter optimization of compact finite-difference schemes to provide improved representation of a range of spatial scales. However, they considered exact time advancement, which leads to an incorrect evaluation of the dispersive error characteristics. Tam and Webb (1993) introduced finite-difference schemes whose parameters are optimized to reduce truncation error for first order spatial derivative. Again,

space and time discretizations were treated independently, which results in an incorrect numerical dispersion relation. Besides, they used a four time level method which introduces two spurious numerical modes apart from the physical mode (as explained in T. K. Sengupta and Dipankar (2004)). In the realm of ML-based schemes, Bar-Sinai et al. (2019) and Kochkov et al. (2021) employed ML models to predict coefficients of a finite-difference method, leading to accurate numerical solutions on relatively coarse grids. However, they did not establish conditions for the stability of the resulting scheme. Using neural networks, Stevens and Colonius (2020) improved a fifth-order shock-capturing Weighted Essentially Non-Oscillatory (WENO) scheme (Jiang and Shu, 1996). Their ML model predicted optimal perturbations to WENO coefficients. Nevertheless, the only constraint applied to the perturbed coefficients was an affine transformation which guarantees that they sum up to one. Hence, the resulting numerical method was only first-order accurate. Kossaczká et al. (2021) proposed instead to perturb the smoothness indicators of the WENO method. Their resulting scheme presented less diffusion and less overshoot in shocks than the original WENO scheme while maintaining high order of accuracy in the smooth regions of the flow. Bezgin, Schmidt, et al. (2021) applied neural networks to adaptively reconstruct nonlinear fluxes in the context of nonclassical shock waves, i.e., shock waves which violate the Lax entropy condition (LeFloch, 2002).

This thesis concerns developing ML-based numerical schemes on irregular and unstructured grids typical of industrial CFD. First, a novel spectral analysis that can be applied to non-linear systems of equations and irregular meshes is proposed. It is worth noting that the proposed analysis does not consider aliasing errors, which results from evaluation of product terms in a finite resolution grid (T. Sengupta, 2013). Such a spectral analysis establishes conditions for optimizing numerical schemes locally in time and space. An ML model is then used to predict optimal values of a parameter of a traditional numerical method used in LES. Finally, applications to problems governed by the Convection/Burgers' equations on 1D irregular meshes and the Convection/Euler equations on 2D irregular triangular meshes demonstrate how the ML-based numerical scheme can outperform a traditional one accuracy-wise.

This manuscript is structured as follows:

Chapter 1 - This chapter introduces the reader to the Machine Learning field. It will be seen that commonly used neural network models like the Multilayer

Perceptron (MLP), the Convolutional Neural Network (CNN), and the Graph Neural Network (GNN) arose from problems outside the scope of scientific discovery but have found many applications therein due to their specific features.

Chapter 2 - This chapter briefly reviews the development of numerical schemes for hyperbolic conservation laws, typical to CFD. Subsequently, the data-driven numerical scheme of Bar-Sinai et al. (2019) and Kochkov et al. (2021) is presented. Their method employed a CNN model to explore the space of possible finite-difference spatial discretization coefficients obtained via Taylor series expansions. It is applied to the system of compressible Euler equations in 2D cartesian grids. The method does not guarantee stability despite improving accuracy levels on a particular set of problems.

Chapter 3 - A theoretical framework that establishes accuracy and stability conditions for coupled ML-PDE systems is developed in this chapter. A novel spectral analysis based on local spatio-temporal features named the Local Transfer-function Analysis (LTA) is proposed. One-dimensional problems governed by the Convection and Burgers' equations are studied under LTA. The considered numerical scheme belongs to the family of Taylor-Galerkin schemes used in LES.

Chapter 4 - The Two-step Taylor-Galerkin C (TTGC) numerical scheme of Colin and Rudgyard (2000) is modified to incorporate a cell-valued parameter provided by a GNN model, which is constrained to respect stability conditions predicted by LTA. The resulting numerical method is termed ML-TTGC and can address some limitations of the original TTGC scheme by successfully damping spurious oscillations on irregular meshes and preserving amplitudes of high wavenumber waves in the linear convection problem.

Chapter 5 - Finally, an extension of the method to higher dimensions is sought in Chapter 5. In particular, an implementation of the TTGC scheme for 2D Convection and Euler problems on unstructured meshes is coupled to a GNN model identical to the one used in the 1D experiments of Chapter 4. The resulting framework confirms ML-TTGC's capacity to damp wiggles on irregular meshes. Moreover, in the case of a double

shear layer problem governed by the Euler equations, ML-TTGC proves to be more stable than the original scheme.

Fig. 1 summarizes the manuscript's structure and highlights the relationships between the chapters.

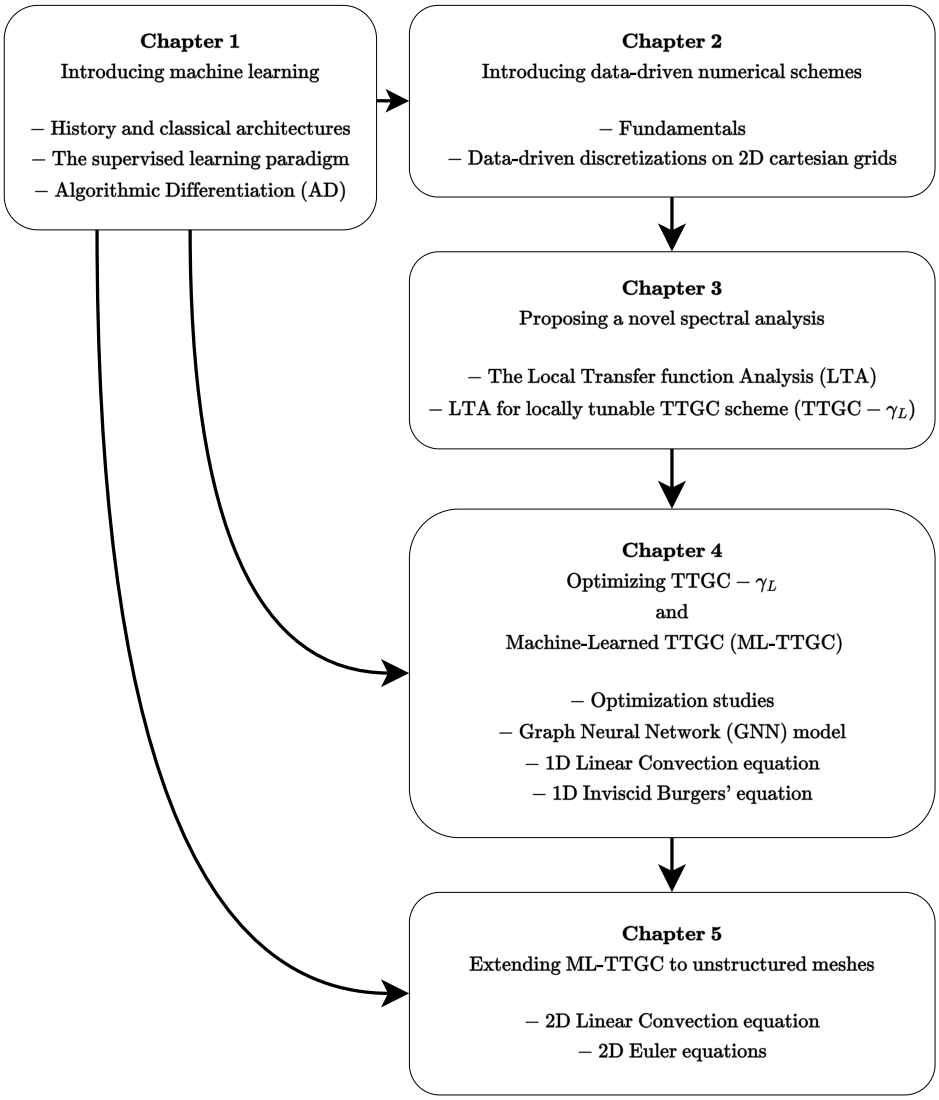


Figure 1: Manuscript's structure. Arrows denote relationships between chapters. More precisely, an arrow from chapter A to chapter B shows that concepts defined in chapter A will be employed in chapter B.

Chapter 1

Introducing machine learning

This chapter surveys the field of Machine Learning (ML). Sec. 1.1 presents standard neural network architectures like multilayer perceptrons and convolutional neural networks. They have been applied by researchers in many scientific applications, despite originating in the image recognition realm. This section also presents graph neural networks. Unstructured data (typical to industrial CFD applications) are naturally expressed in graphs, such that graph neural networks are employed in the data-driven numerical schemes developed in Chapters 4 and 5. These neural network models are trained using gradient-based methods. Sec. 1.2 introduces the reader to algorithmic differentiation, an efficient way to compute gradients of the objective function for training with respect to the ML model parameters. This chapter ends in Sec. 1.3 by presenting some algorithmic differentiation frameworks.

1.1 Machine learning: from checkers to scientific computing

IBM engineer Arthur Samuel is credited for popularizing the term *machine learning* in his attempts to program computers to learn from data during the 1950s. He believed that learning devices would suppress the need for detailed programming effort in problem-solving, as machine-learned rules replace hard-coded algorithmic steps. The realm of games seemed to him a convenient way to develop the first learning techniques due to their relative

Figure 1.1: Arthur Samuel and IBM 700 (1956). From Gladchuk (2020).

boundedness in comparison to life problems in general. In 1952, Samuel then created a checkers-playing program that would not build all possible paths until the game's conclusion, but rather evaluates a scoring function estimating winning chances at any given game state (Samuel, 2000). Such a scoring function also depended on a history of game actions, from which the machine is expected to continuously improve while playing. Samuel's methods relate to modern techniques of the so-called *reinforcement learning* field (Richard S. Sutton, 2018). Remarkably, his program was able to play better than himself after some 10 hours of device-playing time. Fig. 1.1 shows him with an IBM 700 and a checkers board.

Later on, the American psychologist Frank Rosenblatt (1958) invented the perceptron, a learning unit inspired from human brain neurons that would trigger the modern *deep learning* field. Rosenblatt's perceptron was designed to become a machine for pattern recognition (capable of distinguishing geometrical shapes, letters of the alphabet, among others), under the ideas stated in the following words :

> " [...] our objective has been to discover a physical system, or abstract model, which will be capable of "perceiving" its environment, and learning to recognize those objects or events which

> it has perceived in the past. However, since it is our purpose to understand the actual mechanisms employed by the brain, rather than simply to construct a new type of computing device, the perceptron models are constrained in their organization and dynamic properties by what is known of the biological nervous system. Rather than attempting to "invent" or "construct" a machine which will calculate such things as similarities or geometrical properties of stimuli, the approach has been to begin with a hypothetical network of idealized neurons, or nerve cells, resembling the brain in its general organization, and then analyze the system mathematically to determine whether or not it possesses "psychological" properties of interest. " (From Rosenblatt (1962), p. 574)

Its original model consisted of three layers S, A, R connected in series to compose a perception mechanism (Van Der Malsburg, 1986), as illustrated in Fig. 1.2. The layer S corresponds to a *sensorial surface* (e.g., eye's retina) that reacts to environment stimuli thanks to its set of sensory units. Their response then fed layer A as a set of *association* cells responsible for detecting properties that are particular to the object to be recognized. Finally, layer R of *recognition* units receives the outputs from A and should fire only for a specific type of pattern projected on S (e.g., the letter **C** shown in Fig. 1.3(a)).

In mathematical terms, the output signal y from a given R-unit can be expressed as (Bishop, 2007)

$$y := h\left(\sum_i w_i\ \phi(x_i)\right) \equiv h\left(\boldsymbol{w}^\top \phi(\boldsymbol{x})\right), \tag{1.1}$$

where $\boldsymbol{x}$ is the input signal to the A-layer. This signal is a vector of numerical values that represent the object to be recognized. Hereafter, any tensor of numerical values which identify a measurable property is referred to as a *feature*. The input features $\boldsymbol{x}$ are transformed by some non-linear function ϕ (e.g., a sigmoid). Finally, the variable $\boldsymbol{w}$ stands for the (adaptive) weights of the A-layer [1] and h is a step function such that

$$h(z) := \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{if } z < 0. \end{cases}$$

[1] Hardware-wise, the weights were modelled by variable resistors (also known as potentiometers) to be tuned during the learning process.

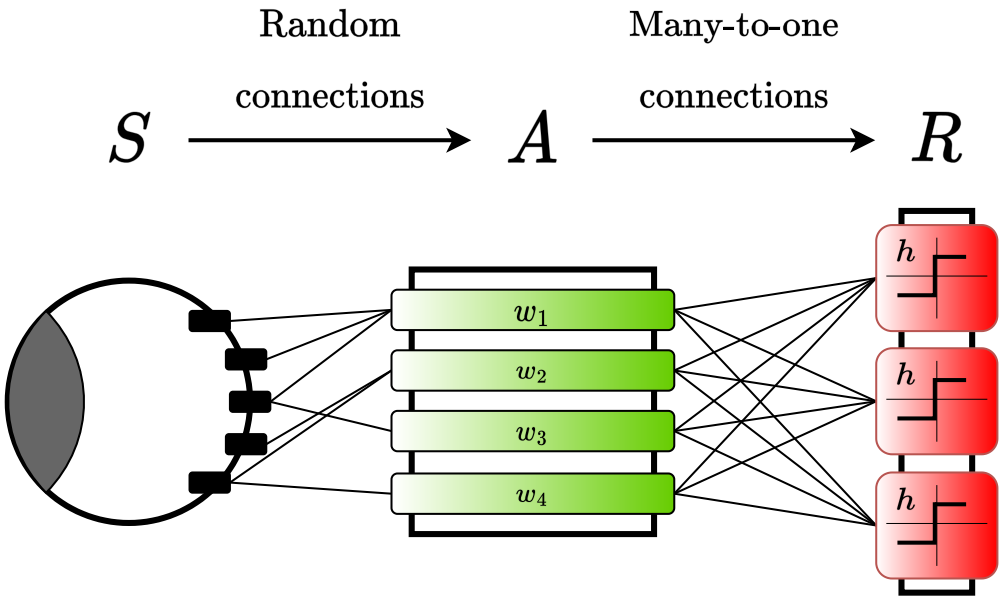


Figure 1.2: Schematic representation of the original (photo-)perceptron. Layers S, A and R refer to the sets of *sensory*, *association* and *recognition* units/cells, respectively. In the example above, w_1, w_2, w_3, w_4 are the weights of the A-layer and h is the threshold function accounting for neuron "firing".

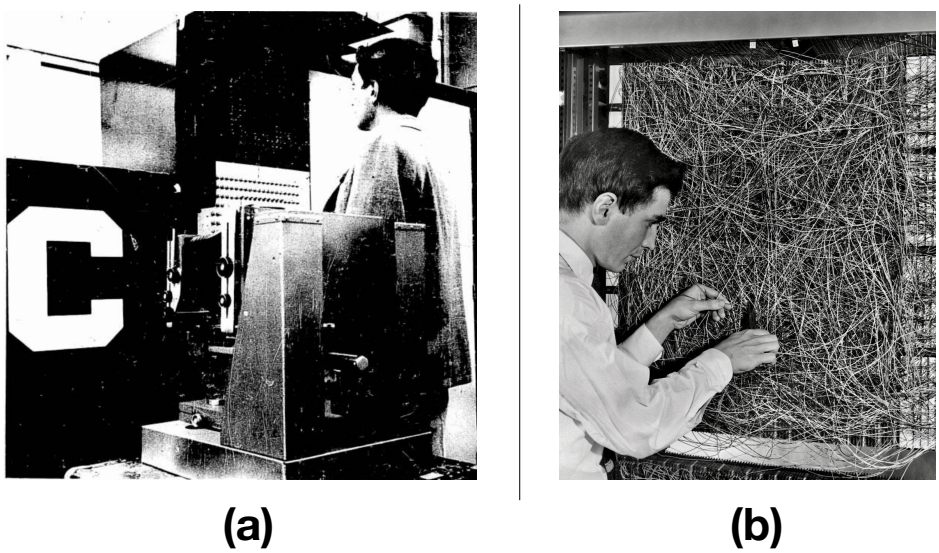


Figure 1.3: (a) Camera device capturing an image of a letter **C** drawn on a wall (1962). From Rosenblatt (1962). (b) Frank Rosenblatt manipulating wired connections of the Mark I Perceptron machine (1960). From dnvdk (2022).

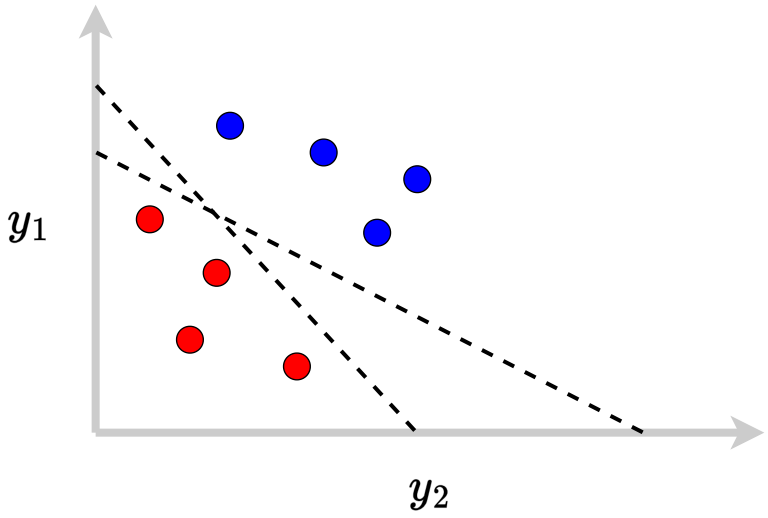


Figure 1.4: The sets of blue and red dots are linearly separable, as indicated by the dashed straight lines. Infinitely many separation lines exist and the perceptron learning algorithm cannot distinguish between them.

The threshold function h accounts for neuron "firing" for a specific pattern on screen.

The perceptron learning algorithm can be described as follows: for a given pattern $\boldsymbol{x}$ on screen, the output signal y is computed and, if the pattern is classified incorrectly (for instance, the neuron "fired" while it should not), a change of the weights values $\boldsymbol{w}$ take place. More precisely, the following updating rule is called for each misclassified pattern $\boldsymbol{x}$ in a dataset:

$$\boldsymbol{w} \longleftarrow \boldsymbol{w} + \eta\, \phi(\boldsymbol{x})\, t, \tag{1.2}$$

where t is an identifier for the class of the pattern and η is some step size. The perceptron was designed for binary classification, so that $t \in \{-1, +1\}$. If the dataset the model learns from is linearly separable [2], the *perceptron convergence theorem* as stated in Rosenblatt (1962) guarantees that the learning algorithm above will find an exact solution in a finite number of steps. However, in practice, it can be difficult to distiguish a nonseparable problem from one that is simply slow to converge. Besides, many solutions to a separable problem can exist (as shown in Fig. 1.4), and a perceptron's converged state will depend on the weights initialization. Besides these limitations, Rosenblatt's perceptron as a neurophysiological model was criticized by White (1963), which stated that it lacked detailing of the temporal characteristics of a neurone firing and that most of the reported experiments dealt with figures that illuminate a large fraction of the retina whereas the device performance in other cases remained uninvestigated. The most impactful review of this work came only later on in 1969, though: Minsky and Papert (1969) in their famous book *Perceptrons* show that the original perceptron was able to learn only linearly separable problems. Despite acknowledging that increasing the number of A-layers could help solving non-linear ones, the reflections developed inside this book led to a substancial decline in research funding for neural networks.

The period between 1969 and the early 1980s is usually recognized as an *Artificial Intelligence (AI) winter*, where the field was surrounded by pessimism and criticism. Renewed interest would come after the works of Hopfield (1982) on associative memory systems. Hopfield networks are models that can reconstruct data after being fed with partial versions of the same data. For example, if the dataset some Hopfield model has learned from is

[2] In binary classification, two sets of points are linearly separable if and only if there exists a line in the Euclidean plane separating them. This is better illustrated in Fig. 1.4.

made of photograph samples, the given model exposed to a piece of one of these photographs is capable of reconstructing the original. It was theoretically proven that a Hopfield network can reliably reconstruct up to $0.138 \times N$ samples, where N is the number of learning units in the model (Stuart Russell, 2010). Finally, the works of Rumelhart, Hinton, et al. (1986) come as a major milestone in AI research. They proposed a new architecture called the multilayer perceptron, which is depicted in the next section.

1.1.1 Multilayer perceptrons

As opposed to Rosenblatt's original system, currently named single-layer perceptron [3], *multilayer perceptrons* (MLPs) have at least three A-R sets of layers. In addition to this, the threshold function [4] of each perceptron composing an MLP is arbitrary. Finally, MLPs can either perform (multiclass) classification or regression, while Rosenblatt's perceptron was restricted to binary classification tasks. By solving the XOR problem (to predict outputs of *exclusive or* logic gates) and by detecting input symmetry (to determine if two strings are symmetric about their center), among other problems (Rumelhart, McClelland, et al., 1988), MLPs have re-ignited interest in neural networks for machine learning. Fig. 1.5 illustrates an MLP in the modern sense. For the sake of rigorousness, the reader will find below a formal definition of the MLP architecture :

Definition 1 (Multilayer perceptron (MLP))**.** An N-layer multilayer perceptron is the composition

$$d_1 \circ d_2 \circ \ldots \circ d_N \tag{1.3}$$

of functions

$$\begin{aligned} d_k : \mathbb{R}^{N_k^{\text{in}}} &\longrightarrow \mathbb{R}^{N_k^{\text{out}}} \\ \boldsymbol{x} &\longmapsto \sigma_k \left(\underline{\boldsymbol{w_k}}\ \boldsymbol{x} + \boldsymbol{b_k}\right), \end{aligned} \tag{1.4}$$

$$\begin{aligned} &(k \in [\![1, N]\!],\ N > 2) \\ &(N_k^{\text{in}} \in \mathbb{N},\ N_k^{\text{out}} \in \mathbb{N}) \end{aligned}$$

[3] The S-, A- and R-layers from Fig. 1.2 are grouped to compose a single-layer perceptron.

[4] In the modern terminology, it is named activation function.

where σ_k is a non-linear activation function, $\underline{\boldsymbol{w_k}} \in \mathbb{R}^{N_k^{\text{out}} \times N_k^{\text{in}}}$ is a matrix of trainable weights and $\boldsymbol{b_k} \in \mathbb{R}^{N_k^{\text{out}}}$ is a vector of trainable biases. Each d_k is named a *dense* layer of the MLP. In order to exist, the composition (1.3) must obey the constraint

$$N_k^{\text{in}} = N_{k-1}^{\text{out}} \qquad (k > 1).$$

The layer d_1 is called the *input* layer, d_N is named the *output* layer, and the layers d_k with $k \in [\![2, N-1]\!]$ are referred to as the *hidden* layers of the MLP. In the particular case where $N_k^{\text{in}} = N_k^{\text{out}} = N^{\text{feat}}\ \forall k \in [\![2, N-1]\!]$, one says that N^{feat} is the number of hidden features of the MLP.

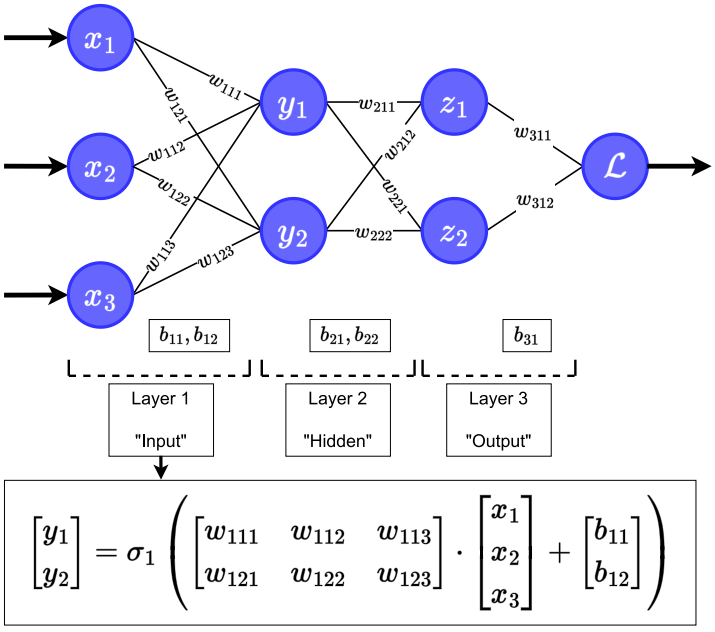


Figure 1.5: Schematic representation of a 3-layer MLP with its sets $\{w_{\texttt{idx\_layer, idx\_output, idx\_input}}\}$ of weights and $\{b_{\texttt{idx\_layer, idx\_output}}\}$ of biases. Connections between nodes of the computational graph store weights that take part in the matrix-vector multiplication explicited in the bottom part of the image. Since all nodes from one layer are linked to all nodes of the subsequent one, they are usually referred to as densely-connected or simply *dense* layers.

Image recognition, the primary task addressed by Rosenblatt's perceptrons can instead be tackled by MLPs. For instance, Lecun et al. (1998) showed that a simple 2-layer MLP can reach an accuracy level of 95.3% on a 10-label classification problem to recognize handwritten digits. The famous MNIST (Modified National Institute of Standards and Technology) dataset which contains $70,000$ samples of 28×28 pixel images of digits written by American Census Bureau employees and high-school students was used to train the MLPs. Fig. 1.6 shows examples of images from this dataset.

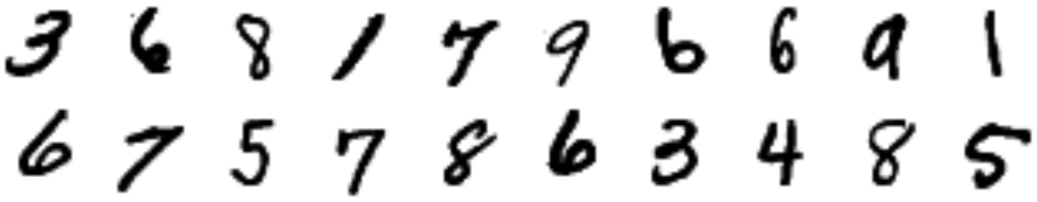


Figure 1.6: Examples from the MNIST dataset. From Lecun et al. (1998).

Lecun et al. (1998) proposed another architecture to solve the MNIST recognition problem with far fewer trainable parameters for a given level of accuracy. This ML model has replaced hand-engineered feature extraction methods to identify objects in images and will be depicted in the next section.

1.1.2 Convolutional neural networks

It is possible to constrain MLPs to solve the MNIST recognition problem more efficiently. These constrained networks were termed *convolutional neural networks* (CNNs). The idea is to group spatially correlated pixels from the images and process them through shared weights. Fig. 1.7 illustrates the concept with an example. CNNs employ fewer trainable parameters than MLPs to reach similar accuracy levels in an image recognition task. On the one hand, each of the output pixels of a dense layer is a function of all of its input pixels. On the other hand, each of the output pixels of a convolutional layer is a function of a small amount of its input pixels. More precisely, the value of a pixel in the output of a convolutional layer is influenced only by a local neighborhood of pixels from the input. This inherent bias embedded in such CNN architectures towards the local neighborhood of pixels is called the *locality inductive bias*. Furthermore, the trainable kernel and biases of a convolutional layer are independent of a pixel's position in an image. CNNs

transform portions of an image independent of their location in the picture itself [5], a property called *spatial translation invariance*.

A formal definition of a CNN follows:

Definition 2 (Convolutional neural network (CNN))**.** An N-layer convolutional neural network is the composition

$$c_1 \circ c_2 \circ \ldots \circ c_N \tag{1.5}$$

of functions

$$\begin{aligned} c_k : \mathbb{R}^{D_k^{\text{in}}} &\longrightarrow \mathbb{R}^{D_k^{\text{out}}} \\ \underline{\boldsymbol{x}} &\longmapsto c_k(\underline{\boldsymbol{x}})[m,n,q] \\ &:= \sigma_k \left(\boldsymbol{b_k}[q] + \sum_{i=-\Delta_k^x}^{\Delta_k^x} \sum_{j=-\Delta_k^y}^{\Delta_k^y} \sum_p \underline{\boldsymbol{w}}_k[i,j,p,q] \; \underline{\boldsymbol{x}}[m+i,n+j,p] \right), \end{aligned} \tag{1.6}$$

$$\begin{aligned} &(m \in [\![1, N_x]\!],\ n \in [\![1, N_y]\!],\ p \in [\![1, N_k^{\text{in}}]\!],\ q \in [\![1, N_k^{\text{out}}]\!]) \\ &(k \in [\![1, N]\!],\ N > 2) \\ &(D_k^{\text{in}} \equiv N_x \ \times \ N_y \ \times \ N_k^{\text{in}}) \\ &(D_k^{\text{out}} \equiv (N_x - \Delta_k^x + 1) \ \times \ (N_y - \Delta_k^y + 1) \ \times \ N_k^{\text{out}}) \\ &(N_k^{\text{in}} \in \mathbb{N},\ N_k^{\text{out}} \in \mathbb{N},\ N_x \in \mathbb{N},\ N_y \in \mathbb{N},\ \Delta_k^x \in \mathbb{N},\ \Delta_k^y \in \mathbb{N}) \end{aligned}$$

where σ_k is a non-linear activation function, N_x and N_y are the number of pixels composing the image along the x and y directions, respectively, $\underline{\boldsymbol{w}_{\boldsymbol{k}}} \in \mathbb{R}^{(2\Delta_k^x+1) \ \times \ (2\Delta_k^y+1) \ \times \ N_k^{\text{in}} \ \times \ N_k^{\text{out}}}$ is a trainable convolutional kernel/filter and $\boldsymbol{b_k} \in \mathbb{R}^{N_k^{\text{out}}}$ is a vector of trainable biases. Each c_k is named a *convolutional* layer of the CNN. In order to exist, the composition (1.5) must obey the constraint

$$N_k^{\text{in}} = N_{k-1}^{\text{out}} \qquad (k > 1).$$

The layer c_1 is called the *input* layer, c_N is named the *output* layer, and the layers c_k with $k \in [\![2, N-1]\!]$ are referred to as the *hidden* layers of the CNN. In the particular case where $N_k^{\text{in}} = N_k^{\text{out}} = N^{\text{feat}}\ \forall k \in [\![2, N-1]\!]$, one says that N^{feat} is the number of hidden features of the CNN.

[5] For example, a CNN employed as a binary classifier to distinguish images of cats from images of dogs will be able to identify a cat's ear independent of its position in the frame.

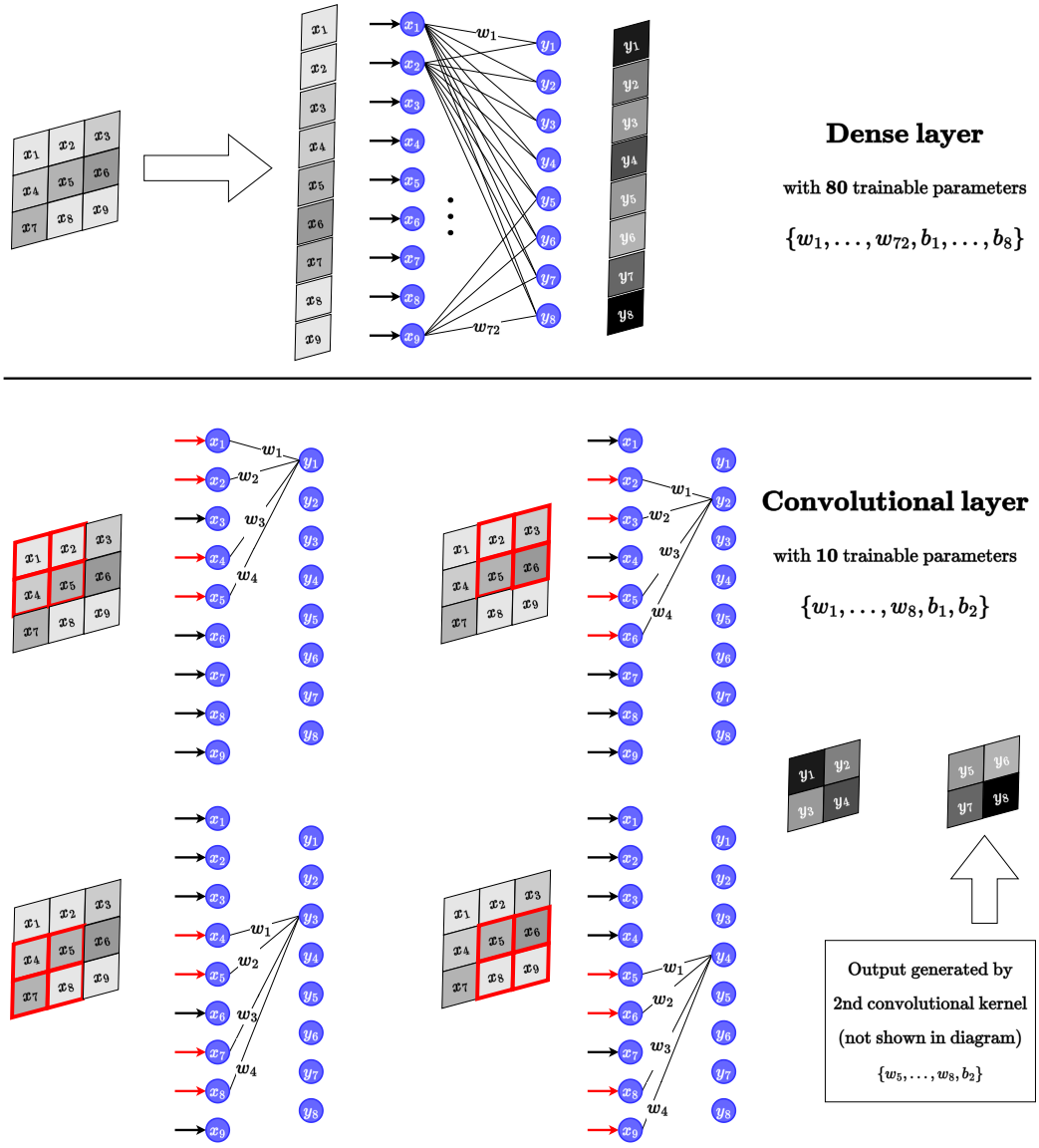


Figure 1.7: Schematic comparison between a dense layer (top) and a convolutional layer (bottom). A 3×3 image with pixel values $\{x_1, \ldots, x_9\}$ is given as input and converted to an 8-pixel representation with values $\{y_1, \ldots, y_8\}$. While the dense layer requires all-to-all connections between input and output neurons, the convolutional layer shares a relatively small set of weights (also known as convolutional kernel/filter) among spatially correlated pixels of the input image (highlighted in red). This allows CNNs to perform image recognition tasks with far fewer trainable parameters than corresponding MLPs.

In a scientific context, many applications for CNNs can be found. In the realm of premixed turbulent combustion modelling, for instance, Lapeyre et al. (2019) trained CNNs to estimate sub-grid flame surface density. Their CNN model was able to efficiently extract flame topology and predict sub-grid scale wrinkling, outperforming classical algebraic models. In the field of composite materials, Ejaz et al. (2022) trained CNNs to map images of copper-carbon nanotube networks to corresponding electrical and thermal conductivities. They highlight the runtime savings obtained from running a CNN surrogate model for conductivity as opposed to a convectional numerical simulation. Regarding data-driven discretizations, the main focus of this thesis, Bar-Sinai et al. (2019) and Kochkov et al. (2021) employed CNNs to approximate spatial derivatives in coarse grids based on downsampled dynamics from high-resolution numerical simulations. In Chapter 2, the reader will find an application of this method. It is restricted to structured grids, though, where instantaneous physical fields can feed a CNN as simple images. In order to overcome this restriction, a neural network architecture that handles grids as graphs can be used. The next section is dedicated to describing this architecture, particularly suitable for industrial CFD where unstructured grids are often used.

1.1.3 Graph neural networks

Battaglia et al. (2018) presented the CNN properties of spatial translational invariance and locality inductive bias within a more general framework. In such a framework, learning concerns the representation of entities and their relations. In the specific case of CNNs, the pixels of an image are the entities whose relations are synthesized by the locality inductive bias and the spatial translation invariance properties. In the more general case where the relations between entities are arbitrary, they proposed employing graphs for their representation. The set of nodes composing a graph stand for the entities, while its set of edges represent their relations. The resulting ML model destined to learn from the entities/relations representation in a graph was named *graph neural network* (GNN). In a GNN, the relations between entities are encoded as *messages* carried by the edges. These messages are exchanged between nodes and transformed across the layers of a GNN, which are termed *message-passing* layers. Tbl. 1.1 summarizes the entities and their relations for the neural network models defined so far. Finally, Fig. 1.8 depicts the operations performed in a message-passing layer of a GNN.

Component	Entities	Relations	Relational inductive bias	Invariance
Dense layer	Units (neurons)	All-to-all	Weak	-
Convolutional layer	Pixels	Local	Locality	Spatial translation
Message-passing layer	Nodes	Edges	Arbitrary	Node/edge permutations

Table 1.1: Overview of presented neural network models under the relational inductive bias framework of Battaglia et al. (2018). Inspired from Battaglia et al. (2018).

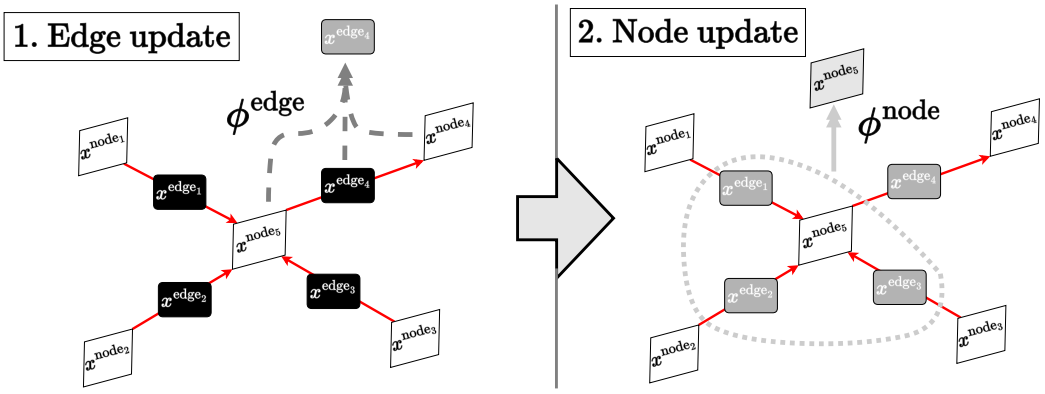


Figure 1.8: Schematic representation of operations performed in a message-passing layer. First, update of the edge features (i.e., the messages) takes place by the action of ϕ^{edge} on the current features of each edge, along with the features of the sender and receiver nodes composing it. Last, ϕ^{node} updates node features by acting on the current features of each node along with the updated features of the incoming edges (i.e., the updated incoming messages). The building blocks ϕ^{edge} and ϕ^{node} are MLPs or CNNs, typically.

A formal definition of a GNN follows:

Definition 3 (Graph neural network (GNN))**.** An N-layer graph neural network is the composition

$$g_1 \circ g_2 \circ \ldots \circ g_N \tag{1.7}$$

of functions

$$\begin{aligned} g_k : \left(\mathbb{R}^{D_k^{\text{in, node}}}, \mathbb{R}^{D_k^{\text{in, edge}}}\right) &\longrightarrow \left(\mathbb{R}^{D_k^{\text{out, node}}}, \mathbb{R}^{D_k^{\text{out, edge}}}\right) \\ (\underline{\boldsymbol{x}}^{\text{node}}, \underline{\boldsymbol{x}}^{\text{edge}}) &\longmapsto g_k(\underline{\boldsymbol{x}}^{\text{node}}, \underline{\boldsymbol{x}}^{\text{edge}}) \\ &:= (\phi_k^{\text{edge}} \circ \phi_k^{\text{node}})(\underline{\boldsymbol{x}}^{\text{node}}, \underline{\boldsymbol{x}}^{\text{edge}}), \end{aligned} \tag{1.8}$$

$$\begin{aligned} &(k \in [\![1, N]\!],\ N > 2) \\ &(D_k^{\text{in, node}} \equiv N^{\text{node}} \times N_k^{\text{in, node}}) \\ &(D_k^{\text{in, edge}} \equiv N^{\text{edge}} \times N_k^{\text{in, edge}}) \\ &(D_k^{\text{out, node}} \equiv N^{\text{node}} \times N_k^{\text{out, node}}) \\ &(D_k^{\text{out, edge}} \equiv N^{\text{edge}} \times N_k^{\text{out, edge}}) \\ &(N_k^{\text{in, node}} \in \mathbb{N},\ N_k^{\text{out, node}} \in \mathbb{N},\ N_k^{\text{in, edge}} \in \mathbb{N},\ N_k^{\text{out, edge}} \in \mathbb{N}) \\ &(N^{\text{node}} \in \mathbb{N},\ N^{\text{edge}} \in \mathbb{N}) \end{aligned}$$

where N^{node} and N^{edge} are the number of nodes and edges in the graph, respectively, ϕ_k^{edge} is a building block responsible for updating edge features and ϕ_k^{node} is a building block responsible for updating node features. More precisely, ϕ_k^{edge} and ϕ_k^{node} are independent MLPs (as per Def. 1) or CNNs (as per Def. 2). The reader is referred to Battaglia et al. (2018) for details on the actions of ϕ_k^{edge} and ϕ_k^{node}. Each g_k is named a *message-passing* layer of the GNN. In order to exist, the composition (1.7) must obey the constraints

$$\begin{aligned} N_k^{\text{in, node}} &= N_{k-1}^{\text{out, node}} \qquad (k > 1) \\ N_k^{\text{in, edge}} &= N_{k-1}^{\text{out, edge}} \qquad (k > 1). \end{aligned}$$

The layer g_1 is called the *input* layer, g_N is named the *output* layer, and the layers g_k with $k \in [\![2, N-1]\!]$ are referred to as the *hidden* layers of the GNN. In the particular case where $N_k^{\text{in, node}} = N_k^{\text{out, node}} = N_k^{\text{in, edge}} = N_k^{\text{out, edge}} = N^{\text{feat}}\ \forall k \in [\![2, N-1]\!]$, one says that N^{feat} is the number of hidden features of the GNN.

GNNs have found successful application to problems dealing with unstructured data. Social networks for instance are naturally represented by graphs: Fan et al. (2019) built social recommender systems based on GNNs. In a scientific context, Deepmind's AlphaFold (Jumper et al., 2021) employed GNNs to accurately predict protein 3D structure resulting from an amino acid sequence. Belbute-Peres et al. (2020) accelerate predictions of fluid flows around airfoils by combining GNNs with a differentiable CFD solver (namely, SU2 by Economon et al. (2016)). Unstructured meshes can be naturally assimilated into graphs. This motivates the use of GNNs for developing data-driven numerical schemes on 1D irregular and 2D unstructured grids in Chapters 4 and 5.

The ML models defined so far can learn to perform a given task by adjusting the values of their weights and biases, a process called *training*. Learning methods can be organized in three main paradigms (Jordan and Mitchell, 2015): supervised, unsupervised and reinforcement learning. In supervised learning, the ML model learns by comparing its predictions to available reference data. In unsupervised learning, no reference data exist, and the model learns only by analyzing its inputs under assumptions about its structural properties (e.g., algebraic, combinatorial or probabilistic). Finally, in reinforcement learning, reference data is absent as well but the inputs contain indications as to whether an action of the ML model is correct or not. The development of data-driven numerical schemes in Chapters 2, 4, and 5 makes use of supervised learning because either the analytical solution dynamics is available for the cases studied or reference data can be obtained from a high-fidelity numerical solver. The next section explains the training procedure in the supervised learning paradigm.

1.1.4 The supervised learning paradigm

In supervised learning, the model parameters are optimized by minimizing a cost, which is a measure of the distance between the network prediction to some reference value. Such a cost is a function of the neural network trainable parameters and is called the *loss function* in the ML community. The training procedure modifies the values of the trainable parameters in order to minimize the mean loss function value across a dataset. The optimization

problem could be formulated as

$$\mathcal{W}^{\star} := \text{argmin}_{\mathcal{W}} \left[\frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}(\mathcal{W}, u_i, r_i) \right], \tag{1.9}$$

where $\mathcal{W}$ is the set of trainable parameters of the neural network. The symbol u_i represents an element of the dataset $\mathcal{U}$ which the model learns from, also known as *training set*. The loss function $\mathcal{L}$ compares the model's predictions from the given input u_i with a reference value r_i. The symbol r_i represents the corresponding sample from the dataset $\mathcal{R}$ of reference values. Finally, the variable N_s stands for the number of samples of the datasets $\mathcal{U}$ and $\mathcal{R}$. Usually, gradient-based optimization techniques are employed to find $\mathcal{W}^{\star}$. More precisely, reverse mode of algorithmic differentiation (see Sec. 1.2.2) [6] evaluates the gradient of the loss function with respect to the neural network parameters in a scheme known as the backpropagation algorithm. The name comes from the fact that the gradient computation starts from the output and propagates back to the inputs of the neural network model. Once this gradient is computed, an update rule is applied to the trainable parameters following the opposite direction of the gradient (similar to Eq. (1.2)). In its simplest form,

$$\mathcal{W} \longleftarrow \mathcal{W} - \eta \cdot \frac{\partial}{\partial \mathcal{W}} \left[\frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}(\mathcal{W}, u_i, r_i) \right], \tag{1.10}$$

where η is the gradient descent step size, also known as the *learning rate* of the training procedure. The process is repeated iteratively until specific stopping criteria are met. Given that training sets can reach sizes of the order of thousands (as MNIST) to millions (as ImageNet by Russakovsky et al. (2015) for generic categorical object recognition) of samples, applying the updating rule (1.10) is unfeasible in practice. An approximation of the gradient descent procedure takes place by computing gradients only for a randomly selected batch of samples at a time. Let $\mathcal{U}_b \subset \mathcal{U}$ be a batch of training samples of size N_b and $\mathcal{R}_b \subset \mathcal{R}$ the corresponding batch reference set. The trainable parameters are updated as

$$\mathcal{W} \longleftarrow \mathcal{W} - \eta \cdot \frac{\partial}{\partial \mathcal{W}} \left[\frac{1}{N_b} \sum_{j=1}^{N_b} \mathcal{L}(\mathcal{W}, u_j, r_j) \right]. \tag{1.11}$$

[6] It was introduced by Linnainmaa (1970) and first applied to neural networks training in 1974 by Werbos (1974) (nevertheless, this work was not cited by Rumelhart, Hinton, et al. (1986)).

When the batch size $N_b = 1$, meaning that the trainable parameters $\mathcal{W}$ are updated one sample at a time, the learning process is said to occur by *online training*, and mainly relates to systems where data is generated as a function of time (e.g., stock price prediction as shown in Hoi et al. (2021)). In this case, the method that uses the updating rule (1.11) to solve the optimization problem (1.9) is called *stochastic gradient descent* (Goodfellow et al., 2016). When the batch size $N_b > 1$, the method is called *mini-batch gradient descent*. In mini-batch gradient descent, reducing training time is made possible by distributing the computation of the gradients of the loss function among multiple processors, each of which handling one sample from the batch. Parallelization across the batch dimension is a common feature among modern AD tools in ML frameworks (e.g., TensorFlow and PyTorch, see Sec. 1.3).

Random batches $\mathcal{U}_b$ are sampled without replacement from the training set $\mathcal{U}$ until the set is exhausted, which is called an *epoch*. Typically, a training procedure involves running multiple epochs, and the training is terminated when a specific stopping criterion is met. A widely used stopping criterium compares the mean loss function value of the training set $\mathcal{U}$ with that of a different dataset, which is called the *validation set* (noted $\mathcal{V}$ herein). During training, the model is expected to improve performance metrics for both sets $\mathcal{U}$ and $\mathcal{V}$. However, after some point, it will keep improving them only for the training set $\mathcal{U}$. Generalization is an ML model's capacity to perform well on data it has never seen before (Chollet, 2018). The neural network is losing its generalization capacity from the point where its performance for the validation set $\mathcal{V}$ starts degrading. The training procedure must stop before that happens. Fig. 1.9 illustrates how the performance metrics of sets $\mathcal{U}$ and $\mathcal{V}$ may evolve along epochs. Finally, Alg. 1 summarizes the training procedure described throughout this section.

Accurate gradient computation is essential to obtain a reliable neural network model from the training procedure. An efficient technique to accurately evaluate gradients for computer programs is algorithmic differentiation, described in the next section.

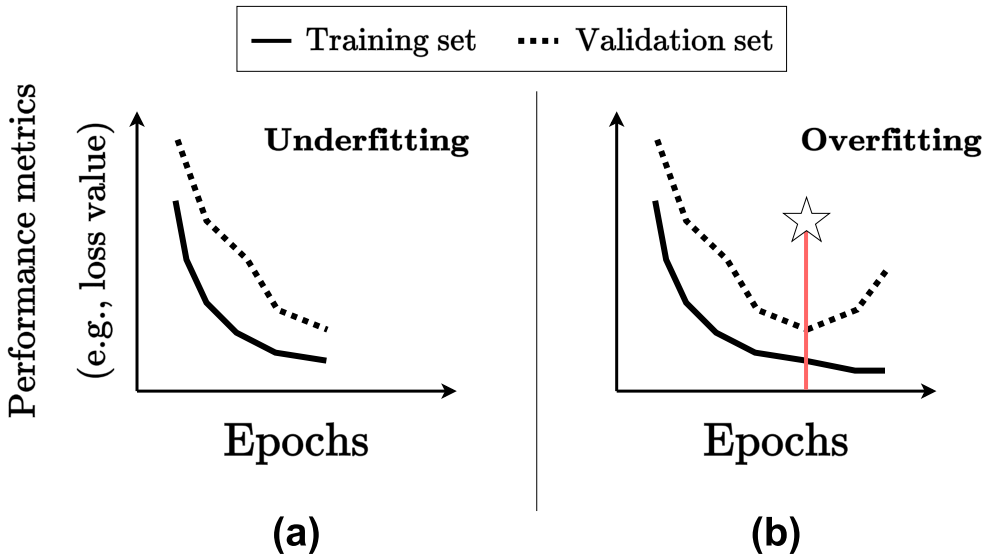


Figure 1.9: Performance metrics for training and validation sets at the end of each epoch during the training procedure. In (a), the neural network model can still potentially improve its predictions for both sets, in which case it is said to be underfitted. In (b), after a given point, it worsens validation set predictions for a number of epochs, in which case it is said to be overfitting on the training set. The optimal set of trainable parameters for the neural network can be found then just before the model starts overfitting, and is indicated by a star in the figure.

Algorithm 1 The training procedure in supervised learning

Data: training set $\mathcal{U}$, validation set $\mathcal{V}$, number of batches N_{batches}, batch size N_b
ML model: trainable parameters $\mathcal{W}$
Training: loss function $\mathcal{L}$, optimizer OPT, performance metrics P, stopping criteria S

▷ Training loop
for epoch $\in 1, \ldots, N_{\text{epochs}}$ **do**
    ▷ Train network for a single epoch

    for batch $\in 1, \ldots, N_{\text{batches}}$ **do**
        ▷ Train network for a single batch

        $L \leftarrow 0$
        for sample $\in$ batch **do**
            $L \leftarrow L + \frac{1}{N_b}\ \mathcal{L}(\mathcal{W}, u_{\text{sample}}, r_{\text{sample}})$
        end for ▷ Estimate loss of the entire batch

        $\mathcal{W} \leftarrow \text{OPT}(\mathcal{W},\ \partial L/\partial \mathcal{W})$ ▷ Update trainable parameters

    end for

    ▷ Evaluate the network on the entire validation set every epoch
    $L_{\text{val}} \leftarrow 0$
    for $v \in \mathcal{V}$ **do**
        $L_{\text{val}} \leftarrow L_{\text{val}} + P(\mathcal{W}, v)$
    end for

    ▷ Evaluate the network on the entire training set every epoch
    $L_{\text{train}} \leftarrow 0$
    for $u \in \mathcal{U}$ **do**
        $L_{\text{train}} \leftarrow L_{\text{train}} + P(\mathcal{W}, u)$
    end for

    ▷ Check stopping criteria
    if $S(L_{\text{val}}, L_{\text{train}}) ==$ True **then**
        save($\mathcal{W}$) ▷ Save current ML model state
        Break ▷ Interrupt training loop execution
    end if

end for

1.2 An introduction to Algorithmic Differentiation (AD)

Algorithmic Differentiation (AD) comprises a collection of methods for converting a program that computes numerical values of a function into a program that computes numerical values of the function's derivatives. These derivatives can include first-order derivatives, such as the function's gradient or the Jacobian of a set of constraints, higher-order derivatives like the Hessian matrix multiplied by a directional vector, or even nested derivatives (Bartholomew-Biggs et al., 2000). In AD, the derivatives are computed with approximately the same accuracy and efficiency as the original function evaluations. This is what distinguishes AD from two other ways of computing derivatives numerically. They are (Baydin et al., 2018)

- Numerical Differentiation (ND): it consists in evaluating the program at some sample points and using finite difference formulas to obtain approximations of its derivatives. For example, the method `dfdx` defined as

```
1  function dfdx(x0 :: Float64)
2    h = 1e-6
3    return ( f(x0 + h) - f(x0 - h) ) / 2h
4  end
```

  uses ND to approximate the derivative of the method `f` representing a scalar map $f : \mathbb{R} \mapsto \mathbb{R}$ at some real value x_0. More precisely, `dfdx` makes use of a second order finite difference approximation:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \mathcal{O}(h^2). \tag{1.12}$$

  The approximation (1.12) results from the truncation of the Taylor series expansions of $f(x_0 + h)$ and $f(x_0 - h)$ around the value x_0, and the error associated with it is called the *truncation error*. Asymptotically, as $h \to 0$, the truncation error diminishes. On the other hand, `f(x0 + h) - f(x0 - h)` becomes closer to machine precision, which introduces the so-called *round-off error*. It grows as $h \to 0$. The presence of both round-off and truncation errors is one of the main issues behind ND, since their asymptotic behaviors are opposite to each other. Fig. 1.10 illustrates it.

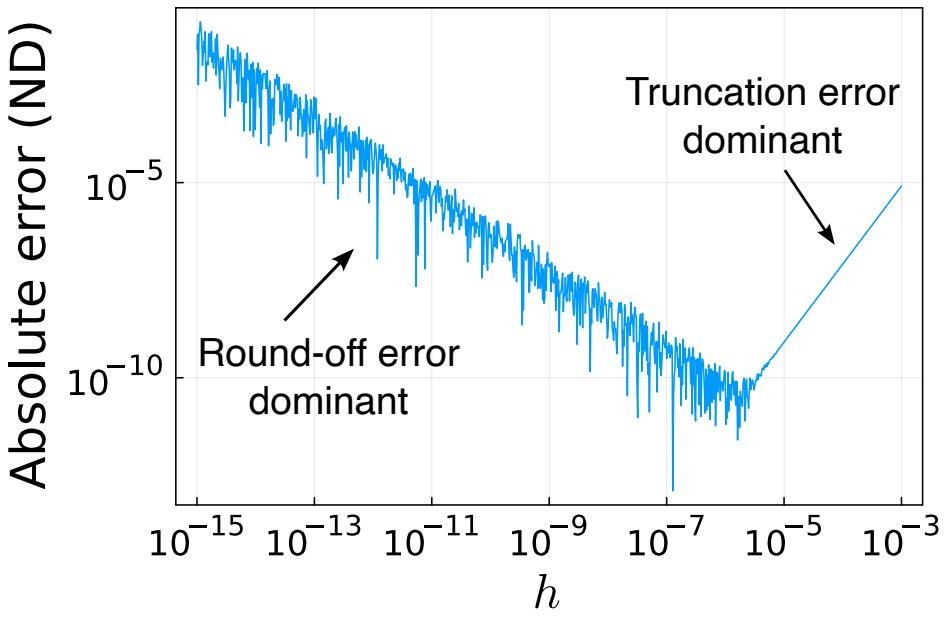


Figure 1.10: Absolute error of the ND-evaluated derivative of the function $f(x) := \exp(3x)$ at $x = 0.2$ using the approximation (1.12).

- Symbolic Differentiation (SD): it makes use of hard-coded differentiation rules for a finite set of operations and functions (e.g., $+$, $-$, $\times$, $\div$, log, sin, ...). This way, SD does not suffer from the numerical errors present in ND. To illustrate it, one can consider the real-valued function $f(x) := \exp(3x)$. In ND, truncation and round-off errors are present for employing truncated Taylor series expansions (see approximation (1.12) and Fig. 1.10). In SD, on the other hand, one can simply code the differentiation rule for exponentials and avoid these sources of error. In SD, the derivative of f is directly evaluated from its analytical expression $f'(x) = 3\exp(3x)$. Nevertheless, SD presents some limitations. The program to be differentiated must not contain control flow mechanisms (e.g., loops, conditionals). Moreover, careless application may lead to increasingly long symbolic expressions and performance issues, a phenomenon known as *expression swell*. To illustrate it, one can consider the function

$$g(x_1, \ldots, x_n) := \prod_{i=1}^{n} x_i, \tag{1.13}$$

with $n > 1$ ($\forall i \in [\![1, n]\!],\ x_i \in \mathbb{R}$) (Speelpenning, 1980). The gradient of g is expressed as:

$$\frac{\partial g}{\partial x_i} = \begin{bmatrix} x_2\, x_3\, \ldots\, x_n \\ x_1\, x_3\, \ldots\, x_n \\ \vdots \\ x_1\, x_2\, \ldots\, x_{n-1} \end{bmatrix}. \tag{1.14}$$

In a (naive) implementation of SD, one builds the symbolic expression for Eq. (1.14) without considering common subexpressions. In this case, its memory allocation grows exponentially with n.

In order to avoid the numerical errors present in ND and the expression swell issue in (naive) SD, AD in its basic form proposes to break the computer program down into elementary pieces to which SD will be applied and store intermediate numerical results whenever necessary. By interpreting the program as a composition of differentiable pieces, the chain rule of calculus can be applied to derive gradients for the entire chain of operations. Laue (2019) states that there is an equivalence of AD and SD when the implementation of SD stores intermediate results as well as pointers to identify common subexpressions.

In practical implementations of AD, track records of operations called *evaluation traces* (also known as Wengert (1964) lists) are built. The chain of operations constituting the computer program to be differentiated is referred to as the *forward primal* evaluation trace (or *forward pass*). Fig. 1.11 shows the forward primal evaluation trace for a sample program. To compute gradients, two distinct evaluation traces can be built: (i) the forward tangent (Sec. 1.2.1) and (ii) the adjoint (Sec. 1.2.2). Here, the variable naming convention of Griewank and Walther (2008) is used: v_{i-N} with $i \in [\![1, N]\!]$ are the N input variables, v_j with $j \geq 1$ are the so-called intermediate variables and y_k with $k \in [\![1, M]\!]$ are the M output variables of the evaluation trace.

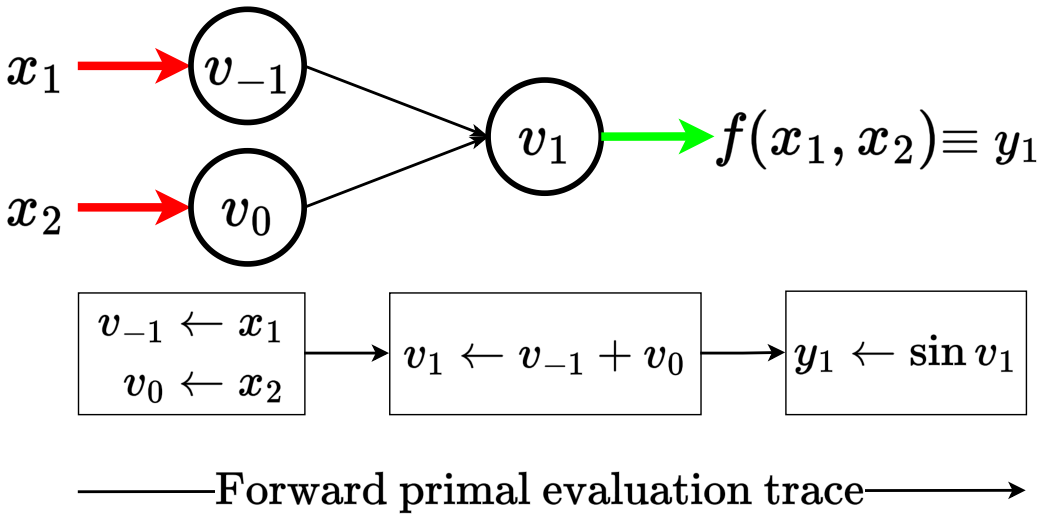


Figure 1.11: Computational graph and forward primal evaluation trace for a program evaluating $f(x_1, x_2) := \sin(x_1 + x_2)$, where $x_1, x_2 \in \mathbb{R}$.

