

ABSTRACT

Title of Dissertation: JUDICIOUS INTEGRATION
OF DEEP LEARNING FOR
SOLVING PARTIAL
DIFFERENTIAL EQUATIONS

Philip Charles
Doctor of Philosophy, 2026

Dissertation Directed by: Professor Deep Ray
Department of Mathematics

Computing numerical solutions to partial differential equations (PDEs) has been an ongoing area of mathematical research for over one hundred years. Currently, with more powerful computers and algorithms, the world of numerical PDE solvers is more rich than ever before. The most recent developments involve incorporating machine learning, a very powerful computational technique that has found applications in nearly every science and engineering field. This dissertation explores how deep learning, in particular, can be carefully leveraged to develop and improve numerical PDE solvers. There are many ways in which one can navigate the intersection of traditional numerical solvers and deep learning, from addressing a computational bottleneck in an existing solver to completely replacing the solver with a trained network.

The first area we consider is hyperbolic conservation laws, where shocks can develop in the solutions in finite time. We are interested in designing a high-order entropy stable numerical solver which requires a specialized reconstruction algorithm satisfying certain properties. As will

be discussed, there are only a handful of existing reconstructions that can work, but they have their own issues. The key idea here is that the reconstruction can be replaced with a properly constrained deep-learned algorithm, while the bulk of the numerical solver and its properties, e.g. high-order accuracy, is retained.

Moving on to more general PDEs, we consider a framework where we are interested in learning the underlying solution operator. Instead of using a traditional finite element-type numerical solver, we seek to develop a neural network that mimics the weak form of the PDE. In particular, we wish to emulate the Petrov-Galerkin weak formulation by having a network that learns the solution operator, which recovers the optimal solution with respect to a suitable norm.

In summary, we are using mathematical theory to inform our approach of incorporating deep learning, rather than treating it as a purely data-driven black box. As we will discuss, this demonstrates interesting new prospects in the ever-expanding field at the intersection of deep learning and numerical PDEs.

JUDICIOUS INTEGRATION OF DEEP LEARNING FOR SOLVING PDES

by

Philip Charles

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:

Professor Deep Ray, Chair/Advisor
Professor James Baeder
Professor Maria Cameron
Professor Ramani Duraiswami
Professor Haizhao Yang

© Copyright by
Philip Charles
2026

Acknowledgments

I would like to thank all those who have supported me in making this dissertation possible. First and foremost, I thank my advisor Dr. Deep Ray for his careful guidance, encouragement, and mentorship over the years of my doctoral research. He has helped to shape me into the researcher I am today by instilling a sense of independence and understanding of scientific machine learning.

I would like to mention and thank the following individuals with whom I have collaborated. I thank Dr. Tom Hughes, Dr. Mats Larson, Dr. Assad Oberai, and Jeff Cochran for their feedback on my numerical results with PG-VarMiON. I thank Dr. Michael Abdelmalik, Hugo Melchers, and Joost Prins for their help with understanding the Nutils solver and providing code for the best implementation of B-splines. I thank Dr. Yue Yu for her insights on best practices for training operator networks and for providing code to train FNOs.

I would like to thank my wife Amaris for supporting me throughout graduate school and believing in my abilities. I also thank my friends and family for supporting me over the years.

Finally, I acknowledge UMD's Zaratan cluster and the UMD Math Department's HPC cluster, as they both made the computationally expensive work possible.

Table of Contents

Acknowledgements	ii
Table of Contents	iii
List of Tables	vi
List of Figures	viii
List of Important Abbreviations	xiv
Chapter 1: Introduction	1
I Deep Learning Based Entropy Stable WENO	5
Chapter 2: Conservation Laws and Entropy Stability	6
2.1 Weak formulation	7
2.2 Entropy Conditions	9
2.3 The Riemann Problem	10
2.4 Examples	12
2.4.1 Linear advection equation	12
2.4.2 Burgers' equation	13
2.4.3 Euler equations	14
Chapter 3: Entropy Stable Finite Difference Schemes	16
3.1 Entropy conservative schemes	17
3.1.1 Numerical fluxes for the Euler equations	19
3.2 High-order entropy conservative fluxes	20
3.3 Entropy stable schemes	21
3.3.1 Dissipation operator	22
3.3.2 High-order accurate diffusion operators	22
3.4 Sign-preserving reconstructions	24
3.4.1 ENO interpolation	24
3.4.2 SP-WENO and its variants	25
Chapter 4: DSP-WENO	33
4.1 Deep learning	33

4.2	Vertex Selection Algorithm	38
4.3	Network Details	44
4.3.1	Data Selection	45
4.3.2	Network Architecture and Training	46
4.3.3	Properties of DSP-WENO	47
4.4	Numerical Results	48
4.4.1	Reconstruction Accuracy	49
4.4.2	Scalar conservation laws	51
4.4.3	Euler Equations	58
4.5	High-order extensions	70
Chapter 5: Conclusion for DSP-WENO		75
5.1	Future Directions	77
II Operator Learning with the Petrov-Galerkin formulation		79
Chapter 6: Variational Forms		80
6.1	Statement of the variational form	80
6.2	Finite-dimensional projection	82
6.3	Petrov-Galerkin formulation	83
6.3.1	Details	84
6.3.2	Limitations of Galerkin methods	85
6.4	Optimal Petrov-Galerkin	86
Chapter 7: Operator Learning		89
7.1	DeepONet	90
7.2	VarMiON	92
7.3	FNO	96
Chapter 8: PG-VarMiON for PDE with fixed parameterization		98
8.1	PG-VarMiON Formulation	98
8.2	Training	102
8.3	Theoretical Analysis	104
8.4	Numerical Results	115
8.4.1	Training and testing data	116
8.4.2	Network architectures	118
8.4.3	1D Dirichlet Problems	123
8.4.4	2D Problems	137
Chapter 9: PG-VarMiON for PDE with varying parameterization		161
9.1	PG-VarMiON Formulation	161
9.2	Training	162
9.3	Theoretical Analysis	163
9.4	Numerical Results	170
9.4.1	Training and testing data	171

9.4.2	Network architectures	171
9.4.3	1D Problems	174
9.4.4	2D Problem	187
9.5	Computational resources	194
9.6	Comparison to spline-based FEM	200
9.6.1	Nutils: direct computation	201
9.6.2	Nutils: Petrov-Galerkin formulation	203
9.7	Error Estimate Numerics	204
Chapter 10: PG-VarMiON Conclusion		207
10.1	Future Directions	209
Chapter 11: Final Conclusions		210
Appendix A: Logarithmic average		212
Appendix B: Additional DSP-WENO errors		214
Appendix C: Proof of Theorem 4.1		216
Appendix D: PG-VarMiON with OOD parameterization data		222

List of Tables

3.1	Summary of the possible Ω_Θ along with the sign property and accuracy constraints defining the associated feasible region Ω_C . Here $\mathcal{L}$ is given by (3.27).	32
4.1	Functions used in constructing the training dataset where $\mathbb{U}[p, q]$ denotes the uniform distribution on the interval $[p, q]$	45
4.2	Reconstruction accuracy for inclined sine wave.	50
4.3	L^1 errors and convergence rates for linear advection smooth tests 1 and 2.	53
4.4	L^1 errors in density for advecting isentropic vortex with different reconstructions.	63
8.1	Summary of training settings for all networks in the 1D pure diffusion problem.	124
8.2	1D diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.	126
8.3	1D advection-diffusion problem: Mean relative L^2 projection error (in %) on the test set for the pure sine trial basis with varying dimension size.	130
8.4	Summary of training settings for all networks in the 1D advection-diffusion problem with homogeneous Dirichlet boundary conditions.	132
8.5	1D advection-diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.	136
8.6	Summary of training settings for all networks in the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.	142
8.7	2D advection-diffusion problem (homogeneous Dirichlet BC): Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *.	143
8.8	2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.	150
8.9	2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 test error (in %) with the various methods trained on zero Neumann data. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *.	157
8.10	2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 , H^1 seminorm, and H^1 test errors (in %) with PG-VarMiON trained with respect to each norm. The standard deviation of error is in parentheses.	159

9.1	Summary of training settings for all networks in the 1D advection-diffusion problem with varying parameterization data g	175
9.2	1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *. FNO-UN denotes FNO trained and tested on un-normalized input data.	177
9.3	1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded. FNO-UN denotes FNO trained and tested on un-normalized input data.	183
9.4	1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *. FNO-UN denotes FNO trained and tested on un-normalized input data.	188
9.5	Summary of training settings for PG-VarMiON and FNO in the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.	192
9.6	2D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.	193
9.7	Approximate computational times (in seconds) for training the various operator networks for selected problems.	200
9.8	1D advection-diffusion problem with varying c and κ : Inference times (in seconds) for a single Dataset 1 sample for (a) Nutils and (b) various operator networks.	202
9.9	2D advection-diffusion problem: Inference times (in seconds) for a single Dataset 2 sample for (a) Nutils and (b) various operator networks.	203
9.10	1D advection-diffusion problem with varying c and κ : Sources of error in Theorem 9.2.	205
B.1	L^2 errors and convergence rates for linear advection smooth tests 1 and 2.	215
B.2	L^∞ errors and convergence rates for linear advection smooth tests 1 and 2.	215
D.1	1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) for projection and PG-VarMiON for in-distribution and OOD velocities. The standard deviation of error is in parentheses.	222

List of Figures

2.1	Characteristic lines for simple waves that form the Riemann solution.	12
3.1	Example of a reconstruction that a) satisfies the sign property and b) violates the sign property. The red values represent the original cell center values.	24
4.1	Schematic for the DSP-WENO reconstruction algorithm.	39
4.2	Feasible region for Case 2 (<i>dark grey</i>) and Case 3 (<i>light grey</i>). The $\mathcal{O}(h)$ box $\mathcal{H}$ (outlined by red dots) is centered about the origin. The vertex selection algorithm further restricts the search space to the blue and green regions for Case 2 and Case 3, respectively. (a) $\psi^+ \in (-1, 0)$ with a zoomed in view in (b) , (c) $\psi^+ \in (-\infty, -1)$ with a zoomed in view in (d)	44
4.3	Reconstruction of a simulated discontinuity, which is not well-resolved, using: (a) ENO3, (b) SP-WENO, (c) SP-WENOc, (d) DSP-WENO.	51
4.4	Linear advection of shapes: Solution at time $T = 1.4$. Comparison of different reconstruction methods: (a) $x \in [0, 1.4]$, (b) $x \in [0.4, 1]$	54
4.5	Runtime vs. L^1 error for linear advection: (a) Test 1, (b) Test 2.	55
4.6	Burgers' Test 1: Solution at time $T = 0.5$. Comparison of different reconstruction methods: (a) $x \in [-1, 1]$, (b) $x \in [0.3, 0.5]$	56
4.7	Burgers' Test 2: Solution at time $T = 0.4$. Comparison of different reconstruction methods: (a) $x \in [-4, 4]$, (b) $x \in [0.86, 1.65]$	57
4.8	Burgers' Test 3: Relative change in total entropy with the time that the discontinuity appears marked.	58
4.9	Modified Sod Shock Tube Test. Comparison of density profile at final time $T = 0.2$ for different reconstruction methods: (a) $x \in [0, 1]$, (b) $x \in [0.5, 0.8]$	60
4.10	Shu-Osher Test: Comparison of density profile at final time $T = 1.8$ for different reconstruction methods: (a) $x \in [-5, 5]$, (b) $x \in [0, 2.5]$	61
4.11	Lax Shock Tube Test. Comparison of density profile at final time $T = 1.3$ for different reconstruction methods: (a) $x \in [-5, 5]$, (b) $x \in [1.75, 3.75]$	62
4.12	Double Rarefaction Test: Solution with TeCNO4 at time $T = 0.6$ for ρ , u , and p . Comparison of different reconstruction methods for $x \in [-1, 1]$ for ρ , u , and p	63
4.13	2D Riemann problem (conf. 12): Density profiles at time $T = 0.25$ with 30 contour lines between 0.52 and 2.2. (a)-(d) Comparison of different reconstruction methods. (e) One-dimensional slices of the solution using each reconstruction methods through $x = 0.89$ (bottom left) and $x = 0.5$ (bottom right).	65
4.14	2D Riemann problem (conf. 3): Density profiles at time $T = 0.3$ with 30 contour lines between 0.1 and 2.28 using Roe dissipation. Comparison of different reconstruction methods.	67

4.15	2D Riemann problem (conf. 3): Density profiles at time $T = 0.3$ with 30 contour lines between 0.1 and 2.28 using Rusanov dissipation. Comparison of different reconstruction methods.	68
4.16	2D Riemann problem (conf. 3): Zoom-in of the carbuncle-like phenomenon for SP-WENO and DSP-WENO with Roe and Rusanov dissipation.	69
4.17	Kelvin-Helmholtz Instability: Density profiles at time $T = 3$. Comparison of different reconstruction methods for 256×256 mesh.	71
4.18	Kelvin-Helmholtz Instability: Density profiles at time $T = 3$. Comparison of different reconstruction methods for 512×512 mesh.	72
7.1	Architecture of a basic DeepONet.	91
7.2	Architecture of VarMiON.	95
8.1	Schematic of the PG-VarMiON algorithm for fixed PDE parameterization data $\mathbf{g}$ where the only trainable component is the block in green.	103
8.2	Cut-off function used to enforce the homogeneous Dirichlet boundary conditions.	120
8.3	1D diffusion problem: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	126
8.4	1D diffusion problem: 4 samples from DATASET 1, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	127
8.5	1D diffusion problem: 4 samples from DATASET 2, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	127
8.6	1D diffusion problem: 4 samples from DATASET 3, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	128
8.7	1D diffusion problem: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red).	128
8.8	1D diffusion problem: Mean relative test error (in %) with the various operator models as the number of f samples in the training set is varied. The projection error for each dataset is shown in black.	129
8.9	The trial basis comprised of fifteen orthonormalized functions used for the 1D advection-diffusion problem. Note the presence of boundary layers near the right boundary.	131
8.10	1D advection-diffusion problem: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	133
8.11	1D advection-diffusion problem: 4 samples from DATASET 1, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	134

8.12	1D advection-diffusion problem: 4 samples from DATASET 2, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	134
8.13	1D advection-diffusion problem: 4 samples from DATASET 3, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.	135
8.14	1D advection-diffusion problem: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red).	135
8.15	1D advection-diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods.	136
8.16	The mean relative projection errors for the tensorized B-spline basis against the number of samples used in the optimization. These results are computed on the three test sets for the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.	139
8.17	The velocity field for the 2D advection-diffusion solution sample. c_1 and c_2 are the x and y components of the velocity field, respectively. $\ c\ $ is the magnitude of the velocity. The red arrows indicate the direction of the velocity.	140
8.18	The first sixteen functions of the trial basis used for the 2D advection-diffusion problem (homogeneous Dirichlet BC). These are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$	141
8.19	2D advection-diffusion problem (homogeneous Dirichlet BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	144
8.20	3D plots with contours of the forcing functions for 3 test samples.	144
8.21	3D plots with contours of the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations for 3 test samples.	145
8.22	Plots of the errors of the 3 test samples for the projection, PG-VarMiON, L-DeepONet, and FNO approximations.	146
8.23	1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51$, $x = 0.51$	147
8.24	Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$	148
8.25	The first sixteen functions of the trial basis used for the 2D advection-diffusion problem (mixed BC). These are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$	149
8.26	2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	151
8.27	3D plots with contours of the forcing functions and plots of η for 3 test samples.	152
8.28	3D plots with contours of the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations for 3 test samples.	153

8.29	Plots of the errors of the 3 test samples for the projection, PG-VarMiON, L-DeepONet, and FNO approximations.	154
8.30	1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51, x = 0.51$	155
8.31	Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$	156
8.32	2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses. The test sets consist of samples with zero Neumann data.	158
8.33	2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses. The test sets consist of samples with non-zero Neumann data.	158
9.1	Schematic of the PG-VarMiON algorithm for varying PDE parameterization data $\mathbf{g}$ where the only trainable component is the block in green.	162
9.2	The B-spline activation function in the k -th hidden layer of an MLP. Notice that each neuron shares the same B-spline basis, but the linear combination weights are unique to each neuron.	172
9.3	1D advection-diffusion problem with varying $\mathbf{c}$: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	178
9.4	1D advection-diffusion problem with varying $\mathbf{c}$: 3 samples from DATASET 3, with the forcing functions f and velocities $\mathbf{c}$ plotted in the first and second columns. The corresponding reference solutions (black lines) are plotted in the third column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).	179
9.5	1D advection-diffusion problem with varying $\mathbf{c}$: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.4.	180
9.6	1D advection-diffusion problem with varying $\mathbf{c}$: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.4.	180
9.7	The trial basis comprised of sixteen orthonormalized functions used for the 1D advection-diffusion problem. Note the presence of boundary layers near the left and right boundaries.	181
9.8	1D advection-diffusion problem with varying $\mathbf{c}$: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	183

9.9	1D advection-diffusion problem with varying c : 3 samples from DATASET 3, with the forcing functions f and velocities c plotted in the first and second columns. The corresponding reference solutions (black lines) are plotted in the third column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).	184
9.10	1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.9.	185
9.11	1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.9.	185
9.12	1D advection-diffusion problem with varying c and κ : Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	188
9.13	1D advection-diffusion problem with varying c and κ : 3 samples from DATASET 3, with the forcing functions f , κ , and velocities c plotted in the first, second, and third columns. The corresponding reference solutions (black lines) are plotted in the fourth column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).	189
9.14	1D advection-diffusion problem with varying c and κ : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.13.	189
9.15	1D advection-diffusion problem with varying c and κ : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.13.	190
9.16	1D advection-diffusion problem with varying c and κ : Mean relative test error (in %) with the various operator models as the number of f samples in the training set is varied. The projection error for each dataset is shown in black.	190
9.17	2D advection-diffusion problem with varying c : Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.	193
9.18	3D plots with contours of the forcing functions for 3 test samples as well as the magnitude of the velocity. The red arrows indicate the direction of the velocity.	194
9.19	3D plots with contours of the reference solutions, projection, PG-VarMiON, and FNO approximations for 3 test samples.	195
9.20	Plots of the errors of the 3 test samples for the projection, PG-VarMiON, and FNO approximations.	196
9.21	1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51$, $x = 0.51$	197
9.22	Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$. These correspond to the first sample in Figure 9.18.	198

9.23	Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$. These correspond to the second sample in Figure 9.18.	199
9.24	1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 error (in %) for the Nutils solution as the number of degrees of freedom increases for 50 samples in each dataset. The mean PG-VarMiON performance is shown in black.	201
9.25	2D advection-diffusion problem: Comparison of mean relative L^2 error (in %) for the Nutils solution as the number of degrees of freedom increases for 50 samples in each dataset. The mean PG-VarMiON performance is shown in black.	202
9.26	1D advection-diffusion problem with varying c and κ : Comparison of computation time for the optimal weighting functions with Nutils for varying degrees of freedom and accuracy of the Petrov-Galerkin formulation using these computed weighting functions. The Nutils cost is scaled by the cost of PG-VarMiON (actual cost of PG-VarMiON: 0.01 s).	204
9.27	2D advection-diffusion problem: Comparison of computation time for the optimal weighting functions with Nutils for varying degrees of freedom and accuracy of the Petrov-Galerkin formulation using these computed weighting functions. The Nutils cost is scaled by the cost of PG-VarMiON (actual cost of PG-VarMiON: 0.01 s).	205
9.28	1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 test error (in %) between projection and PG-VarMiON for different trial basis sizes for the three test datasets.	206

List of Important Abbreviations

DeepONet	Deep Operator Network
DSP-WENO	Deep sign-preserving weighted essentially nonoscillatory
ENO	Essentially nonoscillatory
FNO	Fourier Neural Operator
OOD	out-of-distribution
PDE	Partial differential equation
PG-VarMiON	Petrov-Galerkin Variationally Mimetic Operator Network
SP-WENO	Sign-preserving weighted essentially nonoscillatory
VarMiON	Variationally Mimetic Operator Network
WENO	Weighted essentially nonoscillatory

Chapter 1: Introduction

There are several classes of numerical algorithms to solve partial differential equations (PDEs), each accompanied by theoretical error estimates and convergence rates. In general, while these methods are mathematically sound, there may be some inefficiencies or bottlenecks present. In developing new numerical solvers or augmenting existing ones, maintaining the mathematical rigor is an objective of paramount importance.

Deep learning, while it has found use in innumerable applications across various scientific domains, is generally considered to produce black-box models. Thus, naively attempting to integrate it with solving PDEs can be met with skepticism as the interpretability can be lacking. Thus, when we incorporate deep learning, we want to diligently maintain as much of the aforementioned rigor as possible in the hope that we can increase generalizability and interpretability.

There are many deep learning-based strategies that are used to solve scientific computing problems. Examples include building surrogate models to solve PDEs [20, 68, 75, 81], learning parameter-to-observable maps [48, 76, 77], solving Bayesian inference problems [22, 38, 82, 92, 102], and learning closure models [6, 79, 96]. In this dissertation, we focus on two problems that can benefit from careful deep learning integration that is mathematically consistent.

The first type of problem corresponds to solving hyperbolic conservation laws, which include prominent examples such as the compressible Euler equations, the shallow water equa-

tions, and the magnetohydrodynamic equations. One important class of solvers is entropy stable schemes. With such methods, the recovered numerical solution is guaranteed to be a physically-relevant solution. There are multiple techniques [14, 27, 97] to construct an entropy stable scheme. Following the work of Tadmor [97], an entropy stable scheme can be formulated by starting with an entropy conservative flux and adding a dissipation operator. In particular, arbitrary high-order accurate finite difference schemes called TeCNO schemes involve two key ingredients. Firstly, an arbitrary high-order accurate entropy conservative flux is required, which can be determined as in [66]. Secondly, the dissipation operator is formed by interpolating the solution at each cell interface and computing the reconstructed jump at the interface. Crucially, the TeCNO scheme is only entropy stable if the reconstruction algorithm used satisfies the sign property, that is, the sign of the reconstructed jump must match the sign of the original jump.

Some reconstructions such as essentially nonoscillatory (ENO) interpolation and sign-preserving variants of weighted ENO (SP-WENO) do satisfy the sign property and can be used to form entropy stable TeCNO schemes. However, beyond these, there are fleetingly few valid reconstructions. Moreover, these methods have their own issues. Specifically, for SP-WENO, the resulting entropy stable scheme can produce solutions where spurious Gibbs oscillations appear near shocks.

To better produce sign-preserving WENO reconstructions, we can use deep learning to generate a weight selection strategy based on data. Deep learning has been used successfully to augment other hyperbolic conservation law solvers. For example, deep-learned troubled-cell indicators such as those in [90, 91] are designed to be integrated into discontinuous Galerkin (DG) schemes to identify elements containing discontinuities. Our method DSP-WENO involves a constrained neural network that always satisfies the sign property, and within these constraints,

it is trained with relevant data to better capture shocks. The constraints of the network ensure the accuracy of the method, leaving the data-driven process to select WENO weights that provide the best shock-capturing behavior.

The second type of problem this dissertation focuses on involves learning solution operators for PDEs. There have been any advancements in the field of operator learning, where the goal is to approximately learn the PDE solution operator that maps PDE data to the solution. Operator networks such as DeepONets [75] and FNOs [68] are neural networks that are trained to learn the solution operator. In particular, these are often used as starting points to develop better operator networks. There are also strategies to improve an operator network’s architecture by encoding more of the mathematical structure of the PDE. An example is VarMiON [81], where the model mimics the variational form of the underlying PDE.

We propose to further improve on the VarMiON structure by instead mimicking the Petrov-Galerkin variational form of the PDE. The Petrov-Galerkin formulation has been developed as a robust alternative to standard Galerkin methods [4], which can fail for certain problems. However, using it to solve many instances of a PDE can be an exceedingly expensive operation. Thus, we design an operator network PG-VarMiON to mimic this formulation such that the bottleneck present can be efficiently handled via a neural network approximation. This allows PG-VarMiON to have improved generalizability capabilities over similar operator networks.

The rest of the dissertation is organized in the following manner:

Part 1 comprises **Chapters 2-5** where deep learning is explored to augment an existing entropy stable PDE solver.

Chapter 2 introduces the basics of hyperbolic conservation laws. The solution frameworks for selected problems are discussed. Further, the notion of entropy stability is introduced.

In **Chapter 3**, we introduce entropy stable schemes and the necessary theory for them. In particular, we focus on the high-order accurate TeCNO solver.

In **Chapter 4**, we discuss DSP-WENO, a novel entropy stable solver that comprises a constrained deep-learned reconstruction algorithm and an existing entropy stable numerical solver.

In **Chapter 5**, we conclude the discussion on DSP-WENO, highlighting impact and future directions.

Part 2 comprises **Chapters 6-10** where deep learning is used to mimic a general PDE solver while leveraging the variational form of the PDE.

Chapter 6 introduces the variational form of the PDE. The Petrov-Galerkin framework is discussed as well along with the notion of optimal weighting functions.

Chapter 7 discusses operator networks and specifically introduces those that are most relevant to this dissertation.

In **Chapter 8**, we discuss the PG-VarMiON framework and show numerical results for diffusion and advection-diffusion problems in one and two dimensions. The focus here is limited to problems where the PDE parameterization data is fixed across samples.

Chapter 9 furthers the discussion on PG-VarMiON by considering more complicated advection-diffusion problems where the PDE parametrization data varies across samples.

In **Chapter 10**, we conclude the discussion on PG-VarMiON, highlighting impact and future directions.

In **Chapter 11**, we offer a final conclusion and discuss the impact of the work presented in this dissertation.

Part I

Deep Learning Based Entropy Stable WENO

Chapter 2: Conservation Laws and Entropy Stability

Hyperbolic systems of conservation laws dictate the behavior of quantities that are conserved in time as the system evolves. These systems of PDEs are ubiquitous across many disciplines, including the compressible Euler equations for simulating the flow past aircraft wings, the shallow water equations for simulating sea waves/tidal waves, and the magnetohydrodynamic equations for simulating supernovas. A generic system of conservation laws can be mathematically expressed in the strong form as

$$\frac{\partial}{\partial t} \mathbf{u}(\mathbf{x}, t) + \sum_{i=1}^d \frac{\partial}{\partial x_i} \mathbf{f}(\mathbf{u}(\mathbf{x}, t)) = \mathbf{0} \quad \forall \mathbf{x} \in \mathbb{R}^d, t \in \mathbb{R}^+ \quad (2.1)$$

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^d,$$

where $\mathbf{u}: \mathbb{R}^d \times \mathbb{R}^+ \rightarrow \mathbb{R}^m$ represents a vector of m conserved variables and $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_d)$ where $\mathbf{f}_i: \mathbb{R}^m \rightarrow \mathbb{R}^m$ is a Cartesian component of the smooth flux, representing the flow of the conserved quantities. The specification of the initial condition defines (2.1) as the Cauchy problem for conservation laws.

Definition 2.1. Let $\mathbf{A}_i(\mathbf{u}) := \mathbf{f}'_i(\mathbf{u})$ be the flux Jacobians. Then the system in (2.1) is called *hyperbolic* if, for any $\mathbf{u} \in \Pi \subset \mathbb{R}^m$ and any $\mathbf{n} \in \mathbb{R}^d$, the matrix

$$\mathbf{A}(\mathbf{u}, \mathbf{n}) = \sum_{i=1}^d \mathbf{A}_i(\mathbf{u}) n_i,$$

has m real eigenvalues $\lambda_1(\mathbf{u}) \leq \dots \leq \lambda_m(\mathbf{u})$ and m linearly independent eigenvectors $\mathbf{r}_1(\mathbf{u}), \dots, \mathbf{r}_m(\mathbf{u})$. The system is “strictly hyperbolic” if the eigenvalues are distinct.

The set Π typically denotes some admissible set, which is dictated by the constraints on $\mathbf{u}$, e.g. positivity of certain quantities. Each pair $(\lambda_i(\mathbf{u}), \mathbf{r}_i(\mathbf{u}))$ corresponding to $\mathbf{A}(\mathbf{u}, \mathbf{n})$ defines a *characteristic field* called the λ_i field. If $\langle \lambda'_i(\mathbf{u}), \mathbf{r}_i(\mathbf{u}) \rangle = 0$ for all $\mathbf{u} \in \Pi$, the λ_i field is *linearly degenerate*. If $\langle \lambda'_i(\mathbf{u}), \mathbf{r}_i(\mathbf{u}) \rangle \neq 0$ for all $\mathbf{u} \in \Pi$, the λ_i field is *genuinely nonlinear*. In this dissertation, we only consider hyperbolic conservation laws with linearly degenerate or genuinely nonlinear characteristic fields.

2.1 Weak formulation

It is well-known that solutions to nonlinear conservation laws can develop discontinuities in finite time even with smooth initial conditions. Thus, solutions to (2.1) must be understood in the weak (distributional) sense.

Definition 2.2 (Weak solution). *A function $\mathbf{u} \in (L^1_{loc}(\mathbb{R}^d \times \mathbb{R}^+))^m$ is a weak solution of the Cauchy problem (2.1) if*

$$\int_0^\infty \int_{\mathbb{R}^d} \left(\langle \mathbf{u}, \frac{\partial}{\partial t} \phi \rangle + \sum_{i=1}^d \langle \mathbf{f}_i(\mathbf{u}), \frac{\partial}{\partial x_i} \phi \rangle \right) dt d\mathbf{x} + \int_{\mathbb{R}^d} \langle \mathbf{u}_0(\mathbf{x}), \phi(\mathbf{x}, 0) \rangle d\mathbf{x} = 0 \quad (2.2)$$

for all $\phi \in (C_0^\infty(\mathbb{R}^d \times \mathbb{R}^+))^m$.

Equation (2.2) is obtained by integrating the inner product of the strong form of the conservation law in (2.1) and the test function ϕ over time and space. With all derivatives transferred onto the test function, we are able to consider a larger admissible set of solutions, including those

that have less regularity. Even so, not every discontinuity is admissible. Let Γ be a surface of discontinuity in the $t - \boldsymbol{x}$ plane for the solution $\boldsymbol{u}$. Let $\tilde{\mathbf{n}} = (n_t, n_1, \dots, n_d) \neq 0$ be the normal vector to Γ . Further, denote $\boldsymbol{u}_+$ and $\boldsymbol{u}_-$ as the limits of $\boldsymbol{u}$ on either side of Γ

$$\boldsymbol{u}_\pm(\boldsymbol{x}, t) = \lim_{\epsilon \downarrow 0} \boldsymbol{u}((\boldsymbol{x}, t) \pm \epsilon \tilde{\mathbf{n}}).$$

Theorem 2.1 (Rankine-Hugoniot (RH) condition). *The weak solution $\boldsymbol{u}$ of (2.1) must satisfy the jump condition*

$$(\boldsymbol{u}_+ - \boldsymbol{u}_-)n_t + \sum_{i=1}^d (\boldsymbol{f}_i(\boldsymbol{u}_+) - \boldsymbol{f}_i(\boldsymbol{u}_-))n_i = 0, \quad (2.3)$$

across the surface of discontinuity Γ .

In one-dimension, assuming that Γ is parametrized as $(t, \xi(t))$, we see $\tilde{\mathbf{n}} = (-s, 1)$ with $s = d\xi/dt$ being the speed of the discontinuity. Thus, the RH condition the speed s as

$$s(\boldsymbol{u}_+ - \boldsymbol{u}_-) = \boldsymbol{f}(\boldsymbol{u}_+) - \boldsymbol{f}(\boldsymbol{u}_-). \quad (2.4)$$

Every classical or strong solution is clearly a weak solution. However, weak solutions are not unique in general, so additional constraints in terms of the *entropy conditions* must be imposed such that a physically relevant weak solution is extracted.

2.2 Entropy Conditions

Assume that for (2.1), $\eta : \mathbb{R}^m \rightarrow \mathbb{R}$ is a convex *entropy function* and $q_i : \mathbb{R}^m \rightarrow \mathbb{R}$ is a Cartesian coordinate of the *entropy flux* function satisfying the compatibility condition

$$\nabla_{\mathbf{u}} q_i(\mathbf{u})^\top = \nabla_{\mathbf{u}} \eta(\mathbf{u})^\top \nabla_{\mathbf{u}} \mathbf{f}_i(\mathbf{u}) \quad i = 1, 2, \dots, d. \quad (2.5)$$

Define $\mathbf{v} = \nabla_{\mathbf{u}} \eta(\mathbf{u})$ as the *entropy variables*. By multiplying (2.1) with $\mathbf{v}$, the additional conservation law

$$\frac{\partial}{\partial t} \eta(\mathbf{u}) + \sum_{i=1}^d \frac{\partial}{\partial x_i} q_i(\mathbf{u}) = 0, \quad (2.6)$$

which is exactly satisfied for smooth solutions (i.e. entropy is conserved). Weak solutions will not necessarily satisfy (2.6) since entropy is dissipated near shocks, and so, the entropy condition states that weak solutions should satisfy the following entropy inequality

$$\frac{\partial}{\partial t} \eta(\mathbf{u}) + \sum_{i=1}^d \frac{\partial}{\partial x_i} q_i(\mathbf{u}) \leq 0, \quad (2.7)$$

which is understood in the weak sense, i.e.,

$$\int_0^\infty \int_{\mathbb{R}^d} \left(\eta(\mathbf{u}) \frac{\partial}{\partial t} \phi + \sum_{i=1}^d q_i(\mathbf{u}) \frac{\partial}{\partial x_i} \phi \right) dt d\mathbf{x} + \int_{\mathbb{R}^d} \eta(\mathbf{u}_0(\mathbf{x})) \phi(\mathbf{x}, 0) d\mathbf{x} \geq 0, \quad (2.8)$$

for all $\phi \in C_0^\infty(\mathbb{R}^d \times \mathbb{R}^+)$ with $\phi(\mathbf{x}, t) \geq 0$. The solution $\mathbf{u}$ of (2.1) is said to be an *entropy solution* if it satisfies (2.7) for all admissible entropy pairs $(\eta(\mathbf{u}), q(\mathbf{u}))$ associated with (2.1).

Scalar conservation laws ($m = 1$) have unique entropy solutions [62], but the uniqueness

is not guaranteed for general systems of conservation laws [67, 18]. However, the entropy conditions provide the only nonlinear estimates currently available for generic systems of conservation laws [21], and are thus essential. In practice, the entropy condition (2.7) is considered with respect to a specific choice of $(\eta(\mathbf{u}), q(\mathbf{u}))$ for a general system (2.1).