1.2.1 Forward mode of AD

Forward mode AD is best suited for the differentiation of programs where the number of output variables is greater than the number of input variables, being mostly used for sensitivity analysis of physical systems. For example, T. W. Hughes et al. (2019) applied the forward mode of AD on Maxwell's equations to compute the sensitivity of a spatial near-field intensity distribution (defined for each cell of a 2D domain) to the dielectric constant of the medium (a scalar value, typically). This mode evaluates the derivatives

$$\dot{v}_{j,i} := \frac{\partial v_j}{\partial v_{i-N}} \qquad (j \geq 1,\ i \in [\![1, N]\!]) \tag{1.15}$$

$$\dot{y}_{k,i} := \frac{\partial y_k}{\partial v_{i-N}} \qquad (k \in [\![1, M]\!]) \tag{1.16}$$

of a given forward primal evaluation trace. In other words, the forward mode of AD computes the derivatives of all the intermediate and output variables with respect to each of the input variables. The resulting evaluation trace is called *forward tangent*. For the sake of simplicity, whenever the number of input variables $N = 1$, the second index of the derivatives $\dot{v}_{j,i}$ and $\dot{y}_{k,i}$ is dropped. Hence, they are represented simply as $\dot{v}_j$ and $\dot{y}_k$, respectively.

As an example, one can consider the sliding ladder problem: it consists on placing a ladder as a rigid and uniform rod in a 2D plane that lays over two perpendicular frictionless walls, as illustrated in the top part of Fig. 1.12. Kapranidis and Koo (2008) showed that as the ladder falls under the action of gravity, there will be a separation point at which it loses contact with the vertical wall (wall normal diminishes). The separation angle and the speed at which the rod hits the floor are respectively given by

$$\theta_{\text{sep}}(\alpha) = \arccos\left(\frac{2}{3}\cos\alpha\right) \tag{1.17}$$

$$s_{\text{f}}(\alpha) = \sqrt{g\ell\left(6\cos\alpha - \frac{4}{9}\cos^3\alpha\right)}, \tag{1.18}$$

where g, ℓ and α stand for local gravity, ladder length and initial ladder angle with the wall, respectively. For fixed values of g and ℓ, one may be interested in knowing the sensitivity of $\{\theta_{\text{sep}},\ s_{\text{f}}\}$ to the parameter α. Equivalently, it means evaluating how much a perturbation in the value of α affects the values of $\{\theta_{\text{sep}},\ s_{\text{f}}\}$. This boils down to computing the gradients $\partial\theta_{\text{sep}}/\partial\alpha$ and $\partial s_{\text{f}}/\partial\alpha$. They are computed at the end of the forward tangent evaluation trace shown at the bottom of Fig. 1.12.

Finally, the number of forward tangent passes equals the number N of input variables, i.e., it is independent of the number M of output variables (Baydin et al., 2018).

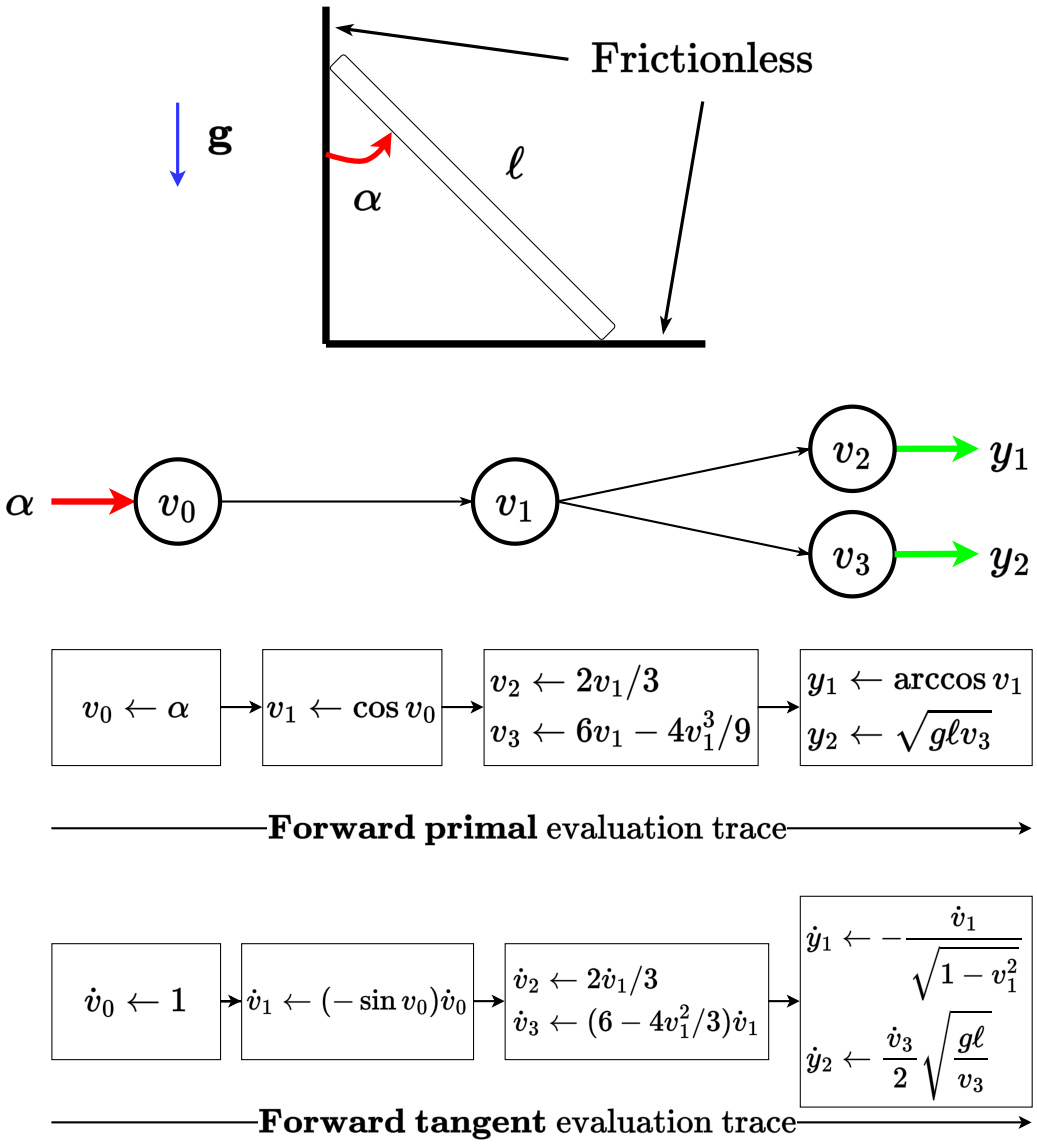


Figure 1.12: (Top) Schematic representation of the sliding ladder problem. (Middle and bottom) Computational graph, forward primal, and forward tangent evaluation traces for a program evaluating the angle y_1 (θ_{sep} in Eq. (1.17)) at which the falling ladder loses contact with the vertical wall and the speed y_2 (s_{f} in Eq. (1.18)) at which it hits the floor.

1.2.2 Reverse mode of AD

The Finnish mathematician and computer scientist Seppo Linnainmaa introduced the reverse mode of AD for efficiently computing derivatives of composite functions in his master's thesis in 1970 in Finnish (Linnainmaa, 1970). The English speaking world was largely unaware of the method until the publication of the English translation (Linnainmaa, 1976) [7]. In this paper, he applied reverse mode of AD to evaluate error propagation in floating point arithmetic. The reverse mode of AD is best suited for differentiating programs where the number of input variables is much higher than the number of output ones. This is the case for neural networks, where one is interested in minimizing a scalar-valued loss function by changes in the values of the trainable weights and biases. The number of trainable parameters reached the outstanding figure of 530 billion for Megatron-Turing NLG 530B, a large-scale generative language model (Smith et al., 2022). The first application of reverse mode of AD to neural networks training is credited to Werbos (1974). This mode evaluates the derivatives

$$\overline{v_{k,j}} := \frac{\partial\, y_k}{\partial\, v_j} \qquad (k \in [\![1, M]\!],\ j \geq 1 - N) \tag{1.19}$$

of a given forward primal evaluation trace. In other words, the reverse mode of AD computes the derivatives of each of the output variables with respect to all the intermediate and input variables. The resulting evaluation trace is called *adjoint* and each $\overline{v_{k,j}}$ is named a *reverse derivative* of this trace. For the sake of simplicity, whenever the number of output variables $M = 1$, the first index of the reverse derivatives $\overline{v_{k,j}}$ is dropped. Hence, they are represented simply as $\overline{v_k}$.

As an example, a dense layer (as stated inside Def. 1) is considered. It outputs a scalar value (a loss function value, typically) and is composed of the set $\{w_1, w_2, w_3\} \in \mathbb{R}^3$ of (trainable) weights and $b_1 \in \mathbb{R}$ as (trainable) bias. Given $\{x_1, x_2, x_3\} \in \mathbb{R}^3$ as input, the output of such a dense layer can be expressed as

$$\mathcal{L}(w_1, w_2, w_3, b_1) = \sigma(w_1\, x_1 + w_2\, x_2 + w_3\, x_3 + b_1), \tag{1.20}$$

where $\forall x \in \mathbb{R},\ \sigma(x) := \max(0, x)$. The activation function σ is called Rectified Linear Unit (ReLU). It was introduced by Fukushima (1969) in the

[7] The reverse mode of AD was rediscovered independently by Speelpenning (1980).

context of visual feature extraction using neural networks. The function σ is not differentiable everywhere. By definition, its derivative σ' is expressed as

$$\sigma'(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0, \end{cases} \tag{1.21}$$

and has no defined value at $x = 0$. However, when using a computer program to execute the adjoint evaluation trace, one needs to define a numerical value for the derivative σ' at $x = 0$. The ML libraries TensorFlow, PyTorch, and JAX (see Sec. 1.3.1-1.3.2) enforce $\sigma'(0) = 0$. Bertoin et al. (2021) reported that the choice of value for σ' at $x = 0$ influences the training results, and that the choice $\sigma'(0) = 0$ provides the best accuracy levels for neural networks trained via stochastic gradient descent (see Sec. 1.1.4). Fig. 1.13 shows the computation of all reverse derivatives related to the computational graph of the dense layer whose output is expressed in Eq. (1.20).

Finally, the number of adjoint passes equals the number M of output variables, i.e., it is independent of the number N of input variables (Baydin et al., 2018). This justifies its use in the machine learning field, where the loss function typically returns a scalar value and the number of trainable weights and biases is large.

The reverse mode of AD is more computationally efficient than the forward mode in the development of ML-based numerical schemes, which is the main focus of this thesis. In Sec. 1.3, the reader will find an overview of AD tools with reverse-mode support.

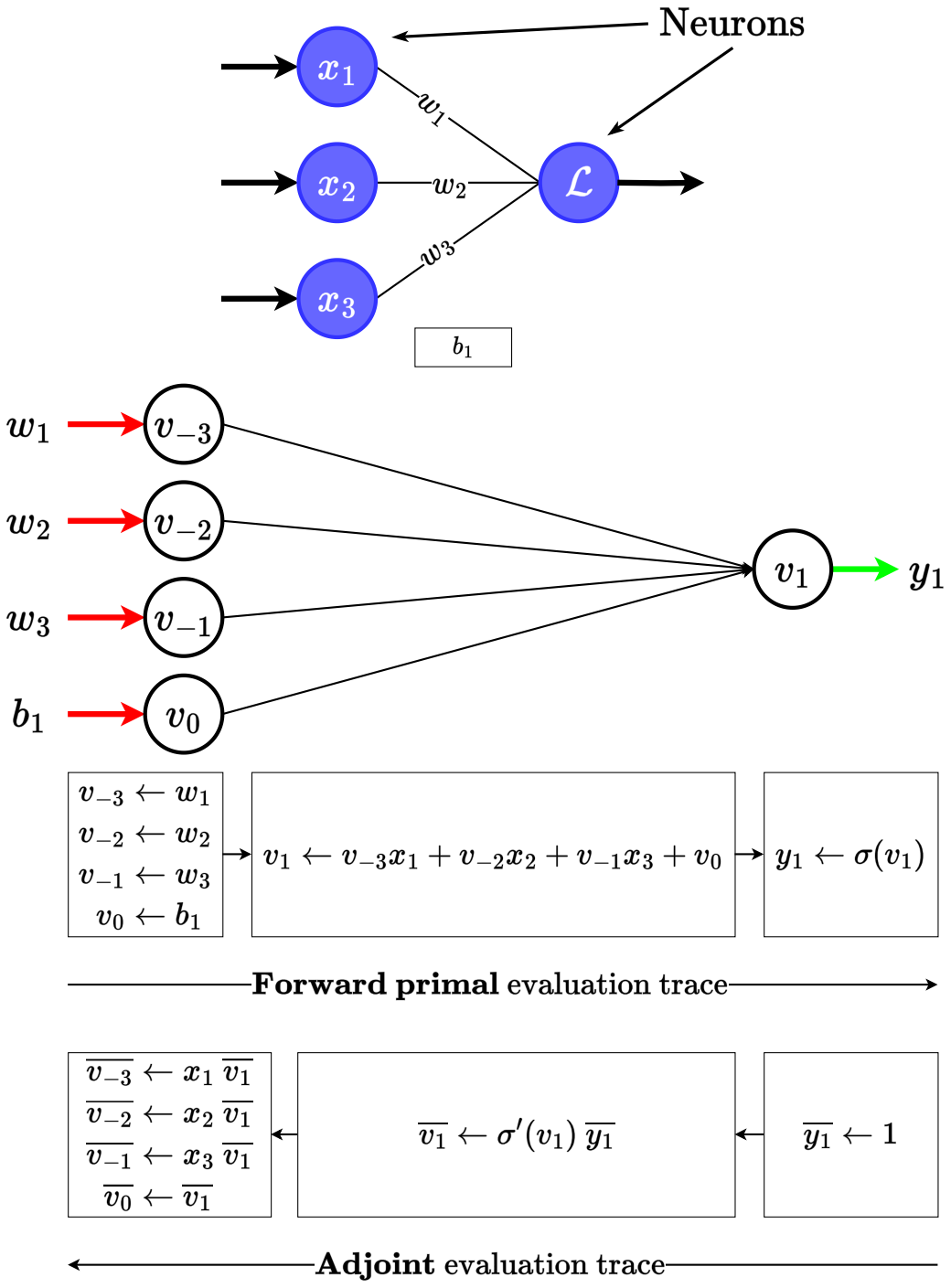


Figure 1.13: (Top) Schematic representation of a dense layer. (Middle and bottom) Computational graph, forward primal and adjoint evaluation traces for a program evaluating the (scalar) output of the dense layer.

1.3 An overview of AD frameworks

This section will familiarize the reader with existing differentiation tools that could be applied to obtain the adjoint of codes composing ML systems and PDE solvers. Among them, TensorFlow and PyTorch focus more on the former, whereas the others better handle generic control flow typical to the latter.

1.3.1 TensorFlow & PyTorch (Python/C++)

TensorFlow is an open source platform for machine learning originally developed by the Google Brain team to conduct research in AI within the organization, and then publicly released under the Apache 2 license (Metz, 2022). PyTorch is an open source machine learning framework developed by Meta AI. It was released in 2016 under Modified BSD license (*Release alpha-1 release · pytorch/pytorch* 2022). They are grouped for employing model subclassing and object-oriented paradigm in ML pipelines. Besides, users need to become familiar with their Domain Specific Languages (DSLs), under which classic layers in the ML community (e.g., dense and convolutional ones) are pre-defined, but custom layers are created at the cost of a rather verbose code dependent on the availability of elemental building blocks by the DSL.

1.3.2 JAX (Python)

JAX is a just-in-time compiler of Python/NumPy-similar code released by the Google Brain team (Frostig et al., 2018). Like TensorFlow and PyTorch, JAX introduces a DSL that is intended to be similar to plain Python/NumPy in order to ease more complex control flow differentiation, typical to coupled ML-PDE systems. However, it only handles pure functions, i.e., input arguments cannot be mutated in the scope of the method. This may turn the coding experience cumbersome.

1.3.3 Tapenade (Fortran/C)

Tapenade is a source-to-source AD engine released by Hascoët and Pascual (2013). Adjoint source code is accessible and can be modified for performance enhancement through parallelization. However, no pre-defined adjoints of

typical building blocks of neural networks like dense and convolutional layers are available. This might repel ML users, especially from industry. The development of the tool toward ML applications would ease the integration of existing legacy industrial CFD codes (mainly written in Fortran/C languages) to neural networks and eventually open a new path to high-performance machine learning for scientific applications. The reader can find in Appendix A an example of Tapenade-generated adjoint of a program.

1.3.4 Zygote (Julia)

Zygote is a source-to-source differentiation tool in Julia (Bezanson et al., 2017) released by Innes (2018a). It is the AD system of the language's main machine learning framework : Flux (Innes et al., 2018; Innes, 2018b). It handles generic control flow in Julia (apart from array mutation, as explained in Appendix B) and coupling PDE solvers coded in Julia to neural networks is as easy as writing any other Julia code. It is used to differentiate `dU_ttgc_reassignment` in Sec. B.3.

1.3.5 Enzyme (LLVM Intermediate Representation)

Enzyme is an AD tool for differentiating code in LLVM Intermediate Representation (IR) released by Moses and Churavy (2020). LLVM is a compiler and a toolkit for building compilers that transforms source code from a myriad of languages (e.g., Fortran, C++, Swift, Rust, Julia) into an IR that is efficiently converted to machine code (*The LLVM Compiler Infrastructure Project* 2022). It was designed for hardware portability, as it offers machine-independent primitive types (Yegulalp, 2020). Enzyme is used to differentiate `dU_ttgc_mutation` in Sec. B.3.

This chapter detailed the concepts required to understand the ML components of the data-driven numerical schemes developed in Chapters 2, 4, and 5. The next chapter is dedicated to reviewing concepts of numerical methods for PDEs. They precede the spectral analysis proposed in Chapter 3, which helps to define stability constraints for these data-driven numerical schemes.

Chapter 2

Introducing data-driven numerical schemes

The numerical simulation of fluid flows makes use of numerical schemes, which are algebraic relations between the space, time, and flow quantities represented in a computer. This chapter presents classical results related to the numerical approximation of hyperbolic systems of conservation laws essential to Computational Fluid Dynamics (CFD). Among them, a theorem by the Russian mathematician Sergei K. Godunov (1959) states that highly accurate linear numerical schemes cannot obey a property called monotonicity. This property ensures that the numerical scheme cannot generate new extrema other than those eventually present in the initial condition (Hirsch, 2007). Lack of monotonicity harm the reliability of the numerical simulation since it leads to unphysical values of quantities that must remain positive such as density, turbulent kinetic energy, or combustion mixture fraction. Spurious oscillations may also lead to unphysical values of certain quantities (T. K. Sengupta, Bhumkar, et al., 2012). These oscillations are specifically named Gibbs' phenomena when originating from shock waves or sharply discontinuous solutions (T. K. Sengupta, Ganerwal, et al., 2004). Godunov's theorem and the emergence of spurious oscillations (among them, Gibbs' phenomenon) in numerical simulations have significantly impacted the development of numerical schemes used in CFD. It should be highlighted that while state variables display discontinuity near shocks, the corresponding flux terms are continuous functions. Spectral accuracy of schemes is capable of curing Gibbs' phenomenon, as it is done by Essentially Non-Oscillatory

(ENO) and Weighted Essentially Non-Oscillatory (WENO) schemes (Zhang and Shu, 2016) applied to the state variables by one-sided differentiation. ENO and WENO schemes perform particular reconstructions of numerical fluxes which are nonlinear. The neural network models introduced in the previous chapter are nonlinear functions that can be employed to design numerical schemes. Sec. 2.3 presents a state-of-the-art method that approximates the derivatives of the numerical fluxes using a constrained Convolutional Neural Network (CNN) (see Def. 2). The constrained CNN is trained to accurately reproduce the convection of isentropic vortices and to evolve a double shear layer initial condition following the Euler equations (an inviscid version of the Navier-Stokes equations) on 2D cartesian grids.

2.1 Generalities about the numerical approximation of hyperbolic conservation laws

As they relate to transport phenomena, conservation laws of hyperbolic nature play an important role in Computational Fluid Dynamics (CFD) (Hirsch, 2007). In their differential form (no source terms):

$$\frac{\partial \boldsymbol{U}}{\partial t} + \mathbf{div}\ \underline{\boldsymbol{F}}(\boldsymbol{U}) = \mathbf{0}, \tag{2.1}$$

where $\boldsymbol{U}$ is the state variable, $\underline{\boldsymbol{F}}$ is a convective flux, and t is the temporal coordinate. The first part of this chapter concerns only one-dimensional equations such that

$$(\,\boldsymbol{U},\ \underline{\boldsymbol{F}}\,) \equiv (\,u(x,t),\ c\,u(x,t)\,) \tag{2.2}$$
$$(\,\boldsymbol{U},\ \underline{\boldsymbol{F}}\,) \equiv (\,u(x,t),\ u(x,t)^2/2\,), \tag{2.3}$$

where $c \in \mathbb{R}$ and $u : (x,t) \in \mathbb{R} \times [0,\infty) \mapsto u(x,t) \in \mathbb{R}$. The equivalences (2.2) and (2.3) respectively lead to the following Partial Differential Equations (PDEs):

$$u_t + c\,u_x = 0 \tag{2.4}$$
$$u_t + u\,u_x = 0, \tag{2.5}$$

where the subscripts $(\cdot)_t = \frac{\partial}{\partial t}$ and $(\cdot)_x = \frac{\partial}{\partial x}$ denote the partial derivatives with respect to the time and space, respectively. Eq. (2.4) is referred to as the

(linear) Convection equation. It models the transport of a quantity u by the constant convection velocity c. The Convection equation is also regarded as a wave propagation equation, where the wave amplitude u has a phase propagating speed equal to c. Eq. (2.5) is called the (inviscid) Burgers' equation, named after Burgers (1948). It appears in studies of gas dynamics (Shefter and Rosales, 1999) and of traffic flow (Lighthill and Whitham, 1955). Its viscous counterpart [1]

$$u_t + u\,u_x - \nu\,u_{xx} = 0 \qquad (\nu > 0), \tag{2.6}$$

can be viewed as a model for decaying free turbulence (Cole, 1951).

The problem that consists in finding $u := u(x,t)$ solutions to Eq. (2.4) or to Eq. (2.5) for a given initial condition $u^0 := u^0(x)$ is called an Initial Value Problem (IVP). Even when initial conditions are smooth, nonlinear hyperbolic conservation laws can develop jump discontinuities that propagate as shock waves (Dafermos, 2016). An important IVP studied in the theory of hyperbolic PDEs is the Riemann problem, which contains a discontinuity (Toro, 2009). It is formally defined as follows:

Definition 4 (1D Riemann problem)**.** Let $u_L \in \mathbb{R}$, $u_R \in \mathbb{R}$, and $x_0 \in \mathbb{R}$ such that $u_L < u_R$. The IVP with initial condition

$$u^0(x) = \begin{cases} u_L & \text{if } \; x \leq x_0 \;, \\ u_R & \text{if } \; x > x_0 \end{cases} \tag{2.7}$$

is called a *1D Riemann problem*. The variables u_L and u_R are referred to as the *left* and *right states* of the 1D Riemann problem, respectively.

One can also state a version of the problem for periodic domains as follows:

Definition 5 (1D periodic Riemann problem)**.** Let $u_{\min} \in \mathbb{R}$, $u_{\max} \in \mathbb{R}$, $x_0 \in \mathbb{R}$, $x_1 \in \mathbb{R}$, and $\ell \in \mathbb{R}$ such that $0 \leq u_{\min} < u_{\max}$ and $0 < x_0 < x_1 < \ell$. The IVP with initial condition

$$u^0(x) = \begin{cases} u_{\max} & \text{if } \; x \in [x_0, x_1] \;, \\ u_{\min} & \text{if } \; x \in [0, x_0) \cup (x_1, \ell] \end{cases} \tag{2.8}$$

is called a *1D periodic Riemann problem*.

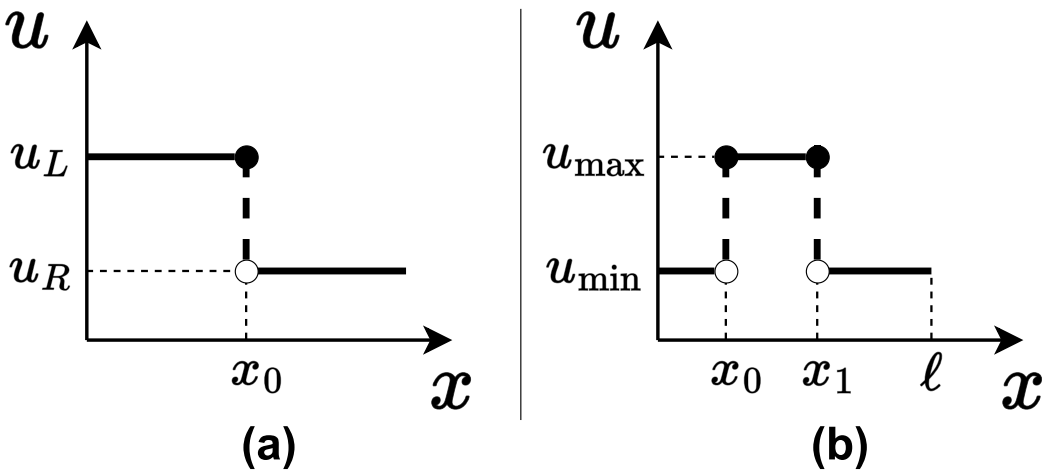


Figure 2.1: Representations of the (a) 1D Riemann problem (Def. 4) and (b) 1D periodic Riemann problem (Def. 5).

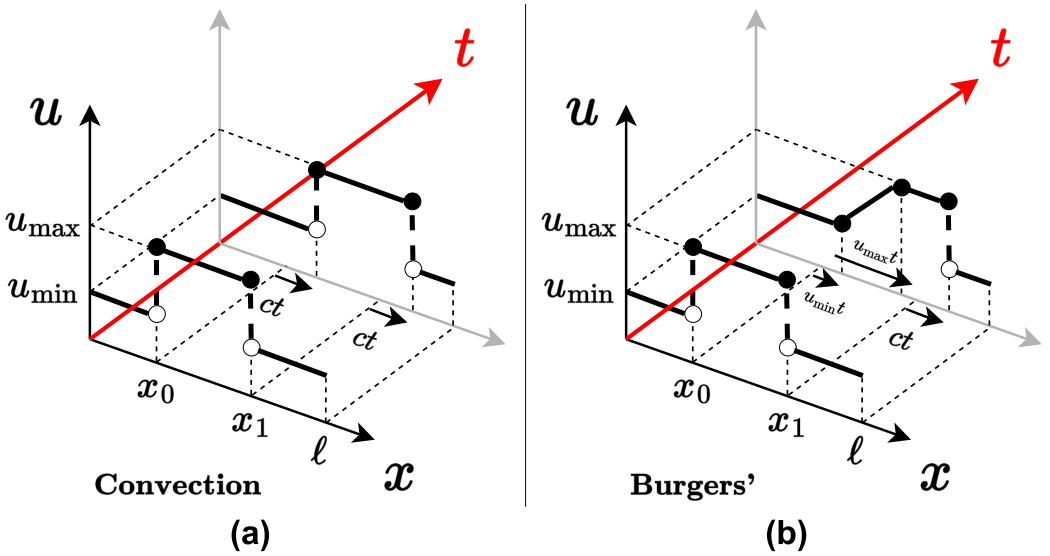


Figure 2.2: Representations of the analytical solutions for the 1D periodic Riemann problem (see Def. 5) following (a) the Convection equation (2.4) and (b) the Burgers' equation (2.5). In (b), $c := (u_{\max} + u_{\min}) / 2$.

Fig. 2.1 illustrates Defs. 4-5. Fig. 2.2 depicts analytical solutions for 1D periodic Riemann problems following the Convection equation (2.4) and the Burgers' equation (2.5). They are obtained using the method of characteristics described in Chapter I (Sec. 4.1) of Godlewski and Raviart (2021). On the one hand, the solution to any IVP following the Convection equation (2.4) simply corresponds to the propagation of the initial condition u^0 at the convection speed c, i.e.,

$$u(x,t) \equiv u^0(x-ct) \quad (t>0).$$

On the other hand, the solution to the 1D periodic Riemann problem following the Burgers' equation (2.5) is given by

$$u(x,t) \equiv \begin{cases} (x-x_0)/t, & x \in (x_0+u_{\min}t, x_0+u_{\max}t] \\ u_{\max}, & x \in [x_0+u_{\max}t, x_1+ct] \\ u_{\min}, & \text{elsewhere} \end{cases} \qquad (t \in (0,t_m)), \quad (2.9)$$

where $c := (u_{\max}+u_{\min})/2$ and $t_m := 2(x_1-x_0)/(u_{\max}-u_{\min})$. The 1D periodic Riemann problem following the Burgers' equation (2.5) admits solutions other than Eq. (2.9). Nevertheless, this is the only physically admissible one for respecting an entropy condition defined by Lax (1971). Eq. (2.9) is called an *entropy solution*. One can prove the uniqueness of the entropy solution to any IVP following the Burgers' equation (2.5) (see Thm. 5.4 in Chapter I of Godlewski and Raviart (2021)). However, it remains an open problem in the case of nonlinear systems of hyperbolic conservation laws.

The 1D Riemann problem is suitable for testing numerical methods that approximate Eqs. (2.4)-(2.5), thanks to the availability of analytical solutions. Its numerical simulation requires representing the state variable u using a finite number of values. The spatial and temporal simulation domains must also be represented using a finite number of points in space and time, respectively. Such a process is called *discretization*. A numerical scheme is a function that employs spatial and temporal discretizations to evolve the discrete values of the state variable. A formal definition follows:

Definition 6 (Numerical scheme)**.** Let I be the set of points representing a spatial discretization and $\{t^n\}_{n\in[\![1,N]\!]}$ be the set of N points representing

[1]The subscript $(\cdot)_{xx} = \frac{\partial^2}{\partial x^2}$ denote the second-order partial derivative with respect to space.

a temporal discretization. A numerical scheme for the PDE (or system of PDEs) E is a function

$$\begin{aligned} S_E : \left(\mathbb{R}^m, \mathbb{R}^{m|K_i|}\right) &\longrightarrow \mathbb{R}^m \\ (\boldsymbol{U}_i^n, \{\boldsymbol{U}_k^n\}_{k\in K_i}) &\longmapsto \boldsymbol{U}_i^{n+1}, \end{aligned} \tag{2.10}$$

where $m \in \mathbb{N}$ is the number of equations, $K_i \subset I$ is a subset of points of the spatial discretization and $|K_i|$ is the number of points in K_i. The variable $\boldsymbol{U}_i^n$ represents the discrete value of the state variable $\boldsymbol{U}$ at time level t^n and spatial point $i \in I$. Furthermore, S_E is called a *linear* scheme if and only if $\exists\, c_k \in \mathbb{R}$ such that

$$\boldsymbol{U}_i^{n+1} := S_E\left(\boldsymbol{U}_i^n, \{\boldsymbol{U}_k^n\}_{k\in K_i}\right) = \sum_{k\in K_i} c_k\, \boldsymbol{U}_k^n.$$

Provided that the numerical solution approximates the solution of the considered PDE (or system of PDEs), it is important to quantify how the approximation error levels evolve with the number of points used in the spatial and temporal discretizations. In general, numerical schemes that require fewer discretization points to achieve a given approximation error value are preferred. In the literature, it is common to compare numerical schemes by their truncation error behaviors using the concept of *order of accuracy* (Strikwerda, 2004). It can be defined for equally spaced cartesian grids as follows:

Definition 7 (Order of accuracy on equally spaced cartesian grids)**.** Let $\boldsymbol{U}$ be a smooth solution to Eq. (2.1). A numerical scheme S_E is said to be p-th order accurate in space and q-th order accurate in time if and only if

$$\left\|\{\boldsymbol{U}_{i,exact}^{n+1}\}_{i\in I} - \{S_E(\boldsymbol{U}_i^n, \{\boldsymbol{U}_k^n\}_{k\in K_i})\}_{i\in I}\right\|_2 = \mathcal{O}\left(\Delta x^{p+1} + \Delta t^{q+1}\right), \quad (\text{as } \Delta t \to 0,\ \Delta x \to 0) \tag{2.11}$$

where Δx and Δt denote the spatial and temporal discretization parameters, respectively. More precisely, Δx is the cartesian grid spacing and $\Delta t = t^{n+1} - t^n\, \forall n$ is a fixed time step size. Unless otherwise stated, p and q are the largest positive integers for which Eq. (2.11) holds. Finally, the left-hand side of Eq. (2.11) is called the *truncation error* of the numerical scheme.

Numerical schemes for hyperbolic conservation laws are also compared in terms of physical admissibility of the numerical solutions they provide. In gas

dynamics, for instance, it is essential that numerical values for density remain positive. In reactive flows, mass fraction values must stay in the interval $[0, 1]$. An important property which helps ensuring that these values remain in a physically admissible interval is that of *monotonicity*. This property states that no new extrema will be generated by the numerical scheme, aside from those present in the initial condition. Nevertheless, Godunov (1959) showed that one cannot formulate a monotone linear scheme for the 1D Convection equation (2.4) with spatial order of accuracy higher than one:

Theorem 1 (Godunov's order barrier)**.** Let $S^{linear}_{Convection}$ be a linear scheme for the 1D Convection equation (2.4) which is p-th order accurate in space. If $p \geq 2$, then $S^{linear}_{Convection}$ cannot be monotone.

Godunov's order barrier Thm. 1 has played an important role in the development of numerical schemes for the 1D Convection equation (2.4) and for the equations of fluid motion in CFD, by extension. The next section depicts numerical schemes of historical relevance.

2.2 Numerical analysis of schemes

A numerical scheme is said to be *consistent* if it tends to the considered PDE (or system of PDEs) as the discretization time and space steps tend to zero. Furthermore, it is said to be *stable* if the numerical solution remains bounded over time. This section is dedicated to show accuracy properties of consistent numerical schemes of historical importance. In what follows, only 1D equally spaced grids (with mesh spacing $\Delta x > 0$) and uniform time discretizations (with time step $\Delta t > 0$) are considered.

2.2.1 First Order Upwind (FOU) and Lax-Wendroff (LW)

The First Order Upwind (FOU) scheme proposed by Courant, Isaacson, et al. (1952) uses Taylor expansions of the perturbed states u_i^{n+1} and u_{i-1}^n in time and space, respectively. Firstly, one can notice that

$$u_i^{n+1} = u_i^n + \Delta t \, u_t^n|_{x=x_i} + \mathcal{O}(\Delta t^2) \tag{2.12}$$

leads to the following expression of the first order derivative in time $u_t^n|_{x=x_i}$:

$$u_t^n|_{x=x_i} = \frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{O}(\Delta t). \tag{2.13}$$

Similarly,

$$u_{i-1}^n = u_i^n - \Delta x\, u_x^n|_{x=x_i} + \mathcal{O}(\Delta x^2) \tag{2.14}$$

leads to the following expression of the first order derivative in space $u_x^n|_{x=x_i}$:

$$u_x^n|_{x=x_i} = \frac{u_i^n - u_{i-1}^n}{\Delta x} + \mathcal{O}(\Delta x). \tag{2.15}$$

Finally, one obtains the first order accurate in time and space FOU scheme for the 1D Convection equation (2.4) as

$$\begin{aligned} S^{\text{FOU}}_{\text{1D Convection}} : \left(\mathbb{R}, \mathbb{R}^2\right) &\longrightarrow \mathbb{R} \\ \left(u_i^n, \{u_k^n\}_{k\in\{i-1,i\}}\right) &\longmapsto u_i^{n+1} := u_i^n - N_c(u_i^n - u_{i-1}^n), \end{aligned} \tag{2.16}$$

where $N_c := c\,\Delta t/\Delta x$.

The FOU scheme can also be used to discretize the 1D inviscid Burgers' equation (2.5). It similarly applies Eq. (2.13) to express the time derivative u_t^n, and uses a Taylor expansion of the associated flux $\underline{\boldsymbol{F}} \equiv f(u) := u^2/2 \ \forall u \in \mathbb{R}$ to express its spatial derivative. The FOU scheme for the 1D Burgers' equation (2.5) reads

$$\begin{aligned} S^{\text{FOU}}_{\text{1D Burgers'}} : \left(\mathbb{R}, \mathbb{R}^2\right) &\longrightarrow \mathbb{R} \\ \left(u_i^n, \{u_k^n\}_{k\in\{i-1,i\}}\right) &\longmapsto u_i^{n+1} := u_i^n - \frac{\Delta t}{2\Delta x}\left[(u_i^n)^2 - \left(u_{i-1}^n\right)^2\right]. \end{aligned} \tag{2.17}$$

Lax and Wendroff (1960) employed second-order Taylor expansions to produce a scheme of better accuracy levels when compared to first-order methods like the FOU scheme. In the Lax-Wendroff (LW) scheme, the first-order time derivative $u_t^n|_{x=x_i}$ is expressed as

$$u_t^n|_{x=x_i} = \frac{u_i^{n+1} - u_i^n}{\Delta t} - \frac{\Delta t}{2} u_{tt}^n|_{x=x_i} + \mathcal{O}(\Delta t^2). \tag{2.18}$$

The time derivative u_{tt}^n in Eq. (2.18) is replaced by an equivalent spatial derivative using the 1D Convection equation (2.4), i.e.,

$$u_{tt}^n \equiv -c\,(u_x^n)_t \equiv -c\,(u_t^n)_x \equiv c^2 u_{xx}^n$$

implies that

$$u_t^n|_{x=x_i} = \frac{u_i^{n+1} - u_i^n}{\Delta t} - \frac{c^2 \Delta t}{2} u_{xx}^n|_{x=x_i} + \mathcal{O}(\Delta t^2). \tag{2.19}$$

Eq. (2.19) leads to the following semi-discrete form of the 1D Convection equation (2.4):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} - \frac{c^2 \Delta t}{2} u_{xx}^n|_{x=x_i} + c\, u_x^n|_{x=x_i} + \mathcal{O}(\Delta t^2) = 0. \tag{2.20}$$

The first and second order spatial derivatives u_x^n and u_{xx}^n in Eq. (2.20) are discretized using the following second-order Taylor expansions:

$$\begin{aligned} u_x^n|_{x=x_i} &= \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + \mathcal{O}(\Delta x^2), \\ u_{xx}^n|_{x=x_i} &= \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2). \end{aligned}$$

Finally, the LW scheme for the 1D Convection equation (2.4) is expressed as

$$\begin{aligned} S_{\text{1D Convection}}^{\text{LW}} : \left(\mathbb{R}, \mathbb{R}^3\right) &\longrightarrow \mathbb{R} \\ \left(u_i^n, \{u_k^n\}_{k\in\{i-1,i,i+1\}}\right) &\longmapsto u_i^{n+1} := u_i^n - \frac{N_c}{2}(u_{i+1}^n - u_{i-1}^n) \\ &\quad + \frac{N_c^2}{2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n). \end{aligned} \tag{2.21}$$

In order to derive the LW scheme for the 1D Burgers' equation (2.5), the time derivative u_{tt}^n in Eq. (2.18) is replaced by a spatial derivative of the associated flux $\underline{\boldsymbol{F}} \equiv f(u) := u^2/2 \;\; \forall u \in \mathbb{R}$, i.e., [2]

$$u_{tt}^n \equiv -(f_x^n)_t \equiv -(f_t^n)_x \equiv -\left(\frac{\partial f^n}{\partial u}\, u_t^n\right)_x \equiv \left(\frac{\partial f^n}{\partial u}\, f_x^n\right)_x$$

[2]For the sake of simplicity, the symbol f^n will be hereafter employed to denote $f(u^n)$.

implies that

$$u_t^n|_{x=x_i} = \frac{u_i^{n+1} - u_i^n}{\Delta t} - \frac{\Delta t}{2}\left(\frac{\partial f^n}{\partial u} f_x^n\right)_x\Bigg|_{x=x_i} + \mathcal{O}(\Delta t^2). \tag{2.22}$$

Eq. (2.22) leads to the semi-discrete form of the 1D Burgers' equation (2.5) below:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} - \frac{\Delta t}{2}\left(\frac{\partial f^n}{\partial u} f_x^n\right)_x\Bigg|_{x=x_i} + f_x^n|_{x=x_i} + \mathcal{O}(\Delta t^2) = 0. \tag{2.23}$$

The spatial derivative f_x^n in Eq. (2.23) is discretized using the following second-order Taylor expansion:

$$f_x^n|_{x=x_i} = \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{2.24}$$

The spatial derivative $\left(\frac{\partial f^n}{\partial u} f_x^n\right)_x$ in Eq. (2.23) can be discretized as follows (Godlewski and Raviart, 2021):

$$\left(\frac{\partial f^n}{\partial u} f_x^n\right)_x\Bigg|_{x=x_i} = \frac{1}{\Delta x^2}\left[\left(\frac{\partial f^n}{\partial u}\right)_{i+1/2}(f_{i+1}^n - f_i^n) - \left(\frac{\partial f^n}{\partial u}\right)_{i-1/2}(f_i^n - f_{i-1}^n)\right], \tag{2.25}$$

where $\left(\frac{\partial f^n}{\partial u}\right)_{i\pm1/2} := \frac{1}{2}\left[\left(\frac{\partial f^n}{\partial u}\right)_i + \left(\frac{\partial f^n}{\partial u}\right)_{i\pm1}\right]$ is the average value of the flux Jacobian in the neighborhood $\{i, i \pm 1\}$.

Finally, by applying Eqs. (2.24)-(2.25) into Eq. (2.23), one can express the LW scheme for the 1D Burgers' equation (2.5) as follows:

$$\begin{aligned} S^{\text{LW}}_{\text{1D Burgers'}}: \left(\mathbb{R}, \mathbb{R}^3\right) &\longrightarrow \mathbb{R} \\ \left(u_i^n, \{u_k^n\}_{k\in\{i-1,i,i+1\}}\right) &\longmapsto u_i^{n+1} := u_i^n - \frac{\Delta t}{4\Delta x}\left[\left(u_{i+1}^n\right)^2 - \left(u_{i-1}^n\right)^2\right] \\ + \frac{\Delta t^2}{8\Delta x^2}&\left\{(u_{i+1}^n + u_i^n)\left[\left(u_{i+1}^n\right)^2 - (u_i^n)^2\right] - (u_i^n + u_{i-1}^n)\left[(u_i^n)^2 - \left(u_{i-1}^n\right)^2\right]\right\}. \end{aligned} \tag{2.26}$$

The LW scheme enjoys less dissipation than the FOU scheme. However, Gibbs' phenomenon is observed when evolving initial conditions with steep

gradients, as shown in Fig. 2.4. It appears as a consequence of Godunov's order barrier Thm. 1. In order to overcome the limitations predicted by the theorem, numerical schemes for the Convection equation (2.4) must be nonlinear. Sec. 2.2.2-2.2.3 introduce nonlinear schemes of historical relevance.

2.2.2 Monotonic Upstream-centered Schemes for Conservation Laws (MUSCL)

Van Leer (1979) introduced the class of Monotonic Upstream-centered Schemes for Conservation Laws (MUSCL), which can achieve spatial second order of accuracy while preventing the emergence of Gibbs' phenomenon in the presence of shocks or sharp discontinuities. Given the semi-discrete form of the hyperbolic conservation law (2.1) below:

$$u_t^n|_{x=x_i} + L(u_i^n) = 0,$$

where

$$L(u_i^n) := \frac{1}{\Delta x}\left[f_{i+1/2}(u_i^n) - f_{i-1/2}(u_i^n)\right] \equiv \frac{1}{\Delta x}\left[f_{i+1/2}^n - f_{i-1/2}^n\right],$$

MUSCL seek suitable nonlinear approximations for the numerical fluxes $f_{i\pm1/2}^n$, defined at the boundaries of a cell surrounding the mesh point i (see Fig. 2.3). The computation of these numerical fluxes depend on values of the state variable extrapolated to the right and left sides of the cell's boundaries, and are denoted as $u_{i\pm1/2}^{R,n}$ and $u_{i\pm1/2}^{L,n}$, respectively. The extrapolations $u_{i\pm1/2}^{R,n}$ and $u_{i\pm1/2}^{L,n}$ are nonlinear functions of the state variable values in the neighborhood $\{i-1, i, i+1, i+2\}$. More precisely,

$$u_{i+1/2}^{R,n} := u_{i+1}^n - \phi(r_{i+1}^n)\left(\frac{u_{i+2}^n - u_{i+1}^n}{2}\right),$$
$$u_{i+1/2}^{L,n} := u_i^n + \phi(r_i^n)\left(\frac{u_{i+1}^n - u_i^n}{2}\right),$$

where $r_i^n := (u_i^n - u_{i-1}^n)/(u_{i+1}^n - u_i^n)$ is an indicator of the local smoothness of the numerical solution and ϕ is a function of this smoothness indicator that limits the slope of the piecewise linear approximation of the state variable inside the cell. The function ϕ is commonly referred to as *flux limiter*. Flux limiters help to prevent the emergence of Gibbs' phenomenon in the numerical solution.

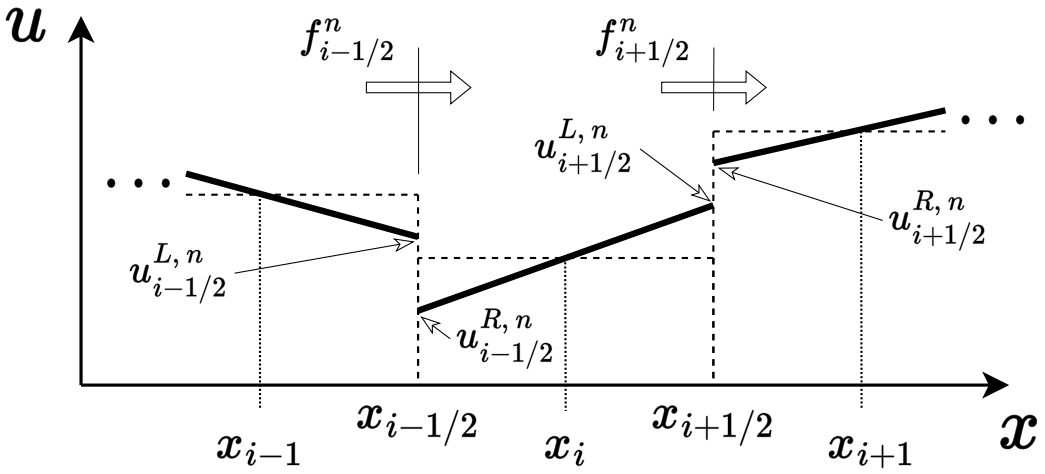


Figure 2.3: Representation of MUSCL numerical fluxes $f^n_{i\pm1/2}$ using the extrapolations $u^{R,n}_{i\pm1/2}$ and $u^{L,n}_{i\pm1/2}$ of the state variable at the boundaries of the cell $[x_{i-1/2}, x_{i+1/2}]$, where $x_{i\pm1/2} := (x_i + x_{i\pm1})/2$.

Fig. 2.4 shows numerical experiments performed with the second-order accurate in space MUSCL of Kurganov and Tadmor (2000), for which

$$f_{i\pm1/2}(u^n_i) := \frac{1}{2}\left\{\left[f\left(u^{R,n}_{i\pm1/2}\right) + f\left(u^{L,n}_{i\pm1/2}\right)\right] - a_{i\pm1/2}\left[u^{R,n}_{i\pm1/2} - u^{L,n}_{i\pm1/2}\right]\right\}, \tag{2.27}$$

where $a_{i\pm1/2}$ is the maximum absolute value of the eigenvalues of the flux Jacobian $\partial f^n/\partial u$. The generalized minmod flux limiter of Van Leer (1979) defined as

$$\phi_\theta(r^n_i) := \max\left[0,\ \min\left(\theta,\ \theta r^n_i,\ 0.5(1 + r^n_i)\right)\right] \qquad (\theta \in [1, 2]) \tag{2.28}$$

is used with $\theta = 2$ (value of the parameter for which the dynamics is least dissipative). To evolve the numerical solution over time, the three-stage, third-order Strong-Stability-Preserving (SSP) explicit Runge-Kutta time discretization method of Gottlieb et al. (2001) is used here. It is given by

$$\begin{aligned} u^{(1)}_i &= u^n_i + \Delta t\, L(u^n_i), \\ u^{(2)}_i &= \frac{3}{4}u^n_i + \frac{1}{4}u^{(1)}_i + \frac{1}{4}\Delta t\, L\left(u^{(1)}_i\right), \\ u^{n+1}_i &= \frac{1}{3}u^n_i + \frac{2}{3}u^{(2)}_i + \frac{2}{3}\Delta t\, L\left(u^{(2)}_i\right). \end{aligned} \tag{2.29}$$

2.2.3 Weighted Essentially Non-Oscillatory (WENO)

The class of Weighted Essentially Non-Oscillatory (WENO) schemes was designed to achieve arbitrarily high order of accuracy in smooth regions while also resolving singularities such as discontinuities and sharp gradients in an accurate and non-oscillatory fashion (Zhang and Shu, 2016). The fifth-order accurate in space WENO (WENO5) scheme of Jiang and Shu (1996) has been used in most applications and is considered here.

Similarly to MUSCL, WENO schemes look for suitable forms of the numerical flux $f^n_{i\pm1/2}$ at cells' boundaries. They reconstruct the numerical flux at a given boundary as a weighted sum of the flux values at neighboring mesh points. The set of weights is obtained as a function of indicators of the local smoothness of the solution, which can adapt the reconstructed flux to accurately capture steep gradients and shocks. The WENO5 method assumes the final form below:

$$u^n_t|_{x=x_i} + L(u^n_i) = 0,$$

where

$$L(u^n_i) := \frac{1}{\Delta x}\left[f_{i+1/2}(u^n_i) - f_{i-1/2}(u^n_i)\right] \equiv \frac{1}{\Delta x}\left[f^n_{i+1/2} - f^n_{i-1/2}\right],$$

with the flux decomposition

$$f^n_{i+1/2} \equiv \widehat{f}^{+,n}_{i+1/2} + \widehat{f}^{-,n}_{i+1/2},$$

and the reconstruction

$$\begin{aligned}
\widehat{f}^{+,n}_{i+1/2} := {} & w^+_0\left(\frac{2}{6}f^+_{i-2} - \frac{7}{6}f^+_{i-1} + \frac{11}{6}f^+_i\right) \\
& + w^+_1\left(-\frac{1}{6}f^+_{i-1} + \frac{5}{6}f^+_i + \frac{2}{6}f^+_{i+1}\right) \\
& + w^+_2\left(\frac{2}{6}f^+_i + \frac{5}{6}f^+_{i+1} - \frac{1}{6}f^+_{i+2}\right), \qquad (2.30) \\
\widehat{f}^{-,n}_{i+1/2} := {} & w^-_2\left(-\frac{1}{6}f^-_{i-1} + \frac{5}{6}f^-_i + \frac{2}{6}f^-_{i+1}\right) \\
& + w^-_1\left(\frac{2}{6}f^-_i + \frac{5}{6}f^-_{i+1} - \frac{1}{6}f^-_{i+2}\right) \\
& + w^-_0\left(\frac{11}{6}f^-_{i+1} - \frac{7}{6}f^-_{i+2} + \frac{2}{6}f^-_{i+3}\right), \qquad (2.31)
\end{aligned}$$

where the expressions of the weights $\left\{w_k^{+/-}\right\}_{k\in[\![0,2]\!]}$ and of the fluxes $\left\{f_{i+k}^{+/-}\right\}_{k\in[\![-2,3]\!]}$ can be found in R. Wang and Spiteri (2007). Fig. 2.4 illustrates the output of numerical simulations performed with the WENO5 spatial discretization method coupled with the third-order SSP explicit Runge-Kutta time-stepping method (2.29).

In order to verify the spatial order of accuracy (see Def. 7) of the implementations of the numerical schemes depicted in this section, one can generate a so-called *grid convergence error plot*. It consists in measuring the mean error of the numerical solution over one box turn of a smooth initial condition projected over equally spaced meshes of different resolutions. Fig. 2.5 illustrates grid convergence error plots for the FOU, LW, MUSCL of Kurganov and Tadmor (2000), and WENO5 schemes. It is worth noting that the WENO5 scheme exhibits third-order accurate behavior for a time step size $\Delta t = \mathcal{O}(\Delta x)$. Such a behavior arises because the time stepping method (2.29) is only third-order accurate, and the total error ϵ of the numerical solution then reads

$$\epsilon = \mathcal{O}\left(\Delta t^3 + \Delta x^5\right) \overset{\Delta t=\mathcal{O}(\Delta x)}{=} \mathcal{O}\left(\Delta x^3 + \Delta x^5\right) = \mathcal{O}\left(\Delta x^3\right).$$

The grid convergence error plot for the WENO5 scheme coupled with the time stepping method (2.29) must be produced using a time step $\Delta t = \mathcal{O}\left(\Delta x^{5/3}\right)$ to reveal the spatial order of accuracy of the implementation of the WENO5 method.

MUSCL and WENO schemes employ suitable nonlinear functions of the values of the state variable to provide accurate numerical solutions without spurious oscillations. However, they present applicability limitations. On the one hand, MUSCL uses nonlinear flux limiters formulated upon heuristic arguments, preventing a given limiter from performing well for all problems (Waterson and Deconinck, 2007). On the other hand, WENO schemes of spatial order of accuracy higher than two may use 3 to 10 times more CPU time than a second-order scheme (Shu, 2009). The recent field of Machine Learning (ML) introduced in the previous chapter relies on nonlinear functions that can be used to address these limitations. Nguyen-Fotiadis et al. (2022) employed neural networks as flux limiters in the discretization of the viscous Burgers' equation and showed that they outperform 11 classical

flux limiters over a range of mesh resolutions. Bezgin, Schmidt, et al. (2022) used neural networks to enhance a third-order WENO scheme. The resulting data-driven numerical method showed better accuracy levels than the more computationally demanding fifth-order WENO5 scheme at critical points. In this context, the next section investigates the ability of the state-of-the-art data-driven spatial discretization method proposed by Bar-Sinai et al. (2019) and Kochkov et al. (2021) to provide accurate and stable dynamics.

2.3 Data-driven spatial discretizations on 2D cartesian grids

The FOU and LW schemes described in Sec. 2.2.1 employ Taylor series expansions to approximate the spatial derivative in the Convection and Burgers' equations (2.4)-(2.5). The derivative at a given point in space is expressed as a weighted sum of the values of the state variable in a neighborhood of such a point. The set of weights that can approximate the spatial derivative at a given order of accuracy is not unique (Fornberg, 1988). Bar-Sinai et al. (2019) and Kochkov et al. (2021) trained neural networks to provide suitable values of the weights for a myriad of problems. More precisely, they aimed to recover features from high-resolution simulations on relatively coarse grids with an appropriate choice of values for the weights. High-resolution simulations of turbulent flows that capture the entire range of eddies' wavelengths are prohibitively expensive. In industrial applications, one performs coarse-grained Large-Eddy Simulations (LES), which account for the effects of unresolved small-scale features using a subgrid-scale model. However, subgrid-scale models are often formulated using ad hoc assumptions (Sagaut, 2006). The ultimate purpose of the works of Bar-Sinai et al. (2019) and Kochkov et al. (2021) is to replace subgrid-scale models with neural networks that provide suitable discretization of the Navier-Stokes equations. Bar-Sinai et al. (2019) demonstrated the approach on 1D problems governed by the viscous Burgers' equation (2.6), the Korteweg-de Vries equation, which is a mathematical model of waves on shallow water surfaces (Korteweg and De Vries, 1895), and the Kuramoto-Sivashinsky equation, which models diffusive-thermal instabilities in laminar flame fronts (Kuramoto, 1978; Sivashinsky, 1977). Kochkov et al. (2021) employed the technique of data-driven spatial discretizations to simulate 2D problems governed by the incompressible Navier-Stokes equa-

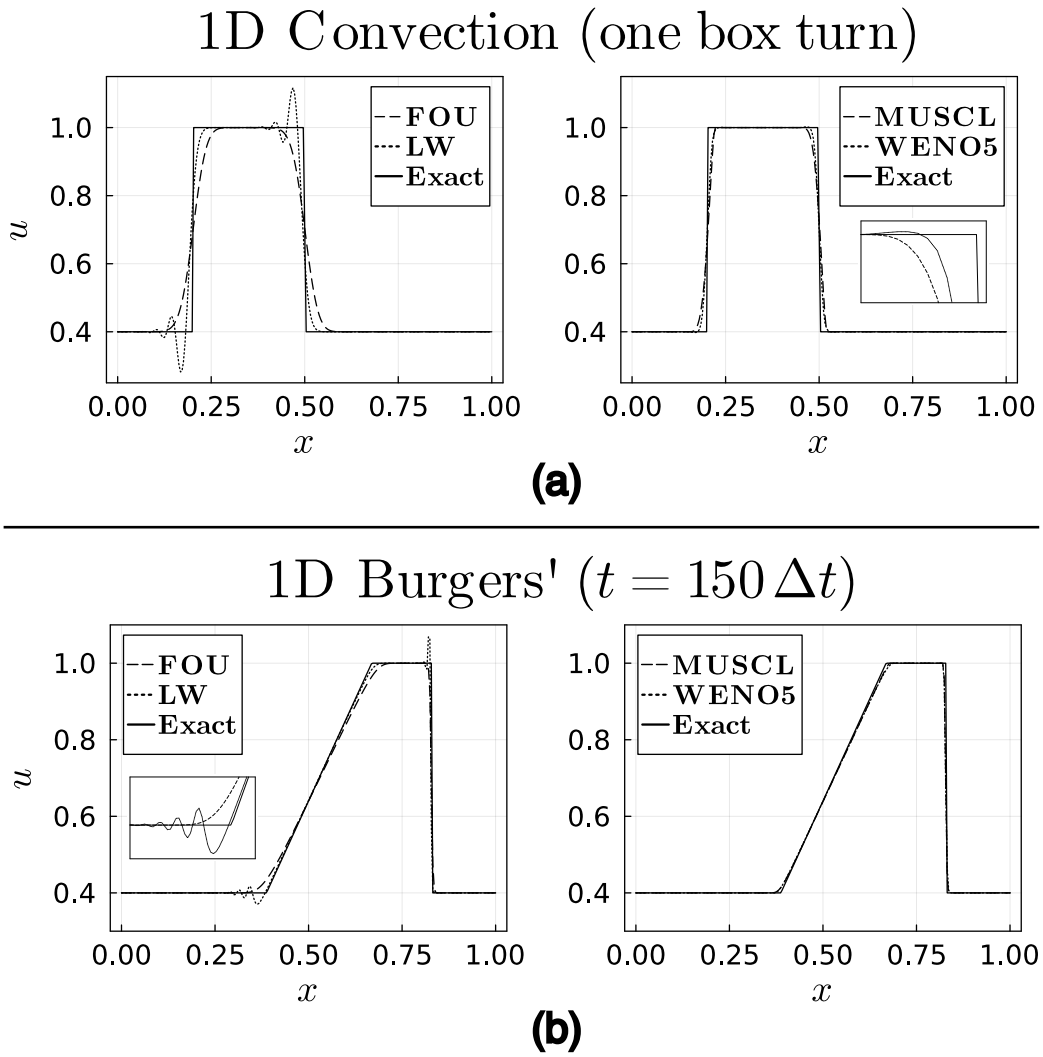


Figure 2.4: (a) One box turn of a 1D periodic Riemann problem (see Def. 5) following the Convection equation (2.4) for all schemes depicted in Sec. 2.2. (b) Numerical solution of a 1D periodic Riemann problem following the Burgers' equation (2.5) at time $t = 150\Delta t$ (where Δt is the simulation time step) for all schemes depicted in Sec. 2.2.