2.3 The Riemann Problem

To understand the behavior of the solution of conservation laws when discontinuities develop, it is typical to analyze the *Riemann problem*. For simplicity, we consider the Cauchy problem for a one-dimensional system of hyperbolic conservation laws

$$\begin{aligned} \frac{\partial}{\partial t} \mathbf{u} + \frac{\partial}{\partial x} f(\mathbf{u}) &= 0, \\ \mathbf{u}_0(x) &= \begin{cases} \mathbf{u}_L & \text{if } x < 0 \\ \mathbf{u}_R & \text{if } x > 0 \end{cases}, \end{aligned} \tag{2.9}$$

where $\mathbf{u}_L$ and $\mathbf{u}_R$ are constant states. This is the simplest initial value problem that can be posed for conservation laws, and the solution of (2.9) serves as the building block for more complicated problems. We assume $\mathbf{u}_L \neq \mathbf{u}_R$, the initial condition is discontinuous. Assuming that the system is strictly hyperbolic, so the flux Jacobian $\mathbf{A}(\mathbf{u}) = \mathbf{f}'(\mathbf{u})$ has real and distinct eigenvalues $\lambda_1(\mathbf{u}) < \dots < \lambda_m(\mathbf{u})$. The solution to the Riemann problem comprises m waves with each one, corresponding to an eigenvalue, emanating from the origin. We can describe the solutions of (2.9) as self-similar, meaning they are of the form $\mathbf{u}(x, t) = \mathbf{w}(x/t)$. Further, the solution consists of $m + 1$ constant states that are separated by m waves. States are connected by one of the three following waves:

- **Shock wave:** The λ_i -wave is a shock wave if it corresponds to a genuinely nonlinear field and connects two states $\mathbf{u}_-$ and $\mathbf{u}_+$ by a single jump discontinuity. To be a shock, the Rankine-Hugoniot condition must be satisfied, which provides the speed S_i at which the shock moves. The characteristic lines $dx/dt = \lambda_i$ on both sides of the shock line meet and merge into the shock wave itself. This is represented by the (Lax) entropy condition [37]:

$$\lambda_i(\mathbf{u}_-) > S_i > \lambda_i(\mathbf{u}_+). \quad (2.10)$$

Figure 2.1a depicts this behavior.

- **Contact wave:** The λ_i -wave is a contact wave if it corresponds to a linearly degenerate field and connects two states $\mathbf{u}_-$ and $\mathbf{u}_+$ by a single jump discontinuity. The RH condition again determines the speed S_i at which the discontinuity moves. Additionally, the characteristic lines on either side of the contact line run parallel to it:

$$\lambda_i(\mathbf{u}_-) = S_i = \lambda_i(\mathbf{u}_+).$$

Figure 2.1b depicts this behavior.

- **Rarefaction wave:** The λ_i -wave is a rarefaction wave if it connects two states $\mathbf{u}_-$ and $\mathbf{u}_+$ through a smooth transition in a genuinely nonlinear field. Here, the characteristic lines diverge from each other without meeting, i.e.,

$$\lambda_i(\mathbf{u}_-) < \lambda_i(\mathbf{u}_+).$$

Figure 2.1c depicts this behavior.

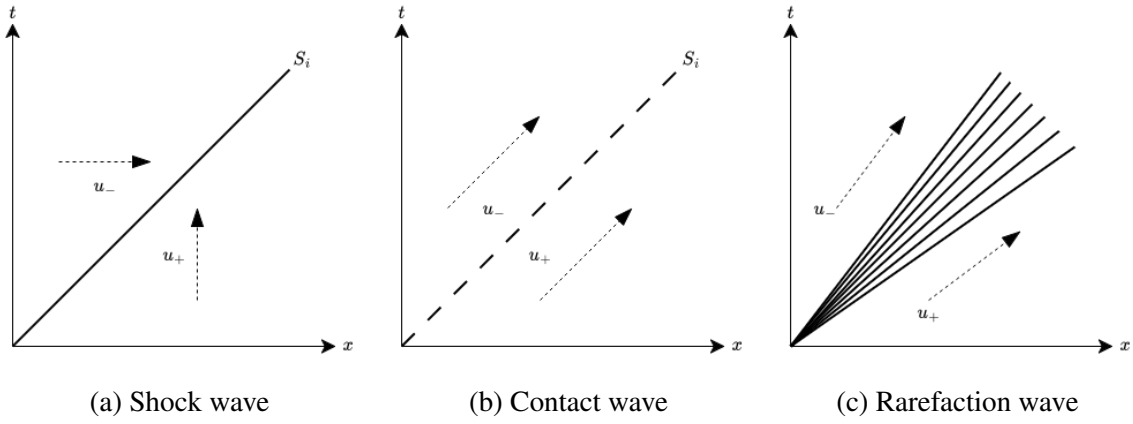


Figure 2.1: Characteristic lines for simple waves that form the Riemann solution.

2.4 Examples

We now describe a few conservation laws.

2.4.1 Linear advection equation

The simplest conservation law of all is the linear advection equation where the flux is $f_i(x) = a_i x$. Thus, it is given by

$$\frac{\partial}{\partial t} u + \mathbf{a} \cdot \nabla u = 0, \quad (2.11)$$

where u is a scalar conserved variable and $\mathbf{a} = (a_1, \dots, a_d)$ is the constant advection velocity.

With initial condition $u(\mathbf{x}, t) = u_0(\mathbf{x})$, the exact solution is

$$u(\mathbf{x}, t) = u_0(\mathbf{x} - \mathbf{a}t),$$

which simply means that the initial condition is advected by the velocity $\mathbf{a}$. For linear advection (and more generally for scalar conservation laws), we choose the quadratic entropy function $\eta(u) = u^2/2$. Of course, this means that the corresponding entropy variable is $v = \eta'(u) = u$. Using the compatibility condition (2.5), the corresponding entropy flux functions can be determined as

$$q_i(u) = \int_u \eta'(z) f'_i(z) dz = \int_u a_i z dz = \frac{a_i u^2}{2}. \quad (2.12)$$

Given that the flux function is linear, it is impossible to develop shocks in the solution. Any discontinuities in the solution must be connected by contact waves.

2.4.2 Burgers' equation

The simplest nonlinear conservation law is the Burgers' equation where the flux is $f_i(u) = \frac{u^2}{2}$. Thus, it is given by

$$\frac{\partial}{\partial t} u + u \nabla u = 0. \quad (2.13)$$

By once again using the quadratic entropy function, the corresponding entropy flux function is determined as

$$q_i(u) = \int_u \eta'(z) f'_i(z) dz = \int_u z^2 dz = \frac{u^3}{3}. \quad (2.14)$$

In this problem, shocks can develop in the solution due to the nonlinear flux, which means that the total entropy can be dissipated.

2.4.3 Euler equations

The Euler equations are an example of a nonlinear system of conservation laws. They are important, as they describe compressible flow mechanics when viscous forces are absent. The system for three spatial dimensions is described by

$$\frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \rho \mathbf{u} \\ E \end{pmatrix} + \nabla \cdot \begin{pmatrix} \rho \mathbf{u} \\ p \mathbf{I} + \rho(\mathbf{u} \otimes \mathbf{u}) \\ (E + p) \mathbf{u} \end{pmatrix} = \mathbf{0}, \quad (2.15)$$

where ρ , $\mathbf{u} = (u_1, u_2, u_3)^\top$, and p represent the fluid density, velocity, and pressure, respectively. Note that $\mathbf{u}$ is distinct from $\mathbf{u}$, with the latter representing the vector of conserved variables. The total energy per unit volume is given by

$$E = \rho \left(\frac{|\mathbf{u}|^2}{2} + e \right). \quad (2.16)$$

where e is the specific internal energy given by a caloric equation of state, $e = e(\rho, p)$. Here, we take the equation of state to be

$$e = \frac{p}{(\gamma - 1)\rho},$$

where $\gamma = c_p/c_v$ denotes the ratio of specific heats.

Physically, the first equation of (2.15) represents the conservation of mass. The second

equation describes the conservation of the three components of momentum, and the last equation describes the conservation of energy.

For systems of conservation laws, the selection of an entropy-entropy flux pair can be difficult and must be done with care. For the Euler equations, a popular choice for the entropy-entropy flux pair [44] is

$$\eta = -\frac{\rho s}{\gamma - 1}, \quad \mathbf{q} = [q_1, q_2, q_3] = -\frac{\rho \mathbf{u}^\top s}{\gamma - 1}, \quad (2.17)$$

where $s = \ln(p) - \gamma \ln(\rho)$ is associated with the thermodynamic specific entropy. The corresponding vector of entropy variables is given by

$$\mathbf{v} = \left(\frac{\gamma - s}{\gamma - 1} - \beta |\mathbf{u}|^2, \quad 2\beta \mathbf{u}^\top, \quad -2\beta \right)^\top. \quad (2.18)$$

where $\beta = \rho/(2p)$.

Chapter 3: Entropy Stable Finite Difference Schemes

Finite difference schemes are a popular family of methods to employ for solving PDEs. In particular, we discuss entropy stable finite difference schemes, which are used to solve conservation laws. We consider a one-dimensional finite difference formulation for ease of discussion, which can easily be extended to higher dimensions. We partition the spatial domain $\Omega = [a, b]$ into N disjoint cells $I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}})$ of uniform length $h = (b-a)/N$ with cell center x_i where

$$x_{i+\frac{1}{2}} = a + ih, \quad \forall 0 \leq i \leq N \quad \text{and} \quad x_i = \frac{x_{i-\frac{1}{2}} + x_{i+\frac{1}{2}}}{2} \quad \forall 1 \leq i \leq N.$$

Keeping time continuous, the semi-discrete finite difference scheme for (2.1) is expressed as

$$\frac{d\mathbf{u}_i(t)}{dt} + \frac{1}{h}(\mathbf{f}_{i+\frac{1}{2}} - \mathbf{f}_{i-\frac{1}{2}}) = 0. \quad (3.1)$$

Here, $\mathbf{u}_i(t)$ approximates the point values of the solution to (2.1) at cell center x_i and time t , while $\mathbf{f}_{i+\frac{1}{2}}$ is a consistent, conservative numerical approximation of the flux $\mathbf{f}$ at the cell interface $x_{i+\frac{1}{2}}$.

Entropy stable schemes are numerical PDE solvers for hyperbolic conservation laws that are designed to provide physically-relevant solutions. This dissertation considers the approach developed in [97], which entails constructing an entropy conservative flux and augmenting with a dissipation term. Another class of solvers are based on the summation-by-parts (SBP) method-

ology where the mesh is discretized with complementary grids where the flux is first interpolated to the intermediate points on another grid and then differentiated back to the original [27]. SBP operators have also been used in the discontinuous Galerkin framework to develop entropy stable schemes [14].

3.1 Entropy conservative schemes

The approach to develop an entropy stable scheme that we consider requires that we start with an entropy conservative scheme [97].

Definition 3.1. *The numerical scheme (3.1) is entropy conservative if it satisfies the discrete entropy equality*

$$\frac{d\eta(\mathbf{u}_i)}{dt} + \frac{1}{h} \left(\tilde{q}_{i+\frac{1}{2}} - \tilde{q}_{i-\frac{1}{2}} \right) = 0, \quad (3.2)$$

where $\tilde{q}_{i+\frac{1}{2}}$ is a numerical entropy flux consistent with entropy flux q in (2.7).

We denote the undivided jump and average across the interface $x_{i+\frac{1}{2}}$ by $\Delta(\cdot)_{i+\frac{1}{2}} = (\cdot)_{i+1} - (\cdot)_i$ and $\overline{(\cdot)}_{i+\frac{1}{2}} = ((\cdot)_{i+1} + (\cdot)_i)/2$, respectively.

Additionally, we introduce the *entropy potential* $\Psi(\mathbf{u}) := \mathbf{v}(\mathbf{u})^\top \mathbf{f}(\mathbf{u}) - q(\mathbf{u})$, where we recall the vector of entropy variables $\mathbf{v}(\mathbf{u}) = \nabla_{\mathbf{u}} \eta(\mathbf{u})$. A sufficient condition for constructing entropy conservative fluxes is provided by the following theorem.

Theorem 3.1. ([97]) *The second-order accurate numerical scheme (3.1) with the two-point numerical flux $\tilde{\mathbf{f}}_{i+\frac{1}{2}} = \tilde{\mathbf{f}}_{i+\frac{1}{2}}(\mathbf{u}_i, \mathbf{u}_{i+1})$ is entropy conservative if*

$$(\Delta \mathbf{v}_{i+\frac{1}{2}})^\top \tilde{\mathbf{f}}_{i+\frac{1}{2}} = \Delta \Psi_{i+\frac{1}{2}}. \quad (3.3)$$

Specifically, it satisfies the entropy condition with the consistent numerical entropy flux given by

$$\tilde{q}_{i+\frac{1}{2}} = (\bar{\mathbf{v}}_{i+\frac{1}{2}})^\top \tilde{\mathbf{f}}_{i+\frac{1}{2}} - \bar{\Psi}_{i+\frac{1}{2}}.$$

The above theorem yields a simple recipe for the entropy conservative flux for scalar conservation laws. Given an entropy function $\eta(\mathbf{u})$, the expression for the flux is

$$\tilde{f}_{i+\frac{1}{2}} = \frac{\Delta \Psi_{i+\frac{1}{2}}}{\Delta \mathbf{v}_{i+\frac{1}{2}}}.$$

Example 3.1. Consider the linear advection equation with flux $f(u) = cu$ with constant c . With the square entropy $\eta(u) = u^2/2$ and corresponding entropy flux $q(u) = cu^2/2$, the entropy conservative flux is

$$\tilde{f}_{i+\frac{1}{2}} = c \frac{u_i + u_{i+1}}{2}. \quad (3.4)$$

Example 3.2. Consider the Burgers' equation with flux $f(u) = u^2/2$. With the square entropy $\eta(u) = u^2/2$ and corresponding entropy flux $q(u) = u^3/3$, the entropy conservative flux is

$$\tilde{f}_{i+\frac{1}{2}} = \frac{u_i^2 + u_{i+1}^2 + u_i u_{i+1}}{6}. \quad (3.5)$$

Remark 3.1. For systems of conservation laws, the relation (3.3) does not necessarily have a unique solution. In [97], the following path-integral formulation was proposed

$$\tilde{f}_{i+\frac{1}{2}} = \int_0^1 \mathbf{f}(\mathbf{v}_{i+\frac{1}{2}}(s)) ds, \quad \mathbf{v}_{i+\frac{1}{2}}(s) = \mathbf{v}_i + s(\mathbf{v}_{i+1} - \mathbf{v}_i).$$

However, this does not always admit a closed-form expression and requires approximations, which can be expensive [29].

3.1.1 Numerical fluxes for the Euler equations

For the Euler equations, a popular choice for the entropy-entropy flux pair [44] is $\eta = -\rho s/(\gamma - 1)$, $\mathbf{q} = [q_1, q_2, q_3] = \eta \mathbf{u}^\top$ where $s = \ln(p) - \gamma \ln(\rho)$ is associated with the thermodynamic specific entropy. The corresponding vector of entropy variables is given by $\mathbf{v} = [-\beta|\mathbf{u}|^2 - (\gamma - s)/(\gamma - 1), 2\beta\mathbf{u}^\top, -2\beta]^\top$ where $\beta = \rho/(2p)$.

To construct the fourth-order entropy conservative flux (3.9), there are a few options for two-point entropy conservative fluxes. For example, there is Roe's entropy conservative flux [53]. However, beyond entropy conservation, there is another desirable property for numerical schemes for the Euler equations that this particular flux lacks. This property is kinetic energy preservation, which means that the evolution of the discrete kinetic energy is consistent with the evolution of the total kinetic energy. Jameson's kinetic energy preserving (KEP) flux [54] was developed to have this property but is not entropy conservative. We refer interested readers to [11, 88] for further details on kinetic energy preserving fluxes.

The second-order flux that we will use in this dissertation is the kinetic energy preserving and entropy conservative (KEPEC) flux [11], which in 1D is expressed as:

$$\mathbf{f} = \begin{pmatrix} F^\rho \\ F^m \\ F^e \end{pmatrix} = \begin{pmatrix} \hat{\rho}\bar{\mathbf{u}} \\ \tilde{p} + \bar{\mathbf{u}}F^\rho \\ F^e \end{pmatrix}, \quad F^e = \left[\frac{1}{2(\gamma - 1)\hat{\beta}} - \frac{1}{2}|\bar{\mathbf{u}}|^2 \right] F^\rho + \bar{\mathbf{u}}F^m, \quad (3.6)$$

where $\tilde{p} = \bar{p}/(2\bar{\beta})$ and $\beta = \rho/(2p)$. $\hat{\rho}$, $\hat{\beta}$ represent the logarithmic averages (see [53, 11] and Appendix A) of the respective positive quantities.

3.2 High-order entropy conservative fluxes

The numerical fluxes described in Section 3.1 are only second-order accurate. Arbitrary high-order entropy conservative fluxes $\tilde{\mathbf{f}}^p$ (of order p) can be constructed using linear combinations of second-order two-point entropy conservative fluxes [66]. This procedure is outlined in the following theorem.

Theorem 3.2 ([66]). *For $p \in \mathbb{N}$, assume that $\alpha_1^p, \alpha_2^p, \dots, \alpha_p^p \in \mathbb{R}$ solve the linear equations*

$$2 \sum_{r=1}^p r \alpha_r^p = 1, \quad \sum_{l=1}^p l^{2s-1} \alpha_r^p = 0 \quad (s = 2, \dots, p), \quad (3.7)$$

and define the flux

$$\tilde{\mathbf{f}}_{i+\frac{1}{2}}^{2p} = \sum_{r=1}^p \alpha_r^p \sum_{s=0}^{r-1} \tilde{\mathbf{f}}(\mathbf{u}_{i-s}, \mathbf{u}_{i-s+r}), \quad (3.8)$$

where $\tilde{\mathbf{f}}$ is a two-point entropy conservative flux satisfying (3.3). Then, the semi-discrete scheme (3.1) with flux $\tilde{\mathbf{f}}^{2p}$ is

- *2pth order accurate. That is, for sufficiently smooth solutions $\mathbf{u}$ we have*

$$\frac{\tilde{\mathbf{f}}_{i+\frac{1}{2}}^{2p} - \tilde{\mathbf{f}}_{i-\frac{1}{2}}^{2p}}{h} = \frac{\partial}{\partial x} \mathbf{f}(\mathbf{u}_i) + \mathcal{O}(h^{2p}).$$

- *entropy conservative. Thus, it satisfies the discrete entropy identity*

$$\frac{d\eta_i}{dt} + \frac{1}{h} \left(\tilde{q}_{i+\frac{1}{2}}^{2p} - \tilde{q}_{i-\frac{1}{2}}^{2p} \right) = 0,$$

where

$$\tilde{q}_{i+\frac{1}{2}}^{2p} = \sum_{r=1}^p \alpha_r^p \sum_{s=0}^{r-1} \tilde{q}(\mathbf{u}_{i-s}, \mathbf{u}_{i-s+r}),$$

with $\tilde{q}$ being the consistent numerical entropy flux corresponding to $\tilde{\mathbf{f}}$.

For example, the fourth-order ($p = 4$) entropy conservative flux is given by

$$\tilde{\mathbf{f}}_{i+\frac{1}{2}}^4 = \frac{4}{3} \tilde{\mathbf{f}}(\mathbf{u}_i, \mathbf{u}_{i+1}) - \frac{1}{6} \left(\tilde{\mathbf{f}}(\mathbf{u}_{i-1}, \mathbf{u}_{i+1}) + \tilde{\mathbf{f}}(\mathbf{u}_i, \mathbf{u}_{i+2}) \right). \quad (3.9)$$

3.3 Entropy stable schemes

Entropy is conserved for smooth solutions and thus using an entropy conservative scheme is meaningful. However, entropy conservative schemes are central schemes and will generate spurious Gibbs oscillations near discontinuities. To control these oscillations, additional dissipation terms must be introduced. Further, entropy is dissipated near discontinuities in accordance to (2.7) while being conserved in smooth regions.

Definition 3.2. *The numerical scheme (3.1) is called entropy stable if it satisfies the discrete entropy inequality*

$$\frac{d\eta(\mathbf{u}_i)}{dt} + \frac{1}{h} \left(q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}} \right) \leq 0, \quad (3.10)$$

where $q_{i+\frac{1}{2}}$ is a consistent numerical entropy flux.

An important class of entropy stable schemes [97, 28] consists of adding an entropy variable-based numerical dissipation term to the entropy conservative numerical flux

$$\mathbf{f}_{i+\frac{1}{2}} = \tilde{\mathbf{f}}_{i+\frac{1}{2}}^{2p} - \frac{1}{2} \mathbf{D}_{i+\frac{1}{2}} \Delta \mathbf{v}_{i+\frac{1}{2}}, \quad (3.11)$$

where $\mathbf{D}_{i+\frac{1}{2}} \succeq 0$ (a positive semi-definite matrix) is evaluated at some suitable averaged states.

Theorem 3.3. [28] *The numerical scheme (3.1) with numerical flux (3.11) is entropy stable. In particular, it satisfies the discrete entropy inequality with numerical entropy flux of the form*

$$q_{i+\frac{1}{2}} = \tilde{q}_{i+\frac{1}{2}}^{2p} - \frac{1}{2} \left(\bar{\mathbf{v}}_{i+\frac{1}{2}} \right)^\top \mathbf{D}_{i+\frac{1}{2}} \Delta \mathbf{v}_{i+\frac{1}{2}}.$$

3.3.1 Dissipation operator

While any positive semi-definite matrix ensures entropy stability in the sense of (3.10), we choose the form $\mathbf{D}_{i+\frac{1}{2}} = \mathbf{R}_{i+\frac{1}{2}} \mathbf{\Lambda}_{i+\frac{1}{2}} \mathbf{R}_{i+\frac{1}{2}}^\top$. Here, $\mathbf{R}$ is a matrix consisting of the right eigenvectors of the flux Jacobian and $\mathbf{\Lambda}$ is a nonnegative diagonal matrix that depends on the eigenvalues of the flux Jacobian. These are evaluated using some suitable averaged values at the cell interface [11]. Specifically, we choose the *Roe-type* diffusion matrix with $\mathbf{\Lambda} = \text{diag}(|\lambda^1|, \dots, |\lambda^d|)$. See [97] for other choices for the diffusion matrix.

3.3.2 High-order accurate diffusion operators

Note that the term $\Delta \mathbf{v}_{i+\frac{1}{2}}$ in (3.11) is $\mathcal{O}(h)$ for smooth solutions. Therefore, the numerical scheme that results from using (3.11) is only first-order accurate regardless of the accuracy of the entropy conservative flux. Since the accuracy of the scheme is limited by the diffusion term, we construct a higher-order diffusion term by suitably reconstructing the jump in entropy variables at the cell interfaces.

We consider the interface at $x_{i+\frac{1}{2}}$ between the cells I_i and I_{i+1} and reconstruct from the left and right of this interface. We define the (locally) *scaled entropy variables* $\mathbf{z} = \mathbf{R}_{i+\frac{1}{2}}^\top \mathbf{v}$

corresponding to this interface. The flux (3.11) can thus be expressed as

$$\mathbf{f}_{i+\frac{1}{2}} = \tilde{\mathbf{f}}_{i+\frac{1}{2}}^p - \frac{1}{2} \mathbf{R}_{i+\frac{1}{2}} \Lambda_{i+\frac{1}{2}} \Delta \mathbf{z}_{i+\frac{1}{2}}. \quad (3.12)$$

Let $\mathbf{z}_i(x)$ and $\mathbf{z}_{i+1}(x)$ be polynomial reconstructions of the scaled entropy variables in I_i and I_{i+1} , respectively. We denote the reconstructed values and jump at the cell interface by

$$\mathbf{z}_{i+\frac{1}{2}}^- = \mathbf{z}_i(x_{i+\frac{1}{2}}), \quad \mathbf{z}_{i+\frac{1}{2}}^+ = \mathbf{z}_{i+1}(x_{i+\frac{1}{2}}), \quad \llbracket \mathbf{z} \rrbracket_{i+\frac{1}{2}} = \mathbf{z}_{i+\frac{1}{2}}^+ - \mathbf{z}_{i+\frac{1}{2}}^-.$$

Replacing the original jump $\Delta \mathbf{z}_{i+\frac{1}{2}}$ in (3.12) by the reconstructed jump $\llbracket \mathbf{z} \rrbracket_{i+\frac{1}{2}}$ will lead to a higher-order accurate scheme. However, this scheme is not guaranteed to be entropy stable. The following lemma provides a sufficient condition on the reconstruction algorithm that ensures entropy stability.

Lemma 3.1. ([28]) *For each interface $x_{i+\frac{1}{2}}$, if the reconstruction satisfies the sign property*

$$\text{sign}(\llbracket \mathbf{z} \rrbracket_{i+\frac{1}{2}}) = \text{sign}(\Delta \mathbf{z}_{i+\frac{1}{2}}), \quad (3.13)$$

then the scheme with the following numerical flux is entropy stable

$$\mathbf{f}_{i+\frac{1}{2}} = \tilde{\mathbf{f}}_{i+\frac{1}{2}}^p - \frac{1}{2} \mathbf{R}_{i+\frac{1}{2}} \Lambda_{i+\frac{1}{2}} \llbracket \mathbf{z} \rrbracket_{i+\frac{1}{2}}. \quad (3.14)$$

High-order entropy stable schemes described in Lemma 3.1 are called *TeCNO* schemes, which rely on reconstructions satisfying the sign property (3.13). Figure 3.1 depicts examples of reconstructions that satisfy and violate the sign property.

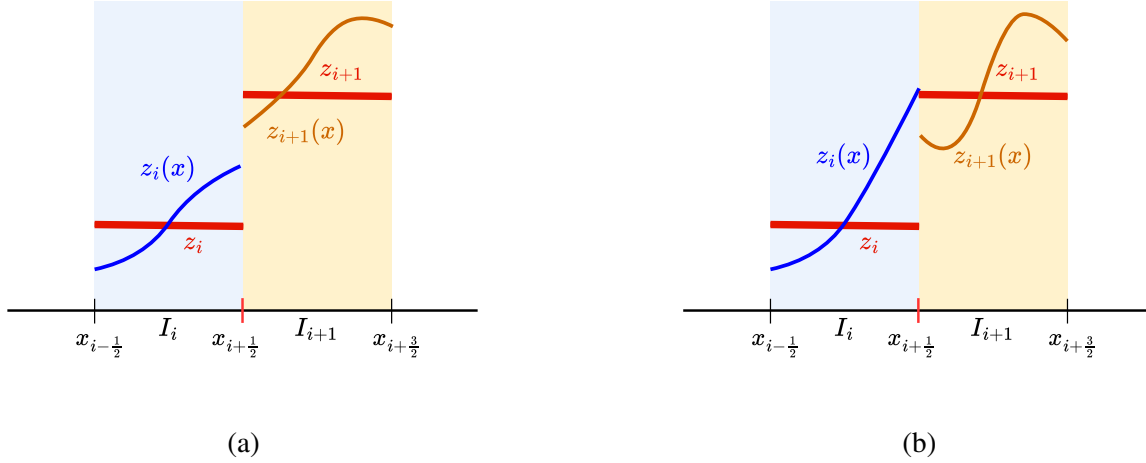


Figure 3.1: Example of a reconstruction that **a)** satisfies the sign property and **b)** violates the sign property. The red values represent the original cell center values.

3.4 Sign-preserving reconstructions

There are only a handful of sign-preserving reconstructions, i.e., reconstructions that satisfy the sign property.

3.4.1 ENO interpolation

ENO interpolation is one such reconstruction algorithm that has been shown to be sign-preserving [30]. The idea of ENO is to select the smoothest stencil from a set of candidates to reconstruct the polynomial. This is especially crucial when interpolating near discontinuities because the inclusion of a cell containing such a feature would lead to a poor reconstruction. Let $z(x)$ be a function whose cell-center values ($z_i = z(x_i)$) we wish to interpolate. Thus, in cell I_i , we seek to construct a polynomial such that $p_i(x_i) = z_i$ and

$$p_i(x_{i-\frac{1}{2}}) = z(x_{i-\frac{1}{2}}) + \mathcal{O}(\Delta x^k), \quad p_i(x_{i+\frac{1}{2}}) = z(x_{i+\frac{1}{2}}) + \mathcal{O}(\Delta x^k).$$

To have a k -th order approximation of $z(x)$, we need a polynomial of at most degree $k - 1$. This requires a stencil S_i consisting of k points. To adaptively select the smoothest stencil of k consecutive cell-centers, we use the Newton divided differences

$$z[x_i] \equiv z_i, \quad z[x_i, \dots, x_{i+j}] \equiv \frac{z[x_{i+1}, \dots, x_{i+j}] - z[x_i, \dots, x_{i+j-1}]}{x_{i+j} - x_i}.$$

The procedure begins with the base stencil $S_i = \{x_i\}$. The stencil expands to the left and becomes $S_i = \{x_{i-1}, x_i\}$ if

$$|z[x_{i-1}, x_i]| < |z[x_i, x_{i+1}]|,$$

and otherwise the stencil expands to the right and becomes $S_i = \{x_i, x_{i+1}\}$. This process repeats until the stencil S_i contains k points, and the interpolating polynomial $p_i(x)$ can be constructed. Note that for cell I_i , there are k possible stencils and thus k possible polynomial approximations.

3.4.2 SP-WENO and its variants

A typical weighted ENO (WENO) reconstruction [55] produces a $(2k - 1)$ th-order accurate reconstruction by taking a convex combination of the $2k - 1$ candidate polynomials considered in the k th-order ENO reconstruction. A third-order WENO has the following form for the left and right reconstructions of the variable z at the interface $x_{i+\frac{1}{2}}$:

$$z_{i+\frac{1}{2}}^- = \frac{w_0(z_i + z_{i+1}) + w_1(3z_i - z_{i-1})}{2}, \quad z_{i+\frac{1}{2}}^+ = \frac{\tilde{w}_0(3z_{i+1} - z_{i+2}) + \tilde{w}_1(z_i + z_{i+1})}{2}, \quad (3.15)$$

where the weights $w_0, w_1, \tilde{w}_0, \tilde{w}_1$ (at each interface) must be chosen to: (i) be convex, (ii) ensure third-order accuracy of the reconstructions $z_{i+\frac{1}{2}}^\pm$ for smooth solutions, and (iii) provide minimal weight to linear polynomials on stencils containing discontinuities. In general, WENO reconstructions do not satisfy the sign property, and their use in the TeCNO framework described by (3.14) does not yield entropy stable schemes.

SP-WENO was the first variant of WENO schemes designed to satisfy the sign property. We define the jump ratios at the interface $x_{i+\frac{1}{2}}$ as $\theta_i^- := \Delta z_{i+\frac{1}{2}} / \Delta z_{i-\frac{1}{2}}$, $\theta_i^+ := 1/\theta_i^-$ and the additional terms

$$\psi_{i+\frac{1}{2}}^+ := \frac{1 - \theta_{i+1}^-}{1 - \theta_i^+}, \quad \psi_{i+\frac{1}{2}}^- = \frac{1}{\psi_{i+\frac{1}{2}}^+}. \quad (3.16)$$

In this framework, the weights in (3.15) are given as

$$w_0 = \frac{3}{4} + 2C_1, \quad \tilde{w}_0 = \frac{1}{4} - 2C_2, \quad w_1 = 1 - w_0, \quad \tilde{w}_1 = 1 - \tilde{w}_0. \quad (3.17)$$

Then the perturbations C_1, C_2 in SP-WENO are determined as

$$C_1(\theta_i^+, \theta_{i+1}^-) = \begin{cases} \frac{1}{8} \left(\frac{\kappa^+}{(\kappa^+)^2 + (\kappa^-)^2} \right) & \text{if } \theta_i^+ \neq 1, \psi^+ < 0, \psi^+ \neq -1 \\ 0 & \text{if } \theta_i^+ \neq 1, \psi^+ = -1 \\ -\frac{3}{8} & \text{if } \theta_i^+ = 1 \text{ or } \psi^+ \geq 0, |\theta_i^+| \leq 1 \\ \frac{1}{8} & \text{if } \psi^+ \geq 0, |\theta_i^+| > 1 \end{cases},$$

and $C_2(\theta_i^+, \theta_{i+1}^-) = C_1(\theta_{i+1}^-, \theta_i^+)$, where

$$\kappa^+(\theta_i^+, \theta_{i+1}^-) := \begin{cases} \frac{1}{1+\psi^+} & \text{if } \theta_i^+ \neq 1, \psi^+ \neq -1 \\ 1 & \text{otherwise} \end{cases}, \quad \kappa^-(\theta_i^+, \theta_{i+1}^-) := \kappa^+(\theta_{i+1}^-, \theta_i^+).$$

We summarize the key properties that SP-WENO satisfies:

1. **Consistency:** The weights obey $0 \leq w_0, w_1, \tilde{w}_0, \tilde{w}_1 \leq 1$, or equivalently $-3/8 \leq C_1, C_2 \leq 1/8$.
2. **Sign Property:** The reconstructed values (3.15) satisfy the condition (3.13). Further, we can show that the reconstructed jump can be re-written as

$$\llbracket z \rrbracket_{i+\frac{1}{2}} = \frac{1}{2}[\tilde{w}_0(1 - \theta_{i+1}^-) + w_1(1 - \theta_i^+)]\Delta z_{i+\frac{1}{2}}, \quad (3.18)$$

which leads to the following constraint (equivalent to the sign property)

$$[\tilde{w}_0(1 - \theta_{i+1}^-) + w_1(1 - \theta_i^+)] \geq 0. \quad (3.19)$$

3. **Negation Symmetry:** The weights remain unchanged under the transformation $z \mapsto -z$.

A sufficient condition to ensure this property is to choose C_1, C_2 to be functions of features invariant to this negation transformation. For example,

$$C_k = C_k(\theta_i^+, \theta_{i+1}^-, |\Delta z_{i+\frac{1}{2}}|, (|z_i| + |z_{i+1}|)) \quad k = 1, 2. \quad (3.20)$$

4. **Mirror Property:** Mirroring the solution about interface $x_{i+\frac{1}{2}}$ should also mirror the weights about the interface. Assuming negation symmetry holds and C_1, C_2 are of the form (3.20), the mirror property is ensured if

$$C_1(a, b, c, d) = C_2(b, a, c, d) \quad \forall a, b, c, d \in \mathbb{R}. \quad (3.21)$$

5. **Inner Jump Condition:** This condition ensures that the reconstruction is locally monotonicity-preserving. For each cell i :

$$\text{sign} \left(z_{i+\frac{1}{2}}^- - z_{i-\frac{1}{2}}^+ \right) = \text{sign} \left(\Delta z_{i+\frac{1}{2}} \right) = \text{sign} \left(\Delta z_{i-\frac{1}{2}} \right), \quad (3.22)$$

whenever the second equality holds. With consistent weights, this property is automatically satisfied.

6. **Bound on jumps:** The reconstructed jump has the bound

$$|\llbracket z \rrbracket_{i+\frac{1}{2}}| \leq 2|\Delta z_{i+\frac{1}{2}}|. \quad (3.23)$$

While SP-WENO leads to an entropy stable scheme, numerical results presented in [89] demonstrate its poor performance near discontinuities in the TeCNO framework. As discussed in [89], this issue can be attributed to the fact that the reconstructed jump $\llbracket z \rrbracket_{i+\frac{1}{2}}$ is zero in most cases, resulting in zero numerical diffusion (see (3.14)) in these regions. While the absence of numerical diffusion may be acceptable in smooth regions, this can cause Gibbs oscillations near discontinuities. The most important problematic cases lie in the so-called *C-region* [89], where

the jump ratios are either $\theta_i^+ < 1$, $\theta_{i+1}^- > 1$ or $\theta_i^+ > 1$, $\theta_{i+1}^- < 1$.