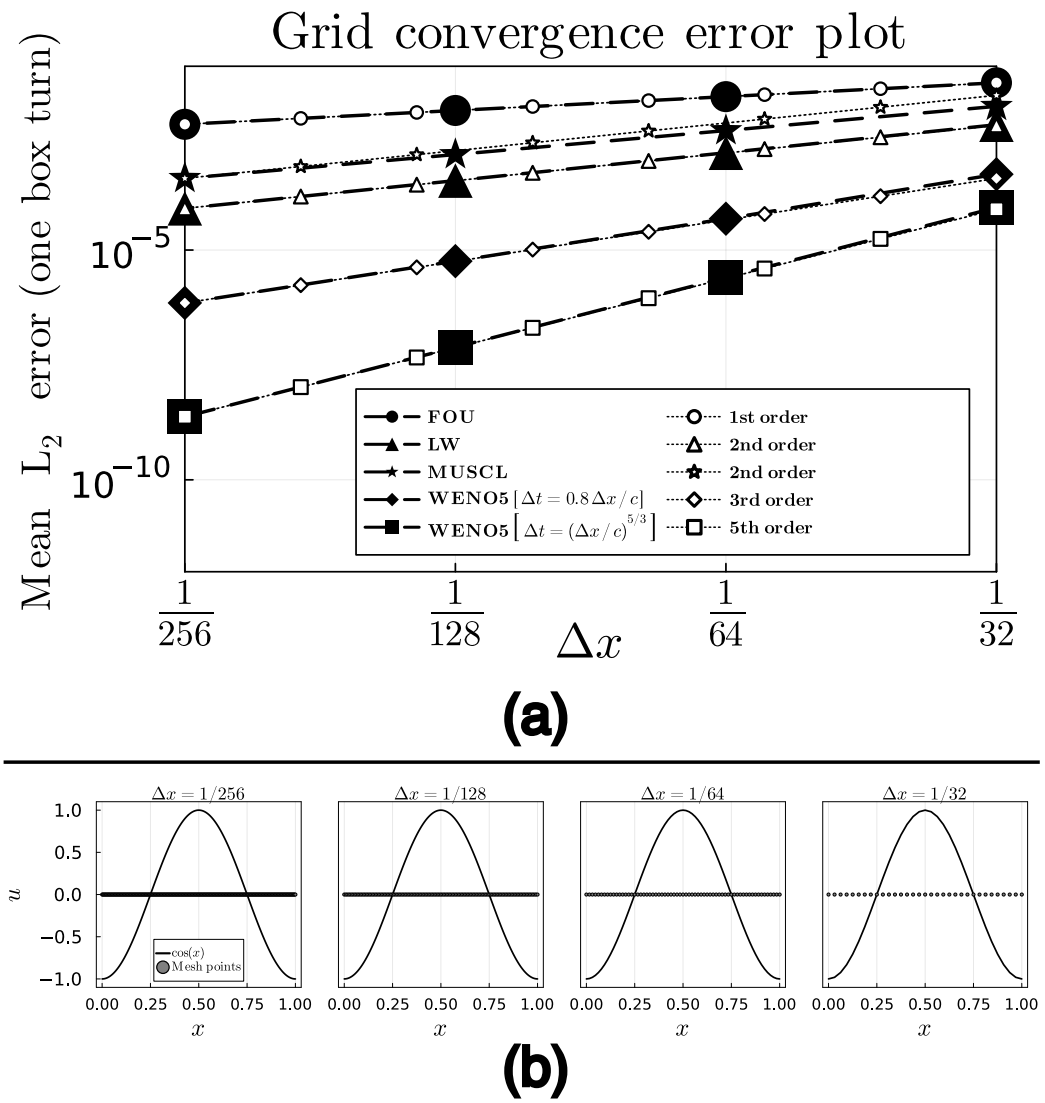


Figure 2.5: (a) Grid convergence error plots for all schemes depicted in Sec. 2.2. (b) Smooth initial condition $u^0 : x \in [0, 1] \mapsto \cos(x)$ used for grid convergence studies.

tions. This section studies periodic problems governed by the 2D Euler equations (an inviscid version of the compressible Navier-Stokes equations) on cartesian grids. They can be expressed using Eq. (2.1) with the following choice of state variable $\boldsymbol{U}$ and convective flux $\underline{\boldsymbol{F}}$:

$$\boldsymbol{U} := \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \quad \underline{\boldsymbol{F}}(\boldsymbol{U}) := \begin{bmatrix} \rho u & \rho v \\ \rho u^2 + p & \rho uv \\ \rho vu & \rho v^2 + p \\ u(E+p) & v(E+p) \end{bmatrix}, \tag{2.32}$$

where ρ, p, and E are the fluid's density, pressure, and energy fields, respectively; and u and v are the components of the flow velocity field along the x and y directions, respectively. A presentation of the considered neural network model that approximates the spatial derivatives of the flux $\underline{\boldsymbol{F}}$ follows.

2.3.1 Neural network model

Fig. 2.6 illustrates the neural network architecture. It consists of a Convolutional Neural Network (CNN) model (see Def. 2) followed by a non-trainable layer called the *Polynomial Accuracy Layer (PAL)*. The density (ρ), the components of the momentum along the x and y directions (ρu and ρv, respectively), and the energy (E) fields of a given initial condition are used as input features of the CNN. Subsequently, the CNN output features are split among eight channels. Each channel corresponds to a spatial derivative to be approximated (one per scalar flux, see Eq. (2.32)). The PAL projects the features of each channel on the space of weights that approximate spatial derivatives at a given enforced order of accuracy. Finally, the third-order SSP Runge-Kutta time-stepping method (2.29) is used to evolve the initial condition over time.

In order to explain how the PAL works, a 1D field $f := f(x)$ is considered, for the sake of simplicity. The PAL computes a set of weights that is used to approximate the ℓ-th derivative $f^{(\ell)}(x_0)$ of the field f for a given number N of points in the neighborhood of x_0. The Taylor series of f around the neighbor $x_n := x_0 + h_n$ ($h_n > 0$) can be expressed as follows:

$$f(x_0 + h_n) = f(x_0) + \ldots + f^{(N-1)}(x_0)\, h_n^{N-1}/(N-1)! + \mathcal{O}(h_n^N). \tag{2.33}$$

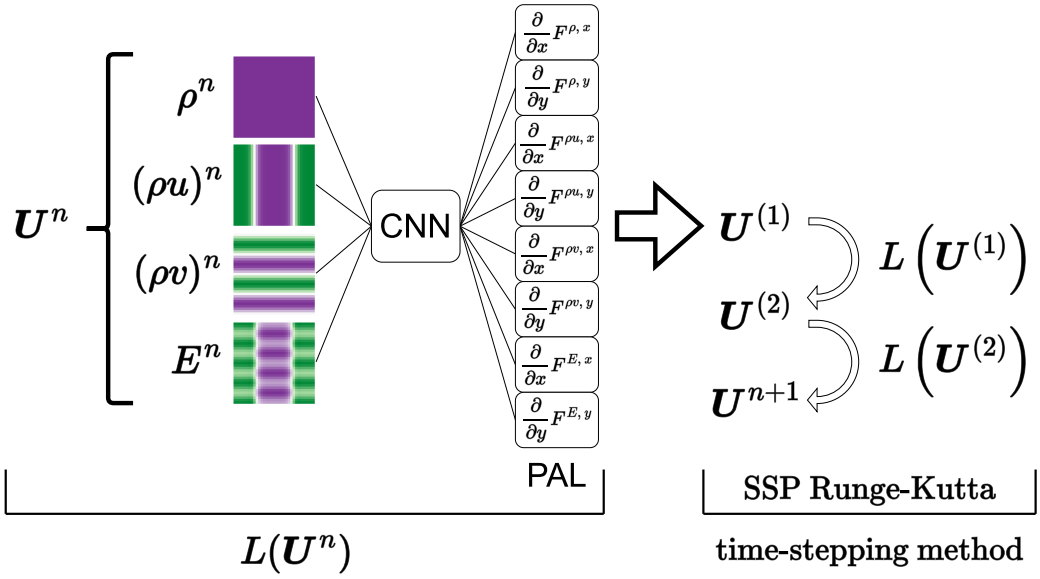


Figure 2.6: Schematic representation of the CNN-PAL neural network architecture for the data-driven spatial discretization of the Euler equations on 2D cartesian grids. The variables $F^{\star,x}$ and $F^{\star,y}$ correspond to the components of the flux of the field $\star$ along the x and y directions, respectively.

By approximating $f^{(\ell)}(x_0)$ as a sum of the neighboring values of f, weighted by a set of coefficients $\boldsymbol{\alpha} := (\alpha_i)_{i\in[\![1,N]\!]}$ to be determined, one obtains:

$$\begin{aligned} f^{(\ell)}(x_0) &\approx \sum_{n=1}^{N} \alpha_n\, f(x_0+h_n) \\ &\overset{\text{Eq. (2.33)}}{\approx} \sum_n \alpha_n\, f(x_0) + \ldots + \sum_n \alpha_n\, \underbrace{f^{(\ell)}(x_0)}\, h_n^\ell/\ell! \\ &\quad + \ldots + \sum_n \alpha_n\, f^{(N-1)}(x_0)\, h_n^{N-1}/(N-1)! \\ &\quad + \mathcal{O}(\alpha_1\, h_1^N). \end{aligned} \tag{2.34}$$

By comparing left- and right-hand sides of Eq. (2.34), the following set of constraints arise:

$$\begin{aligned} \sum_n \alpha_n\, h_n^\ell/\ell! &= 1, \\ \sum_n \alpha_n\, h_n^0/0! &= 0, \\ &\ldots \\ \sum_n \alpha_n\, h_n^{N-1}/(N-1)! &= 0. \end{aligned} \tag{2.35}$$

Alternatively, $\boldsymbol{\alpha}$ is solution to the linear system

$$\underline{\boldsymbol{A}_{\mathrm{f}}}\, \boldsymbol{\alpha} = \boldsymbol{b}_{\mathrm{f}}, \tag{2.36}$$

with

$$\underline{\boldsymbol{A}_{\mathrm{f}}} := \begin{bmatrix} h_1^0 & \ldots & h_N^0 \\ \vdots & \ddots & \vdots \\ h_1^{N-1} & \ldots & h_N^{N-1} \end{bmatrix}_{N\times N}, \quad \boldsymbol{b}_{\mathrm{f}} := \ell! \begin{bmatrix} \delta_{0,\ell} \\ \vdots \\ \delta_{N-1,\ell} \end{bmatrix}_{N\times 1},$$

where $\delta_{n-1,\ell}$ is the Kronecker's delta, $\forall\, n \in [\![1, N]\!]$. The matrix $\underline{\boldsymbol{A}_{\mathrm{f}}}$ is full rank and hence the coefficients that solve the linear system (2.36) are *unique*. The resulting approximation of $f^{(\ell)}$ is $\mathcal{O}(h_1^{N-\ell})$, since Eq. (2.35) ensures that

$$\mathcal{O}(\alpha_1\, h_1^N) \sim \mathcal{O}(h_1^{N-\ell}).$$

However, unique solutions prevent using the input of the PAL (i..e., the CNN output) to approximate spatial derivatives. To circumvent this, one

must enforce lower accuracy in the approximation of the ℓ-th derivative $f^{(\ell)}$, which leads to considering fewer terms in the Taylor expansion (2.33). Equivalently, for a desired order of accuracy $\mathcal{O}(h_1^m)$, with $m < N - \ell$, one has to solve the linear system

$$\underline{\boldsymbol{A}}\,\boldsymbol{\alpha} = \boldsymbol{b}, \tag{2.37}$$

with

$$\underline{\boldsymbol{A}} := \begin{bmatrix} h_1^0 & \dots & h_N^0 \\ \vdots & \ddots & \vdots \\ h_1^{m+\ell-1} & \dots & h_N^{m+\ell-1} \end{bmatrix}_{(m+\ell)\times N}, \; \boldsymbol{b} := \ell! \begin{bmatrix} \delta_{0,\ell} \\ \vdots \\ \delta_{m+\ell-1,\ell} \end{bmatrix}_{(m+\ell)\times 1}.$$

In such a case, the linear system is under-determined and hence admits an infinite number of solutions. Here, any solution $\boldsymbol{\alpha}$ can be written as the sum of an arbitrary fixed solution $\boldsymbol{\alpha}_{\mathrm{p}}$ and a vector $\boldsymbol{\alpha}_{\mathrm{null}}$ from the nullspace of the matrix $\underline{\boldsymbol{A}}$:

$$\underline{\boldsymbol{A}}\,\boldsymbol{\alpha} = \underline{\boldsymbol{A}}\,\boldsymbol{\alpha}_{\mathrm{p}} + \underline{\boldsymbol{A}}\,\boldsymbol{\alpha}_{\mathrm{null}} = \underline{\boldsymbol{A}}\,\boldsymbol{\alpha}_{\mathrm{p}} = \boldsymbol{b}.$$

In this work, $\boldsymbol{\alpha}_{\mathrm{p}}$ is the solution of the fully determined linear system (2.36) (which is solution of the under-determined system (2.37) as well) and $\boldsymbol{\alpha}_{\mathrm{null}}$ is the projection of the output of the CNN on the nullspace of $\underline{\boldsymbol{A}}$ (so that $\underline{\boldsymbol{A}}\,\boldsymbol{\alpha}_{\mathrm{null}} = 0$). This projection is performed as follows: let $\underline{\boldsymbol{A}} \equiv \underline{\boldsymbol{U}}\underline{\boldsymbol{\Sigma}}\underline{\boldsymbol{V}}^{\top}$ be the singular value decomposition of $\underline{\boldsymbol{A}}$, where $\underline{\boldsymbol{U}}$ and $\underline{\boldsymbol{V}}$ are square matrices whose columns are the so-called left- and right-singular vectors of $\underline{\boldsymbol{A}}$, respectively, and $\underline{\boldsymbol{\Sigma}}$ is a diagonal matrix of the so-called singular values of $\underline{\boldsymbol{A}}$. It is a well-known fact that the last $N - \mathrm{rank}(\underline{\boldsymbol{A}})$ columns of $\underline{\boldsymbol{V}}$ span the nullspace of $\underline{\boldsymbol{A}}$. The matrix formed by these columns is denoted $\underline{\boldsymbol{N}_{\boldsymbol{A}}}$. The projection of the output $\boldsymbol{x}$ of the CNN on the nullspace of $\underline{\boldsymbol{A}}$ can be expressed as $\boldsymbol{\alpha}_{\mathrm{null}} := \underline{\boldsymbol{N}_{\boldsymbol{A}}}\,\boldsymbol{x}$. Such a projection ensures that PALs return discretization coefficients $\boldsymbol{\alpha}$ that approximate spatial derivatives in cartesian grids with a constrained order of accuracy.

Only first-order derivatives of the fluxes will be approximated using a PAL (i.e., $\ell = 1$). For equally spaced grids and centered stencils, Eq. (2.37) then reduces to solving

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ -p & -p+1 & \dots & p \\ (-p)^2 & (-p+1)^2 & \dots & p^2 \\ \vdots & \vdots & \ddots & \vdots \\ (-p)^m & (-p+1)^m & \dots & p^m \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} 0 \\ 1/h \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.38}$$

where h is the cell size and $p \in \mathbb{N}$ is such that $N \equiv 2p + 1$ is the number of mesh points of the centered stencil.

A series of experiments with CNN-PAL neural network models that employ Eq. (2.38) was conducted. The parameters common to all the models used in these experiments are depicted in Tbl. 2.1 below. The next section is dedicated to show their outcomes.

Parameter	Value
Number of convolutional layers	4
Number of hidden features	16
Size of convolutional kernel/filter	(3, 3)
Size of input	(4, 33, 33)
Size of stencil (N)	5
Constrained order of accuracy (m)	2

Table 2.1: Parameters common to all CNN-PAL models used in experiments.

2.3.2 Numerical experiments

In this section, CNN-PAL neural network models are trained to predict the dynamics of a given initial condition. It is worth noting that a dimensionless version of the Euler equations is considered. It reads

$$\frac{\partial \boldsymbol{U}'}{\partial t'} + \mathbf{div}' \ \underline{\boldsymbol{F}}'(\boldsymbol{U}') = \mathbf{0}, \tag{2.39}$$

where

$$\boldsymbol{U}' := \begin{bmatrix} \rho' \\ \rho' u' \\ \rho' v' \\ E' \end{bmatrix}, \quad \underline{\boldsymbol{F}}'(\boldsymbol{U}') := \begin{bmatrix} \rho' u' & \rho v' \\ \rho' u'^2 + p' & \rho' u' v' \\ \rho' v' u' & \rho' v'^2 + p' \\ u'(E' + p') & v'(E' + p') \end{bmatrix}, \tag{2.40}$$

and ρ', p' and E' are the dimensionless fields of density, pressure and energy, respectively; u' and v' are the dimensionless components of the flow velocity field along the x and y directions; t' is the dimensionless temporal coordinate; $\mathbf{div}'$ is the dimensionless divergence operator. The dimensionless variables

relate to their original counterparts by the following expressions:

$$\begin{aligned} \rho' &:= \rho/\rho_{\text{charact}}, \\ u' &:= u/u_{\text{charact}}, \\ v' &:= v/u_{\text{charact}}, \\ E' &:= E/(\rho_{\text{charact}}\, u_{\text{charact}}^2), \\ p' &:= p/(\rho_{\text{charact}}\, u_{\text{charact}}^2), \\ t' &:= t/(l_{\text{charact}}/u_{\text{charact}}), \\ \mathbf{div}' &:= l_{\text{charact}}\, \mathbf{div}, \end{aligned} \tag{2.41}$$

where ρ_{charact}, u_{charact}, and l_{charact} are called the characteristic density, speed and length, respectively. The characteristic quantities are specific to each initial condition and simulation domain. The training procedure using the original variables has proven to be numerically unstable, unlike when using their dimensionless counterparts. This can be explained by the fact that the scales of the original variables are typically very dissimilar (with $E \sim \mathcal{O}(10^5)$ whereas $\rho \sim \mathcal{O}(1)$), which may affect the convergence of gradient-based optimization methods used in training (Wan, 2019). The numerical instabilities are related to the exploding gradients problem in ML (Bengio et al., 1994). During backpropagation (see Sec. 1.1.4), the successive multiplication of gradient values with relatively large norm may lead to a value which cannot be represented with finite precision floating point arithmetic. The ML literature encourages performing input normalization to ensure the stability of the training procedure (Shalev-Shwartz and Ben-David, 2014). Moreover, it helps to faster achieve a given accuracy level during training (Sola and Sevilla, 1997). This is because input normalization reduces the distance from the initial state of the trainable parameters of the neural network to the optimal state. The trainable parameters are typically initialized to random values within the $(-1, 1)$ interval. When the input normalization compresses the entire search space into a unitary hypercube, the distance that the backpropagation algorithm (see Sec. 1.1.4) needs to traverse in each iteration is reduced. However, input normalization is a non-physical transformation and the normalized input cannot be used by the PAL. Employing dimensionless variables instead of applying input normalization allows to (partially) address the aforementioned scaling issue in a physical way. Finally, one may notice that the algorithm that implements a numerical scheme for the Euler equations (2.1)-(2.32) can be employed without modifications for the discretization of the dimensionless Euler equations (2.39)-(2.40).

In a first experiment (n.1), a CNN-PAL model is trained to transport a single isentropic vortex by a uniform flow. This case is used to test a numerical scheme's capability to preserve vorticity in unsteady inviscid flows (*VI1 Vortex transport by uniform flow | HiOCFD5* 2022). The initial condition is set by superimposing vortex-related velocity fields over homogeneous fields. The components u^0 and v^0 of fluid's velocity along the x and y directions, as well as the medium temperature field T^0 are given by

$$u^0(x,y) := u_\infty \left[1 - \beta \frac{y - y_0}{R} \exp(-r(x,y)^2/2)\right] \tag{2.42}$$

$$v^0(x,y) := u_\infty \left[\beta \frac{x - x_0}{R} \exp(-r(x,y)^2/2)\right] \tag{2.43}$$

$$T^0(x,y) := T_\infty - \frac{u_\infty^2 \beta^2}{2\, C_p} \exp(-r(x,y)^2) \tag{2.44}$$

$$((x,y) \in [0,1] \times [0,1]),$$

where u_∞ and T_∞ are the initialized homogeneous values of velocity and temperature (with null component of velocity along the y direction), C_p is the fluid's heat capacity at constant pressure, β is the vortex strength, R is the vortex radius, and $r(x,y) := \sqrt{(x - x_0)^2 + (y - y_0)^2}/R$ is the dimensionless distance to the vortex core position (x_0, y_0). The fluid is considered to be an ideal gas. Given the adiabatic nature of the vortex transport, the initial density (ρ^0) and pressure (p^0) fields read

$$\rho^0(x,y) := \rho_\infty (T^0(x,y) \,/\, T_\infty)^{\frac{1}{1-\gamma_{\text{gas}}}} \tag{2.45}$$

$$p^0(x,y) := \rho^0(x,y)\, R_{\text{gas}}\, T^0(x,y), \tag{2.46}$$

where $R_{\text{gas}} = 287.15\ J \cdot kg^{-1} \cdot K^{-1}$ is the ideal gas constant and $\rho_\infty := p_\infty/(R_{\text{gas}}\, T_\infty)$ is the initialized homogeneous value of density, defined for a given initialized pressure p_∞. The value of R_{gas} relates to dry air at $15°C$ (or $288.15\ K$) and $101,325\ Pa$, for which the adiabatic index value $\gamma_{\text{gas}} = 1.4$. Finally, the characteristic quantities of the problem are given by

$$\rho_{\text{charact}} := \rho_\infty,\ u_{\text{charact}} := u_\infty \equiv M_\infty\, c_\infty,\ \text{and}\ \ l_{\text{charact}} := 1,$$

where M_∞ is the Mach number and $c_\infty := \sqrt{\gamma_{\text{gas}}\, R_{\text{gas}}\, T_\infty}$ is the speed of sound in the initialized homogeneous medium. Tbl. 2.2 illustrates the values of parameters used to set the vortex initial condition.

Experiment n.1: Training on a single isentropic vortex Parameter	Value
Vortex initial position (x_0, y_0)	(0.5, 0.5)
Mach number (M_∞)	0.5
Vortex strength (β)	0.2
Vortex radius (R)	0.1

Experiment n.2: Training on a set of isentropic vortices Parameter	Value
Vortex initial position (x_0, y_0)	(0.5, 0.5)
Mach number (M_∞)	$\mathcal{U}(0.1, 0.6)$
Vortex strength (β)	$\mathcal{U}(0.1, 0.8)$
Vortex radius (R)	$\mathcal{U}(0.05, 0.15)$

Experiment n.3: Training on a double shear layer Parameter	Value
Initialized x-velocity $(u^0(x, y))$	$0.5\,u_\infty\{\tanh[20(y-0.25)] - \tanh[20(y-0.75)]-0.2\}$
Initialized y-velocity $(v^0(x, y))$	$0.5\,u_\infty \sin(4\pi x)$
Mach number (M_∞)	0.5

Parameters common to all experiments	Value
Courant-Friedrichs-Lewy condition number (N_c)	0.8
Initialized pressure (p_∞)	10^5
Initialized temperature (T_∞)	300

Table 2.2: Parameter values used in all experiments of Sec. 2.3.2. $\mathcal{U}(a, b)$ stands for a continuous uniform distribution on the real interval (a, b). First introduced by Courant, Friedrichs, et al. (1928), the Courant-Friedrichs-Lewy condition number N_c limits the simulation time step size for a given grid spacing in order to correctly capture wave propagation. The international system of units is used to express all values.

The dynamics of an isentropic vortex initial condition under the Euler equations corresponds to its convection at the speed u_∞ along the x direction. In order to match the analytical vortex dynamics, the baseline CNN-PAL model depicted in Tbl. 2.1 is trained to minimize the distance between its predictions over time and the exact solution. For a given initial condition $\boldsymbol{U}^0$, the loss function reads (see Sec. 1.1.4)

$$\mathcal{L}(\boldsymbol{U}^0) := \frac{1}{N_{\text{step}}} \sum_{k=1}^{N_{\text{step}}} \left\| \boldsymbol{U}^k_{\text{CNN-PAL}} - \boldsymbol{U}^k_{\text{exact}} \right\|_2^2, \tag{2.47}$$

where $\boldsymbol{U}^k_{\text{CNN-PAL}}$ and $\boldsymbol{U}^k_{\text{exact}}$ are the predicted and analytical solution states at time step k, respectively. The variable N_{step} denotes the length of the forecasting window which the CNN-PAL model is trained to accurately predict. Since CNNs benefit from spatial translation invariance (see Sec. 1.1.3) and the vortex dynamics simply corresponds to the convection of the initial condition, one can expect that even a relatively small value of N_{step} leads to a trained CNN-PAL model that is capable to accurately perform long-range forecasts. Two independent CNN-PAL models are trained to predict $N_{\text{step}} = 4$ and $N_{\text{step}} = 16$ time steps. The outcomes of the training procedures including error levels after one box turn of the initial condition are summarized in Fig. 2.7. Additionally, the reader can find in Fig. 2.8 the spatial discretization coefficients (i.e., $\boldsymbol{\alpha}$ solution to Eq. (2.38)) predicted by the CNN-PAL model before and after the training procedure with $N_{\text{step}} = 16$. Both sets of coefficients approximate the spatial derivatives of the fluxes along the x direction at the enforced order of accuracy $m = 2$ (see Tbl. 2.1). However, the ones predicted by the CNN-PAL model after the training procedure lead to a less dissipative evolution of the vortex. Finally, Fig. 2.8 also shows that these coefficients better capture the symmetry of the problem than those of the untrained model.

A second experiment (n.2) consisted of training CNN-PAL models for the convection of a set of vortices (instead of a single one as in experiment n.1). They are trained to predict $N_{\text{step}} = 16$ time steps of the initial condition, which led to better long-range forecasting accuracy levels in experiment n.1. Fig. 2.9 shows the evolution of the mean value of the loss function (2.47) during the training of two independent CNN-PAL models using datasets with $N_s = 32$ and $N_s = 128$ samples (i.e., vortices) [3]. A batch size $N_b = 32$

[3]The reader is referred to Sec. 1.1.4 for an explanation of the training procedure terminology employed here.

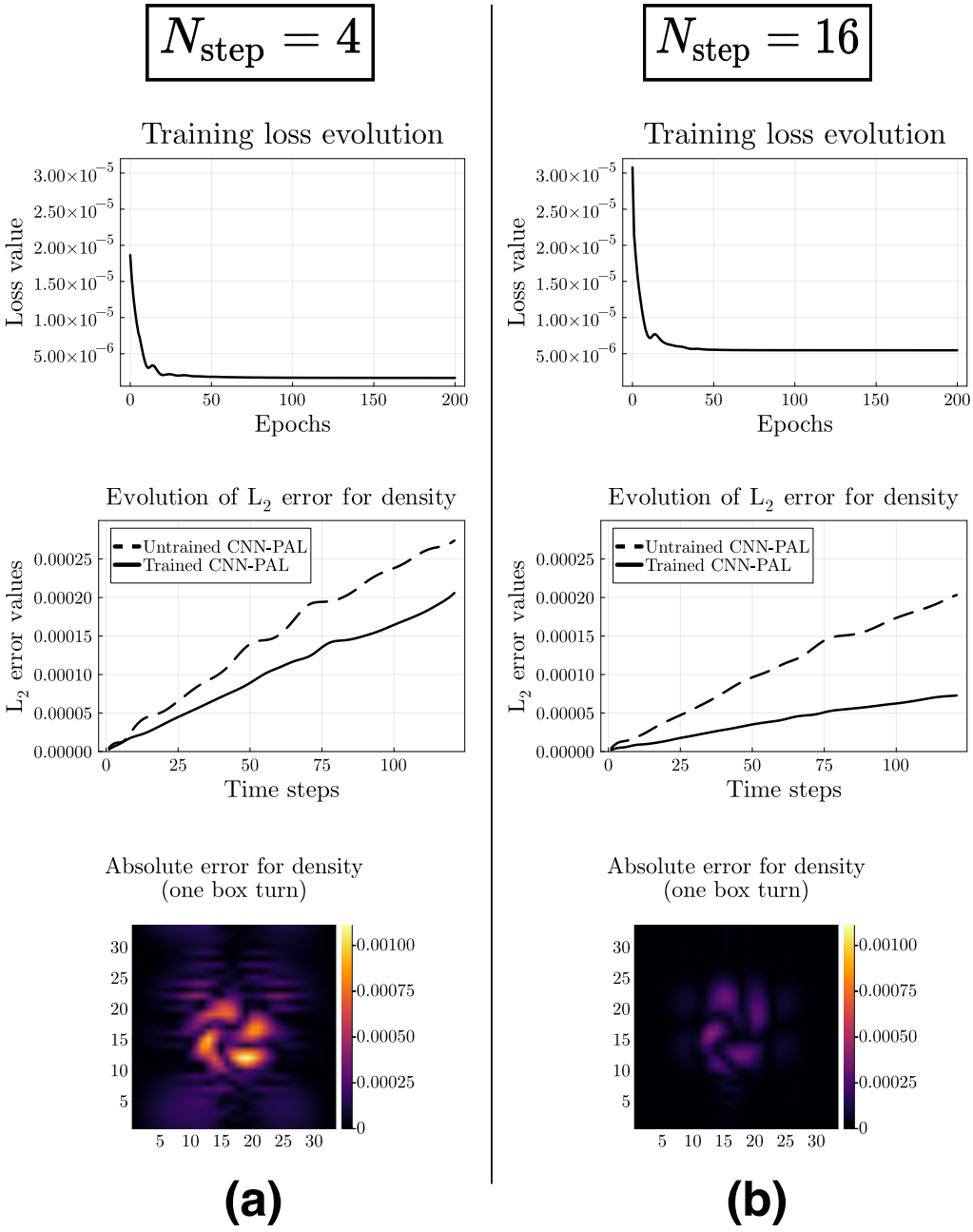


Figure 2.7: Outcomes of experiment n.1 performed with two independent CNN-PAL models trained to predict (a) $N_{\text{step}} = 4$ time steps; (b) $N_{\text{step}} = 16$ time steps of the evolution of the vortex depicted in Tbl. 2.2. (Top) Evolution of training loss values across epochs. (Center) Evolution of L_2 error of density field over time. (Bottom) Absolute error of density field predicted by the trained CNN-PAL models after one box turn of the vortex.

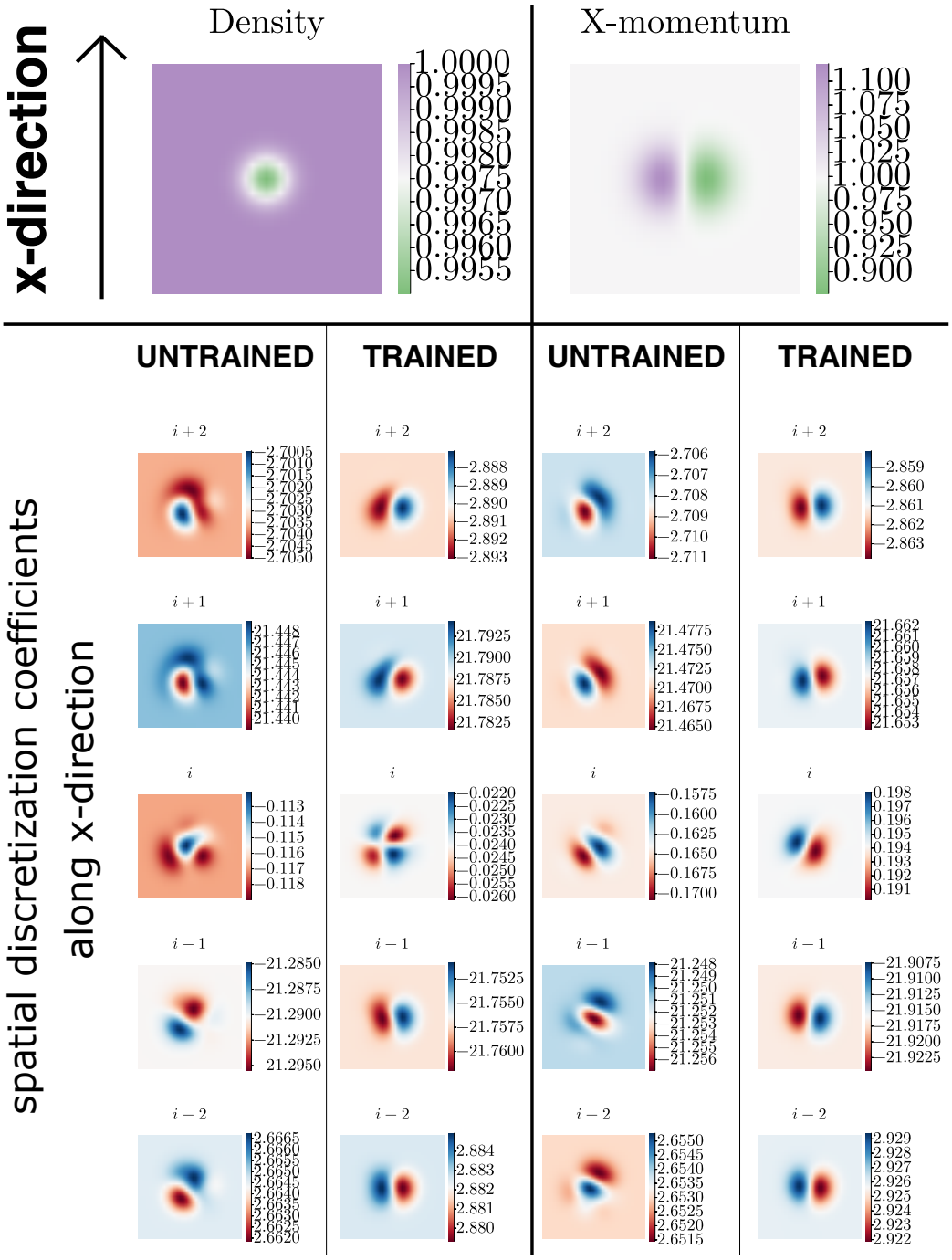


Figure 2.8: Discretization coefficients along the x-direction generated by a CNN-PAL model for the vortex convection analyzed in experiment n.1 (depicted in Tbl. 2.2). The "UNTRAINED" columns refer to the coefficients returned by the CNN-PAL model before training, while the "TRAINED" columns relate to the coefficients returned by the same CNN-PAL model after training. The coefficients are assigned to the neighborhood $\{(i-2,j),(i-1,j),(i,j),(i+1,j),(i+2,j)\}$ around each mesh point (i,j), composing a centered stencil of size $N = 5$ (see Tbl. 2.1).

was used. The training datasets are composed of vortices whose strength (β), radius (R) and mach number (M_∞) were drawn from uniform distributions, as depicted in Tbl. 2.2. To assess the generalization capabilities of the trained CNN-PAL models, one also tracks the evolution of the mean value of the loss function (2.47) for an indepedent set of vortices, not used to train the models. This set is referred to as the validation dataset, which contains 1024 vortices whose strength (β), radius (R) and mach number (M_∞) were drawn from uniform distributions in ranges of values identical to those of the training datasets. Fig. 2.9 shows the evolution of the mean value of the loss function (2.47) evaluated for the validation dataset. One can observe that the difference in the sizes of the training datasets plays an important role in generalization. The CNN-PAL model trained using $N_s = 128$ samples (see Fig. 2.9(b)) better predicts $N_{\text{step}} = 16$ time steps of the previously unseen vortices composing the validation dataset than the CNN-PAL model trained using $N_s = 32$ samples (see Fig. 2.9(a)).

In a last experiment (n.3), CNN-PAL models are trained for a different initial condition. Namely, the (periodic) double shear layer. In this case, the components u^0 and v^0 of fluid's velocity along the x and y directions are given by

$$u^0(x,y) := 0.5\,u_\infty\left\{\tanh\left[20(y-0.25)\right] - \tanh\left[20(y-0.75)\right] - 0.2\right\} \tag{2.48}$$

$$v^0(x,y) := 0.5\,u_\infty \sin\left(4\pi x\right) \tag{2.49}$$

$$((x,y) \in [0,1]\times[0,1]),$$

where u_∞ is some preset value of velocity. The fluid's density, pressure, and temperature are initialized to the homogeneous values of ρ_∞, p_∞, and T_∞, shown in Tbl. 2.2. The characteristic quantities of the problem are given by

$$\rho_{\text{charact}} := \rho_\infty,\ u_{\text{charact}} := u_\infty \equiv M_\infty\, c_\infty,\ \text{and}\ \ l_{\text{charact}} := 1,$$

where M_∞ is the Mach number and $c_\infty := \sqrt{\gamma_{\text{gas}}\, R_{\text{gas}}\, T_\infty}$ is the speed of sound in the initialized homogeneous medium.

Unlike the time evolution of the isentropic vortex initial conditions used in experiments n.1 and n.2, that of the double shear layer cannot be derived analytically. In this experiment, reference dynamics for the training of the CNN-PAL models is produced using a WENO5 scheme implementation for the Euler equations, combined with the SSP Runge-Kutta third-order time-stepping method (2.29). For systems of conservation laws such as the Euler

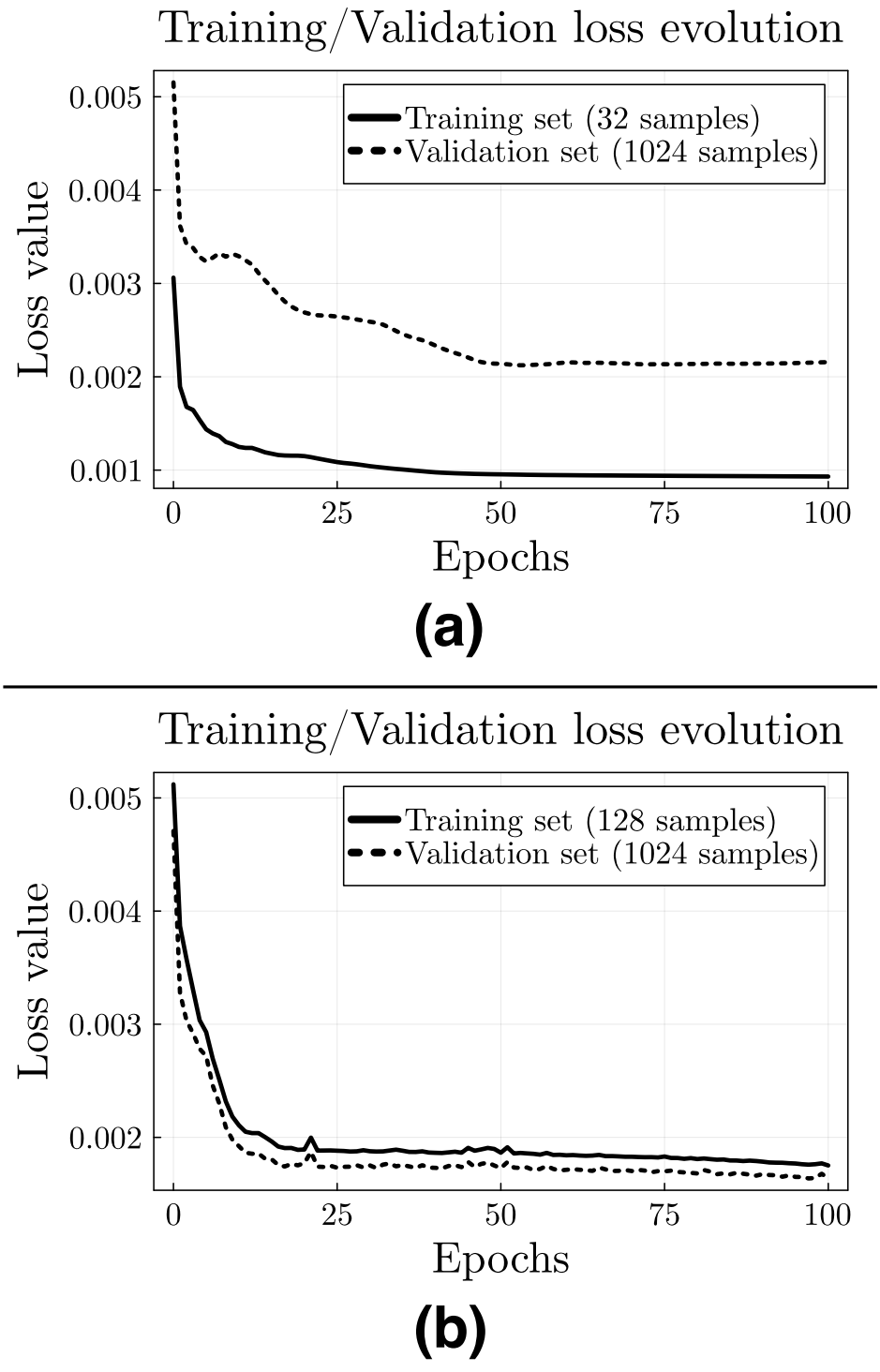


Figure 2.9: Evolution of training and validation loss values across epochs for independent CNN-PAL models trained to predict $N_{\text{step}} = 16$ time steps of the convection of vortices from training datasets composed of (a) $N_s = 32$ samples; (b) $N_s = 128$ samples. Results concern experiment n.2 (see Tbl. 2.2).

equations, the WENO flux reconstruction procedure can be performed in one of the following ways: component- or characteristic-wise. The component-wise reconstruction is the one described in Sec. 2.2.3. The characteristic-wise reconstruction consists in applying the operations of the component-wise reconstruction to the fluxes projected on a so-called *characteristic space*. The projection is performed by the averaged flux Jacobian of the system, which is recomputed at each time step. This makes characteristic-wise WENO schemes much more computationally demanding than their component-wise counterparts. Nevertheless, they produce better nonoscillatory results for spatial orders of accuracy higher than 3 (Shu, 2009). For this reason, a characteristic-wise WENO5 scheme is used here to generate reference data for training. The reader is referred to the Algorithm 4.8 of Shu (1999) for details on the characteristic-wise flux reconstruction procedure. Fig. 2.10 shows grid convergence error plots for the characteristic-wise WENO5 scheme implementation considering the (smooth) isentropic vortex initial condition from experiment n.1.

For a given double shear layer initial condition $\boldsymbol{U}^0$, the loss function reads

$$\mathcal{L}(\boldsymbol{U}^0) := \frac{1}{N_{\text{step}}} \sum_{k=1}^{N_{\text{step}}} \left\| \boldsymbol{U}^k_{\text{CNN-PAL}} - \boldsymbol{U}^k_{\text{WENO5}} \right\|_2^2, \tag{2.50}$$

where $\boldsymbol{U}^k_{\text{CNN-PAL}}$ and $\boldsymbol{U}^k_{\text{WENO5}}$ are the predicted and reference solution states at time step k, respectively. Two independent CNN-PAL models are trained to predict $N_{\text{step}} = 16$ and $N_{\text{step}} = 32$ time steps. The outcomes of the training procedures are summarized in Fig. 2.11. Firstly, one can notice that the plots of the training loss values contain sudden peaks (see the top part of Fig. 2.11). They are due to a reset of the optimizer every time the training procedure is relaunched. Unique training procedures could not be set due to time limit constraints of the cluster where these computations were performed. Despite conducting training procedures with many more epochs than those from experiments n.1 and n.2, one cannot observe improvement in the prediction accuracy through training (see the central part of Fig. 2.11). This happens because the errors of the different fields composing $\boldsymbol{U}$ are placed at different orders of magnitude. The training procedure that employs the loss function (2.50) tends to improve the accuracy of the predictions of the fields with the largest error levels, to the detriment of the others (Sola and Sevilla, 1997). To circumvent this, the loss function is modified to concern

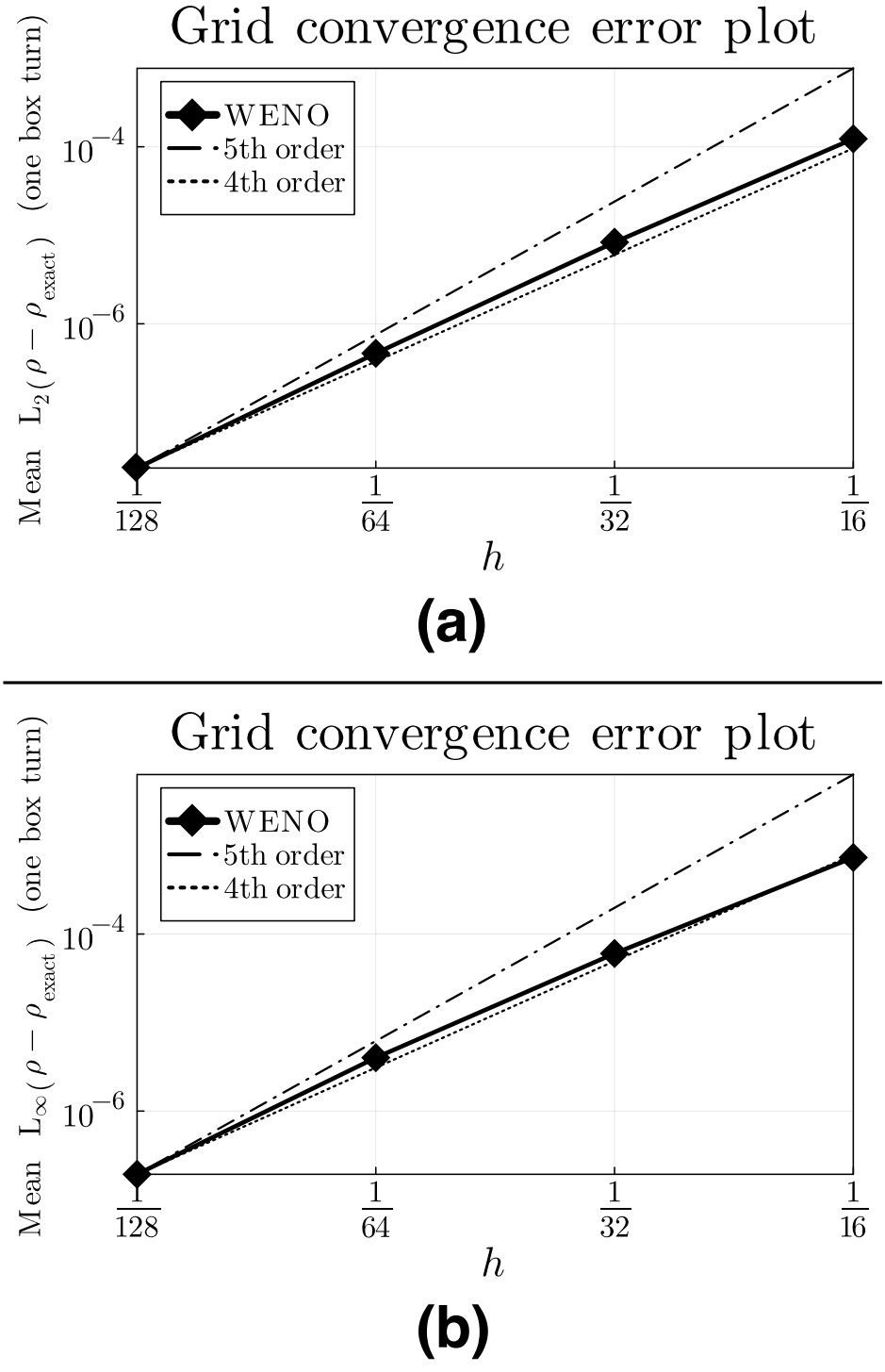


Figure 2.10: Grid convergence error plots for the WENO5 scheme implementation for the Euler equations considering either (a) mean L_2 error or (b) mean L_∞ error of density field over one box turn of the baseline vortex depicted in Tbl. 2.2. The variable h stands for the grid cell size along a single direction.

only the density field, which is the one with the lowest error levels:

$$\mathcal{L}(\boldsymbol{U}^0) := \frac{1}{N_{\text{step}}} \sum_{k=1}^{N_{\text{step}}} \left\| \rho^k_{\text{CNN-PAL}} - \rho^k_{\text{WENO5}} \right\|_2^2 . \tag{2.51}$$

One can expect that training CNN-PAL models using the loss function (2.51) will reduce prediction error levels not only for density, but for all fields that compose $\boldsymbol{U}$, since the time evolution of each field relies on other fields as well via the Euler equations. This is demonstrated by the results shown in Fig. 2.12.

The three experiments described in the previous paragraphs (summarized in Tbl. 2.2) demonstrate the capabilities of a CNN-PAL model to evolve an initial condition in an accurate fashion. However, the enforcement of stability conditions in the design of CNN-PAL models was not addressed by their original authors, namely Bar-Sinai et al. (2019) and Kochkov et al. (2021). The latter claimed that training for the prediction of longer time series improved a model's stability. Fig. 2.13 shows the evolution of the energy field in a long-range forecasting of the double shear layer from experiment n.3. The forecasting is performed by trained CNN-PAL models from all three experiments. One can notice that even for the models trained for the prediction of longer time series, the energy field experience a growth after a certain time, which indicates lack of stability of the trained CNN-PAL models. In the next chapter, a novel spectral analysis that helps to design accurate and stable data-driven numerical schemes is proposed.

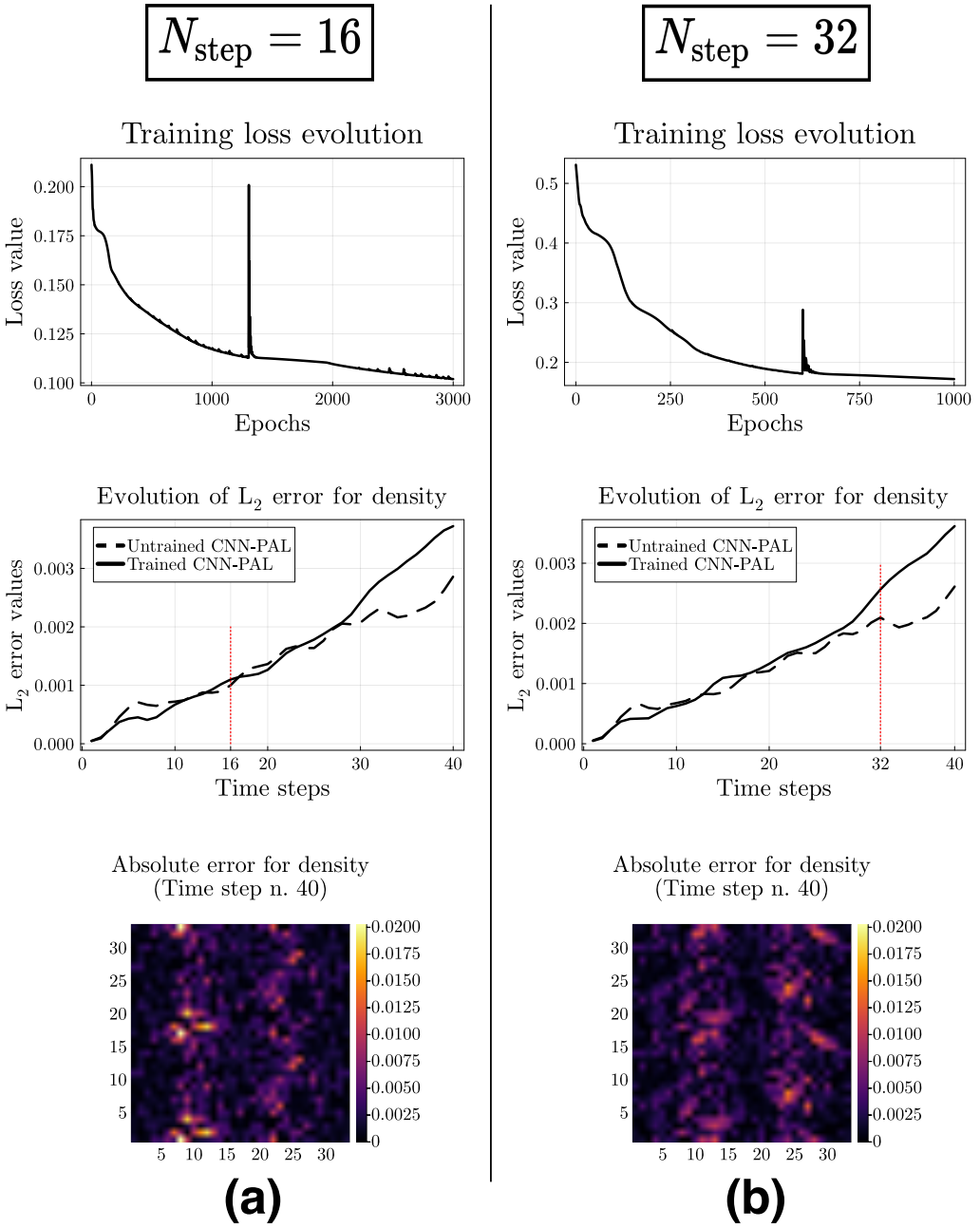


Figure 2.11: Outcomes of experiment n.3 performed with two independent CNN-PAL models trained to predict (a) $N_{\text{step}} = 16$ time steps; (b) $N_{\text{step}} = 32$ time steps of the evolution of the double shear layer initial condition depicted in Tbl. 2.2. The training proceure employs the loss function (2.50). (Top) Evolution of training loss values across epochs. (Center) Evolution of L_2 error of density field over time. (Bottom) Absolute error of density field predicted by the trained CNN-PAL models after 40 simulation time steps.

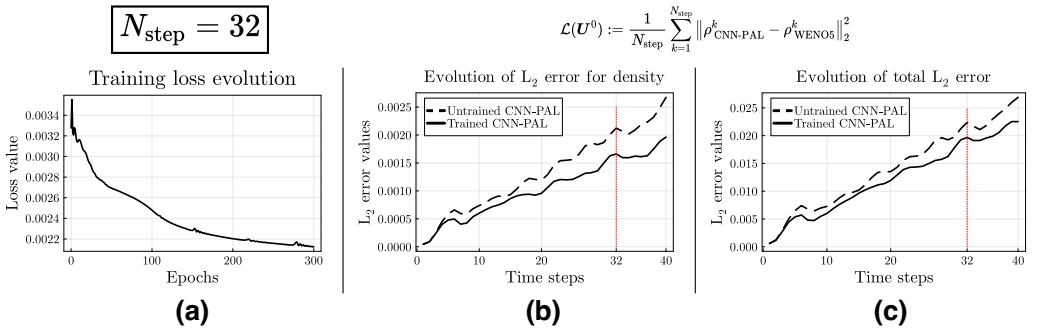


Figure 2.12: Outcomes of experiment n.3 performed with a CNN-PAL model trained to predict $N_{\text{step}} = 32$ time steps of the evolution of the double shear layer initial condition depicted in Tbl. 2.2. The training procedure employs the loss function (2.51). (a) Evolution of training loss values across epochs. (b) Evolution of L_2 error of density field over time. (c) Evolution of the sum of L_2 errors of density, components of momentum along the x and y directions, and energy fields over time.

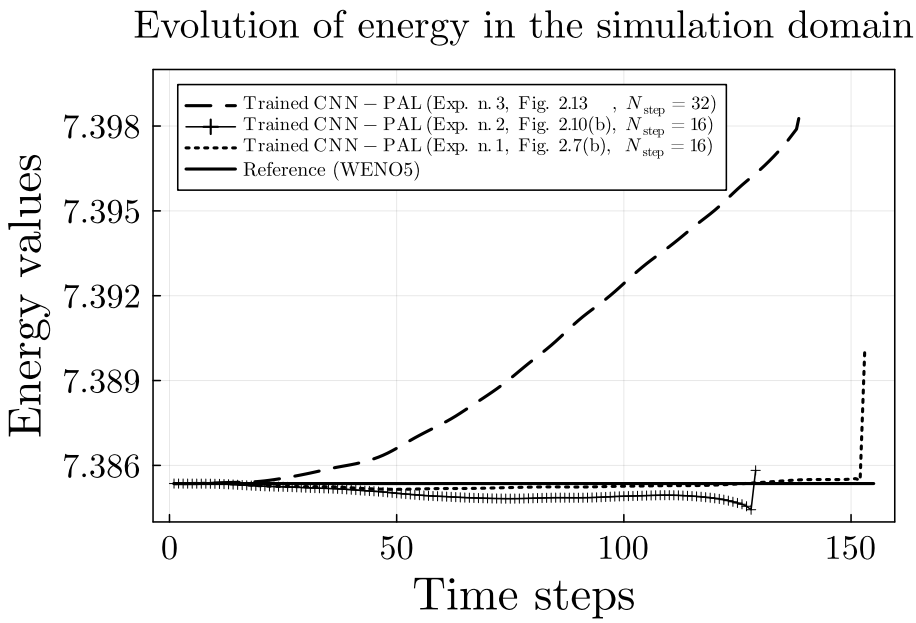


Figure 2.13: Evolution of energy values in the simulation of the double shear layer from experiment n.3 (see Tbl. 2.2). The growth in energy values for the trained CNN-PAL models after a certain time demonstrate numerically unstable dynamics.

Chapter 3

Proposing a novel spectral analysis

The previous chapter showed that the lack of stability constraints in formulating data-driven numerical schemes could lead to unstable predicted dynamics. This chapter introduces a novel spectral analysis that helps to generate stable and accurate ML-based numerical methods called the Local Transfer function Analysis (LTA). LTA transforms the problem of developing stable and accurate numerical schemes into a constraint impedance-matching problem in the spectral domain. Furthermore, using LTA, one can describe the complete dynamics of the discrete numerical solution including some specific numerical artifacts. The considered numerical method belongs to the Taylor-Galerkin family of finite-element schemes. They are employed in CERFACS' solver AVBP for large-eddy simulations in an industrial context, which is the ultimate application of the methodology proposed in this thesis. For the linear convection equation, LTA predicts the generation of spurious waves at the boundary between two distinct uniformly spaced domains and the presence of a local linear instability at the junction. For the inviscid Burgers' equation, LTA is applied at a shock front to explain the emergence of Gibbs' phenomenon near the vicinity of the shock. LTA also shows the lack of numerical dissipation at high wavenumbers for the Two-step Taylor-Galerkin C (TTGC) scheme, which necessitates adding a diffusion term for stability.

3.1 The Taylor-Galerkin family of schemes

The Taylor-Galerkin (TG) schemes were designed to numerically solve convection problems with high numerical accuracy and low numerical dissipation and dispersion errors. The transport of some arbitrary quantity $u = u(x,t)$ over time and space (in 1D) can be modelled using the simple PDE shown in Eq. (3.1):

$$u_t = -cu_x, \tag{3.1}$$

where c is the convective speed and $u(x,0) = u_0(x)$ is the initial solution. This convection operator can also be viewed as a one-sided wave equation (wave moving only to the right). The subscripts $(\cdot)_t = \frac{\partial}{\partial t}$ and $(\cdot)_x = \frac{\partial}{\partial x}$ denote the partial derivatives with respect to the time and space, respectively. For simplicity, spatial periodicity with a period $x \in [0,1]$ is assumed. The first step of the TG formulation involves the approximation of the time derivative in Eq. (3.1) using truncated Taylor series. Two levels of truncation with leading order of $\mathcal{O}(\Delta t^2)$ and $\mathcal{O}(\Delta t^4)$ are shown in Eq. (3.2)-(3.3), respectively:

$$u_t^n = \frac{u^{n+1} - u^n}{\Delta t} - \frac{\Delta t}{2} u_{tt}^n + \mathcal{O}(\Delta t^2) \tag{3.2}$$

$$u_t^n = \frac{u^{n+1} - u^n}{\Delta t} - \frac{\Delta t}{2} u_{tt}^n - \frac{\Delta t^2}{6} u_{ttt}^n - \frac{\Delta t^3}{24} u_{tttt}^n + \mathcal{O}(\Delta t^4), \tag{3.3}$$

where the solution at time $t = t^n$ is denoted by u^n and the solution at a perturbed time $t = t^n + \Delta t$ is u^{n+1}. The temporal derivatives in the truncated Taylor-series expression can be transformed into spatial derivatives using Eq. (3.1). The simplest one-step numerical scheme is obtained from Eq. (3.2) as (Roig, 2007)

$$\delta u^n = -(c\Delta t) u_x^n + \frac{(c\Delta t)^2}{2} u_{xx}^n, \tag{3.4}$$

where $\delta u^n := u^{n+1} - u^n$. Eq. (3.4) is similar to the Lax-Wendroff (FV-LW) scheme used in a finite-volume context. Since the spatial derivatives are obtained using a weak-formulation in the Galerkin finite-element method, one obtains an additional mass-matrix on the RHS of Eq. (3.4), which distinguishes it from the FV-LW. More precisely, a linear basis function ψ also called the hat function (plotted in Fig. 3.1) is considered to project the continuous operators to the discrete space of the basis function. It is defined

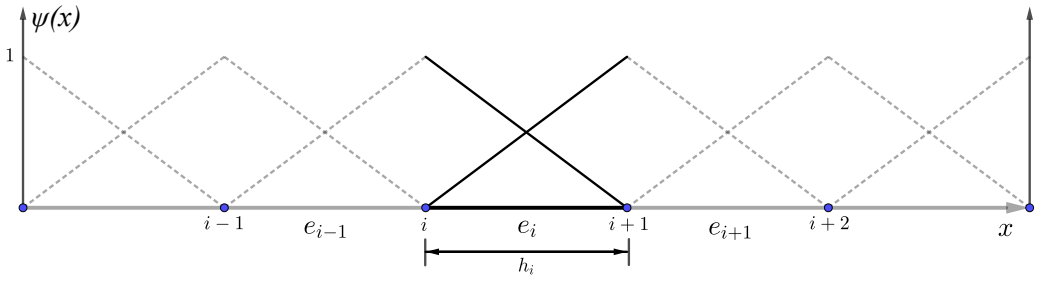


Figure 3.1: 1D linear finite-element shape function (hat function) with nodes and elements enumerated.

as shown in Eq. (3.5), where h_i defines the cell size or the width of the 1D finite-element e_i with end nodes i and $i+1$. Note that the basis function satisfies the sifting property $\psi_i(x_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function.

$$\psi_i(x) := \begin{cases} 0 & \text{for } x < x_{i-1} \text{ or } x > x_{i+1} \\ h_{i-1}^{-1}(x - x_{i-1}) & \text{for } x_{i-1} \leq x < x_i \\ 1 - h_i^{-1}(x - x_i) & \text{for } x_i \leq x < x_{i+1} \end{cases} \tag{3.5}$$

Using this basis allows to express the solution u in terms of its nodal values u_j defined at node j : $u(x) = \sum_{j=1}^{N} u_j \psi_j(x)$. This basis when introduced into the semi-discrete temporal TG formulation and upon taking the inner product (Galerkin projection) with this same basis yields the complete weak formulation of the TG finite-element method. The weak formulation of the scheme (3.4) is shown in Eq. (3.6), where $\langle a, b\rangle = \int ab \; dx$ defines the inner product of two arbitrary functions $a = a(x)$ and $b = b(x)$:

$$\sum_{j=1}^{N} \langle \delta u_j^n \, \psi_j, \psi_i \rangle = -(c\Delta t) \sum_{j=1}^{N} \langle u_j^n \, (\psi_x)_j, \psi_i \rangle - \frac{(c\Delta t)^2}{2} \sum_{j=1}^{N} \langle u_j^n \, (\psi_x)_j, (\psi_x)_i \rangle. \tag{3.6}$$

In discretization (3.6), the last inner product is obtained through integration by parts, eliminating the need for the basis function to be twice-differentiable. According to Sec. 12.15.5 of T. Sengupta (2013), this form is considered to be beneficial when the physical solution admits discontinuities. Furthermore, if the governing equation can be cast in self-adjoint form, the linear algebraic

equation which results from the discretization using integration by parts involves symmetric matrix. This leads to significantly reduced computational effort by achieving faster convergence. Discretization (3.6) is hereafter referred to as TG-LW for convenience. Assuming a regular mesh (i.e., $h_i \equiv h$), the integrals in Eq. (3.6) can be evaluated and simplified for the linear basis function as follows:

$$h^{-1}M_{ij}a_j = \sum_{j=1}^{N}\langle a_j\psi_j, \psi_i\rangle = \frac{a_{i+1} + 4a_i + a_{i-1}}{6h} \tag{3.7}$$

$$K_{ij}a_j = \sum_{j=1}^{N}\langle a_j\psi_{x,j}, \psi_i\rangle = \frac{a_{i+1} - a_{i-1}}{2} \tag{3.8}$$

$$h^{-1}D_{ij}a_j = \sum_{j=1}^{N}\langle a_j\psi_{x,j}, \psi_{x,i}\rangle = \frac{a_{i+1} - 2a_i + a_{i-1}}{h}, \tag{3.9}$$

where a_i is some arbitrary nodal value at node i of function a; M, K and D are the mass, stiffness and damping matrices. The weak form of TG-LW can be written in terms of these matrices as

$$M_{ij}\delta\tilde{u}_j^n = -N_cK_{ij}u_j^n + \frac{N_c^2}{2}D_{ij}u_j^n, \tag{3.10}$$

where $N_c := h^{-1}(c\Delta t)$ is the Courant-Friedrichs-Lewy (CFL) number, a non-dimensional quantity that relates to stability conditions for the numerical scheme as shown in Sec. 3.2.