We have observed that local solution profiles near discontinuities often fall in this C-region. Since SP-WENO will always result in a lack of dissipation for such cases, it becomes vital to ensure the reconstructed jump is nonzero. SP-WENOC seeks to remedy this by introducing a small perturbation $\mathcal{G}$ to ensure that the reconstructed jump is nonzero in the C-region. The modified jump is taken to be of the form

$$\llbracket z \rrbracket_{i+\frac{1}{2}} = \frac{1}{2} [\tilde{w}_0(1 - \theta_{i+1}^-) + w_1(1 - \theta_i^+) + \mathcal{G}] \Delta z_{i+\frac{1}{2}}.$$

where $\mathcal{G}$ is chosen as

$$\mathcal{G} = \left(\min \left(\frac{|\Delta z_{i+\frac{1}{2}}|}{0.5(|z_i| + |z_{i+1}|)}, |\Delta z_{i+\frac{1}{2}}| \right) \right)^3. \quad (3.24)$$

The perturbed jump can be realized by using the following modifications to C_1, C_2

$$\bar{C}_1 = C_1 - \frac{1}{4} \frac{\mathcal{G}}{(1 - \theta_i^+)}, \quad \bar{C}_2 = C_2 - \frac{1}{4} \frac{\mathcal{G}}{(1 - \theta_{i+1}^-)}. \quad (3.25)$$

To keep the weights consistent, the following additional modification is introduced

$$C_1^\# = \min \left(\max \left(\bar{C}_1, -\frac{3}{8} \right), \frac{1}{8} \right), \quad C_2^\# = \min \left(\max \left(\bar{C}_2, -\frac{3}{8} \right), \frac{1}{8} \right). \quad (3.26)$$

SP-WENOC retains all of the properties enumerated for SP-WENO with the exception of the bound on the reconstructed jumps. SP-WENOC certainly injects more diffusion near discontinuities and mitigates the oscillatory behavior to some extent. However, the overshoots present

in solutions, while reduced, are still significant in a multitude of test cases when compared to the performance of ENO reconstructions in the TeCNO framework (see Section 4.4).

The SP-WENO and SP-WENOC formulations are just two possible weight selection strategies satisfying the constraints that guarantee consistency, sign-preservation, and third-order accuracy. Note that none of these constraints describe the behavior of solutions near shocks, thus SP-WENO and SP-WENOC are not necessarily constructed with precise shock-capturing in mind. That is to say, there are potentially other *SP-WENO variants* that perform better near discontinuities while satisfying the constraints.

As noted in [31], the reconstruction problem at an interface can be broken down in several cases depending on the jump ratios $\theta_i^+, \theta_{i+1}^-$. Each case has its own constraints on the perturbations (C_1, C_2) such that consistency, sign-preservation, and third-order accuracy are guaranteed. We define the notion of a *feasible region* based on the values of $\theta_i^+, \theta_{i+1}^-$.

Definition 3.3 (Feasible Region). *Let $\Omega_\Theta \subset \mathbb{R}^2$ such that $(\theta_i^+, \theta_{i+1}^-) \in \Omega_\Theta$. Corresponding to Ω_Θ , we define a feasible region $\Omega_C \subset \mathbb{R}^2$ such that any $(C_1, C_2) \in \Omega_C$ satisfies*

1. *Consistency: $-\frac{3}{8} \leq C_1, C_2 \leq \frac{1}{8}$.*
2. *Sign Property: The choice of (C_1, C_2) leads to weights that satisfy the constraint (3.19).*
3. *Accuracy: Additional order constraints (if necessary) on (C_1, C_2) to ensure that third-order accuracy is achieved near smooth regions.*

Thus, a feasible region is defined for any given values of $\theta_i^+, \theta_{i+1}^-$. In Table 3.1, we list the various cases of interest along with the corresponding sign property and accuracy constraints for third-order SP-WENO formulations. Note that the various Ω_Θ listed in the table (column 2)

form a disjoint partition of $\mathbb{R}^2$. Further, $\psi_{i+\frac{1}{2}}^+$ is used to describe Ω_Θ in cases 2 and 3, which is a function of $\theta_i^+, \theta_{i+1}^-$ by virtue of (3.16). We reintroduce the following notation from [31] (with the $i + \frac{1}{2}$ subscript dropped from ψ^+, ψ^- terms)

$$\mathcal{L} := \begin{cases} \frac{C_1}{\frac{1}{8}(1+\psi^+)} + \frac{C_2}{\frac{1}{8}(1+\psi^-)}, & \text{if } \psi^+ \neq -1 \\ C_1 - C_2 + 1, & \text{if } \psi^+ = \psi^- = -1 \end{cases}, \quad (3.27)$$

which is also used to define the sign property constraints in Table 3.1. The expression of $\mathcal{L}$ can be obtained from the constraint (3.19) through simple algebraic manipulations. For full details, we refer interested readers to [31].

Remark 3.2. *The consistency constraint $-\frac{3}{8} \leq C_1, C_2 \leq \frac{1}{8}$ must always be satisfied. Thus, in cases where there are no constraints due to the sign property or third-order accuracy, consistency will be the only constraint governing the feasible region Ω_C .*

Remark 3.3. *The choice of feasible region is not unique. For instance, any subset of a feasible region is itself clearly a feasible region. It is not straight-forward to assign a physical significance to a feasible region, as it is only meant to act as a way to constrain the reconstruction weights to satisfy the conditions outlined in Definition 3.3. However, care must be taken to not select a feasible region that is too small as it may limit the search space for finding weights that provide better performance near discontinuities.*

Case	Ω_{Θ}	Sign Prop. Const.	Acc. Const.	Remarks
1	$\theta_i^+, \theta_{i+1}^- > 1$	$C_1 = C_2 = \frac{1}{8}$	None	$\tilde{w}_0 = w_1 = 0$
2a	$\theta_i^+ < 1, \theta_{i+1}^- > 1$ $-1 \leq \psi_{i+\frac{1}{2}}^+ < 0$	$\mathcal{L} \leq 1$	$C_1, C_2 \sim \mathcal{O}(h)$	C-region
2b	$\theta_i^+ < 1, \theta_{i+1}^- > 1$ $\psi_{i+\frac{1}{2}}^+ < -1$	$\mathcal{L} \geq 1$	$C_1, C_2 \sim \mathcal{O}(h)$	C-region
3a	$\theta_i^+ > 1, \theta_{i+1}^- < 1$ $-1 \leq \psi_{i+\frac{1}{2}}^+ < 0$	$\mathcal{L} \geq 1$	$C_1, C_2 \sim \mathcal{O}(h)$	C-region
3b	$\theta_i^+ > 1, \theta_{i+1}^- < 1$ $\psi_{i+\frac{1}{2}}^+ < -1$	$\mathcal{L} \leq 1$	$C_1, C_2 \sim \mathcal{O}(h)$	C-region
4a	$\theta_{i+1}^- = 1, \theta_i^+ > 1$	$C_1 = \frac{1}{8}$	None	$w_1 = 0$
4b	$\theta_{i+1}^- = 1, \theta_i^+ \leq 1$	None	None	
5a	$\theta_i^+ = 1, \theta_{i+1}^- > 1$	$C_2 = \frac{1}{8}$	None	$\tilde{w}_0 = 0$
5b	$\theta_i^+ = 1, \theta_{i+1}^- \leq 1$	None	None	
6	$\theta_i^+, \theta_{i+1}^- < 1$	None	None	

Table 3.1: Summary of the possible Ω_{Θ} along with the sign property and accuracy constraints defining the associated feasible region Ω_C . Here $\mathcal{L}$ is given by (3.27).

Chapter 4: DSP-WENO

As described in Section 3.4.2, SP-WENO and its variants will satisfy the sign property when performing the reconstruction on a local stencil. Deep Sign-Preserving WENO (DSP-WENO) leverages deep learning to explore the feasible region of WENO weights for any input to improve shock-capturing performance without the need to handcraft the weight selection strategy. The work in this chapter has been published in [12].

4.1 Deep learning

The typical goal of deep learning is to approximate an unknown function

$$\mathcal{N} : \mathbb{R}^{N_I} \mapsto \mathbb{R}^{N_O}, \text{ given the dataset } \mathbb{T} = \{(\mathbf{X}^{(k)}, \mathbf{Y}^{(k)})\}_{k=1}^K. \quad (4.1)$$

To this end, an artificial neural network $\mathcal{N}_{\Phi} : \mathbb{R}^{N_I} \mapsto \mathbb{R}^{N_O}$, consisting of multiple affine transformations (layers) separated by non-linear activation functions, is trained on a suitable dataset to approximate the unknown function [39]. Each layer comprises a set of neurons that are parameterized by tunable weights and biases. Mathematically, the simplest neural network architecture, known as the multilayer perceptron (MLP), is of the form

$$\mathcal{N}_{\Phi} := \sigma^o \circ \mathcal{A}_{\Phi_{L+1}}^{(L+1)} \circ \sigma \circ \mathcal{A}_{\Phi_L}^{(L)} \circ \sigma \circ \mathcal{A}_{\Phi_{L-1}}^{(L-1)} \circ \dots \circ \sigma \circ \mathcal{A}_{\Phi_1}^{(1)}, \quad (4.2)$$

where $\mathcal{A}_{\Phi_k}^{(k)}$ is an affine transformation representing a layer of the MLP parameterized by its weights and biases Φ_k , σ is a non-linear activation function applied component-wise, and σ^o is an output function that transforms the network output into a usable form. The first L parameterized layers are known as the hidden layers, while the final layer is known as the output layer. $\Phi = \{\Phi_k\}_{k=1}^L$ is the collection of all the trainable weights and biases of the network.

$\mathcal{N}_{\Phi}$ is trained such that an objective/loss function $\Pi(\Phi)$ is optimized. This consists of using a gradient-descent based algorithm to iteratively update the network parameters by computing the gradient of the loss with respect to each parameter. Generally, the training is sensitive to hyperparameters such as the number of layers, layer size, and activation function, so it is imperative to carefully tune the network to achieve desirable results. Assuming that a suitable network architecture is chosen (along with any other hyperparameters), and with sufficient representative data in the training set, the trained network is capable of approximating the true function $\mathcal{N}$ [83, 57, 104].

When considering the problem of learning a variant of the SP-WENO reconstruction, the unknown function $\mathcal{N}$ serves to select the perturbations C_1, C_2 based on local features of the solution. As discussed in Section 3.4.2, SP-WENO and SP-WENOC stipulate possible functions that $\mathcal{N}$ can be, though they may not be optimal. The idea of DSP-WENO is to learn a weight selection strategy by approximating $\mathcal{N}$ with $\mathcal{N}_{\Phi}$ so that shock-capturing behavior is improved. Using the training dataset $\mathbb{T}$ consisting of local features and the actual interface values, we hope to train a neural network such that a suitable variant of the SP-WENO reconstruction is learned.

For the k -th sample in $\mathbb{T}$, $\mathbf{X}^{(k)} \in \mathbb{R}^4$ consists of the four cell center values of some function z in the local stencil centered about (some) $x_{i+\frac{1}{2}}: [z_{i-1}, z_i, z_{i+1}, z_{i+2}]$, while $\mathbf{Y}^{(k)} \in \mathbb{R}^2$ is the true cell interface values $z^\pm(x_{i+\frac{1}{2}})$. Note that the interface values from the left (-) and right (+) will be identical for a smooth function, but they will differ if there is a discontinuity at the interface.

To ensure that all the inputs to the neural network are of the same order of magnitude (essential for generalization), we first scale the cell center values as

$$z_{i+j}^* = \frac{z_{i+j}}{\max(1, (\max_{-1 \leq r \leq 2} (|z_{i+r}|))}}, \quad j = -1, 0, 1, 2. \quad (4.3)$$

A similar input scaling was also considered in [90, 91]. Using the scaled cell center values $[z_{i-1}^*, z_i^*, z_{i+1}^*, z_{i+2}^*]$, we compute $\theta_{i+1}^-, \theta_i^+, |\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*|$. We note that the jump ratios $\theta_{i+1}^-, \theta_i^+$ are invariant to the scaling (4.3). Thus we define the map $\mathcal{S} : \mathbb{R}^4 \rightarrow \mathbb{R}^5$ which transforms $\mathbf{X}^{(k)}$ in the training set to the jump ratios and scaled absolute jumps mentioned above, i.e.,

$$\mathcal{S}([z_{i-1}, z_i, z_{i+1}, z_{i+2}]) = [\theta_{i+1}^-, \theta_i^+, |\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*|]. \quad (4.4)$$

Since $\theta_{i+1}^-, \theta_i^+$ can assume values with significantly large magnitudes, we transform $\theta_{i+1}^-, \theta_i^+$ using the hyperbolic tangent in order to bound the input to the network. The input vector to the neural network is $[\tanh(\theta_{i+1}^-), \tanh(\theta_i^+), |\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*|] \in \mathbb{R}^5$. We define the map $\mathcal{F} : \mathbb{R}^4 \rightarrow \mathbb{R}^5$ which transforms $\mathbf{X}^{(k)}$ in the training set to the input of the multi-layer perceptron (MLP), i.e.,

$$\mathcal{F}([z_{i-1}, z_i, z_{i+1}, z_{i+2}]) = [\tanh(\theta_{i+1}^-), \tanh(\theta_i^+), |\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*|]. \quad (4.5)$$

The output of the network $\mathcal{N}_\Phi$ is the vector $\alpha \in \mathbb{R}^5$ whose components satisfy

$$\alpha^{(k)} = \mathcal{N}_\Phi(\mathcal{F}(X^{(k)})) \quad \text{with} \quad \sum_{s=1}^5 \alpha_s^{(k)} = 1, \quad 0 \leq \alpha_s^{(k)} \leq 1. \quad (4.6)$$

Let us define a function $\mathcal{V} : \mathbb{R}^5 \rightarrow \mathbb{R}^{2 \times 5}$ that maps $\mathcal{S}(X^{(k)})$ to 5 vertices in $\mathbb{R}^2$, i.e.,

$$\mathcal{V}(\mathcal{S}(X^{(k)})) = \nu^{(k)} := \left[\nu_1^{(k)}, \nu_2^{(k)}, \nu_3^{(k)}, \nu_4^{(k)}, \nu_5^{(k)} \right] \quad \nu_s^{(k)} \in \mathbb{R}^2 \quad 1 \leq s \leq 5. \quad (4.7)$$

In (4.7), the $\nu_s^{(k)}$ corresponds to the vertices of a convex polygon (defining the feasible region), thus ensuring that any convex combination of these vertices will result in a vector in $\mathbb{R}^2$ that lies in the (closure) of this convex polygon. These convex regions are formed by at most five vertices. In situations with less than five vertices, some of the $\nu_s^{(k)}$ in (4.7) are replaced by a repeated interior node of the convex region, typically the centroid. Further details about $\mathcal{V}$ can be found in Section 4.2. The output of $\mathcal{N}_\Phi$ is combined with these vertices to obtain the DSP-WENO weight perturbations C_1, C_2 , given by

$$\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \mathcal{V}(\mathcal{S}(X^{(k)})) \mathcal{N}_\Phi(\mathcal{F}(X^{(k)})) = \nu^{(k)} \alpha^{(k)}, \quad (4.8)$$

where it is understood that C_1, C_2 will also be indexed by the sample index k .

Using (3.15) and (3.17), we define the reconstruction function $\mathcal{P} : \mathbb{R}^6 \rightarrow \mathbb{R}^2$, that takes as input the cell center values, $[z_{i-1}, z_i, z_{i+1}, z_{i+2}]$ and the WENO weight perturbations C_1, C_2 to

give the reconstructed values $z_{i+\frac{1}{2}}^\pm$ at the interface $x_{i+\frac{1}{2}}$

$$\hat{\mathbf{Y}}^{(k)} := [z_{i+\frac{1}{2}}^-, z_{i+\frac{1}{2}}^+] = \mathcal{P}(\mathbf{X}^{(k)}, [C_1, C_2]). \quad (4.9)$$

We finally define the objective function in terms of the trainable parameters Φ of the network using a mean squared error (MSE) loss

$$\Pi(\Phi) = \frac{1}{K} \sum_{k=1}^K \Pi_k(\Phi), \quad \Pi_k(\Phi) = \left\| \hat{\mathbf{Y}}^{(k)} - \mathbf{Y}^{(k)} \right\|_2 \quad (4.10)$$

where the $\hat{\mathbf{Y}}^{(k)}$ encapsulate the dependence on Φ and $\|\cdot\|_2$ denotes the Euclidean 2-norm. Thus, the network is trained by solving the minimization problem

$$\Phi^\dagger = \arg \min_{\Phi} \Pi(\Phi).$$

Figure 4.1 summarizes how DSP-WENO computes the reconstructed interface values. Note that the only learnable component of the algorithm is $\mathcal{N}_\Phi$.

Remark 4.1. *The α are hidden variables within the DSP-WENO reconstruction designed to enforce the constraints. There is no notion of what their true values should be nor guarantees of uniqueness, and moreover it is not clear what the mapping that (4.6) approximates would be, as we do not know anything about the properties of this function a priori.*

Remark 4.2. *We have tried the training process with other loss functions, including the mean absolute error (MAE) loss, but ultimately using the loss function (4.10) provides the best performance in our experiments. The choice of the loss function is a hyperparameter, so we do not*

claim that one loss function is superior to another. Moreover, we do not claim that we have recovered the best DSP-WENO network as it might be possible to recover an even better network by further tweaking the hyperparameters.

Remark 4.3. If $\Delta z_{i+\frac{1}{2}} = 0$, θ_{i+1}^- and θ_i^+ are undefined. In this case, we simply set $z_{i+\frac{1}{2}}^- = z_i$, $z_{i+\frac{1}{2}}^+ = z_{i+1}$ leading to the reconstruction jump $\llbracket z \rrbracket_{i+\frac{1}{2}} = 0$. Otherwise if $\Delta z_{i+\frac{1}{2}} \neq 0$, we evaluate the reconstructions using DSP-WENO as described above. This is also the strategy followed for SP-WENO and SP-WENOC.

Remark 4.4. While we have used a neural network to learn the mapping from the local data to the corresponding α , it is possible in principle to use a different parameterized model to learn this mapping. We have decided to use a neural network because it is straight-forward to train for the present problem and has shown great promise in designing shock-capturing algorithms [90, 91]. We note that using a deep learning approach has allowed shock-capturing to be easily incorporated into the weight estimation process of WENO, unlike hand-crafted SP-WENO schemes where shock-capturing is entirely absent.

4.2 Vertex Selection Algorithm

Now, we describe the vertex selection algorithm that defines the map $\mathcal{V}$ in (4.7). Based on the values of θ_i^+ , θ_{i+1}^- , we consider 6 disjoint subdomains, each denoted as Ω_Θ , which partition $\mathbb{R}^2$. For each Ω_Θ , we construct a feasible region Ω_C (see Definition 3.3). Recall that any choice of (C_1, C_2) in Ω_C leads to consistent WENO weights, a guarantee of the sign property, and third-order accuracy in smooth regions.

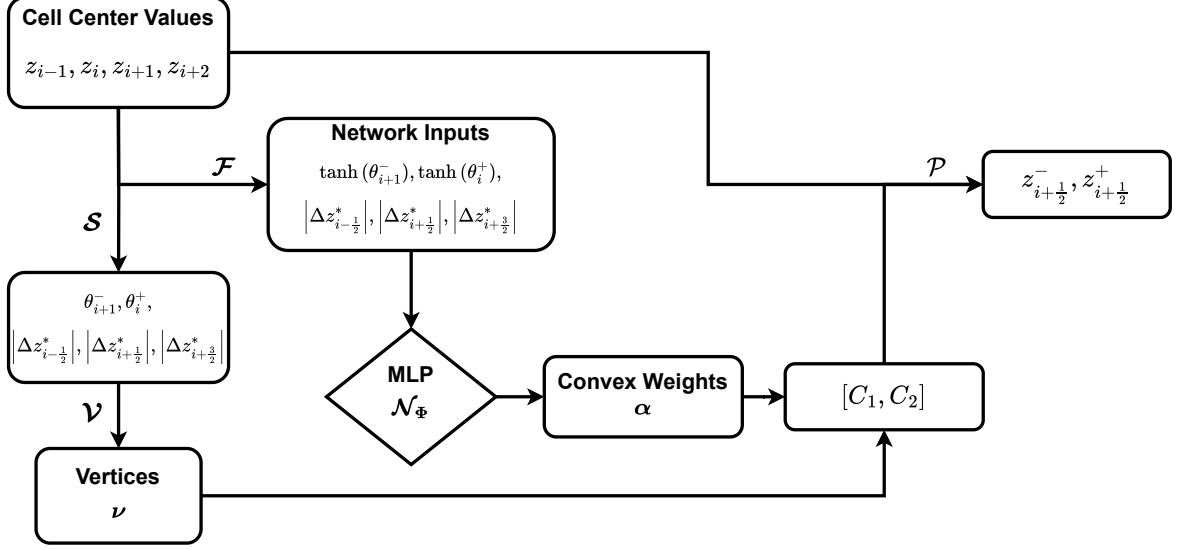


Figure 4.1: Schematic for the DSP-WENO reconstruction algorithm.

Further, it is possible to choose Ω_C (for each case) to form a convex polygon in $\mathbb{R}^2$ with at most 5 vertices. In fact, given any set of cell center values $[z_{i-1}, z_i, z_{i+1}, z_{i+2}]$, we provide an explicit algorithm to obtain the 5 vertices (taking into account repeated interior nodes when needed) of the convex Ω_C . Thus, to ensure that the neural network always outputs the same number of convex weights forming the vector α corresponding to the vertices, we constrain the output dimension of the network to be five and develop the vertex selection algorithm to always provide five vertices of a feasible region. In the various scenarios where the number of vertices of Ω_C is less than five, we augment the set of vertices to a set of five vertices by including either redundant vertices or some averaged state of the figure's vertices. The pseudocode for the algorithm can be found in Algorithm 4.1.

To reiterate, the objective with the neural network approach is to learn how to explore these Ω_C to better select (C_1, C_2) . In comparison, SP-WENO and SP-WENOC have handcrafted weight selection strategies that suffer from significant overshoots near discontinuities in the TeCNO framework. DSP-WENO aspires to mitigate this behavior through a data-driven search in the

feasible regions, which we construct with the vertex selection algorithm.

With the exception of cases 2 and 3 (see Table 3.1), the vertex selection for most scenarios is quite straight-forward. This is due to the fact that all cases besides 2 and 3 do not have an additional accuracy constraint. In particular, for

- **Case 1:** The feasible region Ω_C reduces to a single node $(\frac{1}{8}, \frac{1}{8})$. Thus, all five vertices are taken as this node.
- **Case 4a:** Ω_C reduces the line segment $C_1 = \frac{1}{8}, -\frac{3}{8} \leq C_2 \leq \frac{1}{8}$. Thus we take two vertices to be the right end $(\frac{1}{8}, \frac{1}{8})$, two vertices to be the left end $(-\frac{3}{8}, \frac{1}{8})$, and the final vertex to be the line segment's mid-point $(-\frac{1}{8}, \frac{1}{8})$.
- **Case 5a:** Is identical to Case 4a, but with the role of C_1 and C_2 interchanged.
- **Case 4b, 5b and 6:** There are no constraints (except for consistency) and $\Omega_C = [-\frac{3}{8}, \frac{1}{8}] \times [-\frac{3}{8}, \frac{1}{8}]$. Thus four vertices are chosen as the four corners of this square region, while the fifth vertex is taken to be the centroid of this square.

Cases 2 and 3 are more involved since we require that $C_1, C_2 = \mathcal{O}(h)$ in smooth regions. To address this, we construct an $\mathcal{O}(h)$ region in $\mathbb{R}^2$ as depicted in Figure 4.2. The black dashed lines enclose a box representing the consistency requirements for the perturbations, i.e., $-3/8 \leq C_1, C_2 \leq 1/8$. The $\mathcal{L} = 1$ line shown in orange separates the regions satisfying the sign property for Cases 2 and 3, represented as solid dark and light grey regions respectively. Within these grey regions, we construct smaller sub-regions that also satisfy the accuracy constraints. These sub-regions are shown in blue and green, and form the required feasible regions Ω_C for cases 2 and 3. To construct these sub-regions, we first define the box $\mathcal{H} = [\gamma_2, \gamma_1] \times [\gamma_2, \gamma_1] \subset [-3/8, 1/8] \times$

$[-3/8, 1/8]$ where

$$\begin{aligned}\gamma_1 &= \min \left(\max \left(|\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*| \right), \frac{1}{8} \right), \\ \gamma_2 &= -\min \left(\max \left(|\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*| \right), \frac{3}{8} \right).\end{aligned}\tag{4.11}$$

The region $\mathcal{H}$ is shown as a red dotted box in Figure 4.2. Note that when the z on the local stencil is smooth, $|\Delta z_{i-\frac{1}{2}}^*|, |\Delta z_{i+\frac{1}{2}}^*|, |\Delta z_{i+\frac{3}{2}}^*| = \mathcal{O}(h)$. Thus, we ensure that $\gamma_1, \gamma_2 = \mathcal{O}(h)$, and searching for (C_1, C_2) inside $\mathcal{H}$ would constrain $C_1, C_2 = \mathcal{O}(h)$, which is the order constraint for cases 2 and 3 assuming smoothness for z . When a discontinuity is present, $\mathcal{H}$ can potentially expand to the whole consistency region for C_1, C_2 . With the line $\mathcal{L} = 1$ splitting $\mathcal{H}$, we have that the feasible region for Case 2 consists of the region above $\mathcal{L} = 1$ and that of Case 3 consists of the region below $\mathcal{L} = 1$. We further note that the closer the perturbations are to zero, the more accurate the reconstruction. In smooth regions, this is desirable, but near discontinuities, capturing shocks becomes the priority, so searching a wider search space becomes important.

As shown in Figure 4.2, the line $\mathcal{L} = 1$ splits $\mathcal{H}$ into a triangular and pentagonal Ω_C region, with two out of three and two out of five vertices lying on the line $\mathcal{L} = 1$, respectively. A peculiarity of choosing (C_1, C_2) on this line is that it ensures the left and right reconstructed states at the interface are equal, i.e., $\llbracket z \rrbracket_{i+\frac{1}{2}} = 0$. As discussed in Section 3.4.2, a zero reconstruction jump is undesirable in the vicinity of a discontinuity in the TeCNO framework. When choosing (C_1, C_2) from a triangular region, two of whose vertices are on this line, which biases the selection to be on this line (this is what we also observed in practice). Thus, when constructing a triangular Ω_C above or below $\mathcal{L} = 1$, we do not use the triangle that is a part of $\mathcal{H}$. Instead, we replace one of the vertices lying on $\mathcal{L} = 1$ with another vertex pulled away from this line (without violating

the consistency and sign property constraints) to form a flipped triangular Ω_C that is outside of $\mathcal{H}$. This flipped Ω_C still satisfies $C_1, C_2 = \mathcal{O}(h)$, and thus any point selected in this triangle will also satisfy this constraint. For instance, let us consider Case 2 when $\psi^+ < -1$, where Ω_C is triangular (see Figure 4.2(c),(d)). In this scenario, according to Algorithm 4.1, the vertex outside $\mathcal{H}$ is $(\gamma_2, \frac{1}{8}(1 + \psi^-) - \gamma_1\psi^-)$. For smooth regions, $C_1 = \gamma_2 = \mathcal{O}(h)$. Further, we can deduce that $(1 + \psi^-) = \mathcal{O}(h)$. Since $\gamma_1 = \mathcal{O}(h)$, it is clear that $C_2 = \frac{1}{8}(1 + \psi^-) - \gamma_1\psi^- = \mathcal{O}(h)$. Similar arguments can be made for Case 3 when $-1 < \psi^+ < 0$ where Ω_C is once again triangular (see Figure 4.2(a),(b)). To ensure that we always have five vertices, when the feasible region is a triangle, we augment the set of three vertices with the centroid of the triangle twice.

Remark 4.5. *When constructing the triangular Ω_C for Cases 2 and 3, we do not discard all vertices lying on $\mathcal{L} = 1$. This is because while $\llbracket z \rrbracket_{i+\frac{1}{2}} = 0$ is not desirable near discontinuities, it does lead to better accuracy near smooth regions. Thus, we retain at least one of these vertices to maintain a balance in performance between smooth and discontinuous reconstructions.*

Remark 4.6. *We do not modify the vertices of the pentagonal Ω_C obtained in Cases 2 and 3, as only two of the five vertices are on $\mathcal{L} = 1$. Thus, the pentagonal regions do not suffer from the biasing issue faced by triangular Ω_C .*

Remark 4.7. *The vertex modification algorithm for the triangular Ω_C is not unique. After experimenting with a few configurations of choosing the shifted vertex, we found the one used in Algorithm 4.1 yields the best performance.*

Algorithm 4.1 Vertex Selection Algorithm

Input: Jump ratios θ_{i+1}^- , θ_i^+ and scaled absolute jumps $|\Delta z_{i-\frac{1}{2}}^*|$, $|\Delta z_{i+\frac{1}{2}}^*|$, $|\Delta z_{i+\frac{3}{2}}^*|$

Output: Set of vertices $\nu = [\nu_1, \nu_2, \nu_3, \nu_4, \nu_5]$

```
1: Evaluate  $\psi^\pm$  according to (3.16), and  $\gamma_1, \gamma_2$  according to (4.11)
2: if  $\theta_{i+1}^-, \theta_i^+ > 1$  then
3:    $\nu = [(\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8})]$ 
4: else if  $\theta_{i+1}^- > 1, \theta_i^+ < 1$  then
5:   if  $\psi^+ < -1$  then
6:      $x_* = \frac{1}{8}(1 + \psi^+) - \gamma_1 \psi^+$ 
7:     if  $x_* < \gamma_2$  then
8:        $\hat{x} = \frac{(\psi^-)^2 + \psi^-}{8((\psi^-)^2 + 1)}, \hat{y} = \frac{1}{8}(1 + \psi^-) - \hat{x} \psi^-$ 
9:        $\nu = [(\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y})]$ 
10:    else
11:       $y_*^{(2)} = \frac{1}{8}(1 + \psi^-) - \gamma_1 \psi^-, x_c = \frac{1}{3}(2\gamma_2 + x_*), y_c = \frac{1}{3}(2\gamma_1 + y_*^{(2)})$ 
12:       $\nu = [(\gamma_2, \gamma_1), (x_*, \gamma_1), (\gamma_2, y_*^{(2)}), (x_c, y_c), (x_c, y_c)]$ 
13:    end if
14:  else
15:     $y_* = \frac{1}{8}(1 + \psi^-) - \gamma_1 \psi^-, x_*^{(2)} = \frac{1}{8}(1 + \psi^+) - \gamma_2 \psi^+$ 
16:    if  $y_* < \gamma_2$  then
17:       $\nu = [(\gamma_2, \gamma_1), (\gamma_1, \gamma_1), (\gamma_2, \gamma_2), (\gamma_1, \gamma_2), (0, 0)]$ 
18:    else
19:       $\nu = [(\gamma_2, \gamma_1), (\gamma_1, \gamma_1), (\gamma_2, \gamma_2), (\gamma_1, y_*), (x_*^{(2)}, \gamma_2)]$ 
20:    end if
21:  end if
22: else if  $\theta_{i+1}^- < 1, \theta_i^+ > 1$  then
23:   if  $\psi^+ < -1$  then
24:      $x_* = \frac{1}{8}(1 + \psi^+) - \gamma_1 \psi^+, y_*^{(3)} = \frac{1}{8}(1 + \psi^-) - \gamma_2 \psi^-$ 
25:     if  $x_* < \gamma_2$  then
26:        $\nu = [(\gamma_2, \gamma_1), (\gamma_1, \gamma_1), (\gamma_2, \gamma_2), (\gamma_1, \gamma_2), (0, 0)]$ 
27:     else
28:        $\nu = [(\gamma_1, \gamma_2), (\gamma_1, \gamma_1), (\gamma_2, \gamma_2), (x_*, \gamma_1), (\gamma_2, y_*^{(3)})]$ 
29:     end if
30:   else
31:      $y_* = \frac{1}{8}(1 + \psi^-) - \gamma_1 \psi^-$ 
32:     if  $y_* < \gamma_2$  then
33:        $\hat{x} = \frac{(\psi^-)^2 + \psi^-}{8((\psi^-)^2 + 1)}, \hat{y} = \frac{1}{8}(1 + \psi^-) - \hat{x} \psi^-$ 
34:        $\nu = [(\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y}), (\hat{x}, \hat{y})]$ 
35:     else
36:        $x_*^{(3)} = \frac{1}{8}(1 + \psi^+) - \gamma_1 \psi^+, x_c = \frac{1}{3}(2\gamma_1 + x_*^{(3)}), y_c = \frac{1}{3}(2\gamma_2 + y_*)$ 
37:        $\nu = [(\gamma_1, \gamma_2), (x_*^{(3)}, \gamma_2), (\gamma_1, y_*), (x_c, y_c), (x_c, y_c)]$ 
38:     end if
39:   end if
40: else if  $\theta_{i+1}^- == 1, \theta_i^+ > 1$  then
41:    $\nu = [(\frac{1}{8}, -\frac{3}{8}), (\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, -\frac{3}{8}), (\frac{1}{8}, -\frac{1}{8})]$ 
42: else if  $\theta_i^+ == 1, \theta_{i+1}^- > 1$  then
43:    $\nu = [(\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, \frac{1}{8}), (-\frac{3}{8}, \frac{1}{8}), (-\frac{3}{8}, \frac{1}{8}), (-\frac{1}{8}, \frac{1}{8})]$ 
44: else
45:    $\nu = [(\frac{1}{8}, \frac{1}{8}), (\frac{1}{8}, -\frac{3}{8}), (-\frac{3}{8}, -\frac{3}{8}), (-\frac{3}{8}, \frac{1}{8}), (-\frac{1}{8}, -\frac{1}{8})]$ 
46: end if
```

} Case 1
} Case 2
} Case 3
} Case 4a
} Case 5a
} Cases 4b, 5b, 6

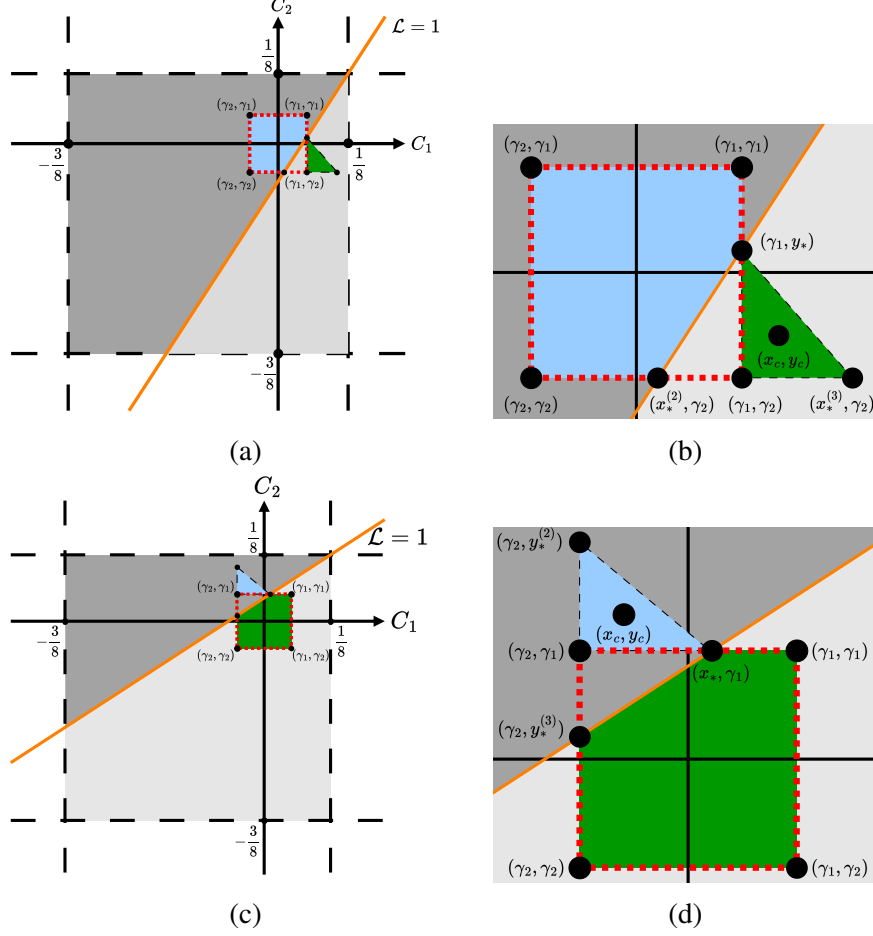


Figure 4.2: Feasible region for Case 2 (*dark grey*) and Case 3 (*light grey*). The $\mathcal{O}(h)$ box $\mathcal{H}$ (outlined by red dots) is centered about the origin. The vertex selection algorithm further restricts the search space to the blue and green regions for Case 2 and Case 3, respectively. **(a)** $\psi^+ \in (-1, 0)$ with a zoomed in view in **(b)**, **(c)** $\psi^+ \in (-\infty, -1)$ with a zoomed in view in **(d)**.