Donea et al. (1987) developed two-step TG schemes in order to increase time discretization accuracy. It starts by noticing that the 4th-order truncation of the time derivative u_t from Eq. (3.3) can be rewritten as

$$u_t^n = \frac{u^{n+1} - u^n}{\Delta t} - \frac{\Delta t}{2}\tilde{u}_{tt}^n + \mathcal{O}(\Delta t^4), \tag{3.11}$$

where

$$\tilde{u}^n := u^n + \frac{\Delta t}{3}u_t^n + \frac{\Delta t^2}{12}u_{tt}^n.$$

This leads to the two-step TG formulation

$$\delta\tilde{u}^n = \alpha\Delta t\, u_t^n + \bar{\alpha}\Delta t^2\, u_{tt}^n \tag{3.12}$$

$$\delta u^n = \Delta t\, u_t^n + \frac{\Delta t^2}{2}\,\tilde{u}_{tt}^n, \tag{3.13}$$

where $\delta\tilde{u}^n := \tilde{u}^n - u^n$ and the parameters $\alpha = \frac{1}{3}$ and $\bar{\alpha} = \frac{1}{12}$ produce the so-called fourth-order TTG4A scheme. By choosing $\bar{\alpha} = \frac{1}{9}$, one obtains a third-order scheme named TTG3. The Galerkin projection using the linear basis function ψ generates the following weak form for TTG3/4A:

$$M_{ij}\delta\tilde{u}_j^n = -\alpha N_c K_{ij} u_j^n + \bar{\alpha} N_c^2 D_{ij} u_j^n \tag{3.14}$$

$$M_{ij}\delta u_j^n = -N_c K_{ij} u_j^n + \frac{N_c^2}{2} D_{ij} \tilde{u}_j^n. \tag{3.15}$$

Colin and Rudgyard (2000) proposed a six-parameter TG scheme (namely, $\{\alpha, \beta, \theta_1, \theta_2, \epsilon_1, \epsilon_2\}$) that gives third-order accuracy with lower overall dissipation (Eq. (3.16)-(3.17)):

$$\delta\tilde{u}^n = \alpha\Delta t u_t^n + \beta\Delta t^2 u_{tt}^n \tag{3.16}$$

$$\delta u^n = \Delta t(\theta_1 u_t^n + \theta_2 \tilde{u}_t^n) + \Delta t^2(\epsilon_1 u_{tt}^n + \epsilon_2 \tilde{u}_{tt}^n). \tag{3.17}$$

By performing the Fourier stability analysis assuming a linear finite-element basis, they obtained the optimal values for the six-parameter model. The optimal values reduce the scheme to the single parameter Two-step Taylor-Galerkin C (TTGC)-γ scheme for which

$$\alpha = \frac{1}{2} - \gamma,\ \beta = \frac{1}{6},\ \theta_1 = 0,\ \theta_2 = 1,\ \epsilon_1 = \gamma,\ \epsilon_2 = 0. \tag{3.18}$$

Note that γ is a free parameter that controls the dissipation at high wavenumbers. The TTGC-γ scheme is third-order accurate on regular meshes. For the parameter setting $\alpha = \frac{1}{3}$, $\beta = \bar{\alpha}$, $\theta_1 = 1$, $\theta_2 = 0$, $\epsilon_1 = 0$ and $\epsilon_2 = \frac{1}{2}$, the TTG4A ($\bar{\alpha} = \frac{1}{12}$) and TTG3 ($\bar{\alpha} = \frac{1}{9}$) schemes of Donea et al. (1987) can be derived from this same six-parameter family. The weak form of TTGC-γ can be written in terms of the mass, stiffness and damping matrices from Eq. (3.7)-(3.8)-(3.9) as follows:

$$M_{ij}\delta\tilde{u}_j^n = -\alpha N_c K_{ij} u_j^n + \beta N_c^2 D_{ij} u_j^n \tag{3.19}$$

$$M_{ij}\delta u_j^n = -N_c K_{ij} \tilde{u}_j^n + \gamma N_c^2 D_{ij} u_j^n. \tag{3.20}$$

3.2 Stability, dissipation and dispersion analysis

Recently, Najafiyazdi et al. (2018) derived a low-dispersion and dissipation TTG scheme starting from the TTGC. They show that using multi-staging

one can prevent the evaluation of the third-order spatial derivative that is tedious to obtain for Euler and Navier-Stokes equations. In addition, they provide the connection between the multistage Runge-Kutta (RK) schemes and the TG family of schemes. In fact, TTGC is a sub-class of the Turan-type RK methods. An important problem not fully addressed by Colin and Rudgyard (2000) is the determination of the optimal γ considering a range of N_c values. The authors utilize a single global optimal value for γ for a fixed N_c. Using a single global value for γ as in Colin and Rudgyard (2000) demands additional artificial dissipation when solving non-linear system with shock or jump discontinuities on irregular meshes (Roux et al., 2010). In this section, spectral properties of TG schemes in general are depicted and a novel spectral analysis based on local spatio-temporal features is proposed in order to theoretically approach the aforementioned problem for the TTGC scheme in particular.

3.2.1 Global Spectral Analysis (GSA)

Traditional spatial and temporal order of error analysis merely gives the leading truncation error in space and time but does not shed light on the stability and spatio-temporal evolution of the numerical solution. The numerical stability of a discretization of a PDE is classically studied using the von Neumann analysis (Charney et al., 1950). It is based on the decomposition of the numerical error into Fourier series. The numerical scheme is said to be stable if the errors do not grow unboundedly over time.

Nevertheless, the von Neumann analysis presents limitations: it is applicable to spatially periodic problems only; it implicitly assumes no interactions among the Fourier modes of the numerical error; and it assumes that the numerical error and the numerical solution follow the same dynamics (Sagaut et al., 2023). T. Sengupta, Ganeriwal, et al. (2003) addressed these limitations by proposing a novel analysis named the Global Spectral Analysis (GSA). GSA provides information on both the stability and spatio-temporal behavior of the numerical solution. This is achieved by transforming the discretized form and the governing equation into the spectral space using a bi-directional Fourier-Laplace transformation:

$$u(x,t) = \int_{\Omega}\int_{\mathcal{K}} U(k,\omega)e^{i(kx-\omega t)}dk\ d\omega, \tag{3.21}$$

where k and ω are the spatial wavenumber and temporal frequency, respectively. The convective problem in Eq. (2.4) transforms to the simplified ex-

pression $\omega = kc$ in the spectral space. This is the physical dispersion relation that relates the phase speed (c) to the wavenumber (k) and circular frequency (ω). In addition to the individual phase speed of a wave it is also possible to define the velocity associated with a group of waves called the group velocity, $v_g := d\omega/dk$. For linear dispersive wave propagation, the energy propagates at the group velocity (and not at the phase speed), making it much more important than the phase. Rayleigh (1899) was instrumental in establishing the theory of group velocity, as opposed to phase speed. In addition, for non-dispersive linear propagation, the phase speed and group velocity are the same. Due to the discretization procedure, the numerical solution rather follows the so-called numerical dispersion relation $\omega_N = c_N k$, where c_N is the numerical phase speed (T. K. Sengupta and Dipankar, 2004). Since numerical solutions are almost always dispersive, matching of numerical and exact group velocities becomes critical to accurately capture the energy dynamics of the system. The amplification factor (G) is another important quantity that measures the amount of dissipation and that affects the dispersion the solution undergoes within a time interval of Δt. More precisely, G affects the value of the numerical phase speed c_N, which differs from its physical value c. Since the linear convection in Eq. (2.4) is non-dissipative and non-dispersive, its amplification factor is $G_{\text{phy}} := e^{-ikc\Delta t} \equiv \cos(kc\Delta t) - i\sin(kc\Delta t)$, for which $|G_{\text{phy}}| = 1$.

To obtain the numerical amplification, the existence of a continuous initial condition $u(x, t^n)$ at some time level t^n is considered. Let $\widehat{U}(k, t^n)$ be the spatial Fourier transform of the function at t^n and $\widehat{U}(k, t^n + \Delta t)$ is the solution at $t^n + \Delta t$. The functions can be defined using their Fourier transforms as shown below:

$$u(x, t^n) := \int_{\mathcal{K}} \widehat{U}(k, t^n) e^{ikx} dk \tag{3.22}$$

and

$$u(x, t^n + \Delta t) := \int_{\mathcal{K}} \widehat{U}(k, t^n + \Delta t) e^{ikx} dk. \tag{3.23}$$

The discrete sampling of this function at points x_i is performed by sifting with the Dirac delta function $\delta(x - x_i)$ transforming the samples into a generalized

function representation as shown in Eq. (3.24)-(3.25):

$$\begin{aligned} u_i^n = u(x,t^n) * \delta(x-x_i) &= \int_X u(x,t^n)\delta(x-x_i)dx \\ &= \int_X \left[\int_{\mathcal{K}} \widehat{U}(k,t^n)e^{ikx}dk\right]\delta(x-x_i)dx \\ &= \int_{\mathcal{K}} \left[\int_X \widehat{U}(k,t^n)e^{ikx}\delta(x-x_i)dx\right]dk \\ &= \int_{\mathcal{K}} \widehat{U}(k,t^n)e^{ikx_i}dk \end{aligned} \tag{3.24}$$

$$u_i^{n+1} = u(x,t^n+\Delta t) * \delta(x-x_i) = \int_{\mathcal{K}} \widehat{U}(k,t^n+\Delta t)e^{ikx_i}dk \tag{3.25}$$

Note that X in the integral is the spatial domain of the problem and $*$ is the generalized product (Kanwal, 2004; Lighthill, 1958). In general, the limits of the integration is $-\infty < k < \infty$ but in the case of equally spaced spatial samples (h), the Nyquist sampling theorem states that the numerical scheme can represent wavenumbers only within the range $-\pi < p < \pi$, where $p := kh$.

The numerical amplification G_{num} is then defined as the ratio of the spatial Fourier transforms at time t^n and $t^n + \Delta t$ as shown below (T. Sengupta, Dipankar, et al., 2007):

$$G_{\text{num}} := \frac{\widehat{U}(k,t^n+\Delta t)}{\widehat{U}(k,t^n)} := \frac{\widehat{U}^{n+1}}{\widehat{U}^n} \tag{3.26}$$

Here the numerical phase c_N and group v_{gn} velocities are obtained using the real $\Re(G_{\text{num}})$ and imaginary $\Im(G_{\text{num}})$ parts of the numerical amplification as follows (T. Sengupta, Dipankar, et al., 2007):

$$\frac{c_N}{c} := \frac{\phi_c}{pN_c} \tag{3.27}$$

$$\frac{v_{gn}}{c} := \frac{1}{N_c}\frac{d\phi_c}{dp}, \tag{3.28}$$

where

$$\phi_c := -\arctan\left(-\frac{\Im(G)}{\Re(G)}\right),\ N_c := \frac{c\Delta t}{h}, \tag{3.29}$$

and

$$\frac{d\phi_c}{dp} := \frac{1}{|G|^2}\left(\Im(G)\frac{d\Re(G)}{dp} - \Re(G)\frac{d\Im(G)}{dp}\right). \tag{3.30}$$

An ideal numerical scheme for the convection equation must have unit value for the above defined ratios, i.e., satisfies the condition $|G_{\text{num}}|/|G_{\text{phy}}| = 1$, $c_N/c = 1$ and $v_{gn}/v_g = 1$. Due to the presence of finite dissipation and dispersion errors in numerical schemes, these ratios tend to deviate from this unit value. T. Sengupta, Dipankar, et al. (2007) give an exact dynamics of the solution error $\epsilon := (u_{\text{num}} - u)$ in the numerical solution u_{num} using the above defined ratios using the error propagation equation shown below:

$$\begin{aligned}\frac{\partial \epsilon}{\partial t} + c\frac{\partial \epsilon}{\partial x} =& -c\left[1 - \frac{c_N}{c}\right]\frac{\partial u_{\text{num}}}{\partial x} \\ &- \int \left(\frac{v_{gn} - c_N}{k}\right)\left[\int_0^k ik' U_0 |G_{\text{num}}|^{\frac{t}{\Delta t}} e^{ik'(x - c_N t)} dk'\right] dk \\ &- \int \frac{\ln\left(|G_{\text{num}}|\right)}{\Delta t} U_0 |G_{\text{phy}}|^{\frac{t}{\Delta t}} e^{ik(x - c_N t)} dk \end{aligned} \tag{3.31}$$

It is interesting to note that the error follows the same dynamics as the exact equation with extra source terms defined by the dispersion and dissipation errors. The main highlight of GSA is that it identifies that the constant phase in Eq. (2.4) does not remain a constant, rather the numerical phase speed (c_N) is found to be a function of wavenumber (k), an attribute causing dispersion of numerical solution (T. K. Sengupta and A. Sengupta, 2016). For a detailed comparison of GSA with other spectral analysis the reader is referred to T. K. Sengupta and Dipankar (2004); T. Sengupta (2013).

3.2.2 Application of GSA to Taylor-Galerkin schemes

In order to obtain the numerical amplification, the spatial Fourier transform is applied to the weak numerical discretization (generalized function representation for the samples) in Eq. (3.19)-(3.20) for the TTGC-γ scheme over equally spaced spatial samples and one obtains the following expression

$$\hat{\tilde{U}}^n = \hat{U}^n - i\alpha N_c \widehat{M}^{-1}\widehat{K}\hat{U}^n + \beta N_c^2 \widehat{M}^{-1}\widehat{D}\hat{U}^n \tag{3.32}$$

$$\hat{U}^{n+1} = \hat{U}^n - iN_c \widehat{M}^{-1}\widehat{K}\hat{\tilde{U}}^n + \gamma N_c^2 \widehat{M}^{-1}\widehat{D}\hat{U}^n, \tag{3.33}$$

Method	Amplification factor (G)	
	Real part $\Re(G)$	Imaginary part $\Im(G)$
Pure convection from Eq. (3.1)	$\cos(pN_c)$	$-\sin(pN_c)$
FV-LW (see Roig (2007))	$1+2^{-1}N_c^2\widehat{D}$	$-N_c\widehat{K}$
TG-LW from Eq. (3.4)	$1+2^{-1}N_c^2\widehat{A}\widehat{D}$	$-N_c\widehat{A}\widehat{K}$
TTG3/4A (see Donea et al. (1987))	$1+2^{-1}N_c^2\widehat{A}\widehat{D}(1+\bar{\alpha}N_c^2\widehat{A}\widehat{D})$	$-N_c\widehat{A}\widehat{K}(1+6^{-1}N_c^2\widehat{A}\widehat{D})$
TTGC-γ from Eq. (3.35)	$1+\gamma N_c^2\widehat{A}\widehat{D}-\alpha(N_c\widehat{A}\widehat{K})^2$	$-N_c\widehat{A}\widehat{K}\left(1+\beta N_c^2\widehat{A}\widehat{D}\right)$

Table 3.1: Amplification factor (real and imaginary part) for indicated numerical method; where $\widehat{A}\equiv\widehat{M}^{-1}$.

where the transformed mass, stiffness and damping matrices are given by

$$\widehat{M}=3^{-1}\left[2+\cos(p)\right],\quad \widehat{K}=\sin(p),\quad \widehat{D}=2\left[\cos(p)-1\right],$$

and $p=kh$, where k is the spatial wavenumber of the solution in the transformed plane. The amplification factor (G_{num}) for the TTGC-γ numerical discretization is shown in Eq. (3.34)-(3.35):

$$\tilde{G}=\frac{\hat{\tilde{U}}^n}{\widehat{U}^n}=1-i\alpha N_c\widehat{M}^{-1}\widehat{K}+\beta N_c^2\widehat{M}^{-1}\widehat{D} \tag{3.34}$$

$$G_{\text{num}}=\frac{\widehat{U}^{n+1}}{\widehat{U}^n}=1-iN_c\widehat{M}^{-1}\widehat{K}\tilde{G}+\gamma N_c^2\widehat{M}^{-1}\widehat{D} \tag{3.35}$$

Notice that the numerical amplification G_{num} is a complex function which varies with N_c and p, i.e., $G_{\text{num}}=G_{\text{num}}(N_c,p)$. Therefore, finite dissipation and dispersion errors in the numerical solution are almost always found. To compare the numerical and physical amplification the following ratios are considered: (i) damping ratio ($|G_{\text{num}}|/|G_{\text{phy}}|$), (ii) phase velocity ratio (c_N/c) and (iii) group velocity ratio (v_{gn}/v_g). A summary of the amplification factor for the various TG numerical schemes is provided in Tbl. 3.1. The contour plot of the three ratios for the numerical schemes shown in Tbl. 3.1 are plotted in Fig. 3.2 versus CFL number (N_c) and non-dimensional wavenumber ($p=kh$). All figures stacked column-wise are the ratios $|G_{\text{num}}|/|G_{\text{phy}}|$, c_N/c and v_{gn}/v_g, while stacked row-wise are the various numerical methods forming a figure matrix.

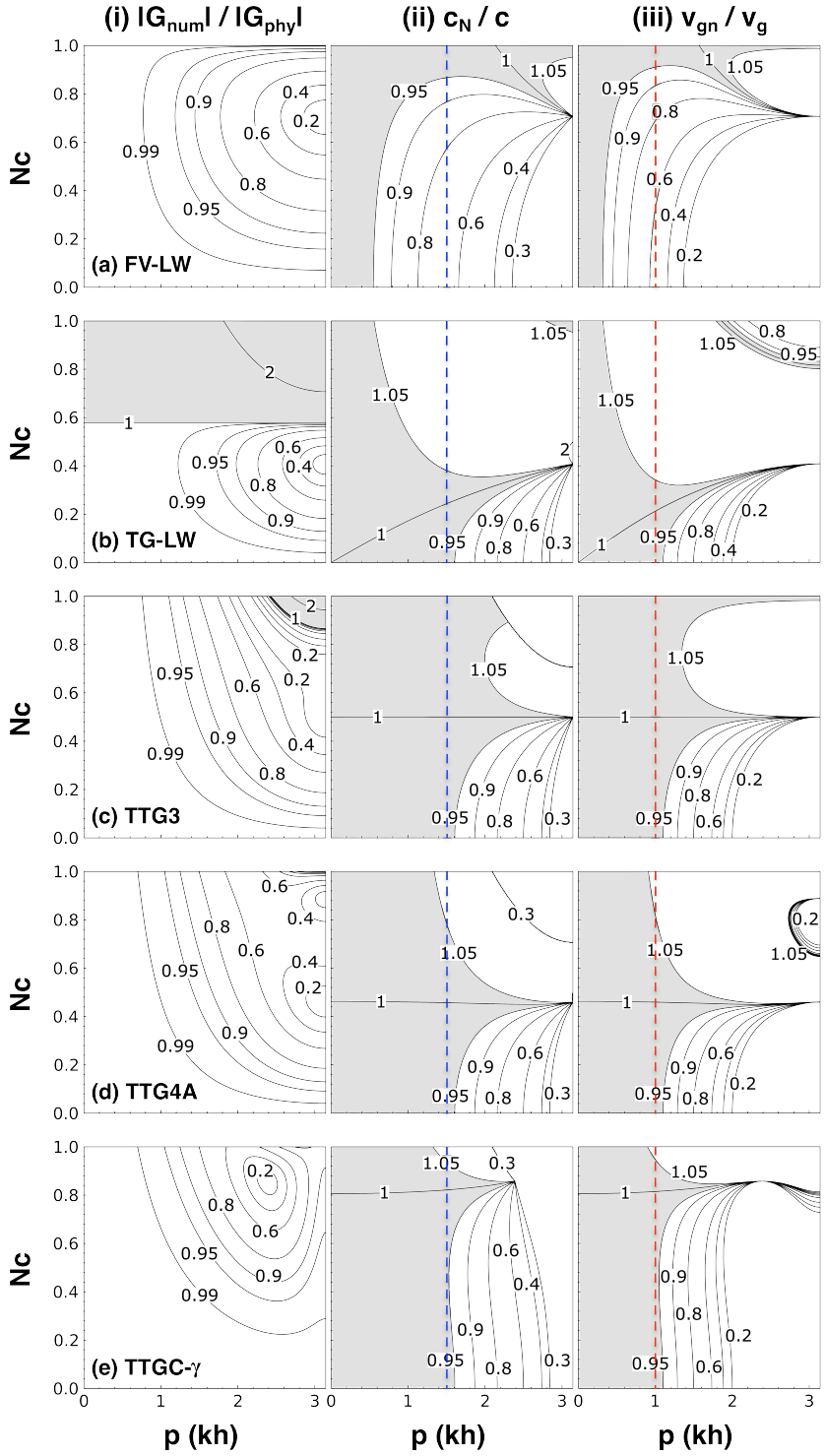


Figure 3.2: Comparison of column-wise (i) $|G_{\text{num}}|/|G_{\text{phy}}|$; grey area shows unstable region $|G| > 1$, (ii) c_N/c; grey area shows region where $0.95 \leq c_N/c \leq 1.05$ and dotted line is $p = 1.5$ and (iii) v_{gn}/v_g; grey area shows GVP region $0.95 \leq v_{gn}/v_g \leq 1.05$ and dotted line is $p = 1$ for the indicated methods (row-wise (a-e)) tabulated in tbl. 3.1.

3.2.2.1 Group velocity preservation

Group velocity preservation is critical compared to preserving the phase speed because, in dispersive systems, the group velocity is the true measure of energy propagation (Rayleigh, 1899). The Group Velocity Preserving (GVP) region (grey shaded area in the v_{gn}/v_g contours of Fig. 3.2) is defined as the numerical group velocity v_{gn} within the $\pm 5\%$ tolerance of the physical value. Fig. 3.2(ii) shows the GVP region for various numerical schemes. It is worth noting that TG schemes have much larger GVP regions compared to the second order FV-LW scheme, which makes them more suitable for unsteady computations. This feature is attributed to the additional mass-matrix term that bestows a compact stencil to the gradient and diffusion terms.

3.2.2.2 Filtering effect of the mass-matrix

Two features are common to the Taylor-Galerkin schemes considered here (TG-LW, TTG3, TTG4A and TTGC-γ), namely: (i) they have similar GVP regions and (ii) their GVP regions are much larger than that of FV-LW (for relatively low values of the CFL condition number). This is mainly attributed to the addition of mass-matrix which produces a compact stencil for the gradient evaluation similar to the compact schemes in finite-difference. This is the reason one can find a stark contrast in the GVP region between the FV-LW and other TG schemes. Note that the additional mass-matrix term ($A \equiv M^{-1}$) improves significantly the GVP region but at the cost of reduced stability in TG-LW scheme. To illustrate this, the values of $\widehat{A}$ are plotted in Fig. 3.3 for the Nyquist range $0 \leq kh \leq \pi$: the inverse mass-matrix term $A \equiv M^{-1}$ compensates the dissipation and dispersion at high wavenumber components by adding anti-diffusion. The role of A in GVP can be confirmed by noticing that the GVP region of the simple one-step TG-LW is quite similar to that of other two-step schemes for relatively low values of the CFL condition number. This is the primary reason approximations to mass-matrix (by lumping) generate unwanted dispersion errors (Guermond and Pasquetti, 2013). Petrov-Galerkin (PG) schemes (Cardle, 1995) exploit this filtering property of mass-matrix to introduce upwind basis function that adds stabilization by modifying the matrix as $M^* = I + 6^{-1}(1-\beta_c)D$, where β_c is a parameter that controls stabilization. For (i) $\beta_c = 0$ one recovers the exact mass-matrix ($M^* = M$), (ii) $\beta_c = 1$ one obtains a block-diagonal form

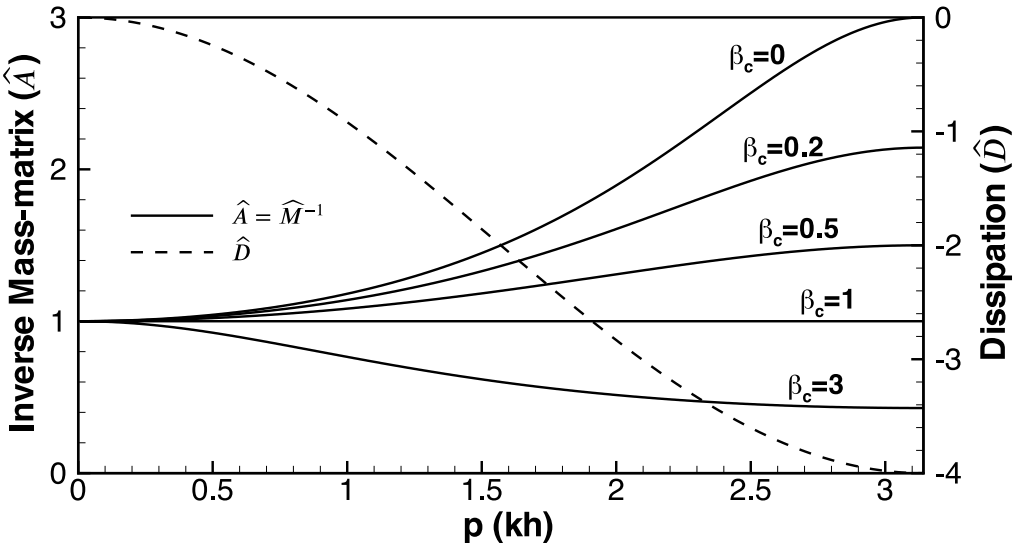


Figure 3.3: Inverse mass-matrix spectral plot ($\widehat{A} = \widehat{M}^{-1}$) compared with the dissipation $\widehat{D}$ to illustrate its anti-diffusive character and the stabilization effect of β_c.

($M = I$) similar to the FV-LW and (iii) $\beta_c > 1$ the stabilization behaves like an implicit filter filtering out the high wavenumbers. The inverse of mass-matrix ($A = M^{-1}$) is plotted in Fig. 3.3 for $0 \leq p \leq \pi$. Essentially changing β_c reduces the anti-diffusion character of M^{-1} resulting in increased dispersion and dissipation.

The dissipation and GVP regions for the TG-LW scheme are plotted in Fig. 3.4 for $\beta_c \in \{0, 0.5, 0.85, 1\}$. The stability of the TG-LW scheme is greatly improved by higher values of β_c. At the same time the operational CFL of the scheme can also be increased significantly by varying the parameter. For convection-diffusion equation, Cardle (1995) gives a variety of choices for β_c based on the type of temporal discretization. Berzins (2001) suggests to use a non-linear function for β_c that preserves the positivity of the mass-matrix for stabilization and shock capturing.

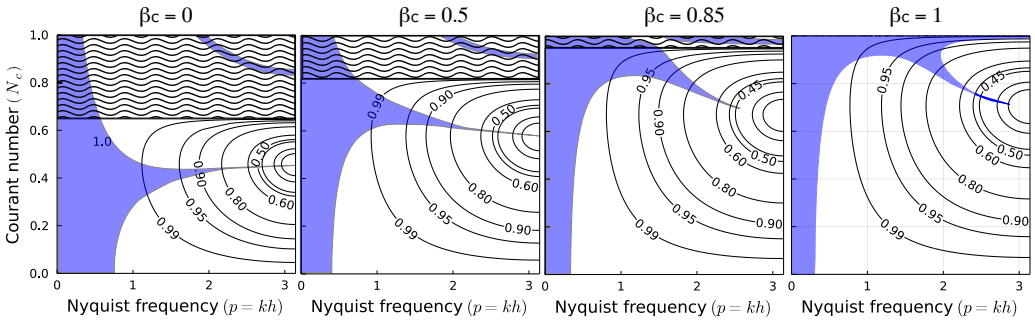


Figure 3.4: Dissipation (solid contour) and GVP region (blue contour) for various β_c values in TG-LW scheme; wavy regions are numerically unstable.

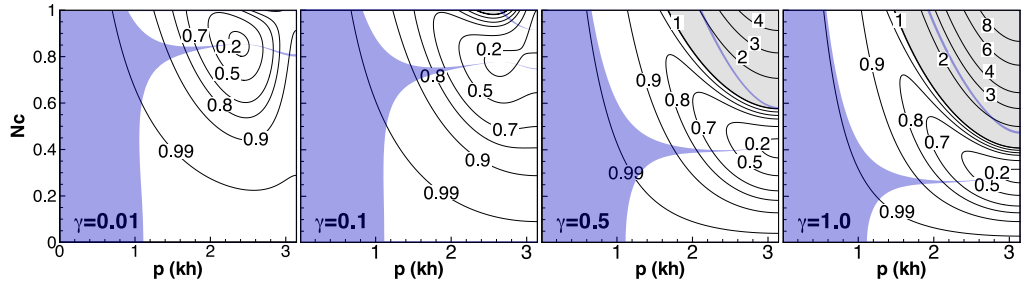


Figure 3.5: $|G_{\text{num}}|/|G_{\text{phy}}|$ contours for indicated values of parameter γ in the TTGC-γ scheme; (i) grey shaded area indicates numerically unstable regions ($|G_{\text{num}}| > 1$) and (ii) blue shaded region indicates the group velocity preserving zone ($0.95 \leq v_{gn}/v_g \leq 1.05$).

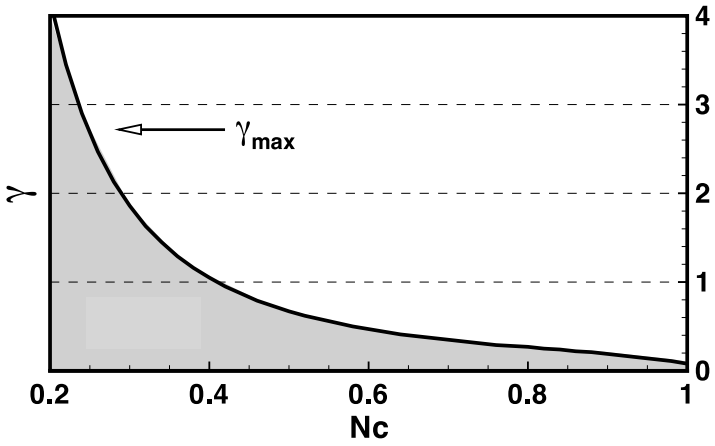


Figure 3.6: Global CFL number versus maximum stable $\gamma = \gamma_{max}$; stable region obtained from GSA indicated in grey shaded area.

3.2.2.3 Need for tuning parameters and adaptive nature of TTGC-γ scheme

An interesting feature is that the TTGC-γ scheme of Colin and Rudgyard (2000) can vary its spectral properties at high wavenumbers by varying γ. The variation of the modulus of the amplification factor of the TTGC-γ scheme by varying γ is shown in Fig. 3.5. This is not possible in TTG3/4A that does not have this tunable parameter. The possibility of redesigning the TTGC-γ such that one varies γ based on the CFL and attenuate spurious waves to some extent is raised. But one must be careful to ensure stability because at higher values of γ the scheme becomes unstable for larger N_c. The range of maximum allowable $\gamma = \gamma_{max}$ can be deduced from the stability analysis and is shown in Fig. 3.6.

3.2.3 Locally tunable TTGC-γ scheme (TTGC-γ_L)

Using GSA, it was shown that parameters with local behavior in the numerical scheme are essential for achieving optimal accuracy. In irregular meshes, even if the convection speed c and time step Δt are kept constant, the local CFL number N_c $(= h^{-1}c\Delta t)$ of the finite-element will differ from one element to another due to variation in local cell size (h). Note that non-linear convective equations even on regular meshes (uniformly spaced) give rise to

variable convective speeds across elements. Therefore, it is clear that for irregular meshes and non-linear convective systems, choosing the right value of γ becomes critical to the successful application of the TTGC scheme. These results motivate the construction of a locally tunable TTGC-γ scheme called TTGC-γ_L. This scheme allows local variations in γ to minimize errors due to numerical dispersion/dissipation contrary to fixing a global γ value in the original TTGC-γ formulation. The weak form of the original TTGC-γ scheme is stated as

$$\sum_{j=1}^{N}\langle\delta\tilde{u}_j^n\psi_j,\psi_i\rangle = -\alpha(c\Delta t)\sum_{j=1}^{N}\langle u_j^n(\psi_x)_j,\psi_i\rangle - \beta(c\Delta t)^2\sum_{j=1}^{N}\langle u_j^n(\psi_x)_j,(\psi_x)_i\rangle \tag{3.36}$$

$$\sum_{j=1}^{N}\langle\delta u_j^n\psi_j,\psi_i\rangle = -(c\Delta t)\sum_{j=1}^{N}\langle\tilde{u}_j^n(\psi_x)_j,\psi_i\rangle - \gamma(c\Delta t)^2\sum_{j=1}^{N}\langle u_j^n(\psi_x)_j,(\psi_x)_i\rangle, \tag{3.37}$$

where $\alpha = 2^{-1}-\gamma$ and $\beta = 6^{-1}$. To build TTGC-γ_L, the coefficients of the original TTGC-γ scheme are redefined using elemental γ values, i.e., γ can vary from one element to another. Given that the support of ψ_i is $[x_{i-1},x_{i+1}]$, the sums in Eqs. (3.36)-(3.37) are expressed in the scope of TTGC-γ_L as

$$\begin{aligned}\sum_{j=1}^{N}\langle\delta u_j^n\psi_j,\psi_i\rangle \longleftarrow\ & \delta u_{i-1}^n\langle\psi_{i-1},\psi_i\rangle \\ & + \delta u_i^n\langle\psi_i,\psi_i\rangle \\ & + \delta u_{i+1}^n\langle\psi_{i+1},\psi_i\rangle \end{aligned} \tag{3.38}$$

$$\begin{aligned}\alpha\sum_{j=1}^{N}\langle u_j^n(\psi_x)_j,\psi_i\rangle \longleftarrow\ & u_{i-1}^n\langle\alpha_{e_{i-1}}(\psi_x)_{i-1},\psi_i\rangle \\ & + u_i^n\left[\langle\alpha_{e_{i-1}}(\psi_x)_i|_{e_{i-1}},\psi_i|_{e_{i-1}}\rangle + \langle\alpha_{e_i}(\psi_x)_i|_{e_i},\psi_i|_{e_i}\rangle\right] \\ & + u_{i+1}^n\langle\alpha_{e_i}(\psi_x)_{i+1},\psi_i\rangle \end{aligned} \tag{3.39}$$

$$\begin{aligned}\gamma\sum_{j=1}^{N}\langle u_j^n(\psi_x)_j,(\psi_x)_i\rangle \longleftarrow\ & u_{i-1}^n\langle\gamma_{e_{i-1}}(\psi_x)_{i-1},(\psi_x)_i\rangle \\ & + u_i^n\left[\langle\gamma_{e_{i-1}}(\psi_x)_i|_{e_{i-1}},(\psi_x)_i|_{e_{i-1}}\rangle + \langle\gamma_{e_i}(\psi_x)_i|_{e_i},(\psi_x)_i|_{e_i}\rangle\right] \\ & + u_{i+1}^n\langle\gamma_{e_i}(\psi_x)_{i+1},(\psi_x)_i\rangle, \end{aligned} \tag{3.40}$$

where γ_{e_i} is the value of γ associated with the finite-element e_i (with end nodes i and $i+1$) and $\alpha_{e_i} := 2^{-1}-\gamma_{e_i}$. Finite element codes on unstructured

grids make use of the so-called *element sub-domain paradigm*, which consists in splitting any integrals into a sum over the sub-domains or elements (Löhner, 2008). The restrictions of the sums (3.38)-(3.39)-(3.40) to an element e_i are expressed as

$$\left(\sum_{j=1}^{N}\langle \delta u_j^n \psi_j, \psi_i\rangle\right)\Bigg|_{e_i} \longleftarrow \delta u_i^n \langle \psi_i|_{e_i}, \psi_i|_{e_i}\rangle + \delta u_{i+1}^n \langle \psi_{i+1}, \psi_i\rangle \tag{3.41}$$

$$\left(\alpha\sum_{j=1}^{N}\langle u_j^n (\psi_x)_j, \psi_i\rangle\right)\Bigg|_{e_i} \longleftarrow u_i^n \langle \alpha_{e_i}\,(\psi_x)_i|_{e_i},\, \psi_i|_{e_i}\rangle + u_{i+1}^n \langle \alpha_{e_i}(\psi_x)_{i+1}, \psi_i\rangle \tag{3.42}$$

$$\left(\gamma\sum_{j=1}^{N}\langle u_j^n (\psi_x)_j, (\psi_x)_i\rangle\right)\Bigg|_{e_i} \longleftarrow u_i^n \langle \gamma_{e_i}\,(\psi_x)_i|_{e_i},\, (\psi_x)_i|_{e_i}\rangle + u_{i+1}^n \langle \gamma_{e_i}(\psi_x)_{i+1}, (\psi_x)_i\rangle. \tag{3.43}$$

Considering the coordinate $x \in [0, h_i]$ for the element e_i, one has $\psi_i(x) := 1 - x/h_i$ and $\psi_{i+1}(x) := x/h_i$. This results in the following expressions for the inner products in the restrictions (3.41)-(3.42)-(3.43):

$$\begin{aligned}
\langle \psi_i|_{e_i}, \psi_i|_{e_i}\rangle &= \int_0^{h_i} (1 - x/h_i)^2 \,\mathrm{d}x = h_i/3 \\
\langle \psi_{i+1}, \psi_i\rangle &= \int_0^{h_i} (1 - x/h_i) x/h_i \,\mathrm{d}x = h_i/6 \\
\langle \alpha_{e_i}\,(\psi_x)_i|_{e_i},\, \psi_i|_{e_i}\rangle &= \int_0^{h_i} (1 - x/h_i)(-\alpha_{e_i}/h_i) \,\mathrm{d}x = -\alpha_{e_i}/2 \\
\langle \alpha_{e_i}(\psi_x)_{i+1}, \psi_i\rangle &= \int_0^{h_i} (1 - x/h_i)(\alpha_{e_i}/h_i) \,\mathrm{d}x = \alpha_{e_i}/2 \\
\langle \gamma_{e_i}\,(\psi_x)_i|_{e_i},\, (\psi_x)_i|_{e_i}\rangle &= \int_0^{h_i} (-\gamma_{e_i}/h_i)(-1/h_i) \,\mathrm{d}x = \gamma_{e_i}/h_i \\
\langle \gamma_{e_i}(\psi_x)_{i+1}, (\psi_x)_i\rangle &= \int_0^{h_i} (\gamma_{e_i}/h_i)(-1/h_i) \,\mathrm{d}x = -\gamma_{e_i}/h_i.
\end{aligned}$$

Finally, the TTGC-γ_L scheme can be represented in a compact and convenient elemental matrix form as shown in Eqs. (3.44)-(3.45):

$$\sum_e \mathbf{P}_e \mathbf{M}_e \delta\tilde{u}_e^n = -\Delta t \sum_e \alpha_e \mathbf{P}_e \mathbf{K}_e u_e^n + \beta \Delta t^2 \sum_e \mathbf{P}_e \mathbf{D}_e u_e^n \tag{3.44}$$

$$\sum_e \mathbf{P}_e \mathbf{M}_e \delta u_e^n = -\Delta t \sum_e \mathbf{P}_e \mathbf{K}_e \tilde{u}_e^n + \Delta t^2 \sum_e \gamma_e \mathbf{P}_e \mathbf{D}_e u_e^n, \tag{3.45}$$

where the mass $\mathbf{M}_e$, stiffness $\mathbf{K}_e$, dissipation $\mathbf{D}_e$ matrices, and the unknown nodal degree of freedom u_e are given by

$$\mathbf{M}_e = \frac{h_i}{3}\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}, \ \mathbf{K}_e = \frac{1}{2}\begin{bmatrix} -c & c \\ -c & c \end{bmatrix}, \mathbf{D}_e = \frac{1}{h_i}\begin{bmatrix} -c^2 & c^2 \\ c^2 & -c^2 \end{bmatrix} \text{ and } u_e = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}. \tag{3.46}$$

Assembly of the local mass, stiffness and damping is performed element-wise to obtain the global ones. The symbol $\mathbf{P}_e$ represents a permutation matrix that maps the local degrees of freedom into the correct row entries in the global matrix. Inspired from the edge-based FEM schemes of Löhner (2008); Luo et al. (1994) and control-volume FEM (CV-FEM) of Baliga and Patankar (1980) and that of Bochev, Peterson, and Gao (2013); Bochev, Peterson, and Perego (2015), a splitting of the integrals into a diagonal term $\mathbf{K}_e^{diag}$ and a dual control-volume edge-flux contribution $\mathbf{K}_e^{cv}$ to the stiffness matrix is introduced and yields the following:

$$\mathbf{K}_e = \begin{bmatrix} -c & 0 \\ 0 & c \end{bmatrix} - \frac{1}{2}\begin{bmatrix} -c & -c \\ c & c \end{bmatrix} = \mathbf{K}_e^{diag} - \frac{1}{2}\mathbf{K}_e^{cv} \tag{3.47}$$

In the split stiffness term, only the off-diagonal edge contribution is weighted using the element local α_e value, i.e., the first step of the TTGC-γ_L becomes

$$\begin{aligned} \sum_e \mathbf{P}_e \mathbf{M}_e \delta\tilde{u}_e^n = &- \alpha\Delta t \sum_e \mathbf{P}_e \mathbf{K}_e^{diag} u_e^n + \frac{\Delta t}{2} \sum_e \alpha_e \mathbf{P}_e \mathbf{K}_e^{cv} u_e^n \\ &+ \beta \Delta t^2 \sum_e \mathbf{P}_e \mathbf{D}_e u_e^n, \end{aligned} \tag{3.48}$$

where $\alpha = 2^{-1} - \gamma$ is a global value as used in the original TTGC-γ of Colin and Rudgyard (2000). Note that this split flux formulation is introduced only for the stiffness term in the first sub-step. The main highlight of the TTGC-γ_L scheme is that the stabilization comes directly from the problem of adjusting γ_e such that the numerical solution matches the physical problem

while maintaining the stability. For $\gamma_e > 0$, the off-diagonal flux residual is weighted by $\alpha_e = 2^{-1} - \gamma_e$ so it behaves like a local flux limiter for the convective term on the first step of TTGC-γ_L. For the second TTGC-γ_L step it provides additional damping (since $\gamma_e > 0$). By adjusting γ_e, unresolved waves can be dissipated while excessive dissipation can be reduced in regions that are fully resolvable by the numerical scheme.

3.2.3.1 Extension to inviscid Burgers' equation

The inviscid Burgers' equation (3.49) in non-conservation form with periodic boundary conditions is considered:

$$u_t + u\, u_x = 0. \tag{3.49}$$

Applying the weak formulation using Taylor-Galerkin expansion to Eq. (3.49), one obtains the following elemental matrix form of the TTGC-γ_L scheme for the inviscid Burgers' equation similarly to the linear convection as

$$\sum_e \mathbf{P}_e \mathbf{M}_e \delta\tilde{u}_e^n = -\Delta t \sum_e \alpha_e \mathbf{P}_e \mathbf{K}_e u_e^n + \beta \Delta t^2 \sum_e \mathbf{P}_e \mathbf{D}_e u_e^n \tag{3.50}$$

$$\sum_e \mathbf{P}_e \mathbf{M}_e \delta u_e^n = -\Delta t \sum_e \mathbf{P}_e \mathbf{K}_e \tilde{u}_e^n + \Delta t^2 \sum_e \gamma_e \mathbf{P}_e \mathbf{D}_e u_e^n, \tag{3.51}$$

where the mass $\mathbf{M}_e$, stiffness $\mathbf{K}_e$, dissipation $\mathbf{D}_e$ matrices and the unknown nodal degree of freedom u_e are given by

$$\mathbf{M}_e = \frac{h_i}{3}\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix},\ \mathbf{K}_e = \frac{1}{6}\begin{bmatrix} -2u_i & u_{i+1} + u_i \\ -(u_{i+1} + u_i) & 2u_{i+1} \end{bmatrix}, \tag{3.52}$$

$$\mathbf{D}_e = \frac{1}{3h_i}\begin{bmatrix} -u_i^2 & u_{i+1}^2 \\ u_i^2 & -u_{i+1}^2 \end{bmatrix} \text{ and } u_e = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}. \tag{3.53}$$

$\mathbf{P}_e$ is a permutation matrix that maps the local degrees of freedom into the correct row entries in the global matrix. By specifying the appropriate initial conditions for the state $u(x, t = 0)$ the solution can be marched forward in time. Similar to the convection discretization, the splitting of the stiffness is introduced in the first TTGC-γ_L step into a diagonal term $\mathbf{K}_e^{diag}$ and a dual control-volume edge-flux contribution $\mathbf{K}_e^{cv}$ for the stiffness matrix that gives

$$\mathbf{K}_e = \frac{1}{3}\begin{bmatrix} -u_i & 0 \\ 0 & u_{i+1} \end{bmatrix} - \frac{1}{6}\begin{bmatrix} -u_{i+1} & -u_{i+1} \\ u_i & u_i \end{bmatrix} = \mathbf{K}_e^{diag} - \frac{1}{2}\mathbf{K}_e^{cv}. \tag{3.54}$$

The splitting and overall form of the final stiffness terms in Eq. (3.47) and Eq. (3.54) are quite similar. The elemental α_e weights the off-diagonal term, whereas a constant global optimal value $\alpha = 2^{-1} - \gamma$ weights the main-diagonal terms (similar to Eq. (3.48)). In the next section, the analytical framework to optimize the local γ_e values that lead to a stable and accurate numerical scheme is presented.

3.2.4 Local Transfer function Analysis (LTA)

GSA gives the global behavior of the numerical solution vis-a-vis the exact one (dissipation, dispersion, energy propagation speed, etc.). However, the analysis of numerical schemes on non-uniform meshes has been restricted to a subset of meshes whose global structure is known (T. K. Sengupta, Rajpoot, Saurabh, et al., 2011; T. K. Sengupta and A. Sengupta, 2016; N. Sharma et al., 2017). This section considers the numerical convection of a wave packet (WP) with a central wavenumber k and phase speed $c = 1$ on a mesh that has a jump discontinuity in the mesh spacing at some location x_0 is analyzed. The mesh is equally distributed with a spacing of h_L and h_R to the left and right of x_0 with a jump in size $\tau = h_R h_L^{-1} = 2$ at x_0. Fig. 3.7 displays the numerical solution at various time steps for the TTGC-γ scheme with a constant $\gamma = 0.01$ and central frequency of $kh_L = 0.8$. The location of x_0 is indicated using the blue dot and the yellow dots are the mesh nodes. The frequency of this WP is chosen such that the wave is fully resolved (i.e., inside the GVP region) on the left and slightly above the GVP region on the right. This results in the generation of upstream propagating numerical waves that reflect off the junction x_0 toward the left boundary. Non-linear equations admit shock solutions that produce CFL (N_c) jumps across the shock even on regular meshes and result in a similar behavior. Due to the local nature of this problem, GSA in its current form cannot be used for this analysis.

This motivates the extension of GSA by a new approach which will be named *Local Transfer function Analysis (LTA)*. In LTA, the local numerical amplification G represents the transfer function (of the discrete form) that dictates the dynamics of all degrees of freedom. Therefore, one can conceive these local transfer functions (elemental or nodal degree of freedom) as transfer function blocks forming a circuit block diagram as shown in Fig. 3.8. The impedance or the local transfer function can be obtained as follows. The

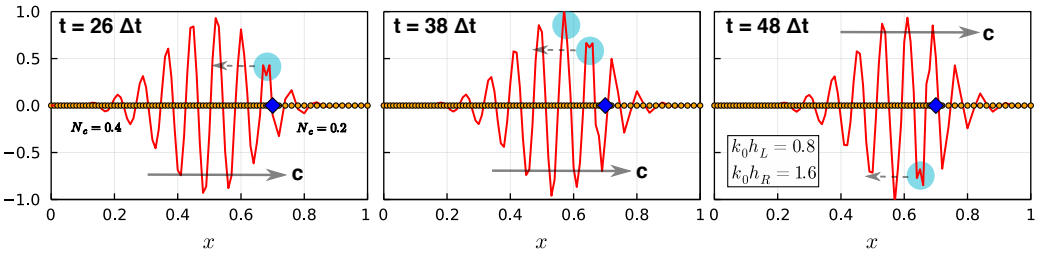


Figure 3.7: Convection of WP on mesh with discontinuity at junction (♦ is the junction) showing upstream reflected waves indicated by ⟵ ● and actual wave propagation direction indicated by ⟶ c; ● are the mesh nodes.

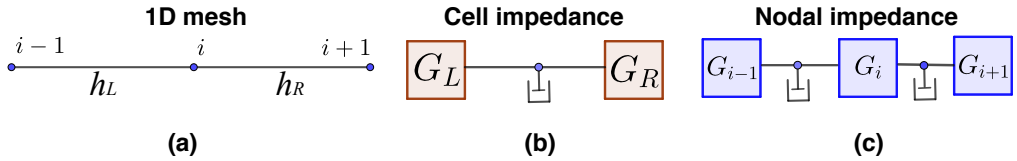


Figure 3.8: Cell and nodal impedance matching problem using local-transfer function; the dash-pot models externally added artificial dissipation.

residual at node i has contributions from the two elements $e_L[u_{i-1}, u_i]$ on the left and $e_R[u_i, u_{i+1}]$ on the right. Let τ be the (non-dimensional) ratio of the element sizes e_L and e_R (i.e., $h_R = \tau h_L$). For linear convection with constant velocity c, one can define the CFL ratios $N_c^R := \tau^{-1} h_L^{-1} c\,\Delta t = \tau^{-1} N_c^L$. In doing so, the mass, stiffness, and damping matrices for each side (left and right elements) can be obtained, which leads to the following form of TTGC:

$$\begin{aligned} &M^L\tilde{u}_i^n - M^L u_i^n + \alpha^L(c\Delta t)K^L u_i^n - \beta(c\Delta t)^2 D^L u_i^n = \\ &\quad -(M^R\tilde{u}_i^n - M^R u_i^n + \alpha^R(c\Delta t)K^R u_i^n - \beta(c\Delta t)^2 D^R u_i^n) \qquad (3.55) \\ &M^L u_i^{n+1} - M^L u_i^n + (c\Delta t)K^L\tilde{u}_i^n - \gamma^L(c\Delta t)^2 D^L u_i^n = \\ &\quad -(M^R u_i^{n+1} - M^R u_i^n + (c\Delta t)K^R\tilde{u}_i^n - \gamma^R(c\Delta t)^2 D^R u_i^n), \qquad (3.56) \end{aligned}$$

where

$$M^L u_i = \frac{h_L}{3}\left(\frac{u_{i-1}}{2} + u_i\right),\ M^R u_i = \frac{\tau h_L}{3}\left(\frac{u_{i+1}}{2} + u_i\right),$$
$$K^L u_i = -\frac{u_{i-1} + u_i}{2},\ K^R u_i = \frac{u_{i+1} + u_i}{2},$$
$$D^L u_i = \frac{u_{i-1} - u_i}{h_L} \text{ and } D^R u_i = \frac{u_{i+1} - u_i}{\tau h_L}. \tag{3.57}$$

Following the generalized function GSA approach, one can substitute the Fourier transform counterpart of the operators to this left/right TTGC form to obtain

$$\widehat{M}^L(\tilde{G} - 1) + \alpha^L N_c \widehat{K}^L - \beta N_c^2 \widehat{D}^L = -\left(\widehat{M}^R(\tilde{G} - 1) + \alpha^R N_c \widehat{K}^R - \beta N_c^2 \widehat{D}^R\right) \tag{3.58}$$

$$\widehat{M}^L(G - 1) + N_c \widehat{K}^L \tilde{G} - \gamma^L N_c^2 \widehat{D}^L = -\tau\left(\widehat{M}^R(G - 1) + N_c \widehat{K}^R \tilde{G} - \gamma^R N_c^2 \widehat{D}^R\right), \tag{3.59}$$

where

$$\widehat{M}^L = \frac{1}{3}\left(\frac{e^{-ip}}{2} + 1\right),\ \widehat{M}^R = \frac{\tau}{3}\left(\frac{e^{i\tau p}}{2} + 1\right),\ \widehat{K}^L = -\frac{e^{-ip} + 1}{2},\ \widehat{K}^R = \frac{e^{i\tau p} + 1}{2},$$
$$\widehat{D}^L = e^{-ip} - 1, \widehat{D}^R = \frac{e^{i\tau p} - 1}{\tau},\ p = kh_L \text{ and } N_c = N_c^L. \tag{3.60}$$

Rearranging into a new form,
TTGC step 1:

$$\tilde{G} = \frac{\widehat{M}^L - \alpha^L N_c \widehat{K}^L + \beta N_c^2 \widehat{D}^L}{\widehat{M}^L + \widehat{M}^R} + \frac{\widehat{M}^R - \alpha^R N_c \widehat{K}^R + \beta N_c^2 \widehat{D}^R}{\widehat{M}^L + \widehat{M}^R}$$
$$\tilde{G} = \tilde{G}_L + \tilde{G}_R \tag{3.61}$$

TTGC step 2:

$$G = \frac{\widehat{M}^L - N_c \widehat{K}^L \tilde{G} + \gamma^L N_c^2 \widehat{D}^L}{\widehat{M}^L + \widehat{M}^R} + \frac{\widehat{M}^R - N_c \widehat{K}^R \tilde{G} + \gamma^R N_c^2 \widehat{D}^R}{\widehat{M}^L + \widehat{M}^R}$$
$$G = G_L + G_R \tag{3.62}$$

Under the assumption of periodicity, Eqs. (3.61)-(3.62) help to idealize the finite element discretization into an electrical/acoustic circuit having local

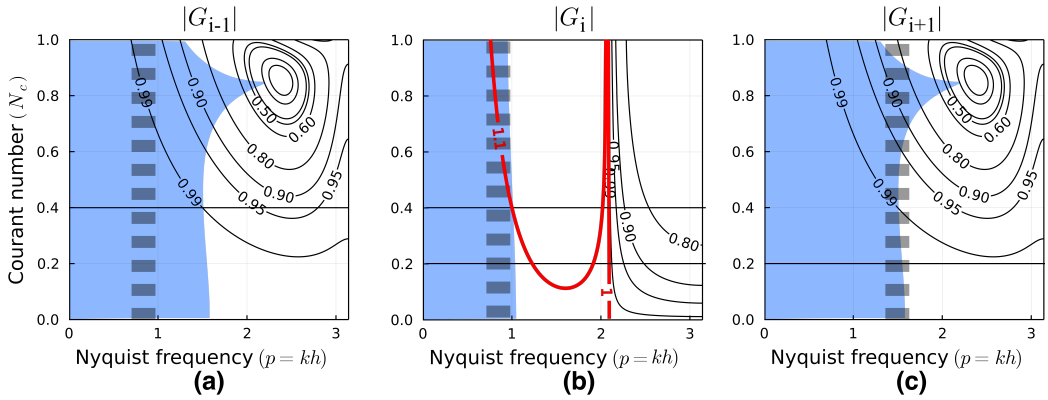


Figure 3.9: $|G|$ contours of LTA nodal impedance (a) $|G_{i-1}|$ (left node), (b) $|G_i|$ (junction node) and and (c) $|G_{i+1}|$ (right); solid shaded region is numerical phase speed within $\pm 5\%$ error of exact phase speed (DRP region), vertical striped region indicates the spectral spread of the WP in terms of local Nyquist frequency p; contours of local instability shown in red (thicker) contour lines $(1, 1.1)$.

impedance of G_L contributing from the left element and G_R from the right element for a given junction i. Therefore, for maximum transmission of the wave power, one has to satisfy the impedance matching condition, i.e., $G_L = G_R^\dagger$, where $(\cdot)^\dagger$ is the complex conjugate. For minimum reflection, the matching condition becomes $G_L = G_R$. Alternatively, one can also match the nodal impedances, i.e., $G_{i+1} = G_i$ or $G_{i+1} = G_i^\dagger$, where the nodal impedance $G_i := G_i|_L + G_i|_R$ is simply the sum of the left/right elemental impedances. Therefore, one has the freedom to change the local characteristics of the numerical scheme by impedance matching for (i) minimizing wave reflection or (ii) minimizing the dissipation of the wave energy at the junction. Note that perfect matching by tuning the numerical scheme might not be possible over the entire range of N_c and p. Additional damping is usually added (especially near discontinuities) in order to filter such problematic waves (idealized as dash-pots in LTA, as illustrated in Fig. 3.8).

3.2.4.1 Further analysis of the junction problem

The $|G|$ contours of the nodal impedance G_{i-1}, G_i and G_{i+1} for the propagation of the WP across a junction of mesh discontinuity (described in the beginning of Sec. 3.2.4 and summarized in Fig. 3.7) is plotted in Fig. 3.9(a-c).

An important feature to notice in Fig. 3.9 is the unstable region (shown in red contour lines) of the numerical amplification G_i at the junction. This implies that the wave gains energy while crossing the junction. It is clearly evident in the numerical solution at $t = 26\Delta t$ and $t = 38\Delta t$ plotted in Fig. 3.7 that the wave gains amplitude from a value $u \approx -0.5$ to a value $u \approx -0.75$. The reason this local instability does not develop into a global one is due to the stability of the impedance to the left and right of the junction and that the numerical phase speed at the junction is non-zero. As a result, the wave simply moves away from the junction before it can gain sufficient amplitude.

The spectral spread of the WP in terms of the local Nyquist frequency p is indicated by the vertical striped region. The values $N_c^{L/R} = \{0.4,\ 0.2\}$ of the local CFL numbers on the left and right side of the junction were deliberately chosen in a region of zero numerical dissipation, i.e., $|G| = 1$ for the entire spectrum of the WP. The nodal impedance G_{i-1} is within the DRP region ($\pm 5\%$) but, in the case of the nodal impedance G_{i+1}, a small fraction of the WP falls outside DRP. This leads to a mismatched impedance that produces reflections. Vichnevetsky and Bowles (1982) show that the semi-discrete form of the convection equation is also a consistent discretization for the wave equation. Hence, it admits both a downstream numerical solution called the p-wave (physical) and an upstream numerical solution called the q-wave (spurious). They also mention that at discontinuities of the computational domain, spurious q-waves are generated, for example, at computational boundaries. The junction problem can also be viewed as a discontinuity in the computational domain that produces spurious reflected q-waves propagating upstream. At the computational boundary, Vichnevetsky and Bowles (1982) employ time-Fourier transforms to derive the reflection ratio $\rho(\omega) = q(\omega)p^{-1}(\omega)$ in terms of the q-wave and p-wave solution to the semi-discrete form. Using the LTA, one can obtain the reflection ratio of the junction problem as shown below:

$$\rho_{i-\frac{1}{2}} = \frac{G_{i-1} - G_i}{G_{i-1} + G_i} \quad \text{and} \quad \rho_{i+\frac{1}{2}} = \frac{G_i - G_{i+1}}{G_i + G_{i+1}} \tag{3.63}$$

Unlike the problem at the computational boundary, the interior junction problem requires matching the two nodal impedance to the left and right of the junction to avoid spurious reflection. Note that the reflection formula in Eq. (3.63) is quite general and can be applied at the computational boundary to obtain the boundary reflection ratio similar to Vichnevetsky and Bowles (1982). $|\rho|$ gives the magnitude of the reflected wave in terms of the incident

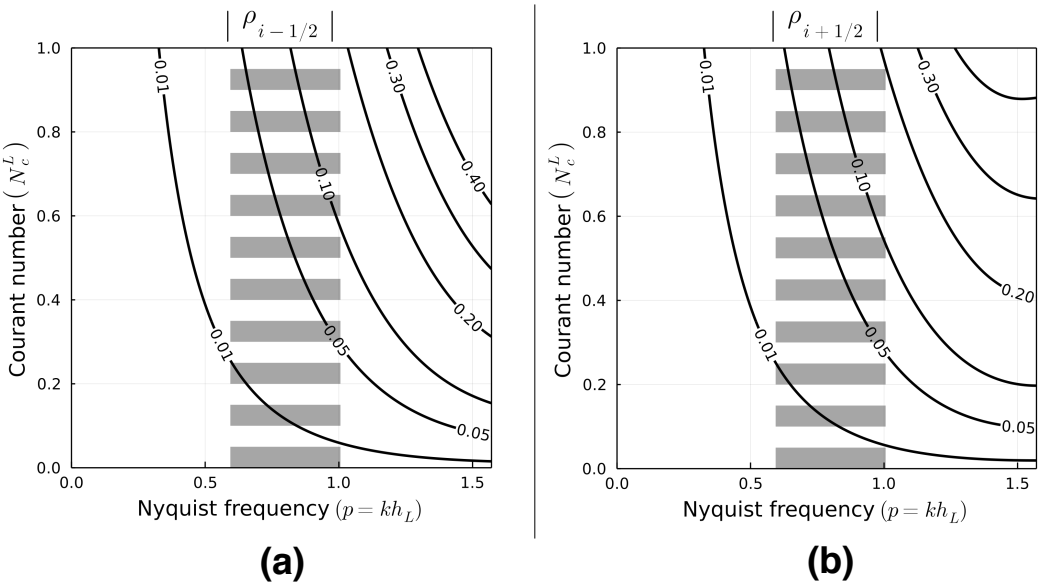


Figure 3.10: Contour of reflection ratio magnitude ($|\rho|$) due to nodal impedance mismatch between (a) $(i-1)$-(i) and (b) (i)-$(i+1)$ for the junction problem for varying CFL ($0 < N_c < 1$) and frequency ($0 < p < \pi/2$).

wave at the junction between the two nodal impedance and is plotted in Fig. 3.10 for the range of CFL number $0 < N_c < 1$ and frequency $0 < p < \pi/2$. For the given spectral spread of the WP the LTA predicts a reflected q-wave with amplitude between 1-5% of the incident wave at the junction, which is confirmed by the numerical simulation (see Fig. 3.7). Note that q-waves (reflections) are present almost for the entire Nyquist spectrum of the incident WP at the junction albeit the amplitudes are very low for $kh_L < 0.5$. For an irregular mesh, several such junctions make the reflected and incident wave dynamics highly complex.

3.2.4.2 LTA and the spatial spectrogram

It is possible to leverage a spatial analogue of the time-frequency spectral analysis, i.e., space-wavenumber (SW) analysis along with LTA to yield some interesting results. The idea is to construct a spectrogram of the function in the spatial domain and analyze the individual spectrogram snapshots using LTA to infer global error behavior. To extract this localized SW, spatially compact Fourier transform (SFT) can be used. SFT is a linear space-

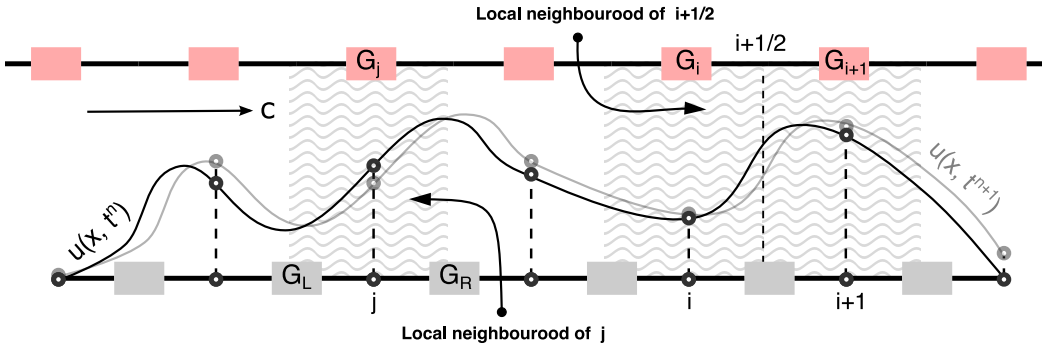


Figure 3.11: Space-wavenumber idealization and local neighborhood of nodal and elemental impedance to the continuous initial condition $u(x, t^n)$ leading to the solution at the next step $u(x, t^{n+1})$.

wavenumber (SW) transform that correlates the spatial signal with a family of waveform that are well concentrated in spatial and wavenumber domain. The localized SW kernel is chosen such that it is centered around a spatial location ξ yielding the following forward SFT, $\widehat{U}(x,k)$ of a function $u(x)$:

$$\widehat{U}(x,k) = \int u(\xi) g(\xi - x) e^{-ikx} d\xi. \tag{3.64}$$

Here $g(\xi, x)$ is the symmetric, normalized (i.e., $\|g\| = 1$) and real valued window function that is compactly supported in space. One can recover the inverse SFT as shown in Eq. (3.65):

$$u(x) = \iint \widehat{U}(x,k) g(x - \xi) e^{ikx} dk d\xi \tag{3.65}$$

and the Parseval's identity for the SFT as shown in Eq. (3.66), respectively:

$$\int |u(x)|^2 dx = \iint |\widehat{U}(x,k)|^2 dx dk. \tag{3.66}$$

Let $u(x,t)$ be a spatio-temporally evolving function which is sampled in time at two time instances t^n and t^{n+1}. Fig. 3.11 shows the two snapshots of the solution $u(x,t^n)$ and $u(x,t^{n+1})$ and an underlying spatial discretization for numerically evolving the solution. One can now make use of the generalized product with the delta function to obtain the numerical amplification in the

SW domain (similar to the approach shown in Sec. 3.2.1):

$$\begin{aligned}\widehat{U}_j(k,t^n) &= \int \widehat{U}(x,k,t^n)\delta(x-x_j)dx \\ &= \int \left[\int \left(u(\xi,t^n)g(\xi-x)e^{-ikx}\right)\delta(x-x_j)dx\right]d\xi \\ &= \int u(\xi,t^n)g(\xi-x_j)e^{-ikx_j}d\xi, \end{aligned} \tag{3.67}$$

and the generalized product of the delta function with the inverse SFT yields

$$\begin{aligned}u_j(t^n) &= \int \left[\iint \widehat{U}(x,k,t^n)g(x-\xi)e^{ikx}dkd\xi\right]\delta(x-x_j)dx \\ &= \iint \widehat{U}_j(k,t^n)g(x_j-\xi)e^{ikx_j}dkd\xi. \end{aligned} \tag{3.68}$$

The window function $g(x_j-\xi)$ must have compact support and it is useful to make its support equal to the Nyquist limit of the underlying discretization (shaded region in Fig. 3.11).

3.2.4.3 Error norm and maximum power impedance matching

Vichnevetsky and Bowles (1982) applied the Parseval's identity to connect the $l2$ norm of solution error in the physical space to the spectral domain. Motivated by his work, the $l2$ solution error will be expressed in terms of the local impedance model used in LTA. It is considered the squared $l2$ norm of the error between the numerical and exact solution at time step $n+1$ assuming a known continuous initial solution at time step n. The error is denoted by $L2(u^{n+1})$ and defined below:

$$\begin{aligned}L2(u^{n+1}) &:= \int |\epsilon(x,t^{n+1})|^2dx = \int |u(x,t^{n+1})-\bar{u}(x,t^{n+1})|^2dx \\ &\approx \sum_j h_{j+1/2}|\epsilon_j^{n+1}|^2 = \sum_j h_{j+1/2}|u_j^{n+1}-\bar{u}_j^{n+1}|^2. \end{aligned} \tag{3.69}$$

The index j denotes the nodal position as illustrated in Fig. 3.11. In LTA, the nodal impedance G_j at node j is idealized as the sum of the elemental impedances to the left (G_L) and right (G_R) of the node i.e., $G_j = G_L + G_R$. Now the error $\hat{\epsilon}^{n+1}$ in the Fourier space is given by

$$\hat{\epsilon}^{n+1} = \widehat{U}^{n+1} - \hat{\widehat{U}}^{n+1} \approx \sum_j \hat{\epsilon}_j^{n+1} = \sum_j \widehat{U}_j^n(G_j - G_{\text{phy}}). \tag{3.70}$$

By applying Parseval's identity to the error ϵ one can connect the $l2$ norm of error in the physical and spectral space as

$$\begin{aligned} L2(u^{n+1}) \approx \sum_j h_{j+1/2}|\epsilon_j^{n+1}|^2 dx &= \sum_j \int |\hat{\epsilon}_j^{n+1}|^2 dk \\ &= \sum_j \int |\widehat{U}_j^n|^2 (G_j - G_{\text{phy}})(G_j^\dagger - G_{\text{phy}}^\dagger) dk. \end{aligned} \tag{3.71}$$

By using the polar form of the complex impedance contributed by the left and right finite-element of node j denoted as $G_L = |G_L|e^{i\phi_L}$ and $G_R = |G_R|e^{i\phi_R}$ then the nodal impedance can be expressed using the polar form as $|G_j|e^{i\phi_j} = |G_L|e^{i\phi_L} + |G_R|e^{i\phi_R}$. Note that the physical impedance for the convection problem is $G_{\text{phy}} = |G_{\text{phy}}|e^{i\phi_{phy}} = e^{-ikc\Delta t}$. Now the $l2$ norm of solution error in Eq. (3.71) can be expressed using this polar form of nodal impedance as

$$\begin{aligned} L2(u^{n+1}) \approx &\sum_j \int |\widehat{U}_j^n|^2 \left(|G_j|^2 + |G_{\text{phy}}|^2\right) dk \\ &- 2\sum_j \int |\widehat{U}_j^n|^2 |G_j||G_{\text{phy}}| \cos\left(\phi_j - \phi_{phy}\right) dk \end{aligned} \tag{3.72}$$

and using the polar form of elemental impedance, the $l2$ norm of solution error can be expressed using the form below:

$$\begin{aligned} L2(u^{n+1}) \approx &\sum_j \int |\widehat{U}_j^n|^2 \left(|G_L|^2 + |G_R|^2 + |G_{\text{phy}}|^2\right) dk \\ &+ 2\sum_j \int |\widehat{U}_j^n|^2 |G_L||G_R| \cos\left(\phi_L - \phi_R\right) dk \\ &- 2\sum_j \int |\widehat{U}_j^n|^2 |G_L||G_{\text{phy}}| \cos\left(\phi_L - \phi_{phy}\right) dk \\ &- 2\sum_j \int |\widehat{U}_j^n|^2 |G_R||G_{\text{phy}}| \cos\left(\phi_R - \phi_{phy}\right) dk. \end{aligned} \tag{3.73}$$

To minimize the error between the numerical and the exact solution, one should minimize the $l2$ error objective $L2(u^{n+1})$. A minimum objective is reached when the minimum satisfies the conditions, $|G_j| \approx |G_{\text{phy}}|$ and $\phi_j = \phi_{phy}$, i.e., the numerical and physical transfer functions match. This

naturally means that the wave power is maximized, since $|G_j| \approx |G_{\text{phy}}|$ ensures minimum dissipation. Comparing Eq. (3.72) and Eq. (3.73) with the individual impedances of the left and right elements in Eq. (3.60) it is trivial to show that the elemental transfer function has tendency to have opposing elemental phase angles $\phi_L \approx -\phi_R$. Therefore, the minimization of $L2(u^{n+1})$ is a good approximation to the maximum power impedance matching $G_L \approx G_R^{\dagger}$. For the minimization of $L2(u^{n+1})$ to perform maximum power matching it is critical to impose the stability constraint on $|G_L|, |G_R|$, i.e., $|G_L|^2 + |G_R|^2 \leq 1$ and bound them close to the physical impedance value. Otherwise, the minimum solution can lead to an unstable scheme.

3.2.4.4 Total-variation diminishing (TVD) condition

Harten (1983) reasoned that the linear stability of a scheme does not imply its convergence to discontinuous solutions of nonlinear problems. He introduced a stronger stability criterion, namely the uniform boundedness of the total variation of the numerical solution. This condition implied convergence to the weak solution. The 1-norm total-variation of a discrete quantity u is given by $TV(u) = \sum_{i=1}^{N} |u_{i+1} - u_i|$. Harten defines a numerical scheme E_h to be total variation diminishing (TVD) if

$$TV(E_h \cdot u) \leq TV(u) \tag{3.74}$$

and showed that numerical schemes with TVD property can successfully capture shocks. Motivated by the work of Harten, a connection between the TVD and the impedance matching condition is sought. It is considered the 2-norm total variation (TV2) defined as

$$\begin{aligned} TV2(u^n) = \frac{1}{2}\int \left|\frac{du}{dx}\right|^2 dx &= \frac{1}{2}\int \lim_{h\to 0}\left|\frac{u^n(x+h) - u^n(x)}{h}\right|^2 dx \\ &\approx \frac{1}{2}\sum_i h_{i+\frac{1}{2}}|\epsilon_h^n|^2 = \frac{1}{2}\sum_i \frac{|u_{i+1}^n - u_i^n|^2}{h_{i+\frac{1}{2}}}. \end{aligned} \tag{3.75}$$

Let $\hat{\epsilon}_h^n$ be the error in the spectral space for the element formed by the nodes $i+1$ and i. Using the fact that translation in x by h becomes simple multiplication by $e^{ikh_{i+1/2}}$ in the spectral space, one obtains

$$\hat{\epsilon}_{h_{i+1/2}}^n = \widehat{U}_{i+1}^n e^{ikh_{i+1/2}} - \widehat{U}_i^n = \widehat{U}_i^n \left(e^{ikh_{i+1/2}} - 1\right) \tag{3.76}$$

Similarly, one can express $\hat{\epsilon}_h^{n+1}$ as

$$\hat{\epsilon}_{h_{i+1/2}}^{n+1} = \widehat{U}_{i+1}^{n+1} e^{ikh_{i+1/2}} - \widehat{U}_i^{n+1} = \widehat{U}_i^n \left(G_{i+1} e^{ikh_{i+1/2}} - G_i \right) \tag{3.77}$$

Using Parseval's identity, one can show that the TV2 from Eq. (3.75) in the spectral space is given by

$$\begin{aligned} TV2(u^n) \approx \frac{1}{2} \sum_i \int |\hat{\epsilon}_{h_{i+1/2}}^n|^2 dk &= \frac{1}{2} \sum_i \int |\widehat{U}_i^n|^2 (e^{ikh_{i+1/2}} - 1)(e^{-ikh_{i+1/2}} - 1) dk \\ &= \sum_i |u_i^n|^2 - \sum_i \int |\widehat{U}_i^n|^2 \cos(kh_{i+\frac{1}{2}}) dk. \end{aligned} \tag{3.78}$$

Similarly, one can obtain the spectral equivalent of $TV2(u^{n+1})$ as

$$\begin{aligned} TV2(u^{n+1}) \approx & \sum_i |u_i^{n+1}|^2 \\ & - \frac{1}{2} \sum_i \int |\widehat{U}_i^n|^2 \left(G_{i+1} G_i^\dagger e^{ikh_{i+1/2}} + G_{i+1}^\dagger G_i e^{-ikh_{i+1/2}} \right) dk \end{aligned} \tag{3.79}$$

For the solution at time $n+1$ to be bounded, the $TV2$ norm of the solution at $n+1$ has to be bounded by the exact solution $TV2(u_{\text{exact}}^{n+1})$, i.e., $TV2(u^{n+1}) \leq TV2(u_{\text{exact}}^{n+1})$. Since the physical amplification for the convection equation is $|G_{\text{phy}}| = 1$, the condition simplifies to

$$TV2(u^{n+1}) \leq TV2(u^n). \tag{3.80}$$

For a stable numerical scheme, one can require $|G_{i+1}|$, $|G_i| \leq 1$, which will automatically satisfy $TV2$ condition if one imposes the stronger element-wise condition:

$$\int |\widehat{U}_i^n|^2 \left(G_{i+1} G_i^\dagger e^{ikh_{i+1/2}} + G_{i+1}^\dagger G_i e^{-ikh_{i+1/2}} - 2\cos(kh_{i+1/2}) \right) dk \leq 0. \tag{3.81}$$

Firstly, a striking similarity of the $TV2$ condition in Eq. (3.80) with the one derived by Harten can be found. Secondly, Eq. (3.81) shows that the $TV2$ equality is valid strictly element-wise if the impedance satisfies the matching condition $G_{i+1} = G_i$. This brings out the one-to-one relationship between the TVD and the impedance matching condition. Satisfying the TVD condition is critical to suppress the spurious waves generated near shock because

it leads to minimum reflection impedance matching (not maximum power transmission) as shown by the local transfer function analysis. To achieve minimum reflection, TVD schemes typically sacrifice the conservation of total energy in the wave (power loss), which is what one finds in practice. The 2-norm $TV2$ is considered (rather than the 1-norm TV) because it directly relates to the total energy of the wave, which is conserved in pure convection. One arrives at the same expression by applying the 1-norm TVD condition of Harten point-wise and squaring to obtain the point-wise 2-norm TVD.

3.2.5 LTA of the inviscid Burgers' equation

From the LTA perspective, both linear and non-linear convection are equivalent because linearization or averaging (Griewank and El-Danaf, 2009; Ramadan and El-Danaf, 2005; Ucar et al., 2017) is used to march the solution from one time step to another. This attribute motivates extending LTA to the non-linear inviscid Burgers' equation shown in Sec. 3.2.3.1. To derive the LTA for non-linear Burgers' equation, it is made the key assumption that the linearization is performed with respect to the time level n (marching solution to $n+1$) and keeps the same throughout the sub-step in time leading to $n+1$. For $N_c < 1$, this is a good approximation, since the changes to the linearization at sub-steps are minimal. A sub-stepwise impedance matching can also be performed but this approach is not shown in the current work. Once the linearization is complete, the convection velocity $c_i = u_i^n$ should be maintained a constant from n to $n+1$ as per the assumption. Let $\tau_{cR}|_i := c_{i+1}/c_i$ and $\tau_{cL}|_i := c_{i-1}/c_i$, be two ratios which measure the deviation in velocity between the nodes $i+1,\ i$ and $i-1,\ i$, respectively. Applying the Fourier-Laplace transform to this linearized system and rearranging, one obtains a two step impedance formula that looks exactly the same as the convection case (shown in Eq. (3.61) and Eq. (3.62)); but the stiffness, damping and CFL number N_c are modified as

$$\widehat{K}^L = -\frac{\tau_{cL}(e^{-ip}+1)}{6},\ \widehat{K}^R = \frac{\tau_{cR}(e^{\tau ip}+1)}{6},$$
$$\widehat{D}^L = \frac{\tau_{cL}^2 e^{-ip}-1}{3}, \widehat{D}^R = \frac{\tau_{cR}^2 e^{i\tau p}-1}{3\tau} \text{ and } N_c = h_L^{-1} c_i \Delta t. \tag{3.82}$$

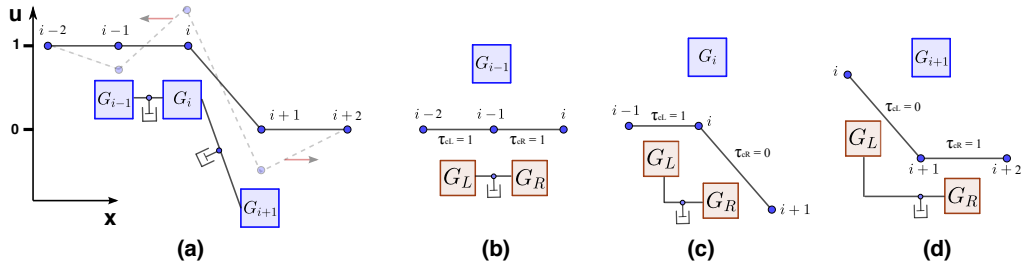


Figure 3.12: LTA model at a Burgers shock (a) initial condition (solid line) and appearance of solution over-/undershoots (dotted line) created by the reflected waves from impedance mismatch and the individual nodal impedance (b-d) from elemental ones.

3.2.5.1 Analysis at the Burgers' shock

A simplified scenario of a Burgers' initial solution containing a shock spread over three nodes $i+1$, i and $i-1$ is considered. The value of u transitions from $u = 1$ (before shock) to $u = 0$ (after shock). Besides, mesh distribution of nodes is assumed to be uniform, i.e., $\tau = 1$. A schematic of the problem is illustrated in Fig. 3.12(a). The LTA dissipation ($|G|$) and phase angle (ϕ) for the three nodal impedances (linearized) shown in Fig. 3.12(b-d) are plotted in Fig. 3.13. Behind the shock, large variations in phase angle in comparison to aft of the shock take place. This indicates that heavier reflections should occur behind the shock compared to the aft of the shock. Fig. 3.14 shows the numerical solution to a moving shock problem with the given initial conditions for the first two steps using TTGC scheme and one can observe exactly the same behavior as predicted by LTA. Book (2005) made similar observations that the phase angle variation was the primary cause for the appearance of over/undershoot at shocks.

$|G|_i$ contours at the junction i reveals a locally linear unstable region ($|G| > 1$) indicated by the grey area in Fig. 3.13. Therefore, the spurious reflections will grow and sustain behind the shock. Comparing the numerical results at $t = \Delta t$ and $t = 2\Delta t$ (see Fig. 3.14), one finds the same behavior as predicted by the LTA. For a standing shock problem, the reflected numerical waves have phase speeds close to zero, resulting in piling up of reflected waves at each time step due to this local instability, which continuously feeds energy leading to numerical blowup. Here, simply adjusting γ is not sufficient to remove this local instability and demands additional external

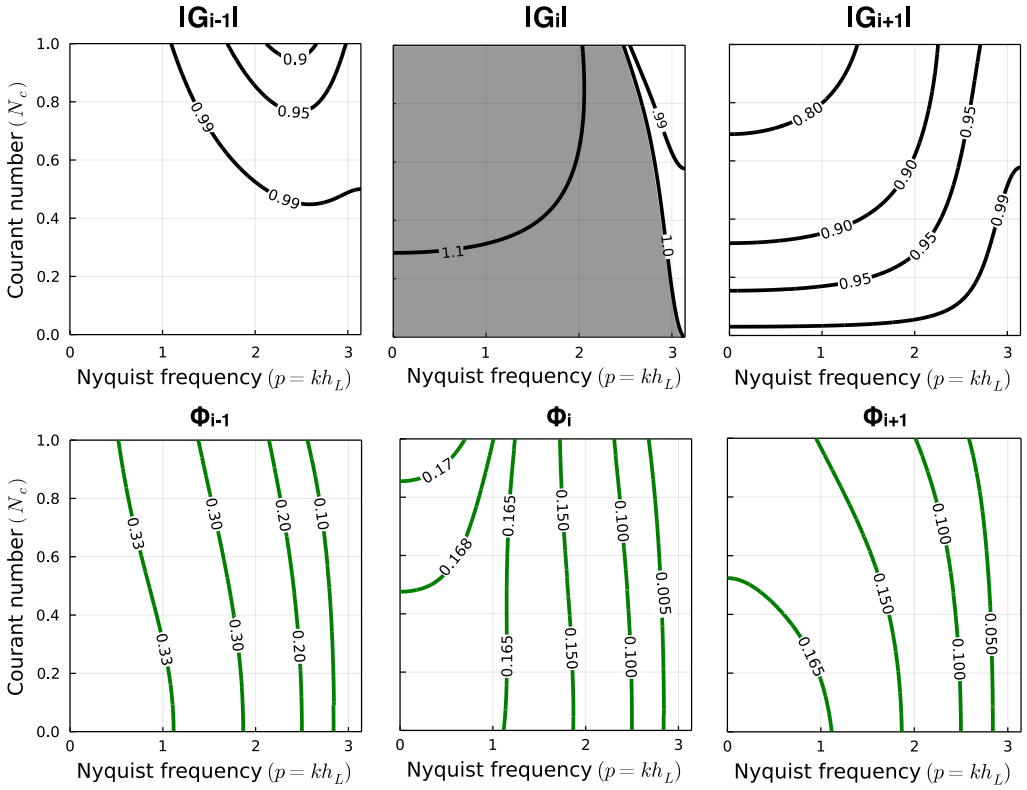


Figure 3.13: Dissipation $|G|$ and phase angle ϕ of the local nodal transfer function G_{i+1}, G_i, and G_{i-1}; grey shaded region are numerically unstable ($|G| > 1$). Note that the phase angle mismatch between G_{i-1} and G_i is much more severe than G_i and G_{i+1}.

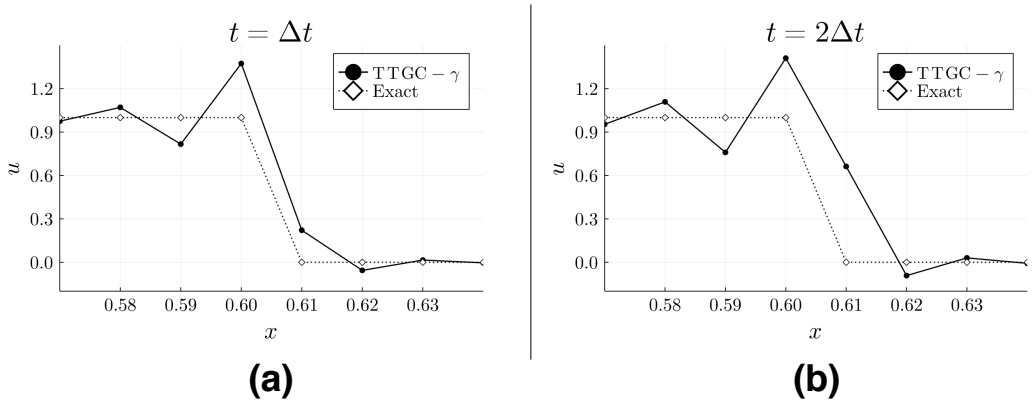


Figure 3.14: Time evolution of numerical solution zoomed near the Burgers' shock for the first (left) and second (right) time step from the initial condition.

dissipation. Note that by adding additional dissipation, one solves a modified inviscid Burgers' equation (Eq. (3.83)) that spreads the shock over multiple cells improving stability (reduced τ_c) and high kh waves are damped which reduces the spurious reflected waves (impedance mismatch).

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 && \text{(in smooth regions)} \\ &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 && \text{(near shocks and high gradients)} \end{aligned} \tag{3.83}$$

To test the improved stability using artificial dissipation, two variants are considered; where an explicit dissipation term is added (i) at the end of every time step and (ii) at the end of every sub-step of the TTGC-γ scheme. The elemental form of the explicit dissipation term $\mathbf{D}_e^*$ and its left/right local transfer function $\hat{D}^{*,L/R}$ are given by

$$\mathbf{D}_e^* = \frac{\nu_e \Delta t}{h_i^2}\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \hat{D}^{*,L} = \mathrm{Pe}^L(e^{-ip} - 1) \quad \text{and} \quad \hat{D}^{*,R} = \mathrm{Pe}^R(e^{ip} - 1), \tag{3.84}$$

where ν_e is the element local value of the coefficient of the artificial viscosity term and $\mathrm{Pe}^{L/R} = \frac{\nu_{L/R}\Delta t}{h^2}$ are the non-dimensional Péclet number (S. Sengupta et al., 2022) of the left and right elements, respectively. Since TTGC-γ has two sub-steps, the effective viscosity in variant (ii) will be twice the amount added in the variant (i) case. $|G|_i$ contour of the nodal LTA at the

shock location i for the variants (i) and (ii) is shown in Fig. 3.15(a)-(b), respectively. A constant Péclet number value of $\mathrm{Pe}^{L/R} = 0.5$ is used for variant (i) and $\mathrm{Pe}^{L/R} = 0.25$ at each sub-step for the variant (ii) case. For the results in the plots, a constant γ value of $\gamma = 0.01$ was maintained. Firstly, it can be observed that variant (i) is unstable both in the low and high wavenumber regimes and variant (ii) is unstable only in the low wavenumber regime. In

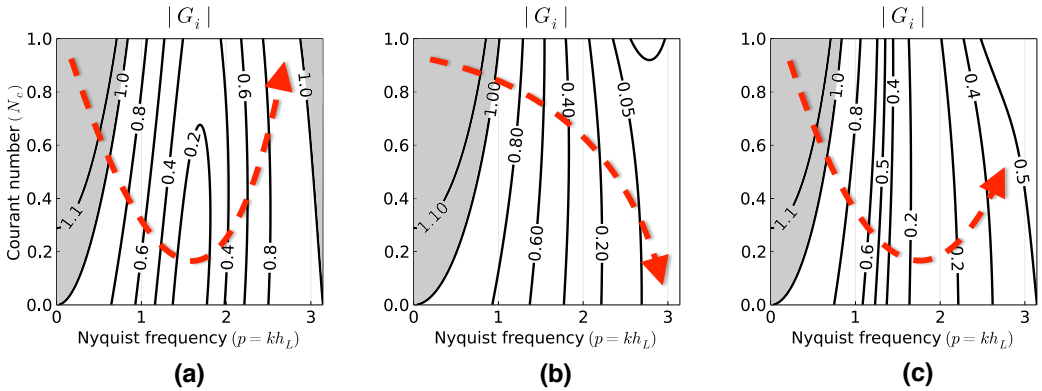


Figure 3.15: $|G|$ contours of local transfer function at shock location i for the TTGC-γ scheme for Burgers' equation (a) for variant (i) with Pe = 0.5, (b) variant (ii) with Pe = 0.25 and (c) variant (ii) with Pe = 0.375.

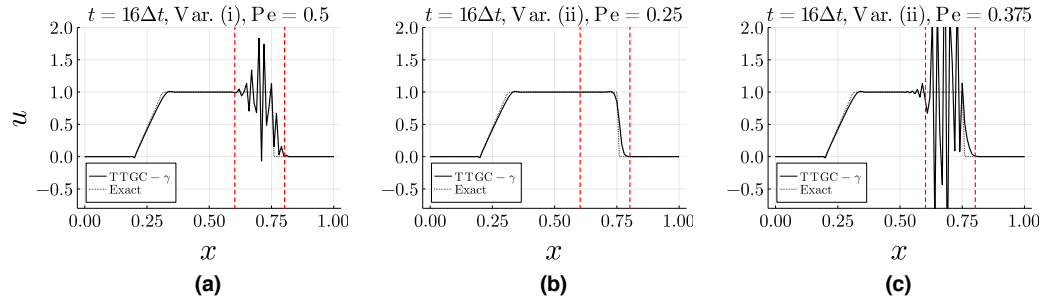


Figure 3.16: Time step $t = 16\Delta t$ for the evolution of a square wave initial condition under Burgers' equation, where Δt is the simulation time step. (a) for variant (i) with Pe = 0.5, (b) variant (ii) with Pe = 0.25 and (c) variant (ii) with Pe = 0.375. Red lines delimit region near shock front where artificial viscosity is added.

the numerical simulations, as shown in Fig. 3.16, variant (i) produced unstable numerical results. On the contrary, even in the presence of the unstable

region at low wavenumber, it was found that variant (ii) was stable even for a stationary shock problem. The numerical stability of variant (ii) becomes clear when one considers the spectral energy cascade of the viscous Burgers' equation shown below (Reid, 1956):

$$\frac{d\left[\widehat{U}(k)\widehat{U}^{\dagger}(k)\right]}{dt} = \frac{d\widehat{E}(k)}{dt} = T(k) - D(k), \tag{3.85}$$

where $\widehat{U}(k)$ is the complex representation of the Burgers' state u in the wavenumber space, $(\cdot)^{\dagger}$ is the complex conjugate, $D(k) = 2\nu k^2 \widehat{E}(k)$ represents the rate at which energy is dissipated by wavenumber k, and $T(k)$ is the net rate at which energy is input into the wavenumber k. For problems in 1D, $T(k)$ can be represented using the following form:

$$T(k) = \sum_{k^*} T(k, k^*) = -ik\Im\left[\widehat{U}^{\dagger}(k)\widehat{U}(k^*)\widehat{U}(k-k^*)\right] \tag{3.86}$$

Note that the term $T(k, k^*)$ is the energy transfer into the wavenumber k due to its interaction with an arbitrary wavenumber k^*. A positive value of $T(k, k^*)$ indicates the interactions add energy into k from k^* and a negative value indicates the removal of energy from k into k^*. For the Burgers' equation, $T(k, k^*)$ is always (i) negative for $k^* > k$, (ii) positive for $k^* < k$ and (iii) zero for $k^* = k$ (Girimaji and Zhou, 1995). This means a given scale with wavenumber k always draws energy from larger scales ($k^* < k$) and continuously loses energy to smaller scales ($k^* > k$). Therefore, any numerical instability in scales at the lower wavenumber k (i.e., $|G(kh)| > 1$) will subsequently be transferred to the scales in the higher wavenumber, which the numerical scheme can dissipate (via the term $D(k)$). As a result, the energy budget in Eq. (3.85) remains bounded thus providing the stability in variant (ii). For certain critical values of ν, Reid (1956) shows that a local or global back-scatter (i.e., reversal of the energy cascade) can occur. By preventing the usual cascade of energy, back-scatter prevents the transfer of energy from lower wavenumbers (k) to higher ones ($k^* > k$); if $|G(kh)| > 1$ and $D(k)$ is insufficient then this will result in a numerical blowup. In numerical experiments, it was found that for $Pe > 0.3$ the variant (ii) was unstable (see Fig. 3.16(c)) showing such a behavior. The instability at lower kh also makes it impossible to remove over-/undershoots for all shock strengths by simply adding dissipation. Nevertheless, one can manage the instability by bounding $E(k)$ using the dissipation term $D(k)$. Vajjala et al. (2020) applied GSA

to the convection-diffusion equation and report a similar route to instability, which they attribute to numerical focusing; a sudden build up of energy in a focused region of the domain. A similar behavior for the Burgers' shock was observed: the build up of errors in lower kh triggers a focused growth of instability near the shock at higher kh. The energy cascade equation is able to clearly explain this numerical focusing or transfer of energy from the unstable low wavenumber region to the higher ones leading to this energy build up.

Using LTA, quantitative and qualitative analyses of numerical schemes on irregular meshes and for non-linear equations were performed. Additionally, LTA suggests that one can optimize numerical schemes either by maximizing power transmission (Sec. 3.2.4.3) or by suppressing spurious waves (Sec. 3.2.4.4). The next chapter is dedicated to exploring the optimization framework resulting from LTA, which can be used to design stable and accurate data-driven numerical schemes.

Chapter 4

Optimizing TTGC-γ_L and Machine-Learned TTGC (ML-TTGC)

In the previous chapter, a novel spectral analysis for numerical schemes called the Local Transfer function Analysis (LTA) was introduced. Using LTA, one can formulate a multi-objective optimization framework for designing optimal numerical schemes with stability constraints. This chapter is dedicated to defining and exploring this framework. Firstly, an optimization procedure is used to tune the parameter γ of the original TTGC scheme by introducing element local (optimal) γ values resulting in the new TTGC-γ_L scheme (shown in Sec. 3.2.3). However, this procedure is computationally expensive, and its application at every time step is inefficient in practice. A message-passing Graph Neural Network (GNN) model (shown in Sec. 1.1.3) is used as a surrogate model to predict the element local optimal values of γ in the TTGC-γ_L. The resulting numerical method is named ML-TTGC. It is used to evolve in time 1D problems governed by the Convection/Burgers' equations. ML-TTGC can diminish dissipation and dispersion errors otherwise produced by the original TTGC scheme. This shows a promising avenue that replaces human-defined rules for numerical methods (often derived for linear systems and on regular grids) with machine-learned discretizations (that will handle non-linear systems and irregular grids).

4.1 The optimization problem resulting from LTA

The LTA results derived in Chapter 3 are used to construct a constrained multi-objective optimization framework for numerical schemes. To demonstrate this framework, the considered problem is that of estimating the optimal γ parameter of the TTGC-γ_L numerical scheme (defined in Sec. 3.2.3). The optimization problem is defined as a weighted sum of the two objectives, (i) maximum power output and (ii) minimum reflection as

$$\begin{aligned} \min_{\gamma} \quad & \theta(G_L - G_R^{\dagger}) + (1-\theta)(G_{i+1} - G_i) \\ s.t. \quad & |G_L|^2 + |G_R|^2 \leq |G_{phy}|^2 \text{ (stability condition)}, \end{aligned} \tag{4.1}$$

where $0 \leq \theta \leq 1$ is the scalarizing weight and $\theta = 1$ implies maximum power while $\theta = 0$ implies minimum reflection. $G_{L/R}$ and $G_{i/i+1}$ are the local left-/right elemental and nodal impedance in 1D (or an edge in 2D). While LTA enabled to clearly define the design parameters, the correct objective function and constraints for designing numerical schemes, practical enforcement of the impedance matching condition to adjust the dispersion and dissipation characteristics is inconvenient in the spectral space. Since the numerical solution is calculated in the time domain and the objective function is defined in the spectral domain it is convenient to enforce these spectral conditions indirectly in the time domain to optimize the local parameter γ in TTGC-γ_L. Using the results from Sec. 3.2.4.3-3.2.4.4, one can show that the equivalent time domain optimization problem is

$$\begin{aligned} \min_{\gamma} \; & \theta L2(u^{n+1}) + (1-\theta)\left[\delta TV(u^{n+1}) + |\delta TV(u^{n+1})|\right] \\ & |G_L|^2 + |G_R|^2 \leq |G_{phy}|^2 \text{ (stability condition)}, \end{aligned} \tag{4.2}$$

where u^{n+1} is the numerical solution at time $n+1$ obtained from a previous time u^n and $\bar{u}^{n+1}$ is a reference or exact solution at time $n+1$. The term $\delta TV(u^{n+1})$ is the difference between the total-variation of the numerical and exact or reference solution, i.e., $\delta TV(u^{n+1}) = TV2(u^{n+1}) - TV2(\bar{u}^{n+1})$. Since TVD enforcement is of interest (for minimum reflection impedance matching), i.e., $TV2(u^{n+1}) \leq TV2(\bar{u}^{n+1})$, only the numerical variations that are higher than the exact or reference solution in the objective are considered. Note that the quantities $L2$ and $TV2$ defined in Eq. (3.75) and Eq. (3.69)

have different mesh scaling of $\mathcal{O}(h)$ and $\frac{1}{\mathcal{O}(h)}$, respectively. Hence, for all the optimization studies the $L2$ and $TV2$ errors are non-dimensionalized using the average element size h_{avg} to bring them to the same order of magnitude, i.e., $(h_{avg})^{-1}L2$ and $h_{avg}\ TV2$, respectively. The optimization problem should be solved at every time step n to estimate the optimal γ that gives the best numerical solution u^{n+1} without violating the constraints. Albright and Shashkov (2020) also found that a balance between L1-norm of the error and 1-norm of total variation (i.e., TV measure) is beneficial in their data-driven optimization of artificial viscosity parameters. LTA corroborates theoretically that a similar competing multi-objective measure of the numerical error using $L2$ and TV is inevitable.

In this work, known exact solutions to the convection problem are used as reference values to obtain the optimal γ. Given the scalar objective in Eq. (4.2) and as many γ as the number of finite-elements, the problem is best suited for adjoint-based gradient optimization. By employing the reverse mode of AD (as explained in Sec. 1.2.2), one can obtain exact gradients at a low computational cost, since it is independent of the number of input parameters (Griewank and Walther, 2008).

These optimal values are then used to build a surrogate model that learns to predict these optimal γ values simply given the solution and discretization. Surrogate models using Machine Learning (ML) have proven to be quite reliable and computationally efficient at representing large dimensional problems (Lapointe et al., 2020; Müller et al., 2019). An important specificity of the data structure here is that unstructured meshes are unsuitable to simple convolutional layers used in many popular neural network architectures, for example, but can instead be represented as a type of graph, as explained in Sec. 1.1.3. Geometric Deep Learning is a dedicated subfield that use graph abstractions to treat relations between entities without prior constraints on their spatial structure, making them most suitable for this task. Battaglia et al. (2018) recently proposed Graph Neural Network (GNN) models for unstructured mesh simulations to provide physically consistent rollouts of dynamical systems (Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020). These GNNs are natural candidates to learn to predict γ directly on the mesh in this study, and are therefore selected employing the representations described later.

A new numerical method named ML-TTGC is proposed, in which the locally tunable TTGC-γ_L's elemental γ values are predicted using the GNN surrogate model. Such a data-driven discretization of the governing equation

benefits from improved convergence properties by enforcing stability into the ML architecture using LTA. This is a key differentiation of ML-TTGC from recent works on data-driven discretization (Bar-Sinai et al., 2019; Kochkov et al., 2021), where stability cannot be guaranteed or enforced easily. More precisely, the neural network can be constrained to return γ values within the stable regions of TTGC-γ_L predicted by the LTA. So for any initial condition, one can enforce stable evolution of the solution. Recent applications of ML models by Stevens and Colonius (2020) and Kossaczká et al. (2021) to enhance Weighted Essentially Non-Oscillatory (WENO) methods exploited the consistency and stability of the original schemes into the ML model. Melland et al. (2021) replaced the artificial viscosity component of an existing staggered-grid Lagrangian hydrodynamics numerical scheme with a learnable function in the form of an artificial neural network. Magiera et al. (2020) introduced a constraint-resolving layer that allows their ML-based surrogate model of a Riemann solver to exactly satisfy the Rankine-Hugoniot condition. An important building block of ML-TTGC is the LTA, which helps identifying the relevant input features for the network. For example, it is known that variations in local CFL number (N_c) and wavenumber spectrum kh of the solution dictate the local optimal value of γ, which in turn changes the impedance. Such a physics-based feature engineering has proven to be useful and effective in the literature. For example, Woo et al. (2022) enhanced time-series forecasting performance by extracting dominating seasonal patterns with Fourier transformations.

4.2 Numerical implementation and verification of the optimization

This section verifies that the optimization settings defined above leads to a stable evolution of the numerical solution free from spurious waves. The existence of the optimum is critical to the success of the ML surrogate in predicting it. All experiments were conducted in a differentiable solver written in the Julia language (Bezanson et al., 2017) and the gradients were obtained using the source transformation AD package Zygote (presented in Sec. 1.3.4). Sequential Least-Squares Quadratic Programming (SLSQP) (Kraft, 1994) gradient-based optimizer in the NLopt (Johnson, 2007) library was used to solve the constrained multi-objective optimization problem. The optimiza-

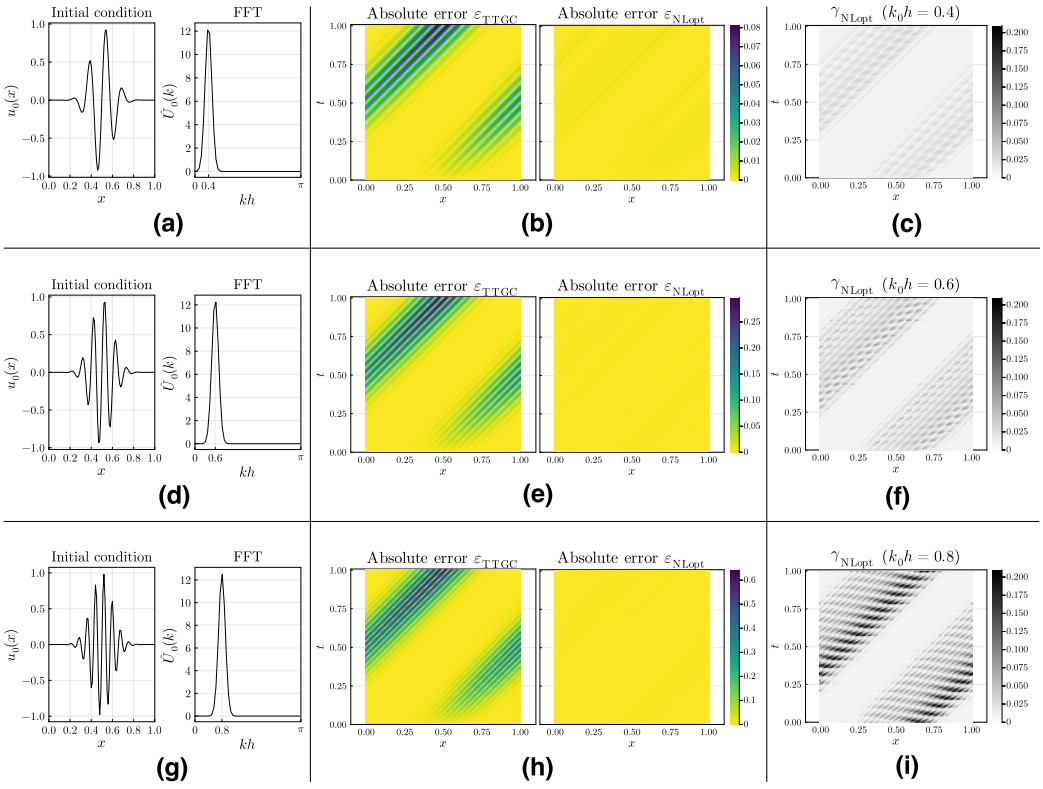


Figure 4.1: Study n.1. Spatio-temporal evolution of numerical error in the solution to the WP convection (Eq. (4.3)); WP and its Fast Fourier Transform (FFT) along with the comparison of numerical errors of optimized and un-optimized TTGC and the optimal γ_{NLopt} obtained using NLopt are plotted for central wavenumbers (a-c) $k_0h = 0.4$, (d-f) $k_0h = 0.6$, and (g-i) $k_0h = 0.8$ respectively.

tion results are compared to the exact solution and to the output of TTGC-γ using the recommended global optimal value of $\gamma = 0.01$ by Colin and Rudgyard (2000). The stability condition $|G| < 1$ translates into bounded γ values to be set in the optimizer. Non-positive and zero value of γ are unstable, so the lower bound $\gamma_{\min} = 0.001$. The upper bound of γ is calculated conservatively using GSA ($\gamma_{\max}$ in Fig. 3.6) for the smallest element size that yields a stable result. The optimization procedure is verified for the linear convection of a Wave Packet (WP) with the following form:

$$u(x,t) = \sin\left[k_0(x - ct - x_0)\right] \exp\left[-\alpha(x - ct - x_0)^2\right] \qquad (x \in [0,1], t \geq 0), \tag{4.3}$$

where k_0 is the central wavenumber of the WP, parameter α controls the spectral spread of the WP and x_0 is the spatial location of the centre of the WP and c is the convection speed.

In the first study (n.1), the TTGC-γ_L scheme applied to an equally spaced mesh is optimized. The convection of WP of relatively low wavenumbers ($k_0 h < 1.0$, inside the GVP region illustrated in Fig. 3.2(e)) on a mesh containing 100 cells (i.e., $h = 0.01$) is analyzed. Fig. 4.1(a, d, g) show the spatial and spectral content of the considered initial conditions.

The spatio-temporal evolution of the numerical error $\varepsilon := |u(x,t) - u_N(x,t)|$ defined by the absolute difference between the analytical $u(x,t)$ and numerical $u_N(x,t)$ solution is plotted for the (i) original TTGC-γ denoted by $\varepsilon_{\text{TTGC}}$ and (ii) the locally optimized TTGC-γ denoted by $\varepsilon_{\text{NLopt}}$ as a dedicated subfield of the graphs shown in Fig. 4.1(b,e,h) respectively for indicated kh in the leftmost column. Firstly, one can observe that the optimized scheme has significantly lower error compared to the original scheme thus indicating that an optimal γ exists. Fig. 4.2 plots the order of error convergence with mesh refinement for the two schemes. The optimized scheme follows the same order of convergence of the original scheme but the entire error curve is shifted (reduced) by an order of magnitude (10^{-1}). The third-order error convergence of TTGC family of schemes is observed only where the wave packet is inside the GVP region. Outside or near the periphery of the GVP ($kh > 1.0$) the order of error fails to conform to the third-order behaviour. Qualitatively, the GVP demarcates the smooth or well-resolved region where order of error is valid (Brooks and T. J. Hughes, 1982).

Fig. 4.1(c, f, i) plot the optimal value of γ obtained for the various WP with central wavenumber $k_0 h = 0.4, 0.6, 0.8$ in the $x-t$ plane. Firstly, one can

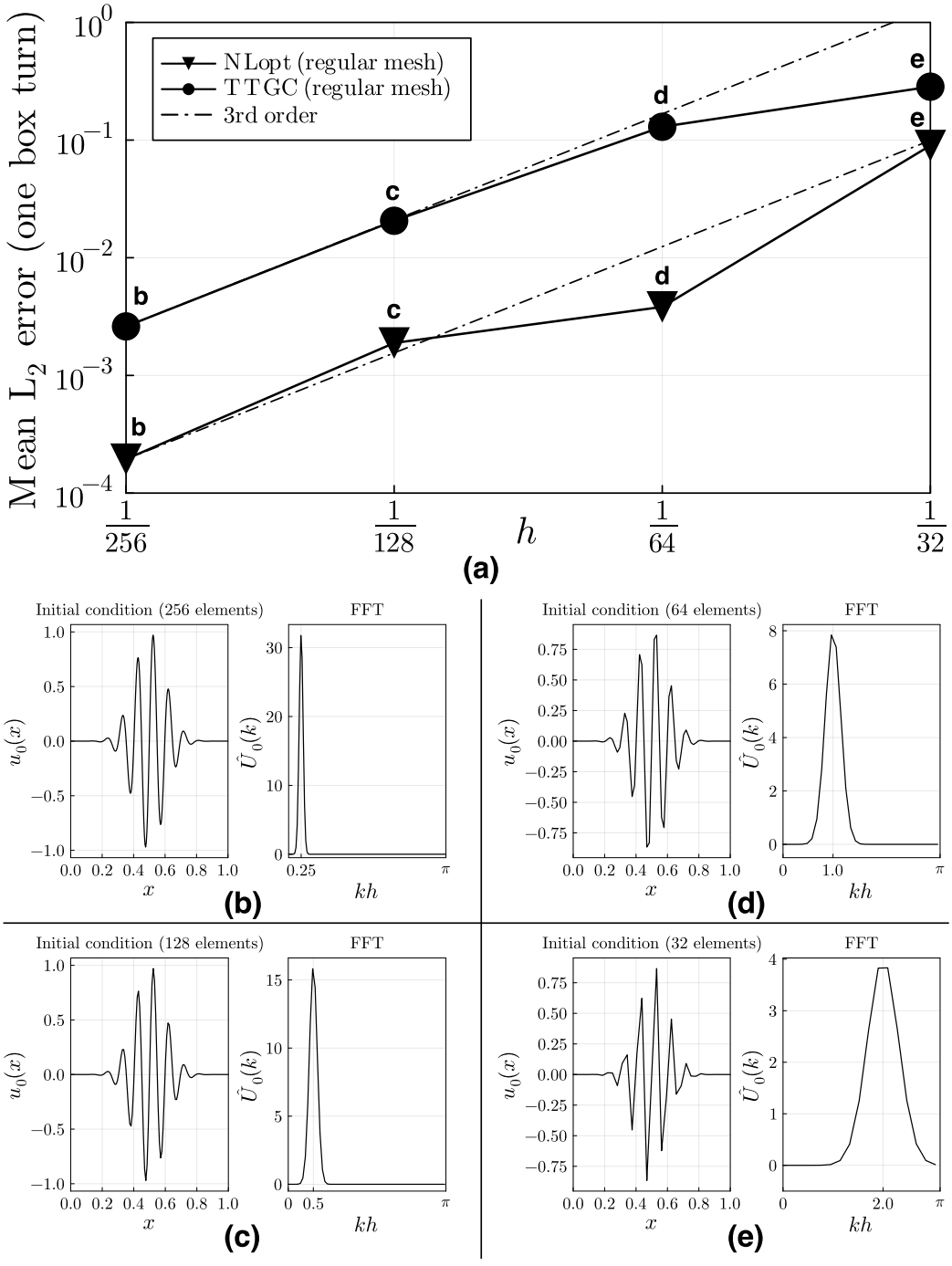


Figure 4.2: (a) Numerical error convergence with mesh refinement (b-e) (finest to coarsest) and the dotted line indicates the third order slope; initial condition and FFT for the mesh refinement levels containing (b) 256, (c) 128, (d) 64 and (e) 32 elements respectively.

observe that the optimal γ is highly correlated with the numerical error and hence the numerical solution itself. Moreover, the optimal value of γ is also localized in the time and space and its magnitude increases proportionally with respect to increase in k_0. These two observations and the error dynamics equation in Eq. (3.31) provide enough evidence that modeling γ based on the spatio-temporal evolution of the numerical solution is feasible.

4.2.1 Multi-objective Pareto optimality

In the second study (n.2), an irregular mesh whose nodes are randomly perturbed is considered. Here, the TTGC-γ scheme generates spurious oscillations even for WP with relatively low wavenumber compared to the Nyquist frequency of the average size ($k_0h = 0.5$). Fig. 4.3(a) shows a WP initial condition in low wavenumber regime ($k_0h = 0.5$) and its corresponding spectral content. To obtain the irregular mesh node location x_i^*, perturbations from a uniform random probability distribution ($\mathcal{U}(0,1)$) toward the positive direction of the x-axis scaled by $\frac{h}{2}$ are imposed on nodes x_i from a uniformly spaced mesh of spacing h, i.e.,

$$x_i^* = x_i + \frac{h}{2}\mathcal{U}(0,1) \tag{4.4}$$

The optimization problem in Eq. (4.2) is solved for different values of parameter θ (the scalarizing weight from Eq. (4.2)) to obtain the Pareto front shown in Fig. 4.3(b). The Pareto front shows that an objective function based purely on the L2-norm of the error is the best choice for evolving waves with low wavenumber content. Note that waves having $kh < 1.2$ are within the GVP region hence they are well resolved and generate lesser q-waves. Enforcing TVD in this region adds more dissipation and thus cannot provide net improvement to the objective. Fig. 4.3(c-d) compare the numerical solution of the TTGC-γ_L (optimized) and the original TTGC-γ at different time instants of the wave convection. For TTGC-γ_L, the optimization problem (4.2) is solved for each timestep. The optimized dynamics shows almost negligible spurious oscillations compared to the original TTGC-γ scheme. The optimal values of γ have a high temporal correlation with the numerical solution and exhibit a similar convection pattern. Therefore, it is important to build this correlation information into the surrogate model to improve its predictions.

In the next study (n.3), the convection of a high-frequency WP using a uniform mesh spacing is considered. Here, the original TTGC-γ scheme is

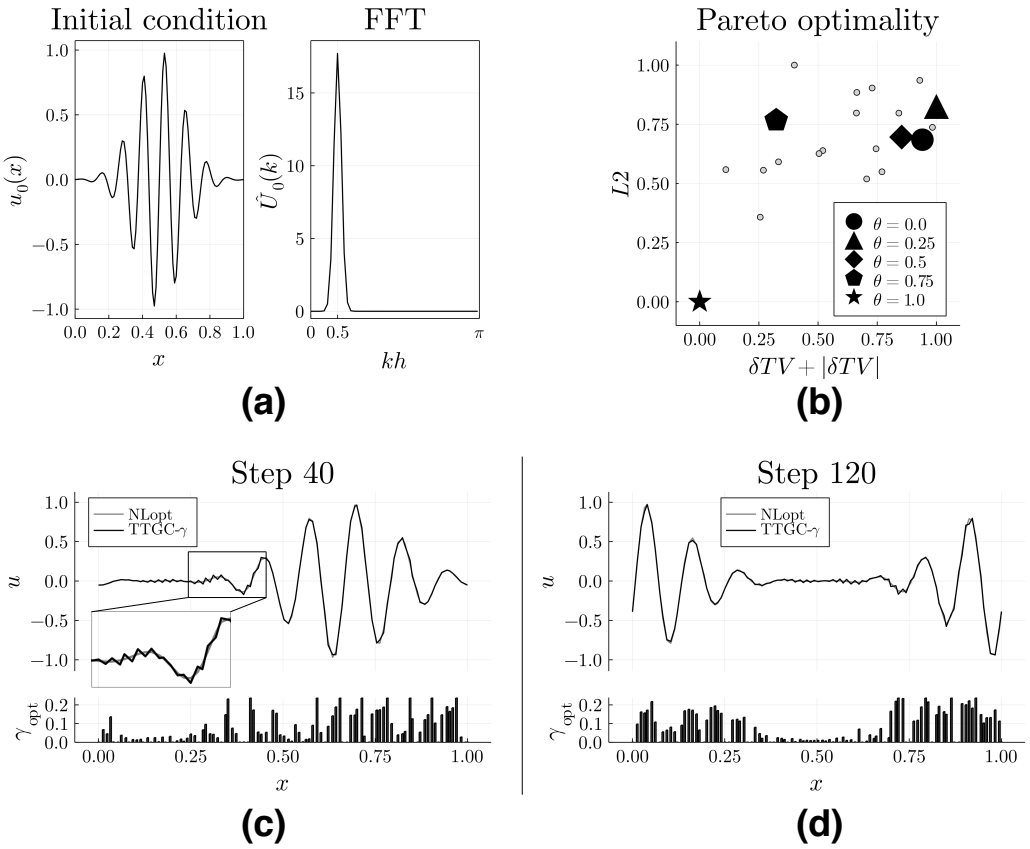


Figure 4.3: Study n.2. (a) Left: Periodic initial condition over an irregular mesh of 101 nodes (or 100 finite-elements). Parameters values are $(k_0 h, \alpha, x_0, c) = (0.5, 25, 0.5, 1)$. Right: Fast-Fourier Transform evaluated over the regular counterpart of the actual mesh; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively.

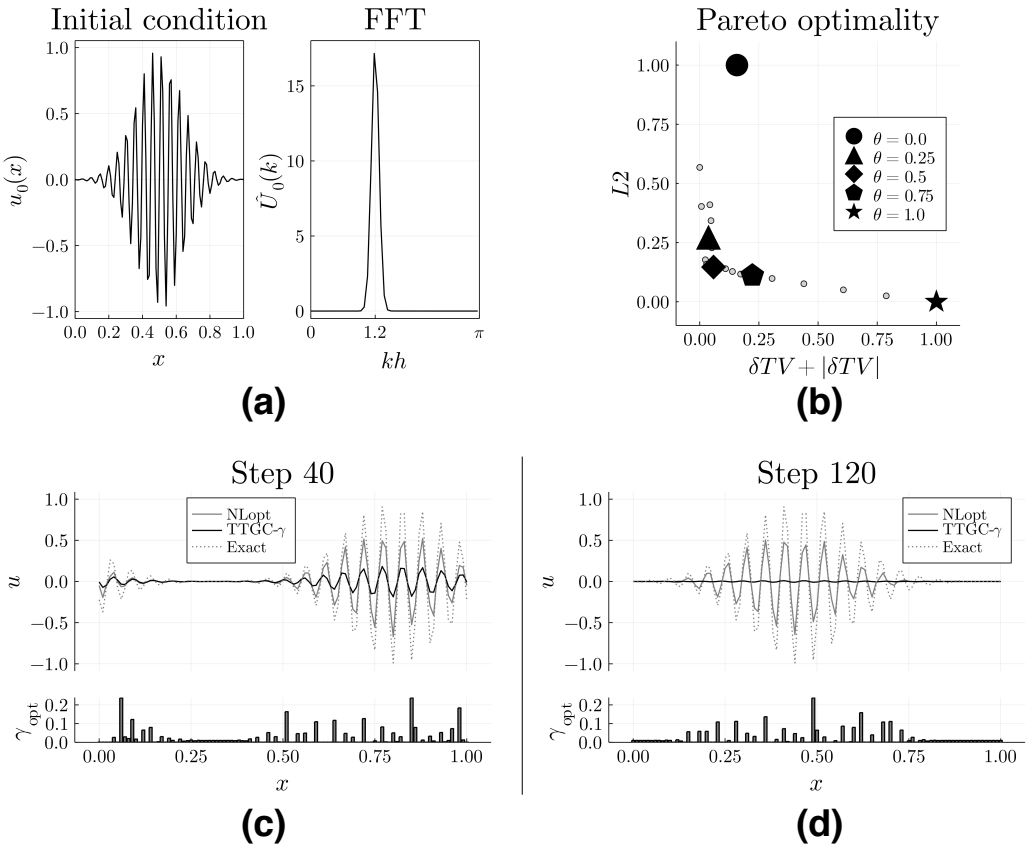


Figure 4.4: Study n.3. (a) Left: Periodic initial condition over a regular mesh of 101 nodes (or 100 finite-elements). Parameters values are $(k_0 h, \alpha, x_0, c) = (0.5, 25, 1.2, 1)$. Right: Fast-Fourier Transform; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of exact solution, NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively.

known to exhibit high numerical dissipation. The initial condition and its Fourier spectrum are plotted in Fig. 4.4(a) and the Pareto front is plotted in Fig. 4.4(b). For (n.3) one finds that $\theta = 0.5$ on the Pareto front yields the lowest TVD and L2 error for the wave convection contrary to (n.2) where $\theta = 1$ was found to be better. For $kh = 1.2$, slightly outside the periphery of the GVP region (where the dissipation as well as the dispersion errors dominate), the optimizer shows that even on regular meshes one can vary γ locally and mitigate dissipation and dispersion errors. For this wavenumber, the original TTGC-γ scheme completely dissipates the wave but the optimized results maintain the wave amplitude as shown by the time evolution in Fig. 4.4(c-d). One observes the similar highly correlated convection pattern of the optimal γ values in time for (n.2) as seen in study (n.1).