4.3 Network Details

The final neural network used in DSP-WENO is the result of careful experimentation and fine-tuning. In this section, the details of the final configuration are discussed, including the composition of the selected data and the network architecture.

No.	Type	$u(x)$	Parameters
1	Smooth	$ax^3 + bx^2 + cx + d$	$a, b, c, d \in \mathbb{U}[-10, 10]$
2	Smooth	$(x - a)(x - b)(x - c) + d$	$a, b, c, d \in \mathbb{U}[-2, 2]$
3	Smooth	$\sin(a\pi x + b)$	$a, b \in \mathbb{U}[-2, 2]$
4	Discontinuous	$\begin{cases} ax + b & \text{if } x \leq 0.5 \\ cx + d & \text{if } x > 0.5 \end{cases}$	$a, b, c, d \in \mathbb{U}[-5, 5]$

Table 4.1: Functions used in constructing the training dataset where $\mathbb{U}[p, q]$ denotes the uniform distribution on the interval $[p, q]$.

4.3.1 Data Selection

To generate training data for the neural network, we sample stencils consisting of four cells from parameterized smooth and discontinuous functions of the form listed in Table 4.1 sampled on meshes with mesh size $h = 1/40, 1/100$, or $1/200$. The dataset has a 50/50 balanced split of smooth and discontinuous data. The section of the dataset consisting of smooth data consists of an equal representation of the three smooth types listed in Table 4.1. For discontinuous data, the jump will either be between the first and second cells, the second and third cells, or the third and fourth cells. In any case, using the four cell center values, we compute the jump ratios θ_{i+1}^- and θ_i^+ and the absolute jumps across the three cell interfaces in the stencil, via $\mathcal{S}$ defined in (4.4). Further, we compute a set of vertices using $\mathcal{V}$ in Section 4.2. Finally, the true values at the cell interface $x_{i+\frac{1}{2}}$ are recorded to serve as target values that DSP-WENO will aim to reconstruct. Overall, the dataset consists of 100,000 samples, roughly corresponding to 50,000 discontinuous samples and 50,000 smooth samples.

Remark 4.8. *We do not use solutions to conservation laws to train the network. We instead generate functions of varying regularity that canonically represent the local solution features we can expect to observe in solutions to conservation laws. Thus, there is an insignificant cost in*

generating the training data. A similar strategy was also considered in [90, 91].

4.3.2 Network Architecture and Training

Given the small input and output dimensions ($N_I = N_O = 5$) for the network, we use a fairly simple network architecture. In particular, we employ an MLP that consists of three hidden layers with five neurons each, leading to an MLP with $\#\Phi = 120$ trainable parameters. The ReLU activation function is used in the hidden layers. Following the output layer, we use the softmax output function to ensure that the network predicts a vector α of convex vertex weights. The predicted α are used to obtain the perturbation according to (4.8), which are then used to reconstruct the (left and right) cell interface values according to (4.9).

To train the neural network, we use the Adam optimizer [58] with optimizer parameters $\beta_1 = 0.5$, $\beta_2 = 0.9$, a learning rate of 10^{-3} , and a weight decay regularization of 10^{-5} . We perform a training/validation/test split in a 0.6/0.2/0.2 proportion where we seek to minimize the reconstruction error (4.10). We use mini-batches of size 500 and reshuffle the training set every epoch. The network is trained using PyTorch for 50 epochs on a 2.60 GHz Intel Core i7-10750H CPU. The training time for training one instance of this MLP is less than 30 seconds. We perform five training runs with different random initializations of the network weight and biases and select the network that performs the best on the test set. We remark that the true test of the network is based on its performance within the TeCNO schemes to solve conservation laws (see Section 4.4).

Remark 4.9. *Other variations of the MLP's architecture (including activation functions) were considered, but ultimately this architecture yielded the best results.*

4.3.3 Properties of DSP-WENO

DSP-WENO by construction satisfies all properties of SP-WENO with the exception of the mirror property. This property is lost due to the nature of the vertex selection algorithm and computation of the convex weights by the neural network. While it is possible to have pursued a construction that preserves the mirror property, we choose to omit it so as not to overly constrain the network. Additionally, DSP-WENO satisfies the following stability estimate for the reconstructed jump:

Lemma 4.1. (Bounds on jumps) *The DSP-WENO reconstructed jump satisfies the following estimate:*

$$\left| \llbracket z \rrbracket_{i+\frac{1}{2}} \right| \leq \frac{1}{2} \left| \Delta z_{i-\frac{1}{2}} \right| + \left| \Delta z_{i+\frac{1}{2}} \right| + \frac{1}{2} \left| \Delta z_{i+\frac{3}{2}} \right| \quad \forall i \in \mathbb{Z}. \quad (4.12)$$

Proof. If $\Delta z_{i+\frac{1}{2}} = 0$, the bound will clearly be satisfied since $|\llbracket z \rrbracket_{i+\frac{1}{2}}| = 0$ in this case by construction of DSP-WENO. Hence, we assume $\Delta z_{i+\frac{1}{2}} \neq 0$. We start by using (3.17) to rewrite the expression of $\llbracket z \rrbracket_{i+\frac{1}{2}}$ in (3.18) as

$$\llbracket z \rrbracket_{i+\frac{1}{2}} = \left(C_1 - \frac{1}{8} \right) \Delta z_{i-\frac{1}{2}} + \left(\frac{1}{4} - C_1 - C_2 \right) \Delta z_{i+\frac{1}{2}} + \left(C_2 - \frac{1}{8} \right) \Delta z_{i+\frac{3}{2}}.$$

By consistency, we have $-3/8 \leq C_1, C_2 \leq 1/8$. Thus, the absolute value of the jump can be bounded as

$$\begin{aligned} \left| \llbracket z \rrbracket_{i+\frac{1}{2}} \right| &\leq \left| C_1 - \frac{1}{8} \right| \left| \Delta z_{i-\frac{1}{2}} \right| + \left| \frac{1}{4} - C_1 - C_2 \right| \left| \Delta z_{i+\frac{1}{2}} \right| + \left| C_2 - \frac{1}{8} \right| \left| \Delta z_{i+\frac{3}{2}} \right| \\ &\leq \frac{1}{2} \left| \Delta z_{i-\frac{1}{2}} \right| + \left| \Delta z_{i+\frac{1}{2}} \right| + \frac{1}{2} \left| \Delta z_{i+\frac{3}{2}} \right|. \end{aligned}$$

□

Remark 4.10. *The estimate (4.12) is not unique to DSP-WENO and is a consequence of the sign property constraint (3.19). Thus, it is also satisfied by every variant of SP-WENO. However the original SP-WENO satisfies a sharper bound which is attributed to the fact the reconstructed jump is zero in most cases [31].*

The estimate (4.12) provides an explicit bound on the size of the reconstructed jump based on the size of the original jumps of the point values in the local stencil. Similar upper bounds have been obtained for ENO reconstruction [30], SP-WENO [31], and SP-WENOc [89].

Remark 4.11. *We note that the DSP-WENO reconstruction (as with any SP-WENO variation) is not positivity-preserving for the system of Euler equations, as there is no constraint present that enforces this. The DSP-WENO method demonstrates that it is possible to learn reconstruction methods by building in constraints of interest, such as the sign property. Further, the same ideas can be used to also build in other desirable properties, e.g. positivity preservation for the Euler system.*

Remark 4.12. *The DSP-WENO designed and trained in one dimension can be deployed in a dimension-by-dimension manner to handle problems in multiple dimensions. In other words, no additional re-training is required.*

4.4 Numerical Results

In this section, several numerical results demonstrate the efficacy of the proposed DSP-WENO approach. We consider both 1D and 2D test cases for the evolution of scalar and systems

of conservation laws. In 2D, we discretize the domain $[a, b] \times [c, d]$ using a uniform mesh (in each dimension) with $h_x = (b - a)/N_x$ and $h_y = (d - c)/N_y$. Ghost cells are introduced to extend the mesh as needed while imposing either periodic or Neumann boundary conditions. In all evolution cases, we use the TeCNO4 numerical flux, consisting of the fourth-order accurate entropy conservative flux (3.9) and various sign-preserving reconstruction methods for the diffusion operator. SSP-RK3 (see [42]) is used for time-marching with the CFL varying across problems. The code used to train DSP-WENO and the C++ code to run the TeCNO schemes can be found at the repository: <https://github.com/pmcharles/DSP-WENO>

4.4.1 Reconstruction Accuracy

Before proceeding to solve conservation laws, we demonstrate the reconstruction accuracy of DSP-WENO. After all, this is the essence of the designed algorithm, and it is vital that it is third-order accurate. We consider the reconstruction of a smooth inclined sine wave $u(x) = \sin(10\pi x) + x$ on $[0, 1]$ as the mesh size h is varied. We reconstruct the values at the cell interfaces, and evaluate the (averaged) error at the interfaces (from the left and right) as

$$\mathcal{E}_h = \frac{1}{N} \sum_{i=1}^N |u_{i+\frac{1}{2}}^- - u(x_{i+\frac{1}{2}})| + \frac{1}{N} \sum_{i=1}^N |u_{i-\frac{1}{2}}^+ - u(x_{i-\frac{1}{2}})|.$$

Table 4.2 shows the errors and corresponding convergence rates for ENO3, SP-WENO, SP-WENOC, and DSP-WENO. All reconstruction methods achieve the expected third-order convergence with the SP-WENO variants exhibiting super-convergence. Note that the errors of DSP-WENO are about an order of magnitude larger than those of SP-WENO and SP-WENOC while being similar to ENO3 on the finer meshes. This leads to a larger reconstruction jump at interfaces

N	ENO3		SP-WENO		SP-WENOC		DSP-WENO	
	$\mathcal{E}_h$	Rate	$\mathcal{E}_h$	Rate	$\mathcal{E}_h$	Rate	$\mathcal{E}_h$	Rate
40	3.47e-2	-	7.27e-2	-	7.41e-2	-	1.65e-1	-
80	4.54e-3	2.93	5.85e-3	3.64	6.37e-3	3.54	3.01e-2	2.45
160	5.84e-4	2.96	4.45e-4	3.72	4.71e-4	3.76	2.83e-3	3.41
320	7.42e-5	2.98	3.29e-5	3.76	3.43e-5	3.78	2.14e-4	3.73
640	9.38e-6	2.98	2.37e-6	3.79	2.46e-6	3.80	1.55e-5	3.78
1280	1.17e-6	3.00	1.68e-7	3.82	1.74e-7	3.82	1.22e-6	3.67

Table 4.2: Reconstruction accuracy for inclined sine wave.

where a discontinuity is present, which plays a critical role in ensuring that the dissipation term in the TeCNO scheme injects sufficient viscosity near shocks to suppress spurious oscillations.

To elucidate this point, we consider a continuous function with a sharp connection between two linear components to mimic a discontinuity:

$$u(x) = \begin{cases} \frac{1}{2}x & \text{if } -0.5 \leq x < 0 \\ \frac{x}{\epsilon} & \text{if } 0 \leq x \leq \epsilon \\ x - \epsilon + 1 & \text{if } \epsilon < x \leq 0.5 \end{cases},$$

where ϵ is chosen as 0.001. We reconstruct on a mesh consisting of 1000 cells, ensuring that a single cell is sitting inside the jump. In other words, we are simulating a scenario where the discontinuity is not perfectly resolved. Figure 4.3 shows the left and right reconstructed interface values for ENO3, SP-WENO, SP-WENOC, and DSP-WENO, zoomed into the region close to the simulated discontinuity. Figure 4.3 also features bar plots that show the actual jump in cell center values and the reconstructed jump at the interface for each method. The left and right reconstructed values are almost identical outside of the $0 \leq x \leq \epsilon$ region for all the methods,

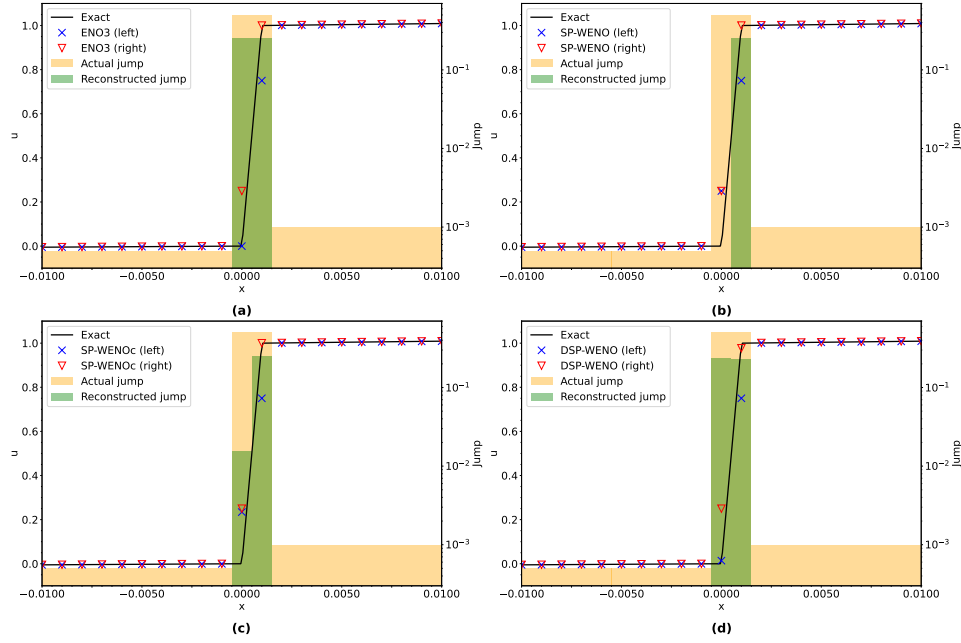


Figure 4.3: Reconstruction of a simulated discontinuity, which is not well-resolved, using: (a) ENO3, (b) SP-WENO, (c) SP-WENOc, (d) DSP-WENO.

and thus not visible in the figure given the scale of the jump. However, inside the pseudo-shock where a cell (let us call it I_0) lies, SP-WENO and SP-WENOc do not fully capture the non-zero jumps. At the left interface of I_0 , SP-WENO results in a very small reconstructed jump, while SP-WENOc leads to a larger jump but which is still an order of magnitude smaller as compared to the jump at the right interface of I_0 . On the other hand, ENO3 and DSP-WENO give rise to larger reconstructed jumps at both interfaces of I_0 , which is the desirable behavior in the vicinity of discontinuities. The impact of the reconstructed jump magnitude can be observed from the numerical results for conservation laws shown in sections 4.4.2 and 4.4.3.

4.4.2 Scalar conservation laws

For scalar conservation laws, we choose the squared entropy $\eta(u) = u^2/2$ with the corresponding entropy variable $v = \eta'(u) = u$. Also, since the scaled entropy variable are the same as

the entropy variable, the finite difference TeCNO4 numerical flux (3.14) is given by

$$f_{i+\frac{1}{2}} = \tilde{f}_{i+\frac{1}{2}}^4 - a_{i+\frac{1}{2}} \llbracket v \rrbracket_{i+\frac{1}{2}}, \quad a_{i+\frac{1}{2}} = \frac{|f'(u_i)| + |f'(u_{i+1})|}{2}. \quad (4.13)$$

4.4.2.1 Linear Advection

We take (3.4) to be the second-order entropy conservative flux used to construct $\tilde{f}^4$ in (3.9).

For all experiments, we set the convective velocity $c = 1$.

Smooth Initial Data: We consider two test cases with smooth initial conditions on $[-\pi, \pi]$ and periodic boundary conditions

- **Test 1:** $u_0(x) = \sin(x)$, simulated until final time $T = 0.5$ with CFL = 0.4.
- **Test 2:** $u_0(x) = \sin^4(x)$, simulated until final time $T = 0.5$ with CFL = 0.5.