The final study (n.4) regards the convection of a square wave, which is defined below:

$$u(x,t) = \mathbb{1}_{[x_0,x_1]}(x - ct) \qquad (x \in [0,1], t \geq 0), \tag{4.5}$$

where x_0 and x_1 are the initial position of the discontinuities of the square wave and $\mathbb{1}$ is the indicator function defined as

$$\mathbb{1}_A(x) := \begin{cases} 1 & \text{if } \ x \in A \ , \\ 0 & \text{if } \ x \notin A \ . \end{cases} \tag{4.6}$$

In gradient-based optimization, discontinuous objective functions are avoided to facilitate smooth descent direction. To ensure a differentiable objective function, a regularized version of the exact solution to the square wave evolution as shown in Eq. (4.7) was used:

$$u_{\text{reg}}(x,t) := \{\tanh[s(x - ct - x_0)] - \tanh[s(x - ct - x_1)]\}/2, \tag{4.7}$$

where parameter s is an arbitrary steepening factor, i.e., the higher the value of s steeper is the regularized square wave. In (n.4), a value of $s = 150$ was used. The Fourier spectrum of the square wave in Fig. 4.5(a) shows that a major proportion of the energy is contained within the GVP region and almost all wavenumbers (broadband) in the Nyquist limit are excited by this waveform. Such a broadband excitation creates spurious waves in the high wavenumber region and even upstream propagating waves, once wavenumbers near the Nyquist limit have negative numerical group velocity (see Fig. 3.2). The Pareto front in Fig. 4.5(b) shows that $\theta \approx 1$ (i.e., pure

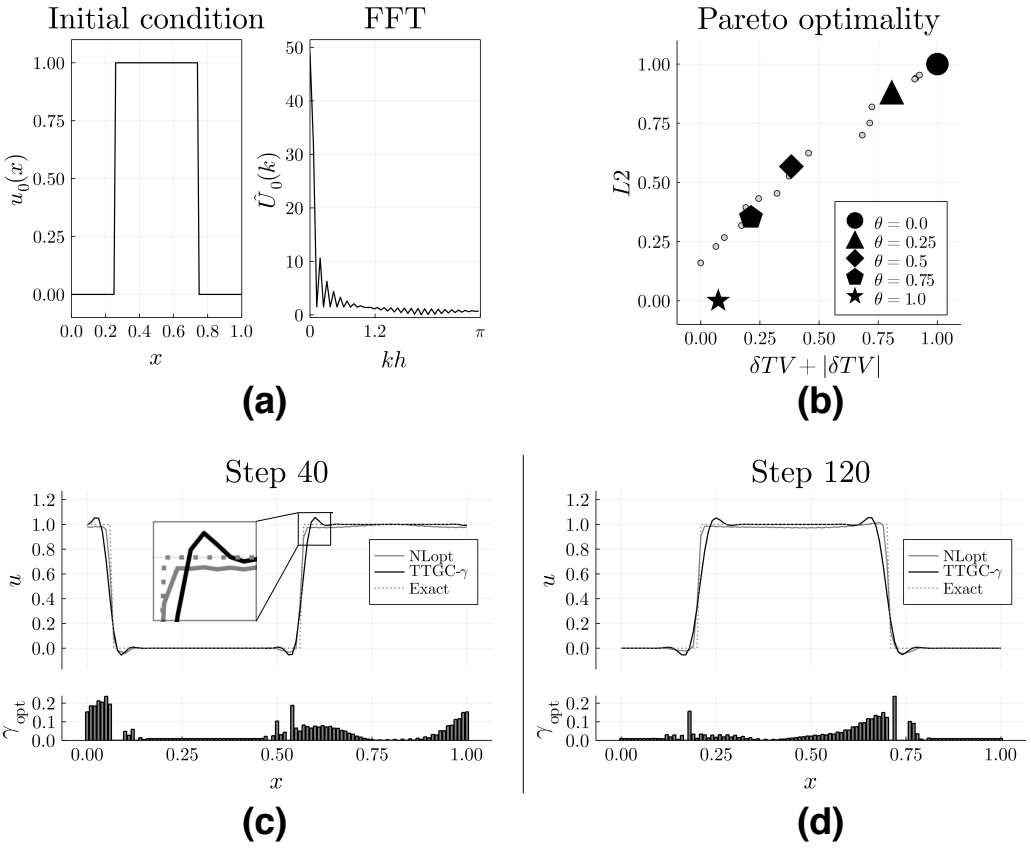


Figure 4.5: Study n.4. (a) Initial condition and its frequency domain counterpart. Parameter $a = 1$; (b) Pareto front for varying $\theta \in [0., 1.]$ comparing normalized time-averaged values of the two objective functions from the optimization problem (mean over 100 steps); (c, d) Comparison of exact solution, NLopt and TTGC-γ outputs at steps n. 40 and 120, respectively.

L2-norm of the error) yields the best trade-off of L2 error for the lowest TVD error. Fig. 4.5(c-d) shows the comparison of the solution evolution of the original TTGC-γ scheme and Pareto optimal one ($\theta = 1$). It can be seen that the latter generates over-/under-shoots near the sharp edges of the square wave of less amplitude than those produced by the TTGC-γ method. For all four studies (n.1-4), one finds the same pattern that the optimal γ is highly correlated temporally and moves with the solution.

In conclusion, by means of the study cases above, the existence of local optimal parameters for the TTGC-γ_L scheme could be numerically verified. These optimal values allowed us to address main issues around the original TTGC-γ scheme, namely the generation of spurious oscillations for irregular meshes and the excessive dissipation of waves with relatively high wavenumber content. Despite the encouraging results, this latter case demonstrated that the impedance matching problem cannot always be fully solved (some wave dissipation is still present, see Fig. 4.4(c-d)). This is due to two scheme features: stability and resolvable regions. The impedance matching problem is constrained by a stability condition (see Eq. (4.2)). In order to keep the scheme stable, the design space of the γ parameter that could be explored has to be bounded. Along with that, the spectral properties of TTGC-γ indicate resolvable regions (i.e., GVP/DRP zones) that do not significantly change for different values of γ (see Fig. 3.5). This means that high-wavenumber content, out of the resolvable regions for the given scheme, can never be faithfully convected. By means of these optimization studies, not only achievements but also the limits of the proposed approach could be assessed.

4.3 Neural network model

The previous section verified that the optimization process on the locally tunable TTGC-γ_L scheme leads to improved results on a range of initial conditions for the convection equation. Here, a surrogate model that will provide results of similar quality is sought. For that purpose, an Encode-Process-Decode Graph Neural Network (GNN) architecture inspired by the work of Pfaff et al. (2021) is used. It is composed of a set V of vertices whose relationship is represented using an ensemble of directed edges. These directed edges are represented as a Bipartite graph connecting the sets of so-called sender S and receiver R vertices. S and R participate in a message-passing scheme that transforms attributes associated with each vertex and

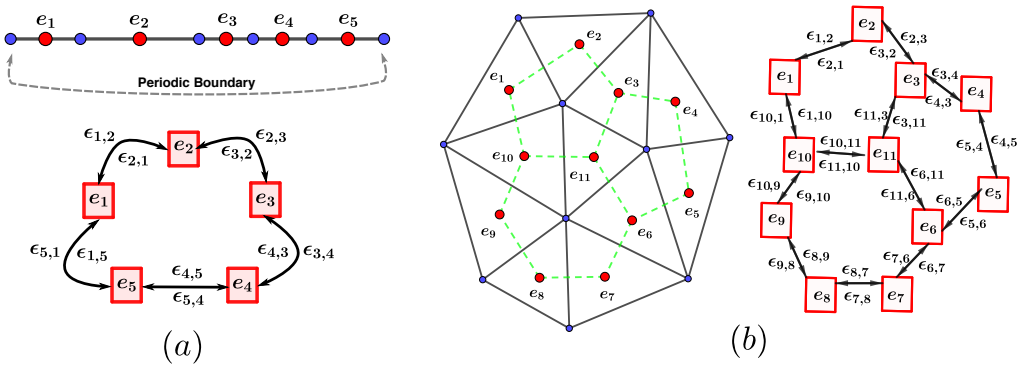


Figure 4.6: (a) 1D primal/dual mesh (top) and GN based on the dual mesh (bottom); (b) Unstructured primal/dual mesh in 2D (left) and GN based on the dual mesh (right). Arrows denote the GN message-passing, red boxes are the GN vertices (which are equivalent to the dual mesh nodes, indicated by red dots), blue dots are the primal mesh nodes and green dotted lines are the dual mesh edges.

edge using neural network layers.

The quantity γ to be predicted is defined element-wise. A natural choice for the set V of vertices would then be the set of finite-elements in the physical mesh, i.e.,

$$V := \bigcup_i e_i.$$

For each vertex e_i, the following vector of input features (f_i^n) is assigned at time step n:

$$\text{(1D)} \qquad f_i^n := [u_i^n,\, u_{i+1}^n,\, \gamma_i^{n-1}]^\top,$$

where u_i^n and u_{i+1}^n are the solution values at the nodes of the element and γ_i^{n-1} is the cell value of the parameter γ that was predicted at the previous time step. In Sec. 4.2, it was found that the optimal γ are highly correlated temporally (see figs. 4.3(c-d), 4.4(c-d), 4.5(c-d)). Therefore, including γ_i^{n-1} in the input features helps the network to find an optimal γ_i^n, which should be close to its previous value. By its inclusion, it was observed that convergence of the optimizer to the objective function during the training improved significantly. An edge ϵ_{ij} in this graph corresponds to a pair of a sender vertex $e_i \in S$ along with a receiver vertex $e_j \in R$. Every pair (e_i, e_j) of

elements sharing a common face is connected by two directed edges, namely ϵ_{ij} and ϵ_{ji}. This is depicted in Fig. 4.6 for 1D and 2D (triangular) meshes. As input edge attribute g_{ij}, the local CFL numbers of the elements e_i and e_j are considered, i.e.,

$$\text{(1D)} \qquad g_{ij} := [N_{c,i}, N_{c,j}]^\top.$$

The above-defined input features are transformed across the Encode-Process-Decode GNN model, as illustrated in Fig. 4.7. This architecture extends Def. 3 of a GNN, which originally only contains message-passing layers. Firstly, an encoding block transports vertices' and edges' features to a higher dimensional latent space (of size L) using independent dense layers. For the applications presented in Sec. 4.4, $L = 32$. The output of the encoder then feeds a core consisting of message-passing layers (that transforms vertices' attributes based on messages carried by edges). Two message-passing layers are used because the results from the entropy condition (in Appendix D) indicate that second-level neighbours $\{i-2,\ i+2\}$ influence impedance matching problem at a given node i. Finally, a decoding block converts vertex attributes to a feature space compatible with our desired final output, namely the element local γ values. The GNN-predicted γ values feed the TTGC-γ_L numerical solver, which outputs an evolved state u^{n+1}. It is worth noting that the decoder's output is re-scaled using a sigmoid function so that γ^n does not violate the stability criteria for the locally tunable TTGC-γ_L, which is key to the generalizability of ML-TTGC. During training, the advanced state is compared to the (regularized) exact solution. More precisely, the loss function can be expressed as

$$\mathcal{L}(u^0) := \frac{1}{N_{\text{step}}} \sum_{k=1}^{N_{\text{step}}} \theta\, L2\left(u^k_{\text{pred}}\right) + (1-\theta)\, \left[\delta TV\left(u^k_{\text{pred}}\right) + \left|\delta TV\left(u^k_{\text{pred}}\right)\right|\right], \tag{4.8}$$

where N_{step} is the number of time steps to predict from a given initial condition u^0, u^k_{pred} is the ML-TTGC output at time k and $\theta \in [0., 1.]$ is the scalarizing weight from Eq. (4.2). The reader may notice that the GNN model could have been trained using a database of optimal γ generated using NLopt. However, generating such a database for various initial conditions is expensive especially when solving the problem in higher-dimensions.

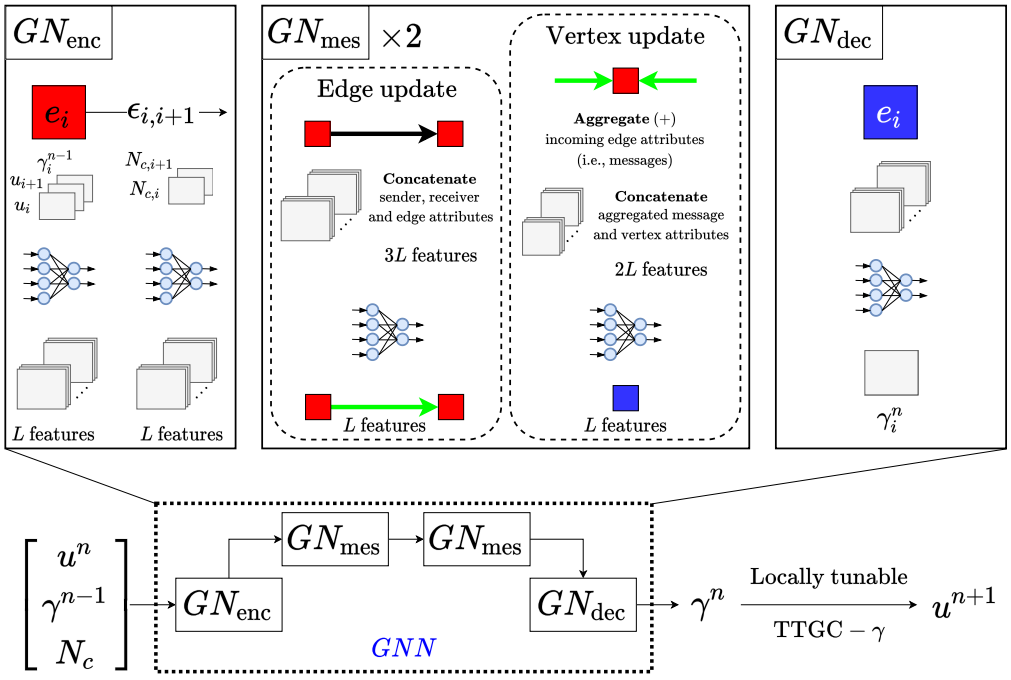


Figure 4.7: ML-TTGC architecture. The GNN instances trained for the studies presented in Sec. 4.4 use an encoding block, two message-passing layers and a decoding block, which update the vertices and edges attributes using dense layers with a hidden feature space of size $L = 32$. Each instance has a total of $10,637$ trainable parameters.

4.4 Application of ML-TTGC

The previous section introduced ML-TTGC: a framework built upon a GNN model that finely tunes the TTGC-γ_L scheme. ML-TTGC will now be used to solve linear and non-linear convection problems. The goal is to assess the proposed architecture's capabilities to provide stable and accurate dynamics for a myriad of initial conditions. More precisely, wiggle-free, less dispersive/dissipative solutions in comparison to those provided by the original TTGC-γ scheme are sought. Tbl. 4.1 below summarizes the set of experiments performed with linear convection and Burgers' equations.

Problem	**Training**	**Test**	**Figures**
Convection 1	$N_c \in (0.7, 0.9)$ Irregular meshes Wave packets $k_0 h \in (0., 0.6)$	$N_c = 0.84$ Irregular meshes Wave packets $k_0 h \in \{0.8.1.5\}$	Fig. 4.8
Convection 2	$N_c \in (0.5, 0.7)$ Regular meshes Wave packets $k_0 h \in (1., 2.)$	$N_c = 0.55$ Regular meshes Wave packet $k_0 h = 1.6$	Fig. 4.9
Burgers'	Square wave $a = 1$	Square wave $a = 2$ Sine wave	Fig. 4.10 Fig. 4.11

Table 4.1: Sets of initial conditions for ML-TTGC applications.

4.4.1 Linear convection equation

The evolution of systems following the linear convection equation as defined in Eq. (3.1) is investigated. It is proposed to solve the convection of a WP initial conditions of the form

$$u(x) = \sin\left[k_0(x - x_0)\right] \exp\left[-\alpha(x - x_0)^2\right] \qquad (x \in [0, 1]), \tag{4.9}$$

where k_0 is the central frequency, α is the spectral spread and x_0 is the spatial location of the peak of the WP. The GNN γ predictor in ML-TTGC

was trained on the forecasting of $N_{\text{step}} = 16$ time steps of a set of 32 samples at wavenumbers $k_0 h \sim \mathcal{U}_{k_0 h}(0, 0.6)$, where $\mathcal{U}_{k_0 h}$ is a uniform distribution in the interval $(0, 0.6)$. Here, h denotes the average mesh size of the irregular grids used in training. They are generated according to Eq. ((4.4)). Once only low wavenumber WP are present in the training set, $\theta = 1$ is chosen as impedance matching balance level in the loss function, following findings in Sec. 4.2. Each sample is evolved under a global CFL $\sim \mathcal{U}_{\text{CFL}}(0.7, 0.9)$ drawn from the uniform distribution $\mathcal{U}_{\text{CFL}}$ in the interval $(0.7, 0.9)$, typical to CFD simulations.

Fig. 4.8 shows the numerical solution of the ML-TTGC and TTGC-γ scheme for the convection problem of WP at CFL $= 0.84$, $k_0 h = 0.8$ (a) and $k_0 h = 1.5$ (b). Note that these WP are not part of the training data set and were deliberately chosen outside the set to demonstrate robustness. One observes that the amplitude of the spurious oscillations is significantly reduced in ML-TTGC compared to the TTGC-γ scheme where amplitudes of the spurious waves are quite significant. Many flow simulations can exhibit sharp gradients such as these, and such numerical wiggles can lead to unphysical values, notably negative quantities. Therefore, it is critical to mitigate them in the numerical solution. This is most commonly achieved by adding artificial dissipation, which significantly increases the solver runtime.

In this first experiment, the dissipation of high-frequency waves (Fig. 4.8(b)) has proven to be unavoidable, due to the spectral properties of the basis scheme at wavenumbers higher than unity. However, it was found that lowering the CFL values at which the simulations are run can address this to some extent. More precisely, the upper limits for γ under which the scheme is stable are increased as the CFL lowers (following GSA, Fig. 3.6), and hence a larger design space leading to less dissipative solutions can be explored by the neural network. In a second experiment, ML-TTGC was separately trained on 16 samples from the solution sets of WP convection at CFL $\sim \mathcal{U}_{\text{CFL}}(0.5, 0.7)$ and $k_0 h \sim \mathcal{U}_{k_0 h}(1., 2.)$. Fig. 4.9 illustrates the high dissipation of TTGC-γ (typical at such range of wavenumbers), which contrasts with ML-TTGC's ability to preserve the wave amplitude for longer time scales. Results shown in Fig. 4.8(b), obtained using an irregular mesh, show higher dissipation of high kh waves; while the results shown in Fig. 4.9(a), obtained using a regular mesh, show less dissipation. The irregularity poses additional challenge as the ML model has to match two competing objective functions, minimum reflection and maximising wave power. Therefore during training the model learns to trade-off dissipation to recover minimum reflection solution (wig-

gle free). On a uniform mesh this limitation is non-existent therefore the ML-TTGC scheme learns to be less dissipative.

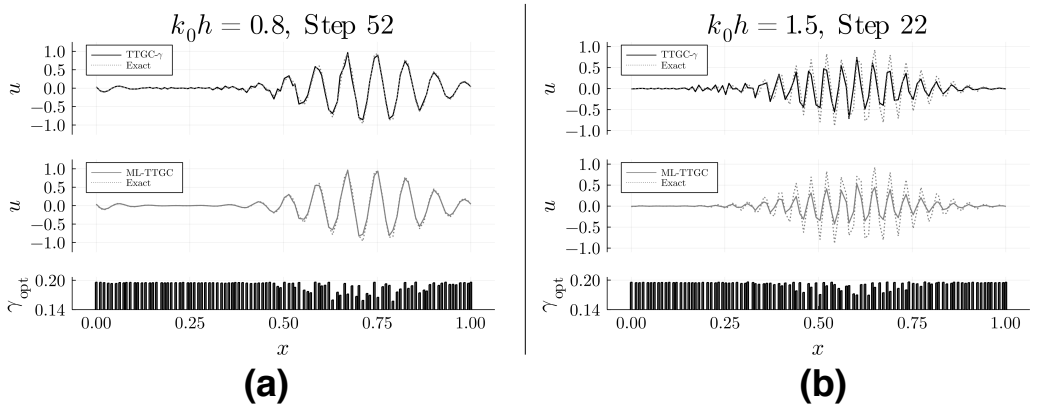


Figure 4.8: (a) Low-frequency WP convection over an irregularly spaced mesh. (b) High-frequency WP convection over irregular mesh. Top: TTGC-γ (Default) and analytical/exact solution. Center: ML-TTGC and analytical/exact solution. Bottom: GNN predicted parameters for locally tunable TTGC-γ_L. Initial conditions are not part of the training set.

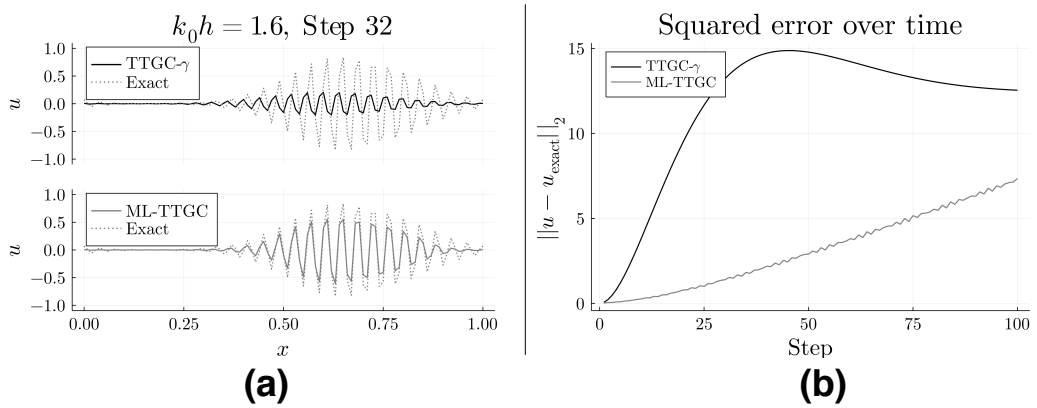


Figure 4.9: (a) High-frequency WP convection over regular mesh. Top: TTGC-γ (Default) and analytical/exact solution. Bottom: ML-TTGC and analytical/exact solution. Initial condition is not part of the training set. (b) L2 norm of the error for TTGC-γ (Default) and ML-TTGC, comparing their dissipation levels over time.

4.4.2 Inviscid Burgers' equation

For the inviscid Burgers' equation, training database is composed of square-wave initial conditions sampled over irregular meshes:

$$u(x) = a\,\mathbb{1}_{[x_0,x_1]}(x) \qquad (x \in [0,1]), \tag{4.10}$$

where $\mathbb{1}$ is the indicator function, as defined in Eq. (4.6), a is the wave amplitude (fixed at 1) and x_0 and x_1 are the initial discontinuities positions that are randomly placed (provided that $x_0 < x_1$). The parameters $\{x_0, x_1\}$ are taken as $x_0 = \frac{1}{5}(1 + \mathcal{U}_0(0,1))$ and $x_1 = \frac{3}{25}(5 + \mathcal{U}_1(0,1))$, where $\mathcal{U}_0$ and $\mathcal{U}_1$ are independent random variables following the uniform distribution in the interval $(0,1)$. Following the method of characteristics, depicted in Fig. D.1(e), their dynamics consist of a progressive rarefaction zone and a moving shock wave that will meet at some point in time. Analytically, the exact solution as a function of time and space can be expressed as

$$u(x,t) = \begin{cases} (x - x_0)/t, & x \in (x_0, x_0 + at] \\ a, & x \in [x_0 + at, x_1 + ct] \\ 0, & \text{elsewhere} \end{cases} \qquad (t \in (0, t_m)), \tag{4.11}$$

where a is the square wave amplitude, $c = a/2$ the shock speed and t_m is the rarefaction-shock meetup time.

Numerically, the simulation time step for both TTGC-γ and ML-TTGC is defined as

$$\Delta t := N_c^{\text{global}} \Delta x_{\min}/a, \tag{4.12}$$

where $N_c^{\text{global}} = 0.84$ and $\Delta x_{\min}$ is the minimum cell size of the considered mesh instance. Results in Fig. 4.10 and Fig. 4.11 compare TTGC-γ and ML-TTGC at the same physical time.

For this particular problem, it was found that the original TTGC-γ produces over-/under-shoots around the shock front and oscillations moving upstream. Using LTA (see Sec. 3.2.5.1), it was shown a mismatch of nodal impedance at the shock with the presence of linear instability was the reason for the over-/under-shoots near the shock. Introducing an artificial dissipation term every sub-step was quite effective in mitigating these numerical issues at the shock. Therefore, it is proposed to add an artificial viscosity term with a locally tunable viscosity parameter into the ML-TTGC scheme.

The viscous terms added after each ML-TTGC sub-step assume the following form:

$$\Delta\tilde{u}^{n+1}_{e,\text{ viscous}} = \Delta t \sum_e \nu_e \, \mathbf{D}^\star_e \, u^n_e \quad \text{(1st sub-step)} \tag{4.13}$$

$$\Delta u^{n+1}_{e,\text{ viscous}} = \Delta t \sum_e \nu_e \, \mathbf{D}^\star_e \, u^n_e \quad \text{(2nd sub-step)}, \tag{4.14}$$

where $\mathbf{D}^\star_e = \frac{1}{h_i^2}\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$.

Since the smooth regions of the flow are well evolved by TTGC-γ, it was decided to tune artificial viscosity only near shock, whereas keeping $\nu = 0$ for cells far from it (when at least 10 cells far from the shock front in the upstream direction and 5 cells downstream). A regularized version of the moving shock wave is used as reference, given by

$$u_{\text{reg, shock}}(x,t) = a\left\{1 - \tanh\left[s(x - x_1 - ct)\right]\right\}/2; \;\; s = 100. \tag{4.15}$$

Due to the relatively limited variability of the training set, it was found that tuning ML-TTGC for the prediction of 20 time steps from 8 initial conditions is enough to give satisfactory results at inference, even at longer time scales, as shown in Fig. 4.10(a). One observes that over-/under-shoots as well as Gibbs' phenomenon are absent by means of the ML-TTGC predicted values for γ and ν. The 'tanh' function was deliberately chosen to regularize the exact solution used in the objective function (Eq. (4.15)) because the exact solution to a moving shock in the viscous Burgers' equation with constant viscosity has a 'tanh' behavior at the shock. It is interesting to notice that the network mimics the viscous Burgers' result by choosing to add constant viscosity (maximum within the set upper bound) in a major part of the shock region.

Fig. 4.10(b) illustrates inference outputs on a generalization test. Here, the jump or amplitude of the shock is doubled when compared to the training samples ($a = 2$). Despite this substantial change to the GNN input features, the resulting dynamics still present significantly fewer undesired oscillations when compared to the original TTGC-γ scheme. Additionally, the GNN model trained only on the square wave was used to predict the dynamics of a sine wave, as shown in Fig. 4.11. The standing shock at the domain boundaries creates a growing numerical instability in the original TTGC-γ scheme, leading to numerical blowup (corroborating the LTA findings in

Sec. 3.2.5.1). ML-TTGC delivers a stable evolution of the numerical solution, which demonstrates our architectures' excellent extrapolation capability.

A priori knowledge of the shock location allowed to restrict the tuning region for viscosity, which helps the network in finding better optima. In general, one should consider the use of shock sensors to determine strong gradient zones of the flow, where the network will be trained on. Recently, Feng et al. (2020) built shock wave detectors based on machine learning. Alternatively, precise estimates of the local wavenumber could be used to identify shock regions. It was found that the one defined in Appendix C was not accurate enough for the cases studied here, though. Ultimately, one can think of a plug-and-play ML-TTGC architecture composed of some layers trained only around shocks while others specialized in handling smooth regions of the flow. They would be conveniently activated to deliver accurate dynamics in general cases.

The capacities of a GNN architecture used as a surrogate model of the element local parameter γ of the TTGC-γ_L scheme were demonstrated. In problems governed by the 1D Convection/Burgers' equations, ML-TTGC outperformed the original TTGC scheme by producing lower levels of dissipation and dispersion errors. In the next chapter, an extension of the ML-TTGC method to higher dimensional problems is proposed.

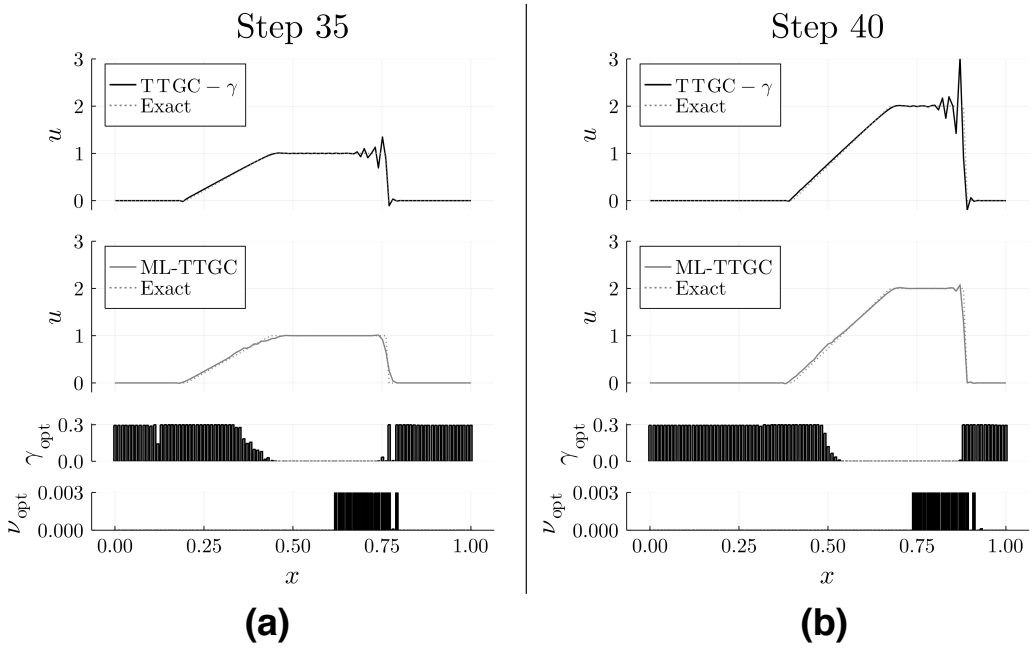


Figure 4.10: Numerical solution obtained at time steps (a) $t = 35\Delta t$ and (b) $t = 40\Delta t$ (TTGC-γ and ML-TTGC scheme with predicted parameters γ_{opt} and ν_{opt}) to the inviscid Burgers' equation for the square-wave initial condition, where Δt is the simulation time step. Parameter values used are $(a, x_0, x_1) = (1., 0.2, 0.65)$ for (a) on the left and $(a, x_0, x_1) = (2., 0.4, 0.75)$ for (b) on the right; data is not part of the training set.

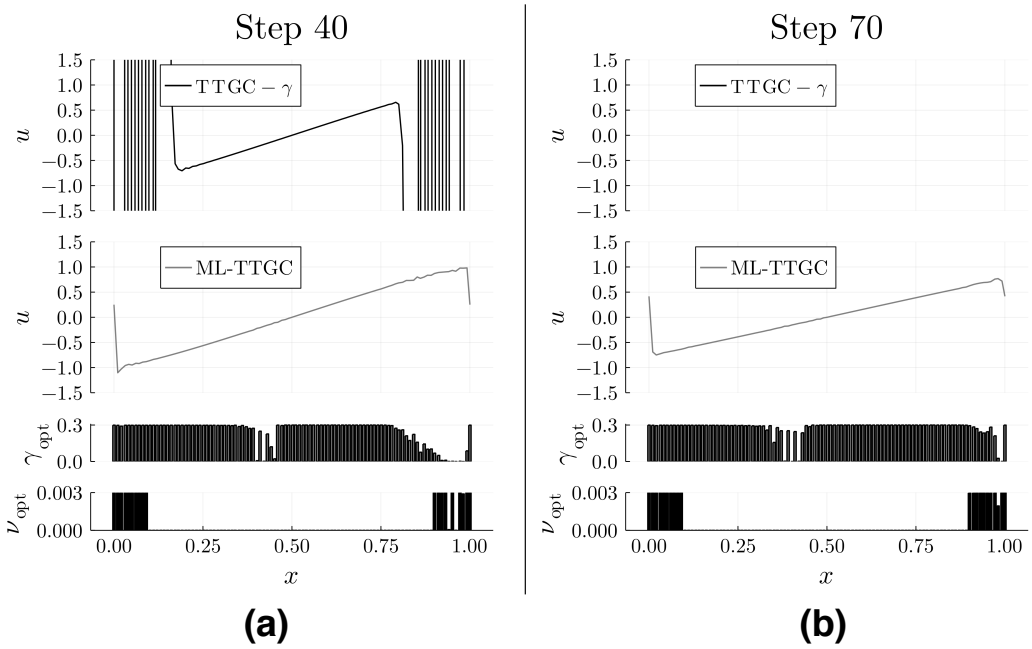


Figure 4.11: Numerical solution obtained at time steps (a) $t = 40\Delta t$ and (b) $t = 70\Delta t$ (TTGC-γ and ML-TTGC with predicted γ_{opt} and ν_{opt}) to the inviscid Burgers' equation for the sine wave initial condition, where Δt is the simulation time step; data is not part of the training set.

Chapter 5

Extending ML-TTGC to unstructured meshes

Chapter 4 proposed a surrogate model based on graph neural networks that locally tunes a version of the Two-step Taylor Galerkin C (TTGC) scheme used in LES. It was employed in 1D problems governed by the Convection/Burgers' equations. In this chapter, extensions of this machine learning model and training procedure to systems governed by the Convection/Euler equations on higher dimensions are sought. Focus is placed on unstructured irregular grids, typical of industrial CFD. Firstly, the TTGC scheme formulation in higher dimensions is briefly presented. The corresponding ML-TTGC numerical method is then defined and applied in benchmarking problems for numerical schemes on 2D triangular meshes. ML-TTGC's ability to damp spurious oscillations is verified, as observed in the 1D simulations. Finally, an ML-TTGC instance trained only on the convection of vortices following the Euler equations demonstrate capacity to evolve a double shear layer initial condition in a stable fashion, where a numerical blowup otherwise occurs when employing the original TTGC numerical scheme.

5.1 TTGC in higher dimensions

By using the divergence theorem, the hyperbolic conservation law (2.1) assumes the following so-called integral form:

$$\frac{\partial}{\partial t}\int_{\Omega} \boldsymbol{U}\ \mathrm{d}\mathcal{V} + \int_{\partial\Omega} \underline{\boldsymbol{F}} \cdot \boldsymbol{n}\ \mathrm{d}\mathcal{S} = \boldsymbol{0}, \tag{5.1}$$

where Ω is an arbitrary control volume, $\partial\Omega$ its boundaries and $\boldsymbol{n}$ is the outward normal to the boundaries. A domain discretization is required to represent $\boldsymbol{U}$ in a finite set of points, and Ω is eventually expressed as the union of polygons (2D) or polyhedrons (3D):

$$\Omega = \bigcup_{K_e \in M} K_e, \tag{5.2}$$

where M is the mesh object and K_e is a cell of this mesh. In what follows, only Triangles (2D) or Tetrahedrons (3D) are considered to be composing a mesh.

In the discretized domain, within a cell K_e, Eq. (5.1) reads

$$\frac{\partial \boldsymbol{U}_e}{\partial t} + \boldsymbol{R}_e = \boldsymbol{0}, \tag{5.3}$$

where $\boldsymbol{U}_e$ is a mean over K_e of the state variable and $\boldsymbol{R}_e$ (as known as *cell residual*) is defined as

$$\boldsymbol{R}_e := \frac{1}{V_e}\oint_{\partial K_e} \underline{\boldsymbol{F}}_h \cdot \boldsymbol{n}\ \mathrm{d}\mathcal{S}, \tag{5.4}$$

with $\underline{\boldsymbol{F}}_h$ a numerical approximation of the convective flux $\underline{\boldsymbol{F}}$ and V_e the volume of K_e (in 2D, it is the triangle area).

Eq. (5.3) is solved in different ways following the numerical scheme of choice. On the one hand, the so-called *cell-centered* schemes store state variables at the center of each cell. On the other hand, *cell-vertex* schemes store state variables at the mesh nodes. In cell-vertex schemes, one solves a nodal form of Eq. (5.3) as follows:

$$\frac{\partial \boldsymbol{U}_j}{\partial t} + \boldsymbol{R}_j = \boldsymbol{0}, \tag{5.5}$$

where $\boldsymbol{U}_j$ is the value of the state variable at some node j and $\boldsymbol{R}_j$ is its associated *nodal residual*. Fig. 5.1 illustrates the difference between cell-centered and cell-vertex schemes.

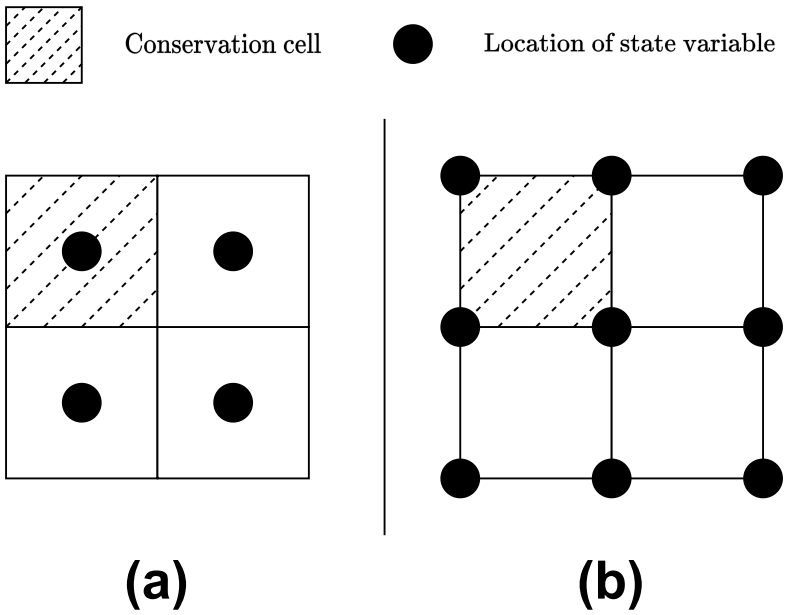


Figure 5.1: Different finite volume schemes: (a) cell-centered, (b) cell-vertex. Inspired from Crumpton et al. (1993).

In particular, the implementation of the TTGC scheme of Colin and Rudgyard (2000) inside the CERFACS solver AVBP is a cell-vertex one. More specifically, it belongs to the class of *residual distribution* schemes (Deconinck and Ricchiuto, 2017), which compute the nodal residual $\boldsymbol{R}_j$ by weighted aggregation of residuals $\boldsymbol{R}_e$ from all cells surrounding node j. Such an aggregation is called the *scatter* operation and is illustrated in Fig. 5.2 below. In order to compute the cell residual $\boldsymbol{R}_e$, it is raised the assumption that convective fluxes vary linearly along faces. In 2D, by considering a triangle cell with nodes $\{1, 2, 3\}$, this translates to an approximation of the flux along the triangle face $\{2-3\}$ (for instance) as

$$\underline{\boldsymbol{F}}_{h,23}(s) = \underline{\boldsymbol{F}}_2 + s \cdot \frac{\underline{\boldsymbol{F}}_3 - \underline{\boldsymbol{F}}_2}{\Delta s_{23}} \qquad (s \in [0, \Delta s_{23}]),$$

where Δs_{23} is the face length. Eq. (5.4) then reads

$$\boldsymbol{R}_e = \frac{1}{V_e}\left[\left(\frac{\underline{\boldsymbol{F}}_2 + \underline{\boldsymbol{F}}_3}{2}\right) \cdot \boldsymbol{n}_{23} + \left(\frac{\underline{\boldsymbol{F}}_3 + \underline{\boldsymbol{F}}_1}{2}\right) \cdot \boldsymbol{n}_{31} + \left(\frac{\underline{\boldsymbol{F}}_1 + \underline{\boldsymbol{F}}_2}{2}\right) \cdot \boldsymbol{n}_{12}\right], \tag{5.6}$$

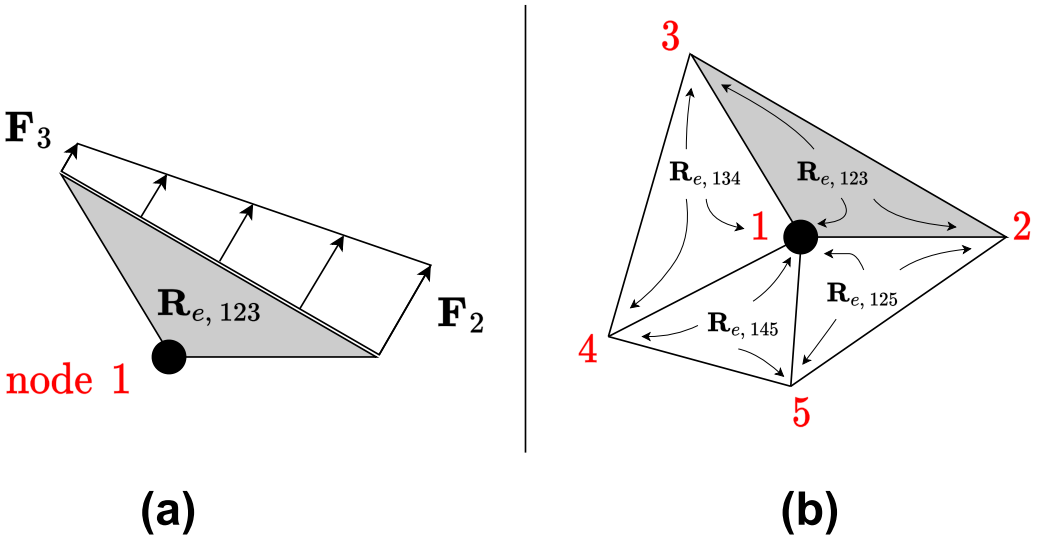


Figure 5.2: Illustration of cell residuals computation via trapezoidal rule (a) and of their scattering to nodes (b).

where $\boldsymbol{n}_{ij}$ is the outward normal to the face $\{i-j\}$ weighted by the face length itself (i.e., $\boldsymbol{n}_{ij} := \hat{\boldsymbol{n}}_{ij}\ \Delta s_{ij}$, where $\hat{\boldsymbol{n}}_{ij}$ is the outward unit vector normal to the face $\{i-j\}$). One can rearrange the flux terms shared among pairs of adjacent faces such that Eq. (5.6) is rewritten as

$$\boldsymbol{R}_e = \frac{1}{2V_e}\left[\underline{\boldsymbol{F}}_1 \cdot (\boldsymbol{n}_{31} + \boldsymbol{n}_{12}) + \underline{\boldsymbol{F}}_2 \cdot (\boldsymbol{n}_{23} + \boldsymbol{n}_{12}) + \underline{\boldsymbol{F}}_3 \cdot (\boldsymbol{n}_{31} + \boldsymbol{n}_{23})\right]. \quad (5.7)$$

Finally, once the face normals satisfy

$$\sum_{\{i-j\}\ \in\ \{\{1-2\},\{2-3\},\{3-1\}\}} \boldsymbol{n}_{ij} = \boldsymbol{0}, \quad (5.8)$$

one is left with the final formula below for cell residuals in 2D triangular meshes:

$$\boldsymbol{R}_e = -\frac{1}{2V_e} \sum_{k\in K_e,\ i,j\neq k} \underline{\boldsymbol{F}}_k \cdot \boldsymbol{n}_{ij}. \quad (5.9)$$

A relation similar to Eq. (5.9) can be found for 3D tetrahedral meshes. In practice, one works with the more generic formula:

$$\boldsymbol{R}_e = -\frac{1}{d\ V_e} \sum_{k\in K_e} \underline{\boldsymbol{F}}_k \cdot \boldsymbol{S}_k, \quad (5.10)$$

where $\boldsymbol{S}_k$ is the so-called *nodal normal* defined as

$$\boldsymbol{S}_k := \sum_{\text{face} \ni k} -\frac{d}{n_{\text{vert}}^{\text{face}}} \, \boldsymbol{n}_{\text{face}},$$

with $n_{\text{vert}}^{\text{face}}$ as the number of vertices in a face (equals 2 for triangles and 3 for tetrahedrons), $\boldsymbol{n}_{\text{face}}$ as the outward normal to the face weighted either by its length (in 2D) or surface (in 3D) and d as the number of spatial dimensions (2 in 2D, 3 in 3D).

The nodal residual $\boldsymbol{R}_j$ is computed by scattering residuals from all cells which the given node j belongs to:

$$\boldsymbol{R}_j := \frac{1}{V_j} \sum_{e \in \mathcal{D}_j} \underline{\boldsymbol{D}}_{j,e} \, V_e \, \boldsymbol{R}_e, \tag{5.11}$$

where $\mathcal{D}_j$ is the set of this node's neighboring cells, V_j is a control volume around the node defined as

$$V_j := \sum_{e \in \mathcal{D}_j} \frac{V_e}{n_{\text{vert}}^{\text{elem}}}, \tag{5.12}$$

with $n_{\text{vert}}^{\text{elem}}$ as the number of nodes in a cell and $\underline{\boldsymbol{D}}_{j,e}$ is a distribution matrix following the numerical scheme of choice:

- Central Differences (CD) - It is the simplest numerical scheme one can obtain from Eq. (5.11) by choosing

  $$\underline{\boldsymbol{D}}_{j,e}^{\text{CD}} := \frac{1}{n_{\text{vert}}^{\text{elem}}} \, \underline{\boldsymbol{Id}},$$

  where $\underline{\boldsymbol{Id}}$ is the identity matrix. It is unconditionally unstable when used with an Euler explicit first order time stepping scheme.

- Lax-Wendroff (LW) - It is a time–space combined discretization scheme: *forward in time* by using first order Euler explicit scheme, and *centered in space* by using second order central differences (along with an additional diffusive term to ensure stability). Its distribution matrix is expressed as

  $$\underline{\boldsymbol{D}}_{j,e}^{\text{LW}} := \frac{1}{n_{\text{vert}}^{\text{elem}}} \, \underline{\boldsymbol{Id}} - \frac{\Delta t}{2 \, d \, V_e} \, \underline{\mathcal{A}}_e \cdot \boldsymbol{S}_{j,e}, \tag{5.13}$$

  where $\underline{\mathcal{A}}_e$ is the Jacobian matrix of convective fluxes, averaged across each element.

Due to the Galerkin projection, Eq. (5.5) assumes the two-step discrete form below for the TTGC scheme shown in Eq. (3.16)-(3.17)-(3.18) (Lamarque, 2007):

$$\int_\Omega \delta\widetilde{\boldsymbol{U}}^n \psi_j \,\mathrm{d}\mathcal{V} = -(0.5-\gamma)\,\Delta t\,\boldsymbol{L}_j(\boldsymbol{U}^n) + \beta\,\Delta t^2\,\overline{\boldsymbol{L}}_j(\boldsymbol{U}^n) \tag{5.14}$$

$$\int_\Omega \delta\boldsymbol{U}^{n+1} \psi_j \,\mathrm{d}\mathcal{V} = -\Delta t\,\boldsymbol{L}_j(\widetilde{\boldsymbol{U}}^n) + \gamma\,\Delta t^2\,\overline{\boldsymbol{L}}_j(\boldsymbol{U}^n), \tag{5.15}$$

where $\delta\widetilde{\boldsymbol{U}}^n := \widetilde{\boldsymbol{U}}^n - \boldsymbol{U}^n$ and $\delta\boldsymbol{U}^{n+1} := \boldsymbol{U}^{n+1} - \boldsymbol{U}^n$, ψ_j is a hat function (as known as P1 element in finite-element terminology) and the operators $\boldsymbol{L}_j$ and $\overline{\boldsymbol{L}}_j$ are parts of the nodal residual that can be expressed in terms of the distribution matrices $\underline{\boldsymbol{D}}^{\mathrm{CD}}_{j,e}$ and $\underline{\boldsymbol{D}}^{\mathrm{LW}}_{j,e}$ as follows:

$$\boldsymbol{L}_j \equiv \sum_{e\in\mathcal{D}_j} \underline{\boldsymbol{D}}^{\mathrm{CD}}_{j,e}\; V_e\; \boldsymbol{R}_e \tag{5.16}$$

$$\overline{\boldsymbol{L}}_j \equiv \frac{2}{\Delta t} \sum_{e\in\mathcal{D}_j} (\underline{\boldsymbol{D}}^{\mathrm{CD}}_{j,e} - \underline{\boldsymbol{D}}^{\mathrm{LW}}_{j,e})\; V_e\; \boldsymbol{R}_e, \tag{5.17}$$

or, more explicitely,

$$\boldsymbol{L}_j \equiv \frac{1}{n^{\mathrm{elem}}_{\mathrm{vert}}} \sum_{e\in\mathcal{D}_j} V_e\; \boldsymbol{R}_e \tag{5.18}$$

$$\overline{\boldsymbol{L}}_j \equiv \frac{1}{d} \sum_{e\in\mathcal{D}_j} (\underline{\mathcal{A}}_e \boldsymbol{R}_e)\cdot \boldsymbol{S}_{j,e}, \tag{5.19}$$

with $\boldsymbol{S}_{j,e}$ as the nodal normal $\boldsymbol{S}_j$ with respect to the element e.

Finally, the left hand sides of Eq. (5.14)-(5.15) are expressed with the help of a mass matrix $\underline{\boldsymbol{\mathcal{M}}}$: $\underline{\boldsymbol{\mathcal{M}}}\,\delta\widetilde{\boldsymbol{U}}^n$ and $\underline{\boldsymbol{\mathcal{M}}}\,\delta\boldsymbol{U}^{n+1}$, respectively. Let $\boldsymbol{\Phi}$ be the resulting nodal residual vector for step 1 of TTGC. The linear system to be solved is

$$\underline{\boldsymbol{\mathcal{M}}}\,\delta\widetilde{\boldsymbol{U}}^n = \boldsymbol{\Phi}. \tag{5.20}$$

The Jacobi iterative method is applied, and is given by (Lamarque, 2007)

$$\delta\widetilde{\boldsymbol{U}}^{(p)} = \delta\widetilde{\boldsymbol{U}}^{(0)} - \underline{\boldsymbol{\mathcal{V}}}^{-1}(\underline{\boldsymbol{\mathcal{M}}} - \underline{\boldsymbol{\mathcal{V}}})\,\delta\widetilde{\boldsymbol{U}}^{(p-1)} \tag{5.21}$$

at iteration step p (with $\underline{\boldsymbol{\mathcal{V}}} := \mathrm{diag}(V_j)$ and $\delta\widetilde{\boldsymbol{U}}^{(0)} := \underline{\boldsymbol{\mathcal{V}}}^{-1}\boldsymbol{\Phi}$).

The next section explains how the numerical method is adapted to accept per-cell values of the parameter γ as done in Sec. 3.2.3 for the 1D case.

5.2 ML-TTGC for 2D Convection/Euler

The 2D Convection and Euler equations have convective fluxes respectively given by (see Eq. (2.1))

$$\boldsymbol{F}_{\text{Convection}}(U) := \begin{bmatrix} c_X\, U \\ c_Y\, U \end{bmatrix}, \tag{5.22}$$

$$\underline{\boldsymbol{F}}_{\text{Euler}}(\boldsymbol{U}) := \begin{bmatrix} \rho u & \rho v \\ \rho u^2 + p & \rho uv \\ \rho vu & \rho v^2 + p \\ u(E+p) & v(E+p) \end{bmatrix}, \text{ with } \boldsymbol{U} := \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \tag{5.23}$$

where c_X and c_Y are the components of the convection velocity field along x and y directions, and ρ, u, v, E and p are the scalar fields of density, x-velocity, y-velocity, (total) energy and pressure, respectively. The variables E and p are related by

$$E = \frac{p}{\gamma_{\text{gas}} - 1} + \frac{1}{2}\rho(u^2 + v^2), \tag{5.24}$$

where γ_{gas} is the adiabatic index of the considered fluid.

Locally tunable versions of the TTGC scheme exposed in the previous section must be derived (as done in Sec. 3.2.3). Eq. (5.14)-(5.15) are reformulated as

$$\int_\Omega \delta\widetilde{\boldsymbol{U}}^n \psi_j \,\mathrm{d}\mathcal{V} = -\Delta t\, \boldsymbol{L}_j^{\text{local}}(\boldsymbol{U}^n) + \beta\, \Delta t^2\, \overline{\boldsymbol{L}}_j(\boldsymbol{U}^n) \tag{5.25}$$

$$\int_\Omega \delta\boldsymbol{U}^{n+1} \psi_j \,\mathrm{d}\mathcal{V} = -\Delta t\, \boldsymbol{L}_j(\widetilde{\boldsymbol{U}}^n) + \Delta t^2\, \overline{\boldsymbol{L}}_j^{\text{local}}(\boldsymbol{U}^n), \tag{5.26}$$

where the operators $\boldsymbol{L}_j^{\text{local}}$ and $\overline{\boldsymbol{L}}_j^{\text{local}}$ include per-cell values of the scheme parameter γ. For the Convection case, these operators can be defined as follows :

$$L_j^{\text{local}} \equiv \frac{1}{3} \sum_{e \in \mathcal{D}_j} \left[(0.5 - \gamma_X)\, (V_e\, R_e)_X + (0.5 - \gamma_Y)\, (V_e\, R_e)_Y \right] \tag{5.27}$$

$$\overline{L}_j^{\text{local}} \equiv \frac{1}{2} \sum_{e \in \mathcal{D}_j} \left\{ \left(\frac{\mathcal{A}_e^{\text{Convection}} \cdot \boldsymbol{S}_{j,e}}{V_e} \right) \left[\gamma_X\, (V_e\, R_e)_X + \gamma_Y\, (V_e\, R_e)_Y \right] \right\}, \tag{5.28}$$

where γ_X and γ_Y are components of the parameter γ along x and y directions, respectively, and $(V_e\, R_e)_X := -\frac{1}{2} \sum_{k \in K_e} F_k^X S_k^X$ and $(V_e\, R_e)_Y := -\frac{1}{2} \sum_{k \in K_e} F_k^Y S_k^Y$.

One may notice that $(V_e\, R_e)_X$ and $(V_e\, R_e)_Y$ result from a splitting of the cell residuals computation that follows Eq. (5.10):

$$V_e\, R_e := -\frac{1}{2}\sum_{k\in K_e} \boldsymbol{F}_k \cdot \boldsymbol{S}_k = -\frac{1}{2}\sum_{k\in K_e} \left(F_k^X S_k^X + F_k^Y S_k^Y\right) \equiv (V_e\, R_e)_X + (V_e\, R_e)_Y. \tag{5.29}$$

For the Euler case,

$$\boldsymbol{L}_j^{\text{local}} \equiv \frac{1}{3}\sum_{e\in\mathcal{D}_j} \left[(0.5-\gamma)\, V_e\, \boldsymbol{R}_e\right] \tag{5.30}$$

$$\overline{\boldsymbol{L}}_j^{\text{local}} \equiv \frac{1}{2}\sum_{e\in\mathcal{D}_j} \left[(\gamma\, \underline{\boldsymbol{\mathcal{A}}}_e^{\text{Euler}} \boldsymbol{R}_e)\cdot \boldsymbol{S}_{j,e}\right]. \tag{5.31}$$

Here, a scalar value for the parameter γ is kept for scheme consistency.

Given the locally tunable versions of the TTGC scheme defined above, a Graph Net (GN) model identical to the one described in Sec. 4.3 is coupled to the solver as a surrogate model that provides optimal values of the parameter γ, except that it expects input features of nature and dimensionality specific to the studied problem. The input features of graph vertex (i.e., mesh element) e_i at each time step n are

$$\text{(2D Convection)} \qquad f_i^n := \left[U_{i1}^n,\ U_{i2}^n,\ U_{i3}^n,\ \gamma_{i,X}^{n-1},\ \gamma_{i,Y}^{n-1}\right]^\top, \tag{5.32}$$

$$\begin{aligned}\text{(2D Euler)} \qquad f_i^n := [&\rho_{i1}^n,\ \rho_{i2}^n,\ \rho_{i3}^n,\\ &(\rho u)_{i1}^n,\ (\rho u)_{i2}^n,\ (\rho u)_{i3}^n,\\ &(\rho v)_{i1}^n,\ (\rho v)_{i2}^n,\ (\rho v)_{i3}^n,\\ &E_{i1}^n,\ E_{i2}^n,\ E_{i3}^n,\\ &\gamma_i^{n-1}]^\top.\end{aligned} \tag{5.33}$$

In other words, γ values from the previous time step and the values of conservative variables at the three nodes composing the mesh cell (a triangle) are provided as information at vertices of the GN. Finally, edges exchanging messages between graph vertices (mesh cells) carry their per-cell defined CFL numbers (N_c), such that the input edge attribute g_{ij} of an edge sending a message from cell i to cell j is

$$g_{ij} := [N_{c,i}, N_{c,j}]^\top. \tag{5.34}$$

Training loss function is also similarly defined as a balance between the squared $l2$-norm of the error over time with respect to the available analytical solution and some Total Variation (TV) error measure specifically set to 2D problems based on the mesh-edge TV. It can be expressed as

$$\mathcal{L}(\boldsymbol{U}^0) := \frac{1}{N_{\text{step}}} \sum_{k=1}^{N_{\text{step}}} \theta \, L2\left(\boldsymbol{U}^k_{\text{pred}}\right) + (1-\theta) \left[\delta TV\left(\boldsymbol{U}^k_{\text{pred}}\right) + \left|\delta TV\left(\boldsymbol{U}^k_{\text{pred}}\right)\right|\right], \tag{5.35}$$

where N_{step} is the number of time steps to predict from a given initial condition $\boldsymbol{U}^0$, $\boldsymbol{U}^k_{\text{pred}}$ is the ML-TTGC output at time k, $\theta \in [0,1]$ is some scalarizing weight,

$$L2(\boldsymbol{U}) := \sum_i \left\|\boldsymbol{U}_i - \overline{\boldsymbol{U}}_i\right\|_2^2 \tag{5.36}$$

with $\overline{\boldsymbol{U}}$ the reference analytical solution, and $\delta TV(\boldsymbol{U}^{n+1}) := TV2(\boldsymbol{U}^{n+1}) - TV2(\overline{\boldsymbol{U}}^{n+1})$ such that

$$TV2(\boldsymbol{U}) := \frac{1}{2} \sum_{i,j} \|\boldsymbol{U}_i - \boldsymbol{U}_j\|_2^2 \tag{5.37}$$

for all pairs $\{i,j\}$ of adjacent mesh nodes. The reader may notice that the definitions used here for $L2$ and $TV2$ (coming from Eq. (5.36) and Eq. (5.37)) reduce to those from the 1D studies in Chapter 3 (Eq. (3.69) and Eq. (3.75), respectively), apart from the mesh geometry dependency. This is done to avoid related expensive computations during training and is assumed to have little influence on final results once only the values of the solution field $\boldsymbol{U}$ are subject to optimization, not the mesh geometry itself.

In what follows, ML-TTGC training/inference is performed on irregular triangular grids generated from perturbing (interior) regular mesh nodes positions. The node is placed somewhere within a circle around its original position, i.e.,

$$\Delta x_i = R_i \cos\phi_i \quad \text{and} \quad \Delta y_i = R_i \sin\phi_i$$

are the displacements imposed to the x and y coordinates of a given node i, respectively. The circle's radius R_i is drawn from the uniform distribution $\mathcal{U}(0, 0.35\,h)$, where h is the (minimal) regular grid spacing, and ϕ_i is a random angle drawn from the uniform distribution $\mathcal{U}(0, 2\pi)$. Fig. 5.3 illustrates irregular mesh generation for ML-TTGC evaluation.

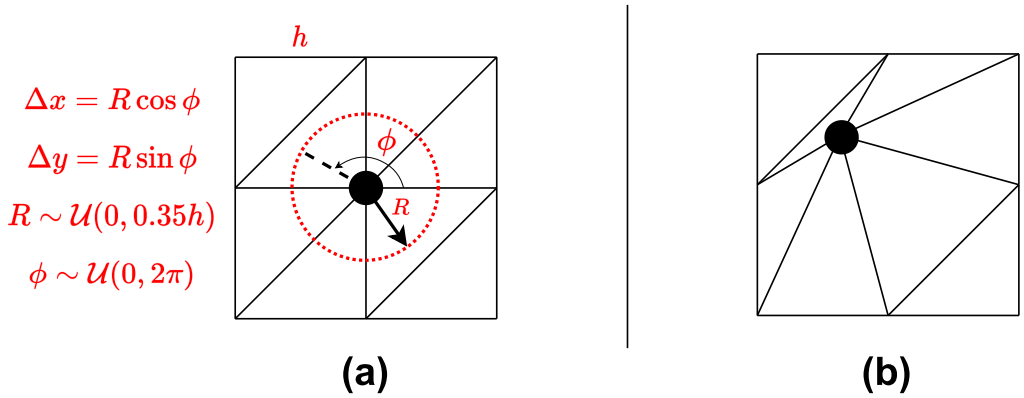


Figure 5.3: (a) Unstructured regular triangular grid and (b) its perturbed counterpart.

For the Convection case, the training set was composed of wave packets of the following form :

$$U^0(x,y) = \sin\left[k_{0,X}\left(x-x_0\right) + k_{0,Y}\left(y-y_0\right)\right] \exp\left\{-\alpha\left[(x-x_0)^2 + (y-y_0)^2\right]\right\} \quad ((x,y) \in [0,1]\times[0,1]), \tag{5.38}$$

where $k_{0,X}$ and $k_{0,Y}$ are the components along the x and y directions of some central wavenumber, α is the spectral spread and (x_0, y_0) is the spatial location of the peak of the WP. Training is performed on conditions similar to those of Sec. 4.4.1, with forecasting of $N_{\text{step}} = 16$ time steps of a set of 32 samples at wavenumbers $k_{0,X}h = k_{0,Y}h$ drawn from the uniform distribution $\mathcal{U}(0, 0.8)$ and $(x_0, y_0) = (0.5, 0.5)$ for all samples. The convection velocity $(c_X, c_Y) = (1., 1.)$ is also a constant for the training set. Fig. 5.4 compares inference outputs for one of the training samples from network models trained either on pure $l2$ loss function (i.e., $\theta = 1$) or on pure TVD one (i.e., $\theta = 0$) against the original scheme and the exact solution. One may notice a significant reduction in the spurious oscillations amplitude as a result of the training procedure, with the pure TVD trained model giving slightly noisier dynamics. The fact that the TTGC scheme is linearly unstable for the junction problem analyzed under the light of LTA in Sec. 3.2.4.1 might explain the reminiscent wiggles on ML-TTGC outputs: wave reflections originated from mesh irregularity will not be fully damped out.

For the Euler case, isentropic vortices of varying strengths and radii are

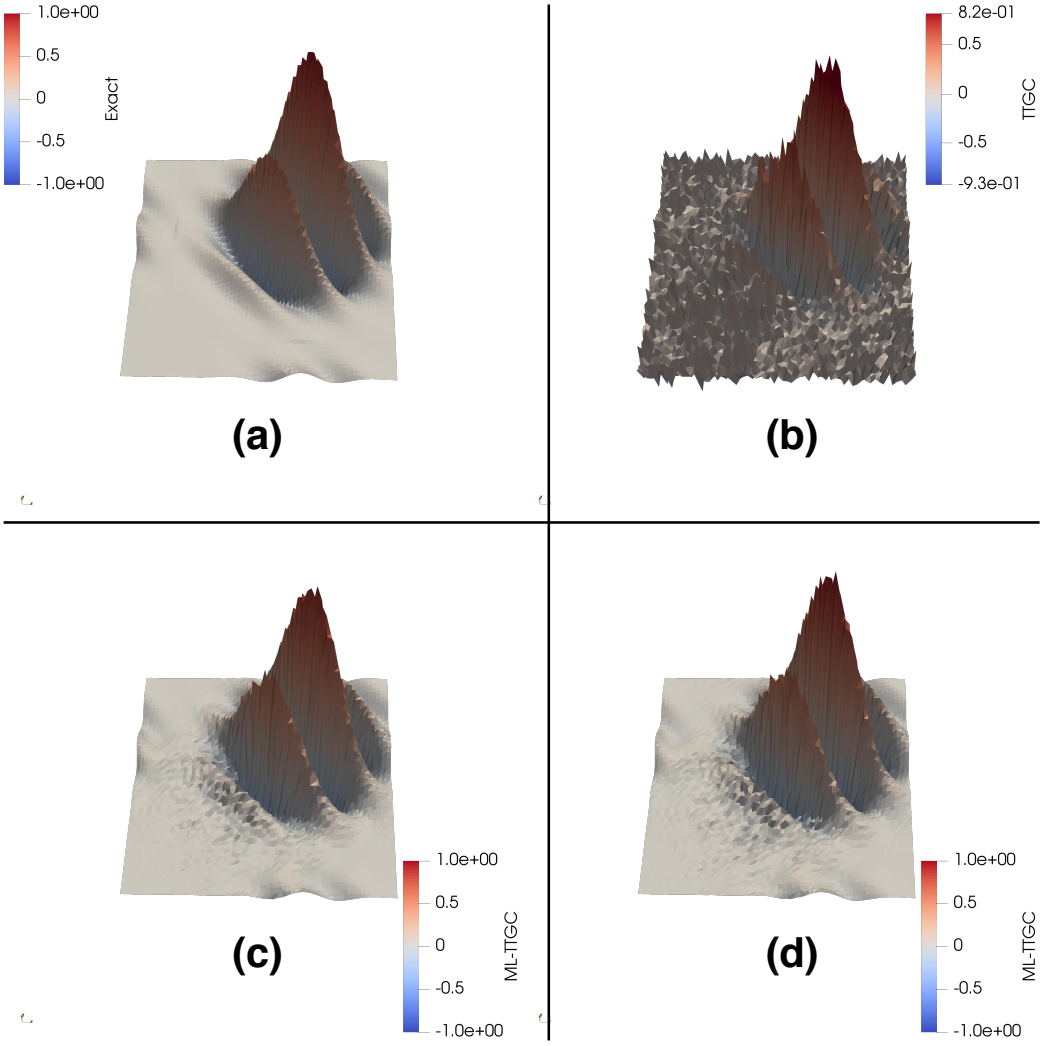


Figure 5.4: Surface plots of wave packet whose dynamics are governed by the 2D Convection equation. Time step n. 360. (a) Analytical solution, (b) Output from TTGC scheme, (c) Output from ML-TTGC scheme trained by pure $l2$ loss function, (d) Output from ML-TTGC scheme trained by pure TVD loss function.

chosen to compose the training set [1]. It consists of 32 samples with constant vortex core position $(x_0, y_0) = (0.5, 0.5)$, and initialized homogeneous temperature $T_\infty = 300\ K$ and pressure $p_\infty = 100,000\ Pa$. The Mach number $M_\infty := u_\infty / \sqrt{\gamma_{\text{gas}}\, R_{\text{gas}}\, T_\infty}$, the vortex strength β and the vortex radius R are drawn from the uniform distributions $\mathcal{U}(0.1, 0.6)$, $\mathcal{U}(0.1, 0.8)$ and $\mathcal{U}(0.05, 0.15)$, respectively. Values of R_{gas} and γ_{gas} are assumed to remain constant in the considered range of pressure and temperature values. The machine learning model is trained to forecast $N_{\text{step}} = 16$ time steps of the vortex convection, for which the analytical dynamics is given by

$$\overline{\boldsymbol{U}}(x, y, t) \equiv \overline{\boldsymbol{U}^0}(x - u_\infty t, y) \qquad (t > 0). \tag{5.39}$$

Fig. 5.5 presents inference results for a vortex from the training set. ML-TTGC was trained on pure $l2$ loss function only, as reportedly giving smoothier dynamics in the Convection case. Once again, wiggles are significantly damped in the predicted dynamics, approaching it from the analytical one. Despite the limited range of initial conditions explored during training, ML-TTGC also showed generalization capabilities for stabilizing the dynamics of a double shear layer initial condition it was never trained on, as exposed in Fig. 5.6.

[1] The reader is referred to Sec. 2.3.2 for a description of the vortex and of the double shear layer initial conditions used here.

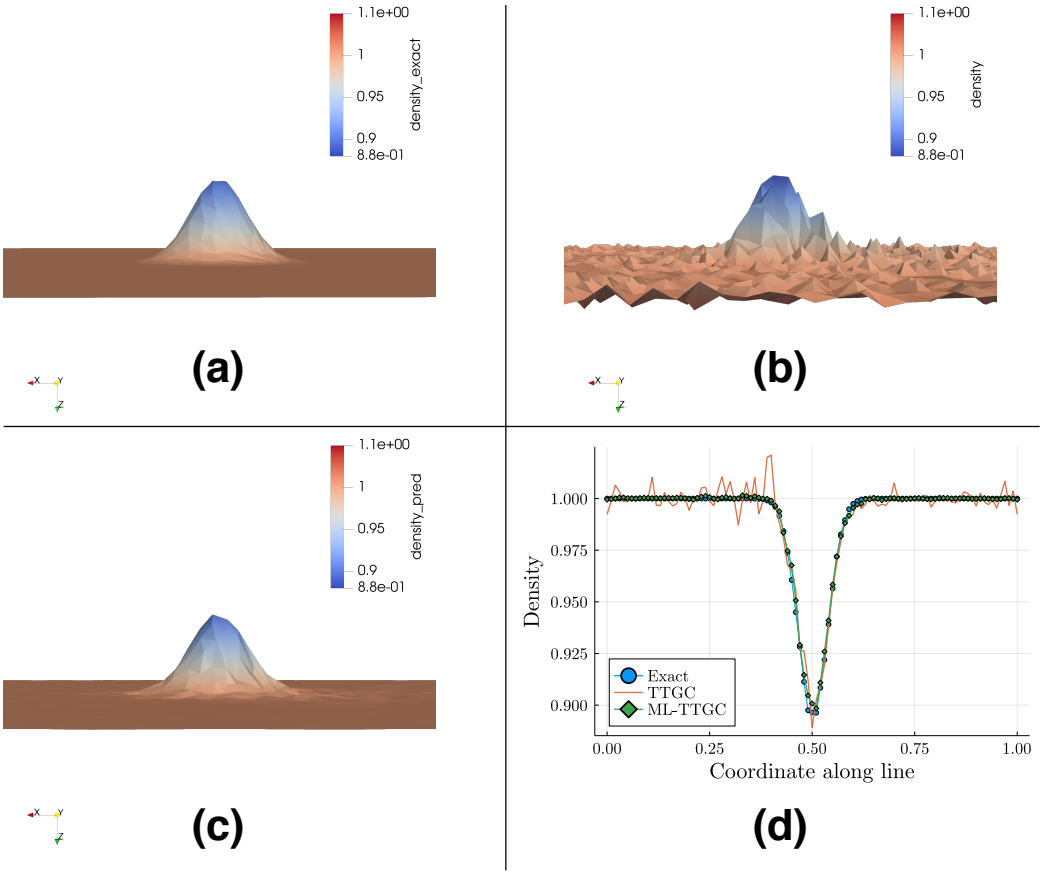


Figure 5.5: (a, b, c) Surface plots and (d) cut along $x = 0.5$ of isentropic vortex after completing one box turn (periodic conditions apply). (a) Analytical density field, (b) Output density field from TTGC scheme, (c) Output density field from ML-TTGC scheme trained on pure $l2$ loss function.

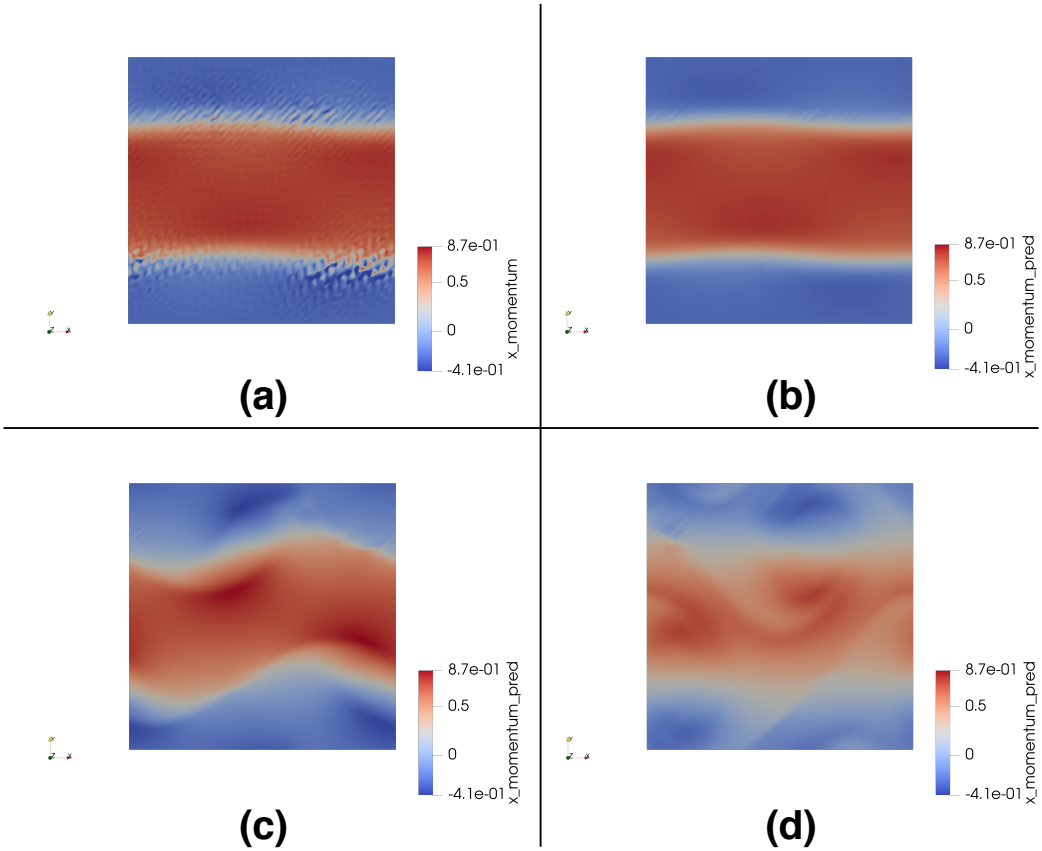


Figure 5.6: Dynamics of fluid's momentum component along the x direction (ρu) for double shear layer problem. (a) TTGC time step n. 600 (just before simulation crash), (b) ML-TTGC time step n. 600, (c) ML-TTGC time step n. 2400, (d) ML-TTGC time step n. 3600.

Conclusions and perspectives

In this work, a new spectral analysis for numerical schemes is introduced based on a local spatio-temporal discretization setting. The Global Spectral Analysis (GSA) (see Sec. 3.2.1) proposed by T. Sengupta, Ganeriwal, et al. (2003) is extended by interpreting the mesh as an electrical circuit whose transfer function is represented by the local numerical amplification factor. Within such an analogy, each mesh element acts as an impedance block providing resistance to the solution propagation. The resulting framework allows to quantify the local spectral properties of a given scheme, applicable to irregular meshes and non-linear systems. It is named Local Transfer-function Analysis (LTA). Taylor series or polynomial order of analysis is only applicable where the order of error makes sense (smooth or well resolved regions) (Brooks and T. J. Hughes, 1982). But practical Large-Eddy Simulations (LES) typically employ coarser, under-resolved meshes that target the capturing of the larger flow scales. Therefore, the need to design CFD schemes for bandwidth similar to Roe (2021) is emphasized. LTA being a spectral approach, it uses bandwidth as the basis of measuring accuracy and provides a complete overview of the error behavior even near high gradients and shocks. It provides practical guidance to mitigate errors that might otherwise trigger numerical instability or compromise solution accuracy.

Using the LTA framework, the optimal design of numerical methods can be formulated as a simple impedance matching problem. It was shown that the time-domain counterpart of such a problem corresponds to setting a balance between Total Variation Diminishing (TVD) and minimal norm of error conditions. Following this finding, an optimization problem based on impedance matching to redesign and tune the TTGC scheme of Colin and Rudgyard (2000) was set out. Differentiable programming methods to compute gradients required by the optimizer and find local optima for the free parameter of the numerical scheme were employed.

While effective, these methods require expensive gradient propagation through the adjoint solver for each new configuration. To alleviate this, a (finite) database of cases where the analytical solution is known is generated and a surrogate model is designed by training a Graph Neural Network (GNN) to predict the local optimal parameters for impedance matching. The data-driven numerical method that arises is termed ML-TTGC. A novel aspect of this architecture is that stability is inbuilt into the ML model through the LTA. ML models being approximate surrogates can lead to predictions in the numerically unstable region (for cases outside the training set), which motivated to constrain their output such that ML-TTGC always provides stable dynamics. Applications to the 1D Convection and Burgers' equations demonstrate both the interpolation and extrapolation of the matched impedance values for cases outside the training set.

Nevertheless, the reader might have noticed that validation sets as defined in Sec. 1.1.4 are absent from ML-TTGC trainings performed in Sec. 4.4 and in Sec. 5.2. This is on purpose, once the capacity to fully solve a restricted set of problems need to be assessed before evaluating generalization features of the coupled ML-PDE system. For instance, it was reported in Sec. 4.4.2 that ML-TTGC without any artificial dissipation term in the numerical scheme side was unable to solve the shock propagation proper to the inviscid Burgers' equation, even if trained on a single sample of a square wave initial condition. Similarly, as shown in Fig. 5.4, tuning of the γ parameter of the TTGC scheme in the convection of 2D wave packets was unable to fully damp all spurious oscillations due to mesh irregularities. Finally, Fig. 5.5 shows that vortex isentropic convection following ML-TTGC still present dispersion errors propagating through the wake of the vortex.

It becomes then clear that the choice of numerical scheme to be tuned by the coupled machine learning model is critical to the success of the proposed approach. Besides, from the experiments reported in Appendix B, it is also clear that an efficient differentiation tool that supports array mutation is required for reasonable performance of the training procedure, both runtime and memory-wise. Despite being able to handle mutating code from a high-level programming language (Julia), the Enzyme library (Sec. 1.3.5) does not offer the same maturity as older tools like Tapenade (Sec. 1.3.3) and may return unexpected gradient values. Future works will focus on the use of Tapenade to develop data-driven discretizations in the context of high-performance CFD, such that the resulting trained ML models can be easily integrated to mature CFD codes like AVBP for industrial applications.

Recently, large-scale applications of ML models in the field of weather forecasting have produced exciting results. Bi et al. (2023) proposed Pangu-Weather: an ML model that can forecast global weekly weather patterns at a significantly faster pace than conventional forecasting methods, while maintaining comparable accuracy. Lam et al. (2022) employed GNNs to generate a 10-day forecast in under 60 seconds. Their GNN model was named GraphCast. It outperformed the European Centre for Medium-Range Weather Forecasts' operational method on 90 percent of the 2760 variables. GraphCast surpassed Pangu-Weather on 99 percent of the 252 targets. Despite the results achieved by these models, they present risks for being exclusively data-driven (Ebert-Uphoff and Hilburn, 2023). One of these risks concerns extreme events, which have an increased likelihood in a changing climate. Besides, these events can lead to highly erratic forecasts, since the performance of fully data-driven ML models usually degrades under conditions not seen during training. Finally, these models do not consider dependencies between forecasting variables, as opposed to the traditional frameworks.

In order to circumvent the aforementioned issues, this work advocates using traditional numerical methods as a basis for the development of ML models in weather forecasting and CFD, in particular, and in physical sciences in general. These hybrid ML models can also produce high-accuracy results with the advantage of being interpretable (Murdoch et al., 2019). In regulated industries such as aerospace, compliance with safety standards and regulations is crucial. Interpretable ML models can facilitate compliance by providing explanations and justifications for the predictions provided, making it easier to demonstrate adherence to safety guidelines. The cited works of Bar-Sinai et al. (2019) and Kochkov et al. (2021) on data-driven spatial discretizations of enforced order of accuracy, Stevens and Colonius (2020) and Kossaczká et al. (2021) on data-driven perturbations of the parameters of WENO methods, and Bezgin, Schmidt, et al. (2021) on data-driven flux reconstruction for a finite-volume scheme are some examples from the state-of-the-art literature showing the potential of interpretable hybrid ML models in solving PDEs. Still recently, Kossaczká et al. (2023) employed neural networks to approximate the spatial truncation error in finite difference schemes. The distinguishing building block of the work in this thesis is employing spectral analysis on both time and space discretizations to better define stability constraints for the hybrid ML model, and also to interpret the optimization of the model as an impedance-matching problem to be solved at each time step.

Bibliography

Albright, J. and Shashkov, M. (2020). "Locally adaptive artificial viscosity strategies for Lagrangian hydrodynamics". *Computers & Fluids*, 205, p. 104580 (cited on p. 129).

artima - The Making of Python (2022) (cited on p. 192).

Baliga, B. R. and Patankar, S. V. (1980). "A new finite-element formulation for convection-diffusion problems". *Numerical Heat Transfer*, 3 (4), pp. 393–409 (cited on p. 107).

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. (2019). "Learning data-driven discretizations for partial differential equations". *Proceedings of the National Academy of Sciences*, 116(31), pp. 15344–15349 (cited on pp. 22, 23, 36, 69, 87, 130, 167).

Bartholomew-Biggs, M., Brown, S., Christianson, B., and Dixon, L. (2000). "Automatic differentiation of algorithms". *Journal of Computational and Applied Mathematics*, 124(1-2), pp. 171–190 (cited on pp. 44, 195).

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). "Relational inductive biases, deep learning, and graph networks". *arXiv:1806.01261 [cs, stat]*. arXiv: 1806.01261 (cited on pp. 36–38, 129).

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). "Automatic Differentiation in Machine Learning: a Survey". *Journal of Machine Learning Research*, 18(153), pp. 1–43 (cited on pp. 44, 48, 51).

Belbute-Peres, F. d. A., Economon, T. D., and Kolter, J. Z. (2020). "Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction" (cited on p. 39).

Bengio, Y., Simard, P., and Frasconi, P. (1994). "Learning long-term dependencies with gradient descent is difficult". *IEEE Transactions on Neural Networks*, 5(2), pp. 157–166 (cited on p. 77).

Berger, M. S. (1977). *Nonlinearity & Functional Analysis: Lectures on Nonlinear Problems in Mathematical Analysis*. Pure and Applied Mathematics, a Series of Monographs and Tex. Academic Press (cited on p. 200).

Bertoin, D., Bolte, J., Gerchinovitz, S., and Pauwels, E. (2021). "Numerical influence of ReLU'(0) on backpropagation". *Advances in Neural Information Processing Systems*, 34, pp. 468–479 (cited on p. 51).

Berzins, M. (2001). "Modified mass matrices and positivity preservation for hyperbolic and parabolic PDEs". *Communications in Numerical Methods in Engineering*, 17(9), pp. 659–666 (cited on p. 102).

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). "Julia: A Fresh Approach to Numerical Computing". *SIAM Review*, 59(1), pp. 65–98 (cited on pp. 54, 130, 195).

Bezgin, D. A., Buhendwa, A. B., and Adams, N. A. (2023). "JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows". *Computer Physics Communications*, 282, p. 108527 (cited on pp. 195, 205).

Bezgin, D. A., Schmidt, S. J., and Adams, N. A. (2021). "A data-driven physics-informed finite-volume scheme for nonclassical undercompressive shocks". *Journal of Computational Physics*, 437, p. 110324 (cited on pp. 22, 167).

Bezgin, D. A., Schmidt, S. J., and Adams, N. A. (2022). "WENO3-NN: A maximum-order three-point data-driven weighted essentially non-oscillatory scheme". *Journal of Computational Physics*, 452, p. 110920 (cited on p. 69).

Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., and Tian, Q. (2023). "Accurate medium-range global weather forecasting with 3D neural networks". *Nature*, 619(7970), pp. 533–538 (cited on p. 167).

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer (cited on pp. 20, 27).

Bochev, P., Peterson, K., and Gao, X. (2013). "A new Control Volume Finite Element Method for the stable and accurate solution of the drift–diffusion equations on general unstructured grids". *Computer Methods in Applied Mechanics and Engineering*, 254, pp. 126–145 (cited on p. 107).

Bochev, P., Peterson, K., and Perego, M. (2015). "A multiscale control volume finite element method for advection–diffusion equations". *International Journal for Numerical Methods in Fluids*, 77 (11), pp. 641–667 (cited on p. 107).

Book, D. L. (2005). *The Conception, Gestation, Birth, and Infancy of FCT*. Ed. by K. Dmitri, L. Rainald, and T. Stefan. Springer-Verlag, pp. 5–27 (cited on p. 121).

Bracewell, R. (1978). *The Fourier Transform and its Applications*. Second. Tokyo: McGraw-Hill Kogakusha, Ltd. (cited on p. 210).

Brooks, A. N. and Hughes, T. J. (1982). "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". *Computer Methods in Applied Mechanics and Engineering*, 32(1-3), pp. 199–259 (cited on pp. 132, 165).

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro,

M. T., and Zhang, Y. (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. arXiv:2303.12712 [cs] (cited on p. 20).

Burgers, J. (1948). "A Mathematical Model Illustrating the Theory of Turbulence". In: *Advances in Applied Mechanics*. Vol. 1. Elsevier, pp. 171–199 (cited on p. 57).

Cardesa, J., Hascoët, L., and Airiau, C. (2020). "Adjoint computations by algorithmic differentiation of a parallel solver for time-dependent PDEs". *Journal of Computational Science*, 45, p. 101155 (cited on p. 195).

Cardle, J. A. (1995). "A modification of the Petrov-Galerkin method for the transient convection-diffusion equation". *International Journal for Numerical Methods in Engineering*, 38(2), pp. 171–181 (cited on pp. 101, 102).

Cassagne, A., Boussuge, J.-F., Villedieu, N., Puigt, G., d'Ast, I., and Genot, A. (2015). "JAGUAR: a New CFD Code Dedicated to Massively Parallel High-Order LES Computations on Complex Geometry". In: *The 50th 3AF International Conference on Applied Aerodynamics (AERO 2015)*. Toulouse, France (cited on p. 195).

Charney, J. G., FjÖrtoft, R., and Neumann, J. V. (1950). "Numerical Integration of the Barotropic Vorticity Equation". *Tellus*, 2(4), pp. 237–254 (cited on p. 95).

Chellapilla, K., Puri, S., and Simard, P. (2006). "High Performance Convolutional Neural Networks for Document Processing". In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Ed. by G. Lorette. Université de Rennes 1. La Baule (France): Suvisoft (cited on p. 20).

Chen, J. and Revels, J. (2016). "Robust benchmarking in noisy environments". *arXiv e-prints*, arXiv:1608.04295 (cited on p. 203).

Chollet, F. (2018). *Deep learning with Python*. OCLC: ocn982650571. Shelter Island, New York: Manning Publications Co (cited on p. 41).

Cole, J. D. (1951). "On a Quasi-Linear Parabolic Equation Occurring in Aerodynamics". *Quarterly of Applied Mathematics*, 9(3), pp. 225–236 (cited on p. 57).

Colin, O. and Rudgyard, M. (2000). "Development of High-Order Taylor–Galerkin Schemes for LES". *Journal of Computational Physics*, 162(2), pp. 338–371 (cited on pp. 23, 94, 95, 104, 107, 132, 153, 165).

Courant, R., Friedrichs, K., and Lewy, H. (1928). "Über die partiellen Differenzengleichungen der mathematischen Physik". *Mathematische Annalen*, 100(1), pp. 32–74 (cited on p. 79).

Courant, R., Isaacson, E., and Rees, M. (1952). "On the solution of nonlinear hyperbolic differential equations by finite differences". *Communications on Pure and Applied Mathematics*, 5(3), pp. 243–255 (cited on p. 61).

Crumpton, P., Mackenzie, J., and Morton, K. (1993). "Cell Vertex Algorithms for the Compressible Navier-Stokes Equations". *Journal of Computational Physics*, 109(1), pp. 1–15 (cited on p. 153).

Dafermos, C. (2016). "Introduction to the Theory of Hyperbolic Conservation Laws". In: *Handbook of Numerical Analysis*. Vol. 17. Elsevier, pp. 1–18 (cited on p. 57).

Deardorff, J. W. (1970). "A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers". *Journal of Fluid Mechanics*, 41(2), pp. 453–480 (cited on p. 19).

Deconinck, H. and Ricchiuto, M. (2017). "Residual Distribution Schemes: Foundations and Analysis". In: *Encyclopedia of Computational Mechanics Second Edition*. Ed. by E. Stein, R. de Borst, and T. J. R. Hughes. Chichester, UK: John Wiley & Sons, Ltd, pp. 1–53 (cited on pp. 153, 205).

dnvdk (2022). *DNVDK*. Tumblr (cited on p. 29).

Donea, J., Quartapelle, L., and Selmin, V. (1987). "An analysis of time discretization in the finite element solution of hyperbolic problems". *Journal of Computational Physics*, 70(2), pp. 463–499 (cited on pp. 93, 94, 99).

Duraisamy, K., Iaccarino, G., and Xiao, H. (2019). "Turbulence Modeling in the Age of Data". *Annual Review of Fluid Mechanics*, 51(1), pp. 357–377 (cited on p. 20).

Ebert-Uphoff, I. and Hilburn, K. (2023). "The outlook for AI weather prediction". *Nature*, 619(7970), pp. 473–474 (cited on p. 167).

Economon, T. D., Palacios, F., Copeland, S. R., Lukaczyk, T. W., and Alonso, J. J. (2016). "SU2: An Open-Source Suite for Multiphysics Simulation and Design". *AIAA Journal*, 54(3), pp. 828–846 (cited on p. 39).

Ejaz, F., Hwang, L. K., Son, J., Kim, J.-S., Lee, D. S., and Kwon, B. (2022). "Convolutional neural networks for approximating electrical and thermal conductivities of Cu-CNT composites". *Scientific Reports*, 12(1), p. 13614 (cited on p. 36).

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. (2019). "Graph Neural Networks for Social Recommendation". In: *The World Wide Web Conference on - WWW '19*. San Francisco, CA, USA: ACM Press, pp. 417–426 (cited on p. 39).

Feng, Y., Liu, T., and Wang, K. (2020). "A Characteristic-Featured Shock Wave Indicator for Conservation Laws Based on Training an Artificial Neuron". *Journal of Scientific Computing*, 83(1), p. 21 (cited on p. 148).

Fornberg, B. (1988). "Generation of finite difference formulas on arbitrarily spaced grids". *Mathematics of Computation*, 51(184), pp. 699–706 (cited on p. 69).

Frostig, R., Johnson, M., and Leary, C. (2018). "Compiling machine learning programs via high-level tracing". In: (cited on p. 53).

Fuks, O. and Tchelepi, H. A. (2020). "Limitations of physics informed machine learning for nonlinear two-phase transport in porous media". *Jour-*

nal of Machine Learning for Modeling and Computing, 1(1), pp. 19–37 (cited on p. 21).

Fukushima, K. (1969). "Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements". *IEEE Transactions on Systems Science and Cybernetics*, 5(4), pp. 322–333 (cited on p. 50).

Girimaji, S. S. and Zhou, Y. (1995). "Spectrum and energy transfer in steady Burgers turbulence". *Physics Letters A*, 202 (4), pp. 279–287 (cited on p. 125).

Gladchuk, V. (2020). *A history of Machine Learning from 1940s to present days.* (Cited on p. 26).

Godlewski, E. and Raviart, P.-A. (2021). *Numerical Approximation of Hyperbolic Systems of Conservation Laws*. Vol. 118. Applied Mathematical Sciences. New York, NY: Springer New York (cited on pp. 59, 64).

Godunov, S. K. (1959). "A difference method for the numerical computation of discontinuous solutions of hydrodynamic equations". *Math. Sbornik*, 47(89), pp. 271–306 (cited on pp. 55, 61).

Goertzel, B. (2014). "Artificial General Intelligence: Concept, State of the Art, and Future Prospects". *Journal of Artificial General Intelligence*, 5(1), pp. 1–48 (cited on p. 20).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (cited on p. 41).

Gottlieb, S., Shu, C.-W., and Tadmor, E. (2001). "Strong Stability-Preserving High-Order Time Discretization Methods". *SIAM Review*, 43(1), pp. 89–112 (cited on p. 66).

Griewank, A. and El-Danaf, T. S. (2009). "Efficient accurate numerical treatment of the modified Burgers' equation". *Applicable Analysis*, 88(1), pp. 75–87 (cited on p. 120).

Griewank, A. and Walther, A. (2008). *Evaluating Derivatives*. Society for Industrial and Applied Mathematics (cited on pp. 46, 129, 197, 214).

Guermond, J.-L. and Pasquetti, R. (2013). "A correction technique for the dispersive effects of mass lumping for transport problems". *Computer Methods in Applied Mechanics and Engineering*, 253, pp. 186–198 (cited on p. 101).

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). "Array programming with NumPy". *Nature*, 585(7825), pp. 357–362 (cited on p. 193).

Harten, A. (1983). "High resolution schemes for hyperbolic conservation laws". *Journal of Computational Physics*, 49(3), pp. 357–393 (cited on p. 118).

Hascoët, L. and Pascual, V. (2013). "The Tapenade Automatic Differentiation tool: Principles, Model, and Specification". *ACM Transactions On Mathematical Software*, 39(3) (cited on p. 53).

Hirsch, C. (2007). *Numerical Computation of Internal and External Flows*. Elsevier (cited on pp. 19, 55, 56).

Hoi, S. C., Sahoo, D., Lu, J., and Zhao, P. (2021). "Online learning: A comprehensive survey". *Neurocomputing*, 459, pp. 249–289 (cited on p. 41).

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences*, 79(8), pp. 2554–2558 (cited on p. 30).

Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer feedforward networks are universal approximators". *Neural Networks*, 2(5), pp. 359–366 (cited on p. 20).

Hughes, T. W., Williamson, I. A. D., Minkov, M., and Fan, S. (2019). "Forward-Mode Differentiation of Maxwell's Equations". *ACS Photonics*, 6(11), pp. 3010–3016 (cited on p. 47).

Innes, M. (2018a). "Don't Unroll Adjoint: Differentiating SSA-Form Programs". *arXiv:1810.07951 [cs]*. arXiv: 1810.07951 version: 1 (cited on p. 54).

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M. C., Joy, N. M., Karmali, T., Pal, A., and Shah, V. (2018). "Fashionable Modelling with Flux". *CoRR*, abs/1811.01457 (cited on p. 54).

Innes, M. (2018b). "Flux: Elegant Machine Learning with Julia". *Journal of Open Source Software* (cited on p. 54).

Isaksson, M. (2020). *Four Common Types of Neural Network Layers* (cited on p. 193).

Jameson, A. (2017). "Origins and Further Development of the Jameson-Schmidt-Turkel Scheme". *AIAA Journal*, 55(5), pp. 1487–1510 (cited on p. 211).

JAX - The Sharp Bits — JAX documentation (2022) (cited on pp. 194, 195).

Jiang, G.-S. and Shu, C.-W. (1996). "Efficient Implementation of Weighted ENO Schemes". *Journal of Computational Physics*, 126(1), pp. 202–228 (cited on pp. 22, 67).

Johnson, S. G. (2007). *The NLopt nonlinear-optimization package*. https://github.com/stevengj/nlopt (cited on p. 130).

Jordan, M. I. and Mitchell, T. M. (2015). "Machine learning: Trends, perspectives, and prospects". *Science*, 349(6245), pp. 255–260 (cited on pp. 20, 39).

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). "Highly accurate protein structure

prediction with AlphaFold". *Nature*, 596(7873), pp. 583–589 (cited on p. 39).

Kanwal, R. P. (2004). *Generalized Functions*. Birkhäuser Boston (cited on p. 97).

Kapranidis, S. and Koo, R. (2008). "Variations of the Sliding Ladder Problem". *The College Mathematics Journal*, 39 (cited on p. 48).

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). "Physics-informed machine learning". *Nature Reviews Physics*, 3(6), pp. 422–440 (cited on p. 20).

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Perez, P. (2022). "Deep Reinforcement Learning for Autonomous Driving: A Survey". *IEEE Transactions on Intelligent Transportation Systems*, 23(6), pp. 4909–4926 (cited on p. 20).

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. (2021). "Machine learning–accelerated computational fluid dynamics". *Proceedings of the National Academy of Sciences*, 118(21), e2101784118 (cited on pp. 22, 23, 36, 69, 87, 130, 167, 195, 205).

Kolmogorov, A. N. (1941). "The Local Structure of Turbulence in Incompressible Viscous Fluid for Very Large Reynolds' Numbers". *Akademiia Nauk SSSR Doklady*, 30, pp. 301–305 (cited on p. 19).

Korteweg, D. J. and De Vries, G. (1895). "On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves". *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 39(240), pp. 422–443 (cited on p. 69).

Kossaczká, T., Ehrhardt, M., and Günther, M. (2021). "Enhanced fifth order WENO shock-capturing schemes with deep learning". *Results in Applied Mathematics*, 12, p. 100201 (cited on pp. 22, 130, 167).

— (2023). "Deep FDM: Enhanced finite difference methods by deep learning". *Franklin Open*, 4, p. 100039 (cited on p. 167).

Kraft, D. (1994). “Algorithm 733: TOMP–Fortran modules for optimal control calculations”. *ACM Transactions on Mathematical Software*, 20(3), pp. 262–281 (cited on p. 130).

Kuramoto, Y. (1978). “Diffusion-Induced Chaos in Reaction Systems”. *Progress of Theoretical Physics Supplement*, 64, pp. 346–367 (cited on p. 69).

Kurganov, A. and Tadmor, E. (2000). “New High-Resolution Central Schemes for Nonlinear Conservation Laws and Convection-Diffusion Equations”. *Journal of Computational Physics*, 160(1), pp. 241–282 (cited on pp. 66, 68).

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. (2022). “GraphCast: Learning skillful medium-range global weather forecasting” (cited on p. 167).

Lamarque, N. (2007). “Schémas numériques et conditions limites pour la simulation aux grandes échelles de la combustion diphasique dans les foyers d’hélicoptère.” PhD thesis. Institut National Polytechnique de Toulouse-INPT (cited on p. 156).

Lapeyre, C. J., Misdariis, A., Cazard, N., Veynante, D., and Poinsot, T. (2019). “Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates”. *Combustion and Flame*, 203, pp. 255–264 (cited on p. 36).

Lapointe, C., Swinburne, T. D., Thiry, L., Mallat, S., Proville, L., Becquart, C. S., and Marinica, M.-C. (2020). “Machine learning surrogate models for prediction of point defect vibrational entropy”. *Physical Review Materials*, 4(6), p. 063802 (cited on p. 129).

Laue, S. (2019). “On the Equivalence of Automatic and Symbolic Differentiation” (cited on p. 46).

Lax, P. (1971). “Shock Waves and Entropy”. In: *Contributions to Nonlinear Functional Analysis*. Elsevier, pp. 603–634 (cited on p. 59).

Lax, P. and Wendroff, B. (1960). "Systems of conservation laws". *Communications on Pure and Applied Mathematics*, 13(2), pp. 217–237 (cited on p. 62).

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*, 86(11), pp. 2278–2324 (cited on p. 33).

LeFloch, P. G. (2002). *Hyperbolic Systems of Conservation Laws*. Basel: Birkhäuser Basel (cited on p. 22).

Lele, S. K. (1992). "Compact finite difference schemes with spectral-like resolution". *Journal of Computational Physics*, 103(1), pp. 16–42 (cited on p. 21).

Lighthill, M. J. (1958). *An Introduction to Fourier Analysis and Generalised Functions*. Cambridge University Press (cited on p. 97).

Lighthill, M. J. and Whitham, G. B. (1955). "On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads". *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 229(1178), pp. 317–345 (cited on p. 57).

Linnainmaa, S. (1970). "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". PhD thesis. Master's Thesis (in Finnish), Univ. Helsinki (cited on pp. 40, 50).

— (1976). "Taylor expansion of the accumulated rounding error". *BIT*, 16(2), pp. 146–160 (cited on p. 50).

List of Nvidia graphics processing units (2022). Page Version ID: 1100220961 (cited on p. 195).

Löhner, R. (2008). *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods, Second Edition*. John Wiley & Sons, Ltd, pp. 1–519 (cited on pp. 106, 107).

Luo, H., Baum, J. D., and Lohner, R. (1994). "Edge-based finite element scheme for the Euler equations". *AIAA Journal*, 32 (6), pp. 1183–1190 (cited on p. 107).

Magiera, J., Ray, D., Hesthaven, J. S., and Rohde, C. (2020). "Constraint-aware neural networks for Riemann problems". *Journal of Computational Physics*, 409, p. 109345 (cited on p. 130).

Magnus, A. E. and Epton, M. A. (1981). *PAN AIR: A computer program for predicting subsonic or supersonic linear potential flows about arbitrary configurations using a higher order panel method. Volume 1: Theory document (version 1.1)*. Tech. rep. NASA (cited on p. 19).

Markidis, S. (2021). "The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?" *Frontiers in Big Data*, 4, p. 669097 (cited on p. 21).

Melland, P., Albright, J., and Urban, N. M. (2021). "Differentiable programming for online training of a neural artificial viscosity function within a staggered grid Lagrangian hydrodynamics scheme". *Machine Learning: Science and Technology*, 2(2), p. 025015 (cited on p. 130).

Metz, C. (2022). "Google Just Open Sourced the Artificial Intelligence Engine at the Heart of Its Online Empire". *Wired* () (cited on p. 53).

Minaee, S., Boykov, Y. Y., Porikli, F., Plaza, A. J., Kehtarnavaz, N., and Terzopoulos, D. (2021). "Image Segmentation Using Deep Learning: A Survey". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1 (cited on p. 20).

Minsky, M. and Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. 1st ed. MIT Press (cited on p. 30).

Moses, W. and Churavy, V. (2020). "Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 12472–12485 (cited on p. 54).

Müller, T., Mcwilliams, B., Rousselle, F., Gross, M., and Novák, J. (2019). "Neural Importance Sampling". *ACM Transactions on Graphics*, 38(5), pp. 1–19 (cited on p. 129).

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). "Definitions, methods, and applications in interpretable machine learning". *Proceedings of the National Academy of Sciences*, 116(44), pp. 22071–22080 (cited on p. 167).

Najafiyazdi, M., Mongeau, L., and Nadarajah, S. (2018). "Low-dissipation low-dispersion explicit Taylor-Galerkin schemes from the Runge-Kutta kernels". *International Journal of Aeroacoustics*, 17(1-2), pp. 88–113 (cited on p. 94).

Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. (2019). "Speech Recognition Using Deep Neural Networks: A Systematic Review". *IEEE Access*, 7, pp. 19143–19165 (cited on p. 20).

Nguyen-Fotiadis, N., McKerns, M., and Sornborger, A. (2022). "Machine learning changes the rules for flux limiters". *Physics of Fluids*, 34(8), p. 085136 (cited on p. 68).

OpenAI (2023). "GPT-4 Technical Report" (cited on p. 20).

Pérez Arroyo, C., Dombard, J., Duchaine, F., Gicquel, L., Martin, B., Odier, N., and Staffelbach, G. (2021). "Towards the Large-Eddy Simulation of a full engine: Integration of a 360 azimuthal degrees fan, compressor and combustion chamber. Part I: Methodology and initialisation". *Journal of the Global Power and Propulsion Society*, (May), pp. 1–16 (cited on p. 19).

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2021). "Learning Mesh-Based Simulation with Graph Networks". In: *International Conference on Learning Representations* (cited on pp. 129, 139).

Pilkington, M. and Keating, P. (2006). "The relationship between local wavenumber and analytic signal in magnetic interpretation". *GEOPHYSICS*, 71 (1), pp. L1–L3 (cited on p. 210).

Poinsot, T. and Veynante, D. (2011). *Theoretical and numerical combustion*. 3. ed. Toulouse: CNRS (cited on p. 19).

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. (2019). "On the Spectral Bias of Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 5301–5310 (cited on p. 21).

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". *Journal of Computational Physics*, 378, pp. 686–707 (cited on p. 20).

Rajpoot, M. K., Sengupta, T. K., and Dutt, P. K. (2010). "Optimal time advancing dispersion relation preserving schemes". *Journal of Computational Physics*, 229 (10), pp. 3623–3651 (cited on p. 21).

Ramadan, M. A. and El-Danaf, T. S. (2005). "Numerical treatment for the modified burgers equation". *Mathematics and Computers in Simulation*, 70(2), pp. 90–98 (cited on p. 120).

Rayleigh, J. (1899). *Scientific Papers*. Scientific Papers v. 1. University Press (cited on pp. 96, 101).

Reid, W. H. (1956). "On the transfer of energy in Burgers' model of turbulence". *Applied Scientific Research*, 6(2-3), pp. 85–91 (cited on p. 125).

Release alpha-1 release · pytorch/pytorch (2022) (cited on p. 53).

Richard S. Sutton, A. G. B. (2018). *Reinforcement Learning: An Introduction, 2nd Edition*. 2nd ed. Bradford Books (cited on p. 26).

Roe, P. (2021). "Designing CFD methods for bandwidth—A physical approach". *Computers & Fluids*, 214, p. 104774 (cited on p. 165).

Roig, B. (2007). “One-step Taylor–Galerkin methods for convection–diffusion problems”. *Journal of Computational and Applied Mathematics*, 204(1), pp. 95–101 (cited on pp. 91, 99).

Rosenblatt, F. (1958). “The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychological Review*, 65(6), pp. 386–408 (cited on p. 26).

— (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books (cited on pp. 27, 29, 30).

Roux, A., Gicquel, L., Reichstadt, S., Bertier, N., Staffelbach, G., Vuillot, F., and Poinsot, T. (2010). “Analysis of unsteady reacting flows and impact of chemistry description in Large Eddy Simulations of side-dump ramjet combustors”. *Combustion and Flame*, 157(1), pp. 176–191 (cited on p. 95).

Rumelhart, D. E., McClelland, J. L., Group, P. R., et al. (1988). *Parallel distributed processing*. Vol. 1. IEEE New York (cited on p. 31).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). “Learning representations by back-propagating errors”. *Nature*, 323(6088), pp. 533–536 (cited on pp. 31, 40).

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). “ImageNet Large Scale Visual Recognition Challenge”. *International Journal of Computer Vision*, 115(3), pp. 211–252 (cited on p. 40).

Sagaut, P. (2006). *Large eddy simulation for incompressible flows: an introduction*. 3rd ed. Scientific computation. Berlin ; New York: Springer (cited on pp. 19, 69).

Sagaut, P., Suman, V., Sundaram, P., Rajpoot, M., Bhumkar, Y., Sengupta, S., Sengupta, A., and Sengupta, T. (2023). “Global spectral analysis: Review of numerical methods”. *Computers & Fluids*, 261, p. 105915 (cited on p. 95).

Samuel, A. L. (2000). "Some studies in machine learning using the game of checkers". *IBM Journal of Research and Development*, 44(1.2), pp. 206–226 (cited on p. 26).

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. (2020). "Learning to Simulate Complex Physics with Graph Networks". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org (cited on p. 129).

Sengupta, S., N.A., S., Mohanamuraly, P., Staffelbach, G., and Gicquel, L. (2022). "Global spectral analysis of the Lax–Wendroff-central difference scheme applied to Convection–Diffusion equation". *Computers & Fluids*, 242, p. 105508 (cited on p. 123).

Sengupta, T. K. and Dipankar, A. (2004). "A Comparative Study of Time Advancement Methods for Solving Navier-Stokes Equations". *Journal of Scientific Computing*, 21(2), pp. 225–250 (cited on pp. 22, 96, 98).

Sengupta, T., Dipankar, A., and Sagaut, P. (2007). "Error dynamics: Beyond von Neumann analysis". *Journal of Computational Physics*, 226(2), pp. 1211–1218 (cited on pp. 97, 98).

Sengupta, T., Ganeriwal, G., and De, S. (2003). "Analysis of central and upwind compact schemes". *Journal of Computational Physics*, 192(2), pp. 677–694 (cited on pp. 95, 165).

Sengupta, T. (2013). *High Accuracy Computing Methods: Fluid Flows and Wave Phenomena*. 1st ed. Cambridge University Press (cited on pp. 22, 92, 98).

Sengupta, T. K., Bhumkar, Y. G., Rajpoot, M. K., Suman, V., and Saurabh, S. (2012). "Spurious waves in discrete computation of wave phenomena and flow problems". *Applied Mathematics and Computation*, 218(18), pp. 9035–9065 (cited on p. 55).

Sengupta, T. K., Ganerwal, G., and Dipankar, A. (2004). "High Accuracy Compact Schemes and Gibbs' Phenomenon". *Journal of Scientific Computing*, 21(3), pp. 253–268 (cited on p. 55).

Sengupta, T. K., Rajpoot, M. K., and Bhumkar, Y. G. (2011). "Space-time discretizing optimal DRP schemes for flow and wave propagation problems". *Computers & Fluids*, 47 (1), pp. 144–154 (cited on p. 21).

Sengupta, T. K., Rajpoot, M. K., Saurabh, S., and Vijay, V. (2011). "Analysis of anisotropy of numerical wave solutions by high accuracy finite difference methods". *Journal of Computational Physics*, 230(1), pp. 27–60 (cited on p. 109).

Sengupta, T. K. and Sengupta, A. (2016). "A new alternating bi-diagonal compact scheme for non-uniform grids". *Journal of Computational Physics*, 310, pp. 1–25 (cited on pp. 98, 109).

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: from theory to algorithms*. New York, NY, USA: Cambridge University Press (cited on p. 77).

Sharma, N., Sengupta, A., Rajpoot, M., Samuel, R. J., and Sengupta, T. K. (2017). "Hybrid sixth order spatial discretization scheme for non-uniform Cartesian grids". *Computers & Fluids*, 157, pp. 208–231 (cited on p. 109).

Sharma, P., Chung, W. T., Akoush, B., and Ihme, M. (2023). "A Review of Physics-Informed Machine Learning in Fluid Mechanics". *Energies*, 16(5), p. 2343 (cited on p. 21).

Shefter, M. and Rosales, R. R. (1999). "Quasiperiodic Solutions in Weakly Nonlinear Gas Dynamics. Part I. Numerical Results in the Inviscid Case". *Studies in Applied Mathematics*, 103(4), pp. 279–337 (cited on p. 57).

Shu, C.-W. (1999). "High Order ENO and WENO Schemes for Computational Fluid Dynamics". In: *High-Order Methods for Computational Physics*. Ed. by M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick, T. J. Barth, and H. Deconinck. Vol. 9. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 439–582 (cited on p. 85).

— (2009). "High Order Weighted Essentially Nonoscillatory Schemes for Convection Dominated Problems". *SIAM Review*, 51(1), pp. 82–126 (cited on pp. 68, 85).

Sivashinsky, G. (1977). "Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations". *Acta Astronautica*, 4(11-12), pp. 1177–1206 (cited on p. 69).

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., Zhang, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., and Catanzaro, B. (2022). "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model" (cited on p. 50).

Sola, J. and Sevilla, J. (1997). "Importance of input data normalization for the application of neural networks to complex industrial problems". *IEEE Transactions on Nuclear Science*, 44(3), pp. 1464–1468 (cited on pp. 77, 85).

Speelpenning, B. (1980). "Compiling fast partial derivatives of functions given by algorithms" (cited on pp. 46, 50).

Steinkraus, D., Buck, I., and Simard, P. (2005). "Using GPUs for machine learning algorithms". In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, pp. 1115–1120 (cited on p. 20).

Stevens, B. and Colonius, T. (2020). "Enhancement of shock-capturing methods via machine learning". *Theoretical and Computational Fluid Dynamics*, 34(4), pp. 483–496 (cited on pp. 22, 130, 167).

Strikwerda, J. C. (2004). *Finite difference schemes and partial differential equations*. 2. ed. Philadelphia, Pa: Society for Industrial and Applied Mathematics (cited on p. 60).

Stuart Russell, P. N. (2010). *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Series in Artificial Intelligence. Prentice Hall (cited on p. 31).

Tam, C. K. and Webb, J. C. (1993). "Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics". *Journal of Computational Physics*, 107(2), pp. 262–281 (cited on p. 21).

The LLVM Compiler Infrastructure Project (2022) (cited on p. 54).

TIOBE Index (2022) (cited on p. 192).

Toro, E. F. (2009). *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. 3rd ed. OCLC: ocn401321914. Dordrecht ; New York: Springer (cited on p. 57).

Ucar, Y., Yagmurlu M., N., and Tasbozan, O. (2017). "Numerical Solutions of the Modified Burgers' Equation by Finite Difference Methods". *Journal of Applied Mathematics, Statistics and Informatics*, 13(1), pp. 19–30 (cited on pp. 120, 214).

Vajjala, K. S., Sengupta, T. K., and Mathur, J. (2020). "Effects of numerical anti-diffusion in closed unsteady flows governed by two-dimensional Navier-Stokes equation". *Computers & Fluids*, 201, p. 104479 (cited on p. 125).

Van Der Malsburg, C. (1986). "Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms". In: *Brain Theory*. Ed. by G. Palm and A. Aertsen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 245–248 (cited on p. 27).

Van Leer, B. (1979). "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method". *Journal of Computational Physics*, 32(1), pp. 101–136 (cited on pp. 65, 66).

VI1 Vortex transport by uniform flow | HiOCFD5 (2022) (cited on p. 78).

Vichnevetsky, R. and Bowles, J. B. (1982). *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*. Society for Industrial and Applied Mathematics (cited on pp. 113, 116).

Wagner, C. A., Hüttl, T., and Sagaut, P., eds. (2007). *Large-eddy simulation for acoustics*. Cambridge aerospace series 20. OCLC: ocm70131103. New York: Cambridge University Press (cited on p. 19).

Wan, X. (2019). "Influence of feature scaling on convergence of gradient iterative algorithm". *Journal of Physics: Conference Series*, 1213(3), p. 032021 (cited on p. 77).

Wang, R. and Spiteri, R. J. (2007). "Linear Instability of the Fifth-Order WENO Method". *SIAM Journal on Numerical Analysis*, 45(5), pp. 1871–1901 (cited on p. 68).

Wang, S., Yu, X., and Perdikaris, P. (2022). "When and why PINNs fail to train: A neural tangent kernel perspective". *Journal of Computational Physics*, 449, p. 110768 (cited on p. 21).

Waterson, N. and Deconinck, H. (2007). "Design principles for bounded higher-order convection schemes – a unified approach". *Journal of Computational Physics*, 224(1), pp. 182–207 (cited on p. 68).

Wengert, R. E. (1964). "A Simple Automatic Derivative Evaluation Program". *Commun. ACM*, 7(8), pp. 463–464 (cited on p. 46).

Werbos, P. (1974). "Beyond regression: new tools for prediction and analysis in the behavioral sciences". *Ph. D. dissertation, Harvard University* (cited on pp. 40, 50).

White, B. W. (1963). "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms". *The American Journal of Psychology*, 76(4), p. 705 (cited on p. 30).

Woo, G., Liu, C., Sahoo, D., Kumar, A., and Hoi, S. (2022). "ETSformer: Exponential Smoothing Transformers for Time-series Forecasting". *arXiv:2202.01381 [cs]*. arXiv: 2202.01381 (cited on p. 130).

Yegulalp, S. (2020). *What is LLVM? The power behind Swift, Rust, Clang, and more* (cited on p. 54).

Zhang, Y.-T. and Shu, C.-W. (2016). “ENO and WENO Schemes”. In: *Handbook of Numerical Analysis*. Vol. 17. Elsevier, pp. 103–122 (cited on pp. 56, 67).

Appendix A

Example of Tapenade-generated adjoint program

Below is a code snippet illustrating the Fortran subroutine `FUNC_B` as the Tapenade-generated adjoint of the subroutine `FUNC` which computes

$$z(\boldsymbol{x}, \boldsymbol{y}) := \sum_i \left[\sin(x_i + 2y_i)\, \cos(x_i - \sin y_i)\right],$$

for some $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{y} \in \mathbb{R}^N$, $N \in \mathbb{N}$:

```
1  MODULE SOLVER_DIFF
2
3    USE ISO_FORTRAN_ENV
4    IMPLICIT NONE
5
6  CONTAINS
7  !  Differentiation of func in reverse (adjoint) mode
8  !  (with options i4 dr8 r4):
9  !   gradient     of useful results: z
10 !   with respect to varying inputs: x y z
11 !   RW status of diff variables: x:out y:out z:in-zero
12   SUBROUTINE FUNC_B(n, x, xb, y, yb, z, zb)
13     IMPLICIT NONE
14     INTEGER(int64), INTENT(IN) :: n
```

```
15      REAL(real64), INTENT(IN)   :: x(n)
16      REAL(real64)               :: xb(n)
17      REAL(real64), INTENT(IN)   :: y(n)
18      REAL(real64)               :: yb(n)
19      REAL(real64)               :: z
20      REAL(real64)               :: zb
21      INTRINSIC SIN
22      INTRINSIC COS
23      INTRINSIC SUM
24      REAL(real64), DIMENSION(n) :: temp
25      REAL(real64), DIMENSION(n) :: tempb
26      REAL(real64), DIMENSION(n) :: tempb0
27      xb = 0.0
28      yb = 0.0
29      temp = x - SIN(y)
30      tempb = COS(x+2*y)*COS(temp)*zb
31      tempb0 = -(SIN(temp)*SIN(x+2*y)*zb)
32      xb = tempb0 + tempb
33      yb = 2*tempb - COS(y)*tempb0
34      zb = 0.0
35    END SUBROUTINE FUNC_B
36
37    SUBROUTINE FUNC(n, x, y, z)
38      IMPLICIT NONE
39      INTEGER(int64), INTENT(IN) :: n
40      REAL(real64), INTENT(IN)   :: x(n)
41      REAL(real64), INTENT(IN)   :: y(n)
42      REAL(real64), INTENT(OUT)  :: z
43      INTRINSIC SIN
44      INTRINSIC COS
45      INTRINSIC SUM
46      z = SUM(SIN(x+2*y)*COS(x-SIN(y)))
47    END SUBROUTINE FUNC
48
49  END MODULE SOLVER_DIFF
```

Appendix B

The mutation problem

Chapter 1 introduced ML-related techniques that are employed in the developments proper to data-driven discretizations. Among them, Algorithmic Differentiation (AD) computes the gradients required to train ML models. The main AD frameworks used in ML pipelines, namely, TensorFlow and PyTorch, have found great adoption for their relative ease of use and scaling capabilities. These frameworks fail to deliver performant differentiation for generic control flow proper to coupled ML-PDE systems, though. This is due to their lack of support to array mutation, i.e., modifying the contents of an array after it has been created. This appendix is dedicated to quantifying the impact of this limitation in chains of operations one can find in PDE solvers for CFD. Building differentiable solvers for unstructured CFD in frameworks that lack support to array mutation is prohibitively expensive. To the extent of the author's knowledge, a quantitative study of this problem has not yet been reported in the literature.

B.1 Presentation of the problem

Guido Van Rossum's efforts in late 1980s to create an easily-extensible, high-level scripting language (*artima - The Making of Python* 2022) helped turning Python into the most popular programming language as of 2022 (according to the *TIOBE Index* (2022)). Despite suffering from lack of native support to performant numerical computing, its ease of use has motivated developers

to build high-level APIs like the NumPy library (Harris et al., 2020) on top of C/C++ procedures that execute fast linear algebra computations. In the realm of machine learning, the same reason led big companies like Google and Meta to launch Python APIs for TensorFlow, JAX and PyTorch. They can enjoy from the fact that most neural network architectures used in image recognition, natural language and time series processing boil down to a relatively small subset of linear algebra kernels whose forward/backward passes can be highly optimized (Isaksson, 2020). Nevertheless, one common limitation faced by these popular frameworks is the lack of full support to array mutation, as illustrated by the code snippets below:

TensorFlow :

```
>>> import tensorflow as tf
>>> tf.__version__
'2.9.1'
>>> x = tf.random.uniform([10])
>>>
>>> x[0] += 1.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tensorflow.python.framework.ops.EagerTensor'
object does not support item assignment
```

PyTorch :

```
>>> import torch
>>> torch.__version__
'1.11.0'
>>> x = torch.rand(10, requires_grad=True)
>>>
>>> x[0] += 1.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: a view of a leaf Variable that requires grad
is being used in an in-place operation.
```

JAX :

```
>>> import jax
>>> jax.__version__
'0.3.13'
>>> key = jax.random.PRNGKey(0)
>>> x   = jax.random.normal(key, (10,))
>>>
>>> x[0] += 1.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "[...]/jax/_src/numpy/lax_numpy.py", line 4568,
  in _unimplemented_setitem
  raise TypeError(msg.format(type(self)))
TypeError: '<class 'jaxlib.xla_extension.DeviceArray'>'
object does not support item assignment.
JAX arrays are immutable.
Instead of ``x[idx] = y'', use ``x = x.at[idx].set(y)''
or another .at[] method
```

Apparent workarounds are provided by certain specific methods, like `tf.tensor_scatter_nd_add` in TensorFlow, `.scatter_add_` in PyTorch, and `.at[]` in JAX. None of them perform actual mutation of the original array, though [1]. They rather execute re-assignements by creating new arrays of same size. This is illustrated in Fig. B.1. Such a limitation is justified by the fact that reverse mode AD in its simplest form requires all intermediate variables of the forward pass to be stored for the execution of the backward pass. This is what has probably led TensorFlow, PyTorch and, to some extent, JAX to not support mutation from the beginning of their developments:

> “ Allowing mutation of variables in-place makes program analysis and transformation difficult. JAX requires that programs are

[1] In JAX, array mutation (an in-place operation) is guaranteed to be performed whenever inside a JIT-compiled function. However, JAX' sharp bits (*JAX - The Sharp Bits — JAX documentation* 2022) may turn the coding experience as a whole quite cumbersome.

> pure functions. " (From *JAX - The Sharp Bits — JAX documentation* (2022))

An excessive number of reassignments in workflows typical to numerical schemes for unsteady problems (where, for instance, the enforcement of boundary conditions would require array mutation at each time step) eventually leads to high memory pressure, which might be one of the main bottlenecks for the development of differentiable PDE solvers. The literature contains works pointing to memory consumption issues for adjoint codes. In their 2021 paper, Kochkov et al. (2021) reported using *gradient checkpointing* [2] at each time step of a JAX-based CFD solver [3] to avoid prohibitive memory requirements during training. Cardesa et al. (2020) also used checkpointing to alleviate memory footprint of the adjoint of the CFD solver JAGUAR (Cassagne et al., 2015). However, while Cardesa et al. (2020) could differentiate very long time-stepping sequences (*ca.* 10^6 time steps) in double precision arithmetics, Kochkov et al. (2021) were limited to only 32 time steps forecasting during training of their coupled ML-PDE system, still using single precision arithmetics. This is mainly due to the native inability of JAX to handle mutating code. More recently, Bezgin, Buhendwa, et al. (2023) claimed that the largest limitation of their differentiable solver in JAX [4] was the available storage space of the GPU, which was an NVIDIA RTX A6000 card with 48 GB of memory, the largest of its series (*List of Nvidia graphics processing units* 2022).

Sec. B.2-B.3 illustrate the mutation problem with applications of the reverse mode of AD. In each case, a given quantity of interest is computed by two programs: one that employs array mutation whereas the other just performs reassignments. Both programs are written in the Julia programming language (Bezanson et al., 2017). Finally, the performances of their adjoints are benchmarked and compared.

B.2 Analysis of a discretization operator

The first problem consists in building a discretization operator of a scalar quantity u over a one dimensional uniform mesh. By means of Taylor expan-

[2] This technique is not detailed here; the interested reader is referred to Sec. 5.5 of Bartholomew-Biggs et al. (2000) for a description.

[3] Available at https://github.com/google/jax-cfd

[4] Available at https://github.com/tumaer/jaxfluids

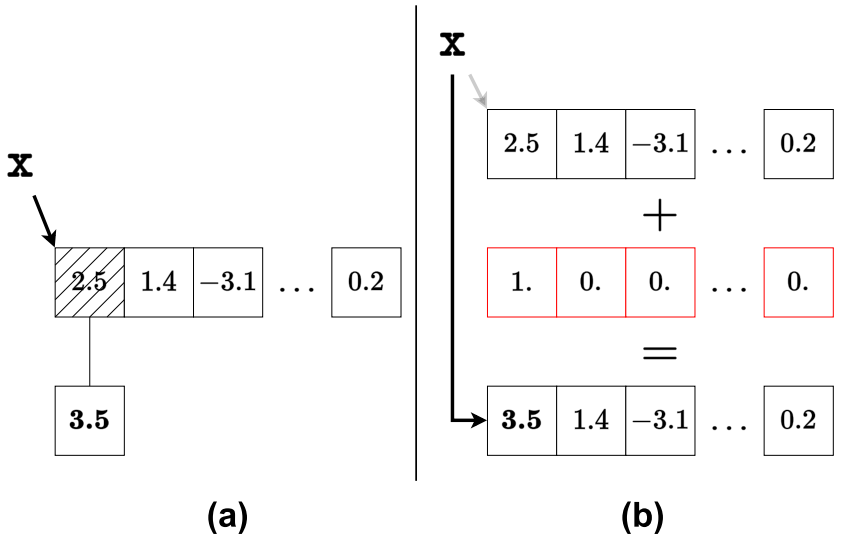


Figure B.1: Illustration of `x[0] += 1.` operation done either (a) in-place (actual mutation of `x`) or (b) by reassignment of `x`. Represented by red boxes, additional memory in the forward pass is required in (b) when compared to (a).

sions, one can define the derivative

$$\left.\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right|_{x=x_i} := \frac{1}{2\Delta x}(u_{i-2} - 4u_{i-1} + 3u_i) + \mathcal{O}(\Delta x^2),$$

where Δx is the mesh spacing, x_i ($i \in [\![1, N]\!]$) is some of the N mesh nodes and $u_i := u(x = x_i)$. Programs that compute the squared $l2$-norm of the operator

$$\mathrm{d}u(x)|_{x=x_i} := u_{i-2} - 4u_{i-1} + 3u_i \tag{B.1}$$

will be crafted. For the sake of simplicity, the constant $2\Delta x$ was ignored. Here, periodic conditions at the boundaries are considered, so that

$$\mathrm{d}u(x_1) := u_{N-1} - 4u_N + 3u_1, \tag{B.2}$$
$$\mathrm{d}u(x_2) := u_N - 4u_1 + 3u_2. \tag{B.3}$$

Fig. B.2 represents the mutating and non-mutating programs. One can notice that the forward pass of `du_reassignment` (Fig. B.2(b)) requires additional

memory storage when compared to that of `du_mutation` (Fig. B.2(a)) due to the allocations of `u_m1` and `u_m2` (lines 6-7 of `du_reassignment`), which are the arrays containing the first- and second-level left neighbors of each element of the array `u`.

Fig. B.3 illustrates the computational graph, the forward, and the backward passes for the program `du_reassignment` (Fig. B.2(b)). Following the variable naming convention of Griewank and Walther (2008) (see Sec. 1.2), the arrays `du` and `u` are assigned to the input variables v_{-1} and v_0:

$$v_{-1} \leftarrow \texttt{du};\ v_0 \leftarrow \texttt{u}. \tag{B.4}$$

The rolling operator `circshift` is then applied to the input variable v_0 (i.e., `u`) to create the arrays `u_m1` and `u_m2`, which will be assigned to the intermediate variables v_1 and v_2:

$$v_1 \leftarrow \texttt{circshift}(v_0, 1);\ v_2 \leftarrow \texttt{circshift}(v_0, 2). \tag{B.5}$$

Moving further on the program execution, line 8 of `du_reassignment` reassigns `du` (represented by v_{-1}) to a new variable v_3 as follows:

$$v_3 \leftarrow v_{-1} + v_2 - 4v_1 + 3v_0. \tag{B.6}$$

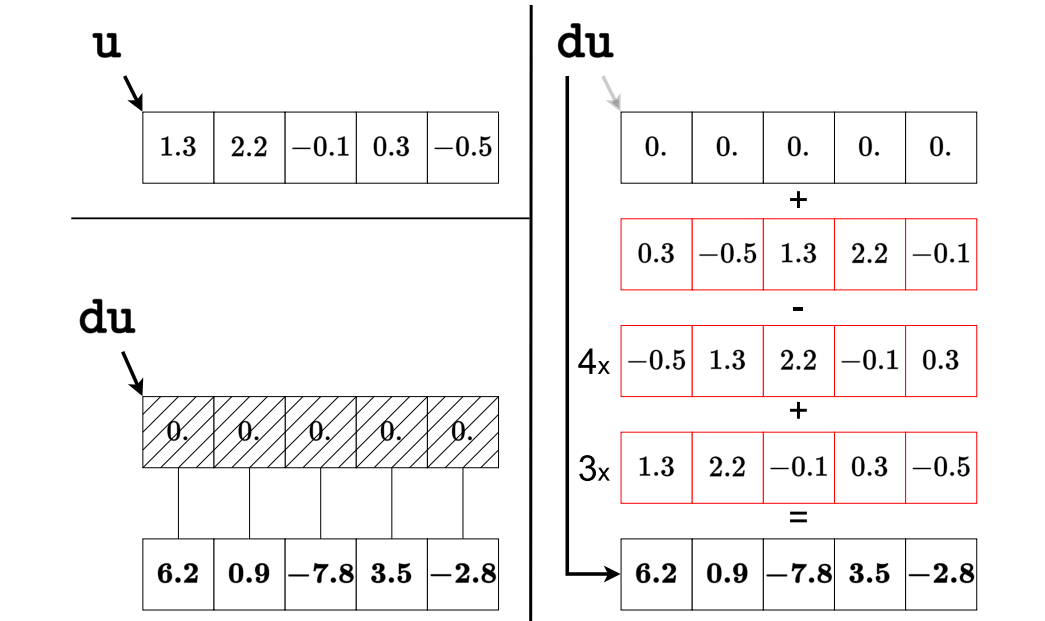
Finally, the squared $l2$-norm of v_3 is assigned to the output variable y (line 10 of `du_reassignment`):

$$y \leftarrow \|v_3\|_2^2 . \tag{B.7}$$

Given the forward pass depicted from Eq. (B.4) to Eq. (B.7), one can build the corresponding backward pass (or adjoint evaluation trace). As explained in Sec. 1.2.2, the evaluation of reverse derivatives $\overline{v} := \partial y / \partial v$ starts from the output y:

$$\overline{y} \leftarrow 1. \tag{B.8}$$

One then applies the chain rule of calculus successively up the forward pass




```julia
function du_mutation(
  u  :: Vector,
  du :: Vector
)

  du[1] += u[N-1] - 4u[N] + 3u[1]
  du[2] += u[N]   - 4u[1] + 3u[2]
  for i = 3:N
    du[i] += u[i-2] - 4u[i-1] + 3u[i]
  end

  y = sum(abs2, du)

  return y

end
```

(a)

```julia
function du_reassignment(
  u  :: Vector,
  du :: Vector
)

  u_m1 = circshift(u, 1)
  u_m2 = circshift(u, 2)
  du  += u_m2 - 4u_m1 + 3u

  y = sum(abs2, du)

  return y

end
```

(b)

Figure B.2: Illustration of how `du` is computed either by (a) `du_mutation` or (b) `du_reassignment`. Represented by red boxes, additional memory in the forward pass is required in (b) when compared to (a). Note that an array `du` initialized to zeros beforehand is given as input to each function. Furthermore, the instruction `sum(abs2, v)` computes the squared $l2$-norm $||\boldsymbol{v}||_2^2 := \sum_i |\boldsymbol{v}_i|^2$ of a given vector $\boldsymbol{v}$.

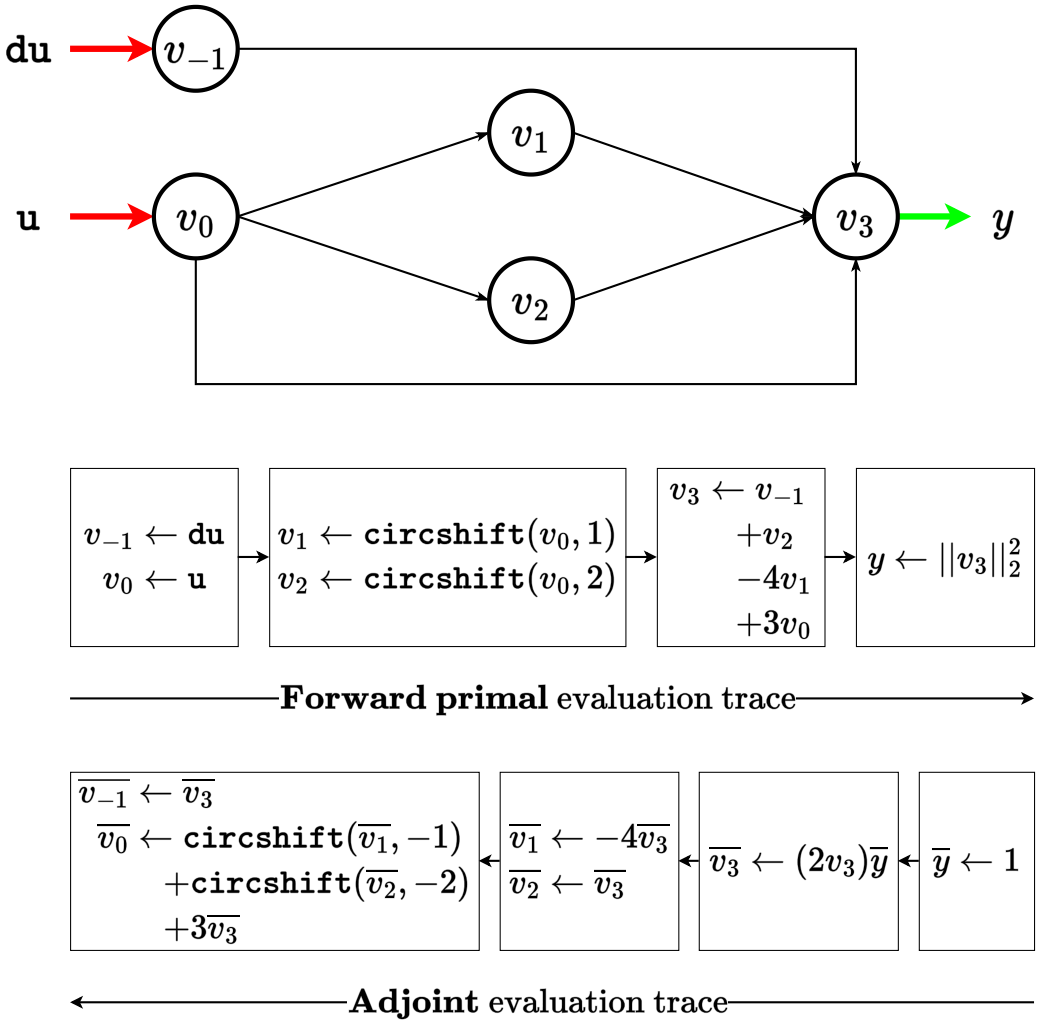


Figure B.3: Computational graph, *forward primal* and *adjoint* evaluation traces for the program `du_reassignment`.

to compute the other reverse derivatives. For v_3, v_2, v_1 and v_{-1}:

$$\overline{v_3} := \frac{\partial y}{\partial v_3} \equiv \left(\frac{\partial y}{\partial v_3}\right)\left(\frac{\partial y}{\partial y}\right) \xleftarrow{\star^5} (2v_3)\ \overline{y} \tag{B.9}$$

$$\overline{v_2} := \frac{\partial y}{\partial v_2} \equiv \left(\frac{\partial v_3}{\partial v_2}\right)\left(\frac{\partial y}{\partial v_3}\right) \xleftarrow{\text{Eq. (B.6)}} (1)\ \overline{v_3} \tag{B.10}$$

$$\overline{v_1} := \frac{\partial y}{\partial v_1} \equiv \left(\frac{\partial v_3}{\partial v_1}\right)\left(\frac{\partial y}{\partial v_3}\right) \xleftarrow{\text{Eq. (B.6)}} (-4)\ \overline{v_3} \tag{B.11}$$

$$\overline{v_{-1}} := \frac{\partial y}{\partial v_{-1}} \equiv \left(\frac{\partial v_3}{\partial v_{-1}}\right)\left(\frac{\partial y}{\partial v_3}\right) \xleftarrow{\text{Eq. (B.6)}} (1)\ \overline{v_3} \tag{B.12}$$

Finally, changes in the input variable v_0 affect variables v_1, v_2 and v_3 of the computational graph (see Fig. B.3), so that the sensitivity $\overline{v_0}$ is a function of the sensitivities $\overline{v_1}$, $\overline{v_2}$ and $\overline{v_3}$:

$$\overline{v_0} := \frac{\partial y}{\partial v_0} \equiv \left(\frac{\partial v_1}{\partial v_0}\right)\left(\frac{\partial y}{\partial v_1}\right) + \left(\frac{\partial v_2}{\partial v_0}\right)\left(\frac{\partial y}{\partial v_2}\right) + \left(\frac{\partial v_3}{\partial v_0}\right)\left(\frac{\partial y}{\partial v_3}\right)$$

$$\xleftarrow{\text{Eqs. (B.5)–(B.6)}} \overline{\texttt{circshift}}(v_0,\ 1,\ \overline{v_1}) + \overline{\texttt{circshift}}(v_0,\ 2,\ \overline{v_2}) + (3)\ \overline{v_3}, \tag{B.13}$$

where $\overline{\texttt{circshift}}$ is the adjoint of the program representing the `circshift` operator. Given `shift` $\in \mathbb{Z}$, an input `w` and the corresponding output

$$\texttt{w\_shifted} := \texttt{circshift}(\texttt{w},\ \texttt{shift}),$$

the role of

$$(\texttt{w},\ \texttt{shift},\ \overline{\texttt{w\_shifted}}) \mapsto \overline{\texttt{circshift}}(\texttt{w},\ \texttt{shift},\ \overline{\texttt{w\_shifted}})$$

is to propagate the contribution of the sensitivity of the output (i.e., $\overline{\texttt{w\_shifted}}$) back to the input `w`, to form the sensitivity $\overline{\texttt{w}}$. Deriving $\overline{\texttt{circshift}}$ is straightforward. Given any valid index `j`, a perturbation in the value of `w_shifted[j]` simply leads to an identical perturbation in the value of `w` at

[5] The derivative of $\boldsymbol{w} \mapsto \|\boldsymbol{w}\|_2^2$ ($\boldsymbol{w} \in \mathbb{R}^N$, $N \in \mathbb{N}$) can be obtained from Fréchet definition (Berger, 1977): $\|\boldsymbol{w} + \boldsymbol{\Delta w}\|_2^2 := \langle \boldsymbol{w} + \boldsymbol{\Delta w}, \boldsymbol{w} + \boldsymbol{\Delta w}\rangle = \langle \boldsymbol{w}, \boldsymbol{w}\rangle + 2\langle \boldsymbol{w}, \boldsymbol{\Delta w}\rangle + \langle \boldsymbol{\Delta w}, \boldsymbol{\Delta w}\rangle = \|\boldsymbol{w}\|_2^2 + \langle \underbrace{2\boldsymbol{w}}, \boldsymbol{\Delta w}\rangle + o(\|\boldsymbol{\Delta w}\|_2)$.

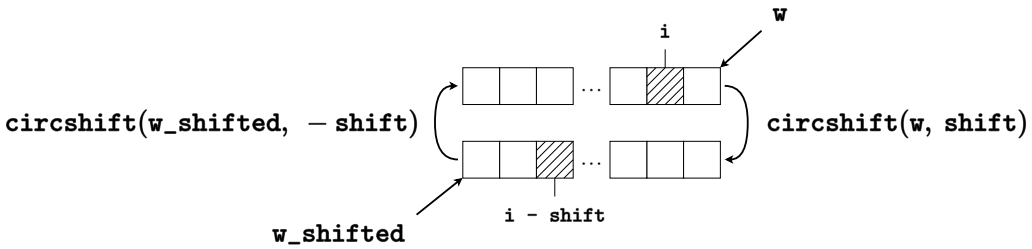


Figure B.4: Schematic representation of `circshift` operation on some array `w`. The sensitivity of the output array `w_shifted` is propagated back to the input array `w` by performing the reverse operation, i.e., $\overline{\texttt{circshift}}(\texttt{w},\ \texttt{shift},\ \overline{\texttt{w\_shifted}}) := \texttt{circshift}(\overline{\texttt{w\_shifted}},\ -\texttt{shift})$ ($\texttt{shift} \in \mathbb{Z}$).

the corresponding index, namely `j + shift` (periodic conditions apply). In other words,

$$\overline{\texttt{w}}\texttt{[j + shift]} = \overline{\texttt{w\_shifted}}\texttt{[j]}$$

or, equivalently,

$$\overline{\texttt{w}}\texttt{[i]} = \overline{\texttt{w\_shifted}}\texttt{[i - shift]},$$

for any valid index `i`. Fig. B.4 summarizes this reflection. The operator $\overline{\texttt{circshift}}$ eventually reads

$$(\texttt{w},\ \texttt{shift},\ \overline{\texttt{w\_shifted}}) \longmapsto \overline{\texttt{circshift}}(\texttt{w},\ \texttt{shift},\ \overline{\texttt{w\_shifted}})$$

$$:= \texttt{circshift}(\overline{\texttt{w\_shifted}},\ -\texttt{shift})$$

and Eq. (B.13) then assumes the final form below:

$$\overline{v_0} \leftarrow \texttt{circshift}(\overline{v_1}, -1) + \texttt{circshift}(\overline{v_2}, -2) + (3)\ \overline{v_3}. \tag{B.14}$$

Given the backward pass depicted from Eq. (B.8) to Eq. (B.14), one can code the adjoint program of `du_reassignment` as illustrated in Fig. B.5(b). The method is named `du_reassignment_adjoint`. It is possible to recognize the reverse derivatives inside the adjoint program by the suffix `b` [6]: `yb` represents the reverse derivative of the output y (i.e., $\overline{y}$), `ub` is the reverse

[6]The suffix `b` stands for *bar* ($\bar{\cdot}$) here, the symbol that distinguishes a reverse derivative from its primal in the text.

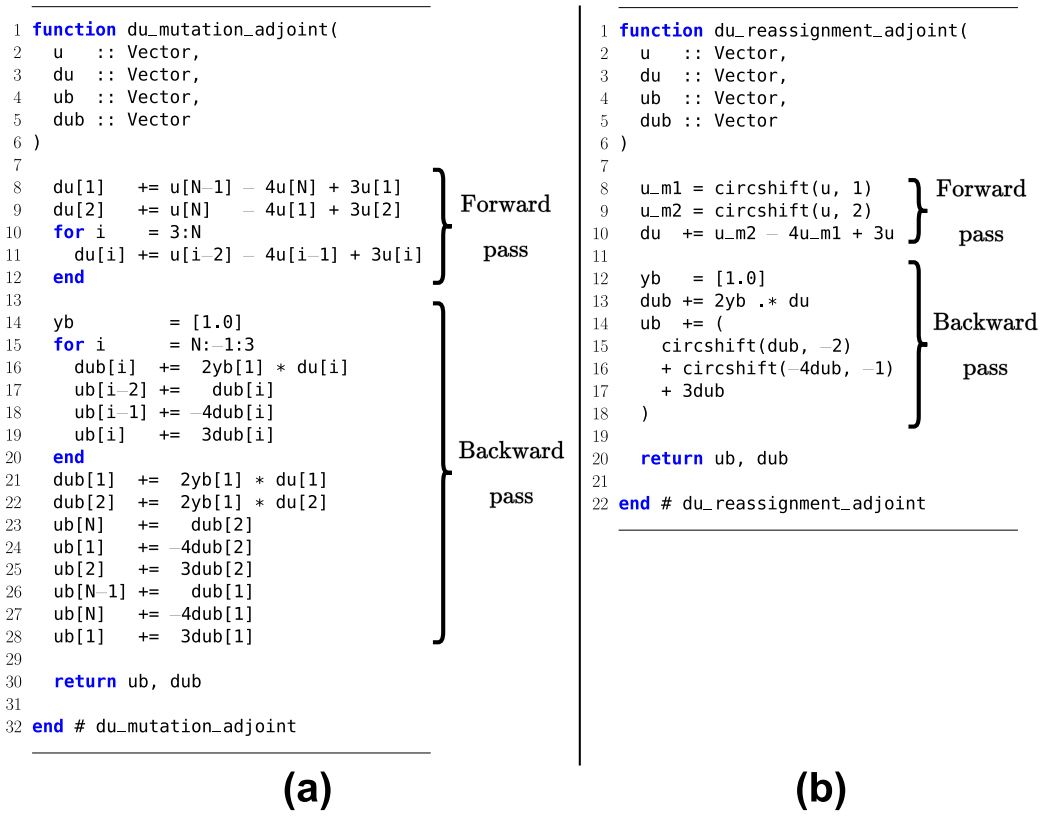


Figure B.5: Adjoint programs of (a) `du_mutation` (Fig. B.2(a)) and (b) `du_reassignment` (Fig. B.2(b)). Note that the arrays `ub` and `dub` respectively corresponding to the reverse derivatives of `u` and `du` are initialized to zeros beforehand and given as inputs to each function.

derivative for the input `u` (i.e., $\overline{v_0}$) and `dub` is the reverse derivative for the input `du` (i.e., $\overline{v_{-1}} \equiv \overline{v_3} \equiv \overline{v_2}$). The adjoint program relates to the evaluation traces in Fig. B.3 as follows: lines 8-10 of `du_reassignment_adjoint` perform the forward pass up to Eq. (B.6), since the value of the output y is not required for executing the backward pass; given the updated value of `du` (i.e., v_3), the program then starts the backward pass in line 12 (following Eq. (B.8)), and applies Eqs. (B.12)-(B.14) to update `dub` and `ub`, respectively.

One can similarly apply the reverse mode of AD to differentiate the method `du_mutation` (Fig. B.2(a)). The resulting adjoint program is shown in Fig. B.5(a). The forward pass is executed first (in lines 8-12) to update

the value of `du` and the backward pass follows from line 14 to line 28.

One can now compare the performances of the two adjoint programs illustrated in Fig. B.5 with respect to their primal programs shown in Fig. B.2. The mutating and non-mutating implementations are compared in terms of total memory usage and runtime. Fig. B.6 gathers benchmarking results for varying lengths N of the array `u`. They were produced in a MacBook Air (Processor: 1.6 GHz Intel Core i5; RAM: 16 GB 2133 MHz LPDDR3) with the version 1.8.5 of the Julia programming language and the package `BenchmarkTools.jl` (Chen and Revels, 2016). Fig. B.5(a) shows that the adjoint program `du_reassignment_adjoint` allocates approximately twice the amount of memory allocated by its primal `du_reassignment`. Meanwhile, the mutating implementations `du_mutation_adjoint` and `du_mutation` exhibit similar memory footprints. Runtime-wise, the mutating implementations present worse performance because their loops are not parallelized.

An extension of the problem to higher dimensions is analyzed, motivated by the fact that LES codes handle turbulence, which is inherently a 3D phenomenon. Adjoints of non-mutating and mutating programs that compute

$$\|\mathrm{d}_x u(x,y,z)\|_2^2 + \|\mathrm{d}_y u(x,y,z)\|_2^2 + \|\mathrm{d}_z u(x,y,z)\|_2^2 \tag{B.15}$$

with [7]

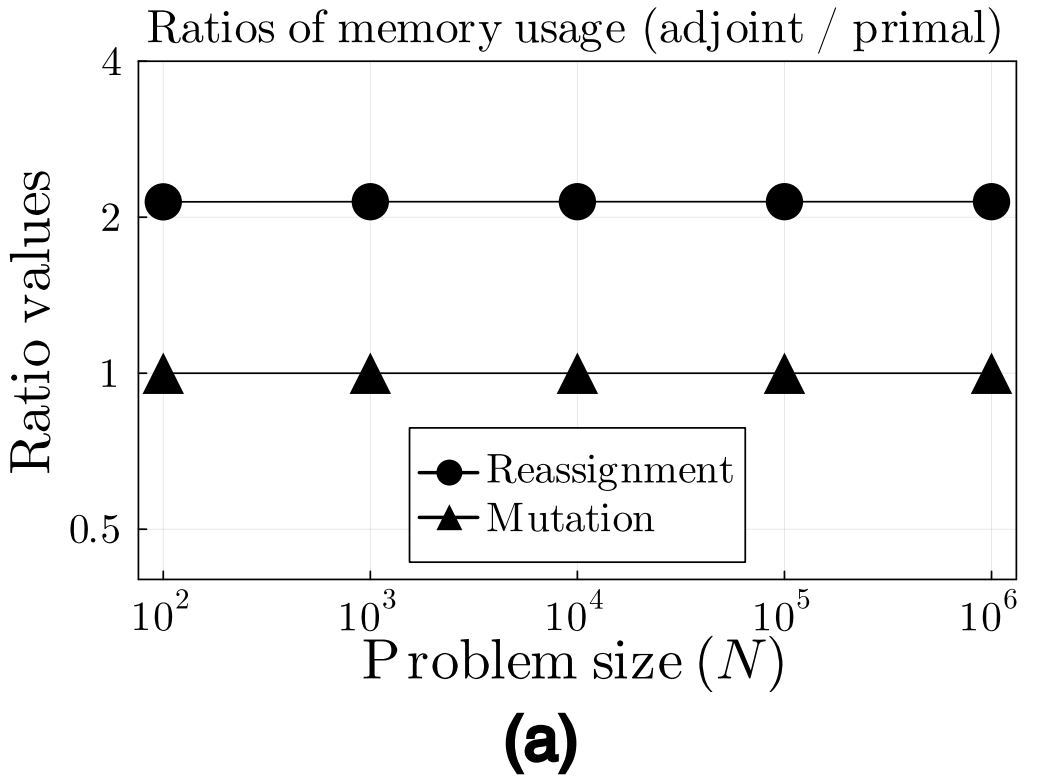
$$\mathrm{d}_x u(x,y,z)|_{x=x_i,y=y_j,z=z_k} := u_{i-2,j,k} - 4u_{i-1,j,k} + 3u_{i,j,k} \tag{B.16}$$
$$\mathrm{d}_y u(x,y,z)|_{x=x_i,y=y_j,z=z_k} := u_{i,j-2,k} - 4u_{i,j-1,k} + 3u_{i,j,k} \tag{B.17}$$
$$\mathrm{d}_z u(x,y,z)|_{x=x_i,y=y_j,z=z_k} := u_{i,j,k-2} - 4u_{i,j,k-1} + 3u_{i,j,k} \tag{B.18}$$

$$(i \in [\![1,N]\!],\ j \in [\![1,N]\!],\ k \in [\![1,N]\!],\ N \in \mathbb{N})$$

are benchmarked [8]. Fig. B.7 compares performances of the adjoint programs with respect to their primals for varying problem sizes (N). The experiments were conducted under conditions similar to those of the 1D case. Memory-wise, the results are equivalent to those found in the 1D experiments. Runtime-wise, the mutating implementations prove to be faster than the non-mutating ones, as opposed to what was observed in the 1D case.

[7] Periodic conditions apply.

[8] The reader can find the adjoint programs in https://github.com/luciano-drozda/reassignment-vs-mutation/.



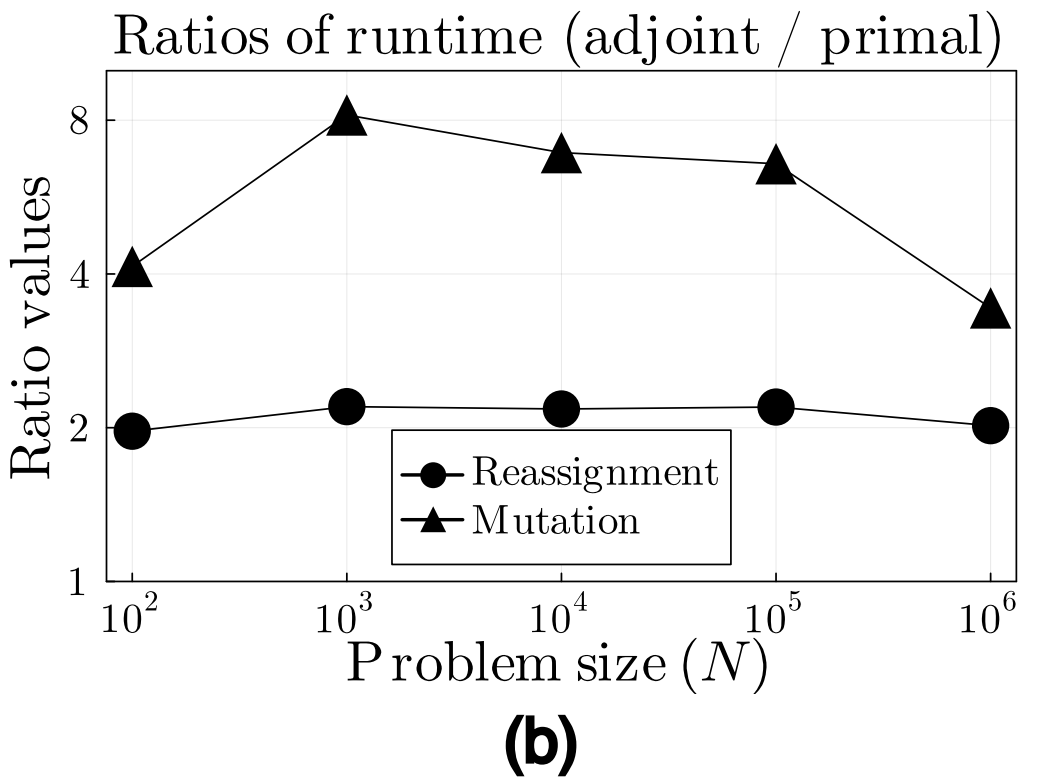


Figure B.6: Benchmarking results for the 1D discretization operator problem regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes.

In the non-mutating 3D implementations, the runtime spent in array allocations becomes dominant when compared to that of the vectorized operations between arrays.

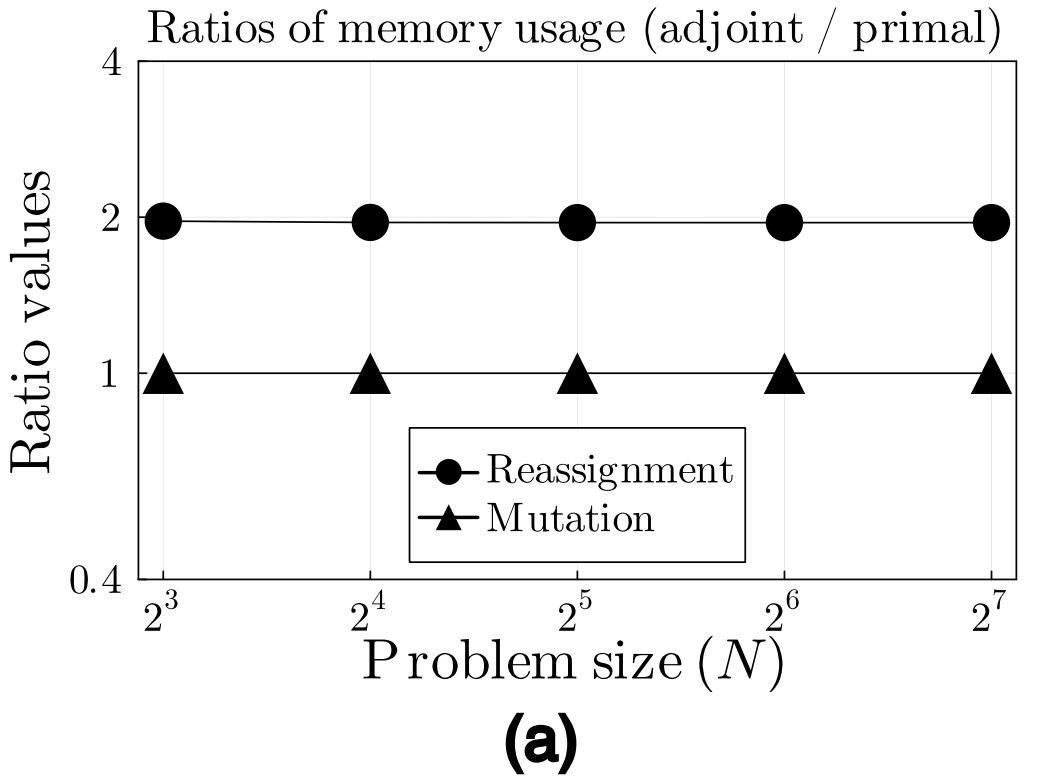
The aftermath of the study conducted in this subsection is that the memory footprint of the adjoint of a non-mutating program when compared to its primal is at least twice that of their mutating counterpart. Along with the elements in the literature already listed (Bezgin, Buhendwa, et al., 2023; Kochkov et al., 2021), such a result corroborates with the assumption that building efficient differentiable PDE solvers requires support to array mutation by the AD tool of choice. In the next subsection, a similar analysis is performed on mutating and non-mutating CFD kernels.

B.3 Analysis of a CFD kernel

In order to evaluate the impact of the mutation problem in a CFD configuration, this section considers a function that updates a numerical solution to the 3D Euler equations on tetrahedral meshes using the Two-step Taylor-Galerkin C (TTGC) discretization scheme depicted in Sec. 5.1. No treatment of boundary conditions is performed here. The non-mutating and mutating programs applying a TTGC update will be hereafter referred to as `dU_ttgc_reassignment` and `dU_ttgc_mutation`, respectively. They both return the squared $l2$-norm of the value $\boldsymbol{\Delta U}$ by which some input numerical solution $\boldsymbol{U}$ should be incremented over one time step. For the sake of shortness, only parts of the algorithms of the methods `dU_ttgc_reassignment` and `dU_ttgc_mutation` are represented in Fig. B.8 [9].

The part of the TTGC scheme highlighted in Fig. B.8 relate to the computation of the nodal residuals $\texttt{Rj} \in \mathbb{R}^{\texttt{neq}\times\texttt{nnode}}$ by scattering of the cell residuals $\texttt{VeRe} \in \mathbb{R}^{\texttt{neq}\times\texttt{ncell}}$ (weighted by the cell volumes), with $\texttt{neq} \equiv 5$ being the number of equations to be solved (one for density, three for momentum and one for energy), `nnode` the number of mesh nodes and `ncell` the number of mesh cells. This is a common operation in the class of residual distribution numerical schemes, which TTGC belongs to (Deconinck and Ricchiuto, 2017). One can notice that the mutating kernel accumulates cells contribution to a zero-initialized array `Rj` inside a loop over mesh cells (lines 5-17 of `dU_ttgc_mutation`). By allowing mutation, one avoids to store `VeRe` for

[9] Full implementations are made available in https://github.com/luciano-drozda/reassignment-vs-mutation/.



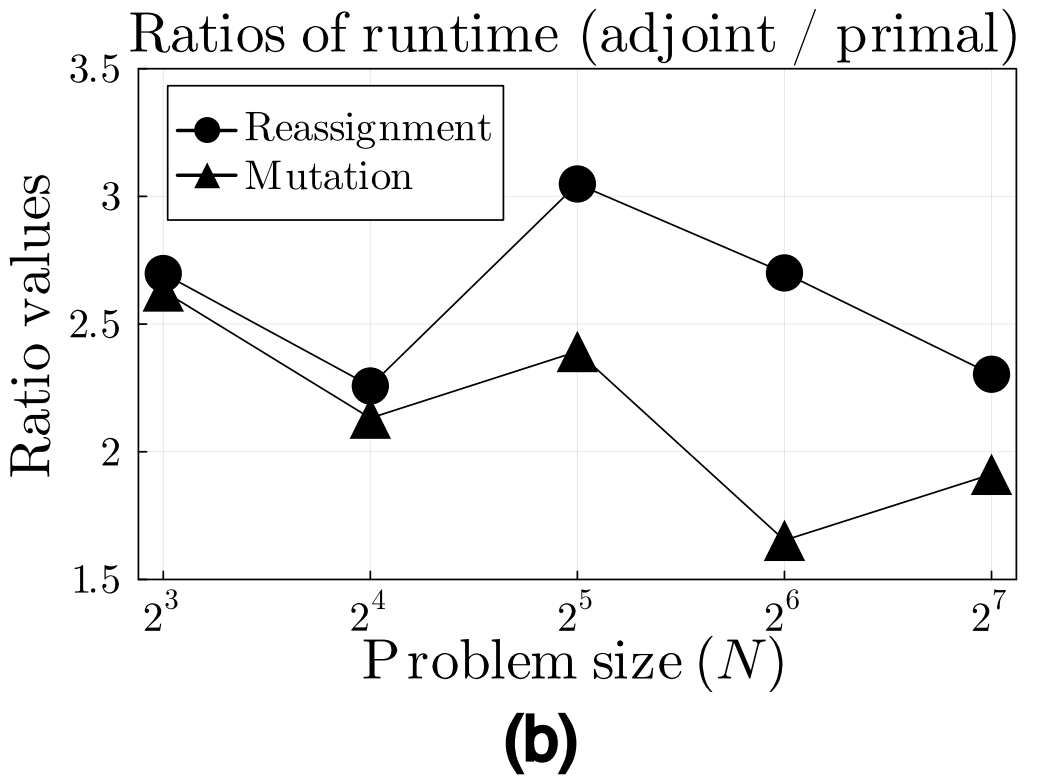


Figure B.7: Benchmarking results for the 3D discretization operator problem regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes.

`dU_ttgc_mutation`

Input arguments: numerical solution $\boldsymbol{U}$; components of nodal normals along the x, y, and z directions $\{\boldsymbol{Sk_X}, \boldsymbol{Sk_Y}, \boldsymbol{Sk_Z}\}$; scalar parameters of the numerical scheme $\{\gamma, \beta\}$; time step Δt

▷ Initialize to zero the nodal residuals ($\boldsymbol{R_j}$)
$\boldsymbol{R_j} \longleftarrow \boldsymbol{0}$

▷ Loop over cells
for cell ∈ mesh **do**

    ▷ Loop over nodes of the current cell
    for node ∈ cell **do**

        ▷ Compute contribution of the current node to the cell's residuals ($\boldsymbol{VeRe}_{\text{cell}}^{\text{node}}$) and to the cell's residuals weighted by the cell-averaged flux Jacobian components along the x, y, and z directions ($\{\boldsymbol{AeRe_X}_{\text{cell}}^{\text{node}}, \boldsymbol{AeRe_Y}_{\text{cell}}^{\text{node}}, \boldsymbol{AeRe_Z}_{\text{cell}}^{\text{node}}\}$)

        ▷ Scatter $\boldsymbol{VeRe}_{\text{cell}}^{\text{node}}$ and $\{\boldsymbol{AeRe_X}_{\text{cell}}^{\text{node}}, \boldsymbol{AeRe_Y}_{\text{cell}}^{\text{node}}, \boldsymbol{AeRe_Z}_{\text{cell}}^{\text{node}}\}$ to all the nodes of the current cell
        for node_ ∈ cell **do**
            $\boldsymbol{R_j}$[node_] += $-\Delta t\,(0.5-\gamma)\,\boldsymbol{VeRe}_{\text{cell}}^{\text{node}}$ / 4
            $\boldsymbol{R_j}$[node_] += $\beta\,\Delta t^2$ (
                $\boldsymbol{AeRe_X}_{\text{cell}}^{\text{node}}$ $\boldsymbol{Sk_X}$[node_, cell]
                +$\boldsymbol{AeRe_Y}_{\text{cell}}^{\text{node}}$ $\boldsymbol{Sk_Y}$[node_, cell]
                +$\boldsymbol{AeRe_Z}_{\text{cell}}^{\text{node}}$ $\boldsymbol{Sk_Z}$[node_, cell]
) / 3
        end for

    end for

end for

[...]

▷ Return the squared $l2$-norm of the TTGC update

(a)

`dU_ttgc_reassignment`

Input arguments: numerical solution $\boldsymbol{U}$; scatter operator $\underline{\boldsymbol{S}}$; scatter operators weighted by the components of the nodal normals along the x, y, and z directions $\{\underline{\boldsymbol{Sje_X}}, \underline{\boldsymbol{Sje_Y}}, \underline{\boldsymbol{Sje_Z}}\}$; scalar parameters of the numerical scheme $\{\gamma, \beta\}$; time step Δt

▷ Compute the cells residuals ($\boldsymbol{VeRe}$) and the cells residuals weighted by the cell-averaged flux Jacobian components along the x, y, and z directions ($\{\boldsymbol{AeRe_X}, \boldsymbol{AeRe_Y}, \boldsymbol{AeRe_Z}\}$)

▷ Scatter $\boldsymbol{VeRe}$ to the mesh nodes in order to obtain the first part of the nodal residuals ($\boldsymbol{L_j}$)
$\boldsymbol{L_j} \longleftarrow \underline{\boldsymbol{S}}(\boldsymbol{VeRe})$ / 4

▷ Scatter $\{\boldsymbol{AeRe_X}, \boldsymbol{AeRe_Y}, \boldsymbol{AeRe_Z}\}$ to the mesh nodes in order to obtain the second part of the nodal residuals ($\boldsymbol{LL_j}$)
$\boldsymbol{LL_j} \longleftarrow \left[\underline{\boldsymbol{Sje_X}}(\boldsymbol{AeRe_X}) + \underline{\boldsymbol{Sje_Y}}(\boldsymbol{AeRe_Y}) + \underline{\boldsymbol{Sje_Z}}(\boldsymbol{AeRe_Z})\right]$ / 3

▷ Compute the nodal residuals ($\boldsymbol{R_j}$)
$\boldsymbol{R_j} \longleftarrow -\Delta t\,(0.5-\gamma)\,\boldsymbol{L_j} + \beta\,\Delta t^2\,\boldsymbol{LL_j}$

[...]

▷ Return the squared $l2$-norm of the TTGC update

(b)

Figure B.8: Portions of the algorithms of the (a) mutating and (b) non-mutating TTGC scheme implementations (3D Euler equations; tetrahedral meshes).

all cells at once, as opposed to what happens inside the non-mutating kernel (see Fig. B.8(b)).

Fig. B.9 presents performance measurements for the adjoints of both methods, including not only memory usage, but also runtime benchmarks. The computations were performed using an Intel(R) Xeon(R) Gold 6140 CPU. The adjoints of `dU_ttgc_reassignment` and `dU_ttgc_mutation` are respectively generated by the Zygote package (which only supports reassignments, as seen in Sec. 1.3.4) and by the Enzyme library (which supports mutation, as seen in Sec. 1.3.5). At the largest problem size under consideration (namely, `nnode` = 33792), Fig. B.9(a) reveals that the adjoint of `dU_ttgc_reassignment` allocates around 250 times the amount of memory allocated by the method, whereas the adjoint of `dU_ttgc_mutation` is only twice as expensive as its primal. Runtime-wise, Fig. B.9(b) shows that the adjoint of the non-mutating kernel is around 100 times slower than its primal, whereas the adjoint of the mutating kernel is less than 4 times slower than its primal. This means that, in general, one can expect the differentiation of non-mutating PDE kernels to perform much worse than that of mutating implementations, both runtime- and memory-wise.

The quantitative observations from Sec. B.2-B.3 should be sufficient to motivate the development of AD tools that support array mutation for enabling efficient training of ML-PDE coupled systems. Tapenade (Sec. 1.3.3) and other tools developed for Fortran/C languages natively support mutation, but seem out of interest of the ML community for their relatively low-level features when compared to those of the Python APIs of the main ML frameworks (namely, TensorFlow, PyTorch and JAX). Once the field of ML for scientific applications leans toward the development of coupled ML-PDE systems (in a quest for more generalizable predictive capabilities of ML architectures acting on physical models), the mutation problem should be given more attention.

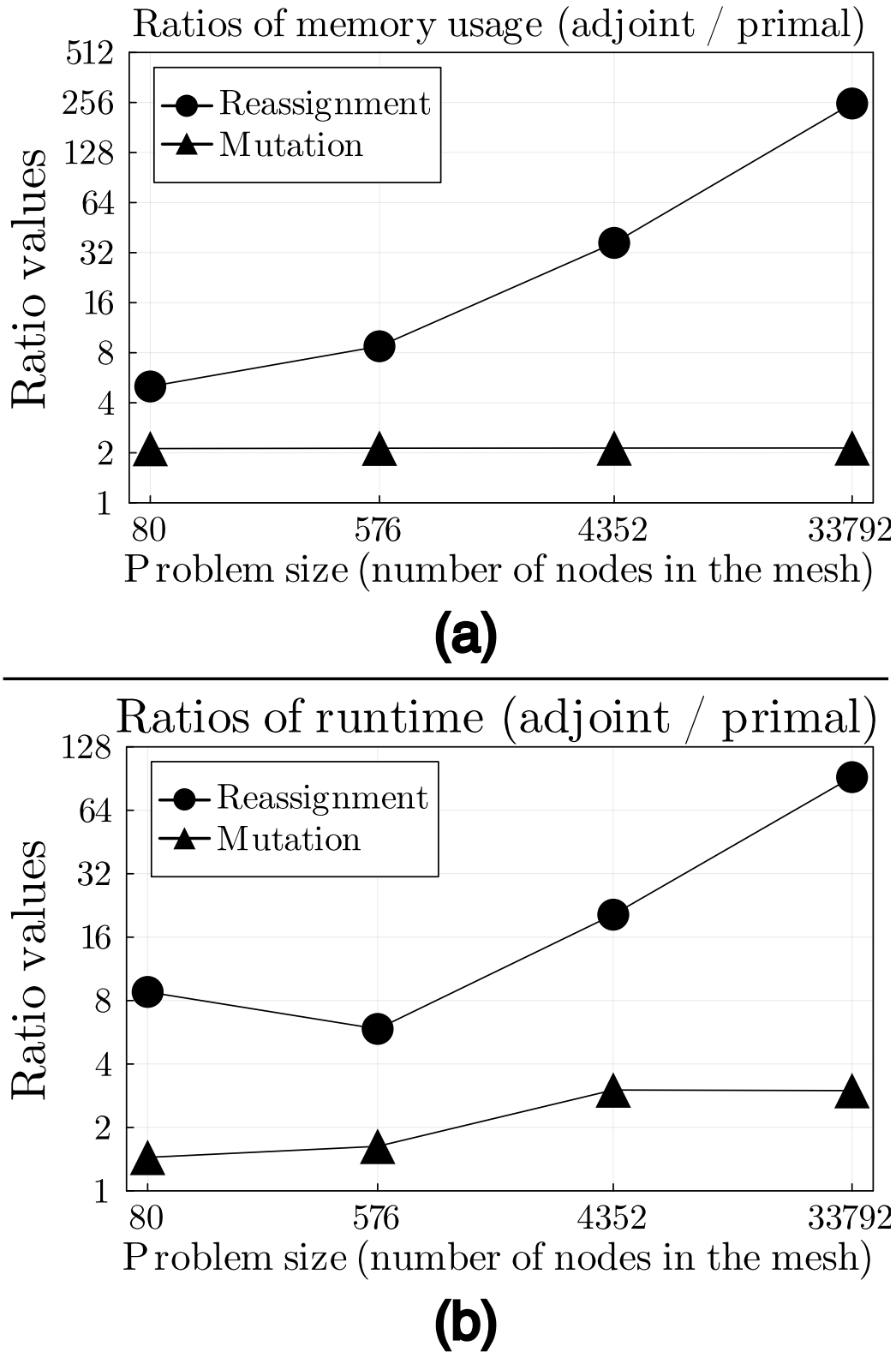


Figure B.9: Benchmarking results for the TTGC scheme (3D Euler equations; tetrahedral meshes) regarding the ratios of (a) memory usage; (b) runtime between adjoint and primal codes.

Appendix C

Local wavenumber analysis and shock sensor of Jameson

The local wavenumber k is defined as the spatial rate of change of the phase angle of the convecting wave $u(x,t)$ (Bracewell, 1978), i.e.,

$$\boldsymbol{k} := \nabla\theta \tag{C.1}$$

This quantity is extensively used in the field of magnetism and geology to identify physical sizes of objects like rock formations under the earth by measuring the aberrations in the phase angle of the spatial magnetic or gravitational field (Pilkington and Keating, 2006). The phase angle variation is obtained using the Eikonal $E = |\nabla u|$, which represents the surface of constant phase angle. Note that the operator $|\cdot|$ is the Euclidean norm. Thus one can rewrite the local wavenumber using the Eikonal as

$$\boldsymbol{k} := \frac{\nabla E}{E} \tag{C.2}$$

The nodal junction i connecting two finite-elements $e_L[i-1,i]$ and $e_R[i,i+1]$ is considered. The local gradient at the node junction i can be approximately evaluated using finite-difference formula as

$$\nabla E_i = \left(\frac{|\Delta u_{i+1/2}|}{h_R} - \frac{|\Delta u_{i-1/2}|}{h_L}\right)\left(\frac{h_L + h_R}{2}\right)^{-1}, \tag{C.3}$$

where $\Delta u_{i+1/2} = u_{i+1} - u_i$ and $\Delta u_{i-1/2} = u_i - u_{i-1}$. If one approximates E at the junction as the average of the values at e_L and e_R, i.e., $E_i =$

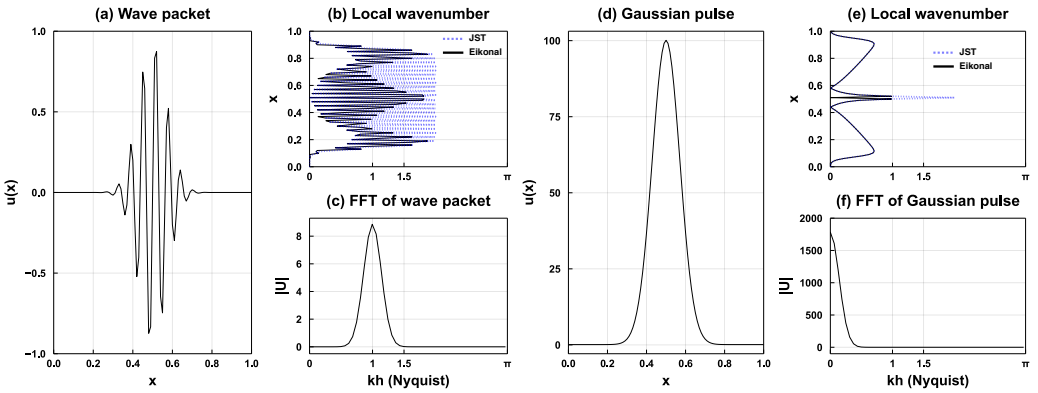


Figure C.1: Comparison of local wavenumber analysis using Eikonal and JST sensor; (a-c) WP function and (d-f) Gaussian pulse function.

$\frac{1}{2}\left(\frac{|\Delta u_{i+1/2}|}{h_R}+\frac{|\Delta u_{i-1/2}|}{h_L}\right)$, one obtains the following numerical approximation for the local wavenumber:

$$(kh)_i^* = 2\frac{|\Delta u_{i+1/2}| - |\Delta u_{i-1/2}|\tau}{|\Delta u_{i+1/2}| + |\Delta u_{i-1/2}|\tau}. \tag{C.4}$$

Eq. (C.4) closely resembles the shock sensor operator from the widely acclaimed JST dissipation scheme of Jameson (2017). Assuming an equally spaced mesh (i.e., $\tau = h_R/h_L \equiv 1$) and only right moving waves (kh is positive), the following inequality with the JST sensor (S_{JST}) holds:

$$S_{JST} = \frac{|\Delta u_{i+1/2} - \Delta u_{i-1/2}|}{|\Delta u_{i+1/2}| + |\Delta u_{i-1/2}|} \geq \frac{|\Delta u_{i+1/2}| - |\Delta u_{i-1/2}|}{|\Delta u_{i+1/2}| + |\Delta u_{i-1/2}|}. \tag{C.5}$$

The shock sensor is a conservative upper bound for the local wavenumber in terms of the Nyquist frequency (on equally spaced meshes). In fact, Jameson (2017) mentions that this sensor provides robust results near shocks compared to other TVD shock sensors and inspired the CUSP family of schemes. To the author's knowledge, this is the first work to reveal this mathematical connection of the local wavenumber to the shock sensor. Fig. C.1(a) shows a WP with centre Nyquist frequency $kh = 1$ whose Fourier spectrum is shown in Fig. C.1(c). The local wavenumber estimates based on the Eikonal and JST sensor values are plotted in Fig. C.1(b). The JST overestimates the local wavenumber since it is an upper bound. The Eikonal estimates are closer to the actual Fourier Nyquist limits. Using the local wavenumber, one can

determine if the variations are within the resolvable limit of the numerical scheme. When variations are beyond the numerical resolution (unresolved) they can be removed using artificial dissipation. Using the uncertainty principle, one can show that a function $u(x)$ and its Fourier transform $\hat{U}(k)$ cannot be supported on arbitrarily small sets as given by the Heisenberg-Pauli-Weyl inequality:

$$\left(\int x^2|u(x)|^2dx\right)^{1/2}\left(\int k^2|\hat{U}(k)|^2dk\right)^{1/2} \geq \frac{1}{4\pi}\|u\|_2^2, \tag{C.6}$$

where $\|u\|_2$ is the $l2$ norm of the function u. Therefore, these spatial estimates are at best approximations to the local wavenumber; when the spatial variations are sharp, the uncertainty in wavenumber grows quite rapidly.

Appendix D

Entropy condition and impedance matching

Non-linear convection problems like the Burgers' equation and compressible Euler equations in gas dynamics admit discontinuous solutions with jumps called shock waves. Fig. D.1(b-d) shows the formation of the shock from a smooth initial solution. Shocks occur at regions where two or more characteristic lines meet (Fig. D.1(e)), i.e., a region where one wave tries to overtake another. Overtaking of waves mathematically leads to multiple solutions. Therefore, the solution that yields the shock wave by enforcing the entropy condition is chosen. There are several ways to formulate this entropy condition, but in this work, the Rankine-Hugoniot (RH) relation is used to enforce the entropy condition. The non-linear conservation law is defined as

$$u_t + [f(u)]_x = 0. \tag{D.1}$$

Then the shock speed ξ is given by the RH condition as

$$\xi := \frac{f(u_1) - f(u_2)}{u_1 - u_2}, \tag{D.2}$$

where u_1 and u_2 are the states before and after the shock (as shown in Fig. D.1). Then for any $u \in [u_1, u_2]$ the shock speed must be bounded by

$$\frac{f(u_1) - f(u)}{u_1 - u} \leq \xi \leq \frac{f(u) - f(u_2)}{u - u_2}. \tag{D.3}$$

This is the entropy condition that ensures uniqueness of the shock solution and eliminates the multiple solution case shown in Fig. D.1(b). For the

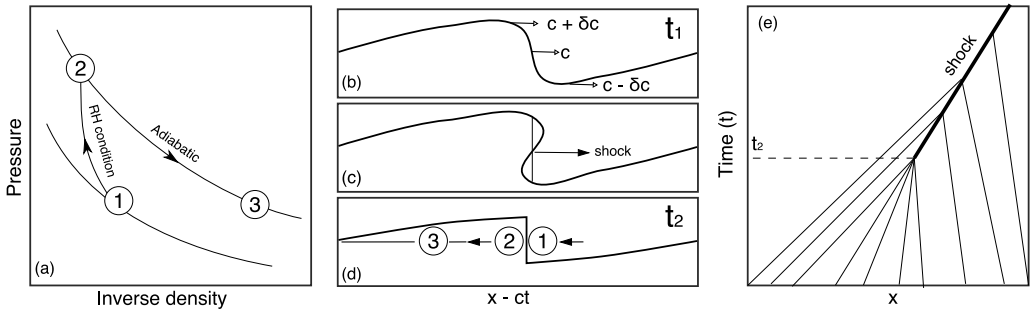


Figure D.1: (a) Thermodynamic cycle of a shock wave, (b) waves overtaking each other, (c) multiple solution at shock, (d) the shock entropy solution and (e) x-t diagram of the shock problem.

inviscid Burgers' equation, the flux function is $f(u) = \frac{1}{2}u^2$ and the entropy condition can be rewritten as

$$u_1 + u \leq 2\xi \leq u + u_2. \tag{D.4}$$

The problem of shock capturing using discrete finite-elements as illustrated in Fig. 3.8 is considered. It is intended to smooth the shock and capture its location between the two elements $e_i \in [i-1, i]$ and $e_{i+1} \in [i, i+1]$, say by matching the impedance to the left and right of node i. The entropy condition in Eq. (D.3) must be satisfied for a valid shock solution between the elements and this simplifies to

$$u_{i-1}^{n+1} \leq u_{i+1}^{n+1}. \tag{D.5}$$

The non-linear Burgers' equation can be linearized for the time step n to $n+1$ and the averaged convection velocity resulting from the linearization (Griewank and Walther, 2008; Ucar et al., 2017) is assumed constant over the given finite-element. The Fourier transform is applied on both sides of the inequality and simplified to yield the entropy condition in spectral space,

$$|G_{i-1}| \leq |G_{i+1}|. \tag{D.6}$$

Note that the impedance G is based on the LTA of linearized convection equation with locally constant (average) convection speed (Griewank and Walther, 2008; Ucar et al., 2017). In addition, the nodal impedance G_{i-1} and G_{i+1} are functions of the element impedance pairs $[G_{i-2}^R, G_{i-1}^L]$ and

$[G^L_{i+2}, G^R_{i+1}]$. With the addition of TVD condition, one can show that the five nodes ($i-2,\ i-1,\ i,\ i+1$, and $i+2$) influence the impedance matching problem for shock capturing.