Table 4.3 shows the errors (measured in the discrete L^1 norm) with various reconstructions for both of these test cases (see Appendix B for errors measured in the discrete L^2 and L^∞ norms). ENO3 works well for Test 1 and achieves third-order convergence with marginally smaller absolute errors when compared to the results of SP-WENO, SP-WENOc, and DSP-WENO. However, the convergence rate with ENO3 deteriorates in Test 2. It has been observed in [93] that ENO3 performs poorly for this test case with the MUSCL scheme due to linear instabilities, and we observe the same issues with ENO3 in the TeCNO framework (also seen in [31]). The other reconstruction methods are stable, as SP-WENO, SP-WENOc, and DSP-WENO do not suffer from this issue. All three reconstruction methods achieve third-order convergence, though the errors of DSP-WENO are about an order of magnitude larger than those of SP-WENO and SP-WENOc. Moreover, SP-WENO and SP-WENOc exhibit super-convergence. This can be attributed to the

	N	ENO3		SP-WENO		SP-WENOC		DSP-WENO	
		Error	Rate	Error	Rate	Error	Rate	Error	Rate
Test 1	100	3.23e-5	-	6.88e-5	-	6.76e-5	-	9.09e-5	-
	200	4.04e-6	3.00	7.60e-6	3.18	7.46e-6	3.18	8.70e-6	3.39
	400	5.05e-7	3.00	8.27e-7	3.20	8.17e-7	3.19	9.11e-7	3.25
	600	1.50e-7	3.00	2.26e-7	3.20	2.27e-7	3.16	2.52e-7	3.17
	800	6.31e-8	3.00	8.73e-8	3.30	8.71e-8	3.32	1.01e-7	3.17
	1000	3.23e-8	3.00	4.22e-8	3.26	4.23e-8	3.24	5.02e-8	3.15
Test 2	100	1.48e-3	-	1.47e-3	-	1.45e-3	-	2.07e-3	-
	200	1.98e-4	2.91	1.62e-4	3.17	1.60e-4	3.18	2.30e-4	3.18
	400	2.58e-5	2.94	1.75e-5	3.21	1.74e-5	3.20	2.69e-5	3.09
	600	8.25e-6	2.81	4.71e-6	3.24	4.71e-6	3.22	7.73e-6	3.07
	800	4.65e-6	1.99	1.84e-6	3.27	1.82e-6	3.30	3.27e-6	2.99
	1000	3.47e-6	1.32	8.94e-7	3.23	8.90e-7	3.22	1.66e-6	3.04

Table 4.3: L^1 errors and convergence rates for linear advection smooth tests 1 and 2.

reconstructed jump often being zero, which effectively disables the local diffusion and results in the local numerical flux entirely consisting of the fourth-order entropy conservative flux.

DSP-WENO is much more dissipative, so in these test cases and in general for smooth regions, we observe larger errors (though the third-order convergence is still maintained). However, this increase in diffusion leads to improved performance over SP-WENO and SP-WENOC when the solution is not smooth, as is demonstrated in the next test case.

Moving Shapes: The domain is $[0, 1.4]$, final time is $T = 1.4$, and $\text{CFL} = 0.2$, with initial profile given by

$$\begin{aligned}
u_0(x) = & 10(x - 0.2)\mathcal{X}_{(0.2, 0.3]}(x) + 10(0.4 - x)\mathcal{X}_{(0.3, 0.4]}(x) \\
& + \mathcal{X}_{(0.6, 0.8]}(x) + 100(x - 1)(1.2 - x)\mathcal{X}_{(1, 1.2]}(x)
\end{aligned}$$

where $\mathcal{X}_{\mathcal{D}}(x)$ denotes the characteristic function on set $\mathcal{D}$. Note that the initial profile is a composition of shapes with different orders of regularity. The results on a mesh with 100 cells with

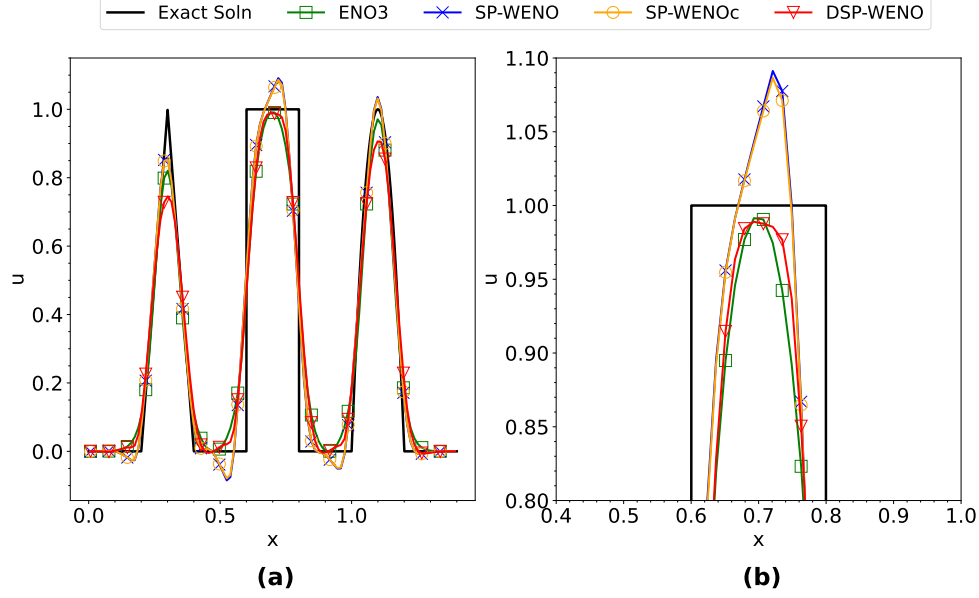


Figure 4.4: Linear advection of shapes: Solution at time $T = 1.4$. Comparison of different reconstruction methods: **(a)** $x \in [0, 1.4]$, **(b)** $x \in [0.4, 1]$.

periodic boundary conditions for various reconstruction methods are shown in Figure 4.4. Overall, ENO3 yields the best performance, as its solution is the closest to the exact solution while avoiding any spurious oscillatory behavior. The solution obtained with DSP-WENO appears to be marginally more dissipative than ENO3, but without exhibiting the large under/overshoots observed with SP-WENO or SP-WENOc.

4.4.2.2 A Note on the Computational Cost

In general, DSP-WENO is a more expensive reconstruction than its SP-WENO and SP-WENOc counterparts. As a first step, we estimate the computational times to reconstruct on 100,000 random samples of local stencil values with the various reconstruction algorithms. These times were 0.852s (DSP-WENO), 0.178s (SP-WENO), 0.192s (SP-WENOc), and 0.839s (ENO3). It is clear that DSP-WENO can be up to approximately 5 times slower than SP-WENO/SP-WENOc, while being comparable to ENO3. Furthermore, we observed that the vertex

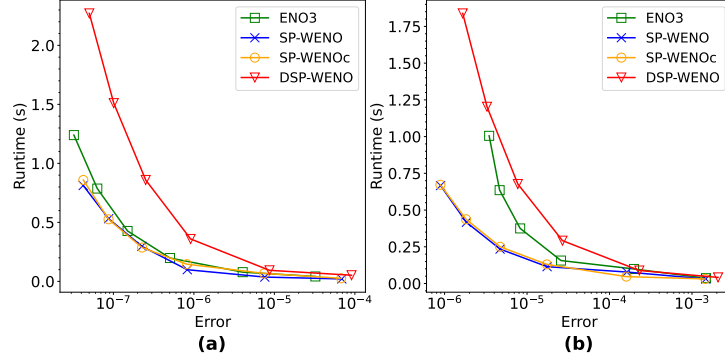


Figure 4.5: Runtime vs. L^1 error for linear advection: **(a)** Test 1, **(b)** Test 2.

search algorithm is the most expensive part of DSP-WENO (95% of the total cost).

Next, we assess the computational runtimes when the reconstructions are used within a TecNO scheme. Figure 4.5 shows the runtimes plotted against the L^1 errors for the reconstruction methods for both smooth linear advection tests of Section 4.4.2.1. Note that the markers here correspond to the various mesh resolutions considered in Table 4.3. We observe that DSP-WENO can be two to three times as expensive than the other methods at finer resolutions. This increased cost, even when compared to ENO3, can be potentially explained by the fact that the current version of the code loops over the cell interfaces in the mesh to compute the reconstructed jump required by the dissipation operator. Thus, the network for DSP-WENO is queried one input at a time, which is a very inefficient way of using a network. We expect that a batched evaluation of the network on the mesh, along with a more efficient (vectorized) implementation of the vertex algorithm, can significantly lower the overall cost of using DSP-WENO.

4.4.2.3 Burgers' Equation

We now consider the Burgers' Equation, i.e., $f(u) = u^2/2$. In this case, the second-order entropy conservative flux corresponding to the quadratic entropy is given by (3.5).

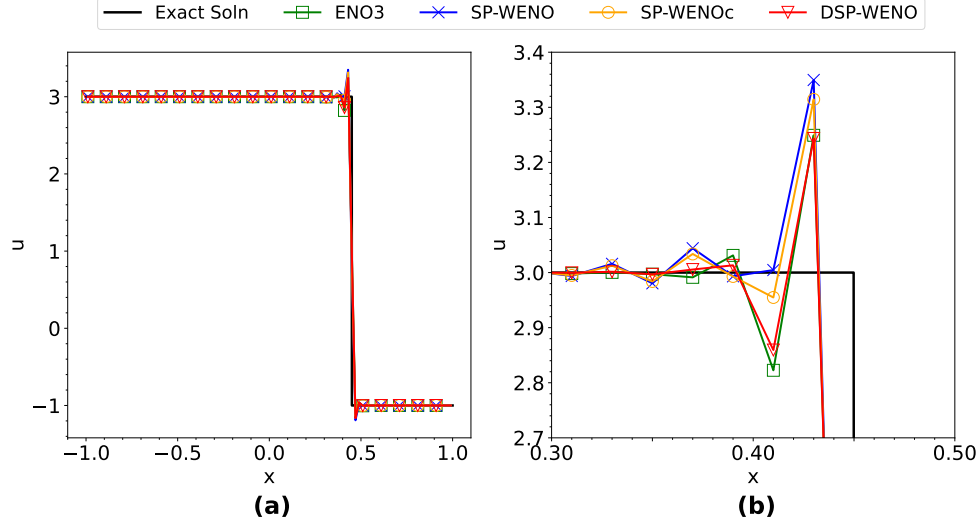


Figure 4.6: Burgers' Test 1: Solution at time $T = 0.5$. Comparison of different reconstruction methods: **(a)** $x \in [-1, 1]$, **(b)** $x \in [0.3, 0.5]$.

Test 1: The domain is $[-1, 1]$, final time is $T = 0.5$, and $\text{CFL} = 0.4$, with initial condition given by $u_0(x) = 3\mathcal{X}_{\{x < 0\}}(x) - \mathcal{X}_{\{x \geq 0\}}(x)$. The mesh consists of 100 cells with Neumann boundary conditions. Figure 4.6 shows that all the solutions experience oscillations. This is a fundamental problem with high-order TeCNO schemes, so there is often no way to completely eliminate the overshooting behavior without introducing additional diffusion. However, we observe that DSP-WENO performs better in mitigating the oscillations leading up to the overshoot.

Remark 4.13. While omitted from Figure 4.6, we found that these oscillations cannot be avoided even with the most restrictive linear reconstruction using the minmod limiter, which also satisfies the sign property [28].

Test 2: The domain is $[-4, 4]$, final time is $T = 0.4$, and $\text{CFL} = 0.4$, with initial profile

$$u_0(x) = 3\mathcal{X}_{[-1, -0.5)}(x) + \mathcal{X}_{[-0.5, 0)}(x) + 3\mathcal{X}_{[0, 0.5)}(x) + 2\mathcal{X}_{[0.5, 1)}(x) + \sin(\pi x)\mathcal{X}_{\{|x| > 1\}}(x)$$

and periodic boundary conditions. This initial condition is taken from [90] and contains both

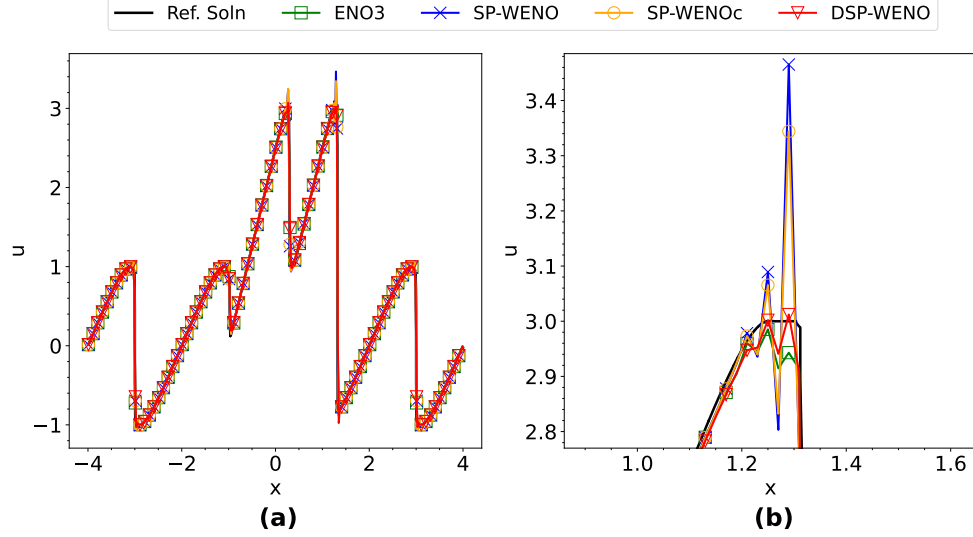


Figure 4.7: Burgers' Test 2: Solution at time $T = 0.4$. Comparison of different reconstruction methods: **(a)** $x \in [-4, 4]$, **(b)** $x \in [0.86, 1.65]$.

smooth and discontinuous features. When solved on a mesh with 400 cells, Figure 4.7 shows that SP-WENO and SP-WENOc perform poorly near the shocks, especially at $x = 1.3$ where the overshoots are relatively large. While the oscillations are still present with DSP-WENO, they are significantly mitigated when compared to the other SP-WENO methods.

Test 3: The domain is $[-1, 1]$, final time is $T = 0.7$, and $\text{CFL} = 0.4$, with initial profile

$$u_0(x) = 1 + \frac{1}{2} \sin(\pi x)$$

and periodic boundary conditions. The interesting aspect of this problem to consider is that though there is a smooth initial condition, the solution evolves to develop a discontinuity at $t = \frac{2}{\pi} \approx 0.637$. Until this time while the solution is smooth, the total entropy should be conserved. Once the discontinuity appears, we expect a sharp decrease in total entropy. We note that the time-stepping scheme does introduce a small amount of numerical diffusion, and to see how the

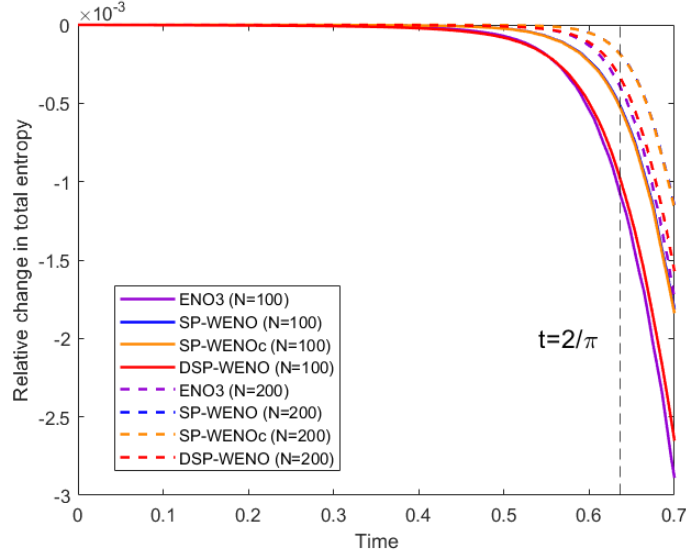


Figure 4.8: Burgers' Test 3: Relative change in total entropy with the time that the discontinuity appears marked.

entropy changes with different methods used, we evaluate the relative change in entropy

$$\frac{E(t) - E(0)}{E(0)}, \quad E(t) := \sum_i \eta_i(t) h \approx \int_{-1}^1 \eta(u(x, t)) dx$$

up to time $T = 0.7$. Figure 4.8 shows that SP-WENO is the least dissipative of the methods with SP-WENOc very close. DSP-WENO is almost as dissipative as ENO3, which makes sense since it is designed to have more dissipation than SP-WENO and SP-WENOc. As the mesh is refined from $N = 100$ cells to $N = 200$ cells, all reconstructions improve in entropy conservation while the solution is smooth.

4.4.3 Euler Equations

We now consider the Euler equations described in Section 2.4.3. In all test cases, we set $\gamma = 1.4$ and use the KEPEC flux (see Section 3.1.1) for the entropy conservative flux.

For the diffusion term in (3.14), we use the Roe-type diffusion operator described in Section 3.3.1 with $\mathbf{\Lambda}_{i+\frac{1}{2}} = \text{diag}(|u - a|, |u|, |u + a|)_{i+\frac{1}{2}}$ where $a = \sqrt{\gamma p / \rho}$ is the speed of sound in air. The various terms in the matrices of the diffusion operator are evaluated at some averaged state at the cell interface. For additional details on the diffusion operator, and the flux formulation in higher dimensions, we refer the readers to [11, 89].

1D Modified Sod Shock Tube: This is a shock tube problem [99] solved on the domain is $[0, 1]$ with initial profile

$$(\rho, \mathbf{u}, p) = (1, 0.75, 1)\mathcal{X}_{\{x < 0.3\}}(x) + (0.125, 0, 0.1)\mathcal{X}_{\{x \geq 0.3\}}(x).$$

It is solved on a domain with 400 cells with Neumann boundary conditions, until a final time of $T = 0.2$ with $\text{CFL} = 0.4$. Figure 4.9 shows that SP-WENO and SP-WENOC exhibit significant overshoots, especially near the shock at $x \approx 0.72$. DSP-WENO markedly improves on the performance of both methods by featuring only very minor overshoots in comparison. Further, the accuracy in shock-capturing is comparable to ENO3, with a sharper resolution of the contact discontinuity at $x \approx 0.57$.

1D Shu-Osher Test: This test case, proposed in [95], contains the interaction of an oscillatory smooth wave and a right-moving shock. The domain is $[-5, 5]$, final time is $T = 1.8$, and $\text{CFL} = 0.4$, with initial profile

$$\begin{aligned} (\rho, \mathbf{u}, p) = & (3.857143, 2.629369, 10.33333)\mathcal{X}_{\{x < -4\}}(x) \\ & + (1 + 0.2 \sin(5x), 0, 1)\mathcal{X}_{\{x \geq -4\}}(x), \end{aligned}$$

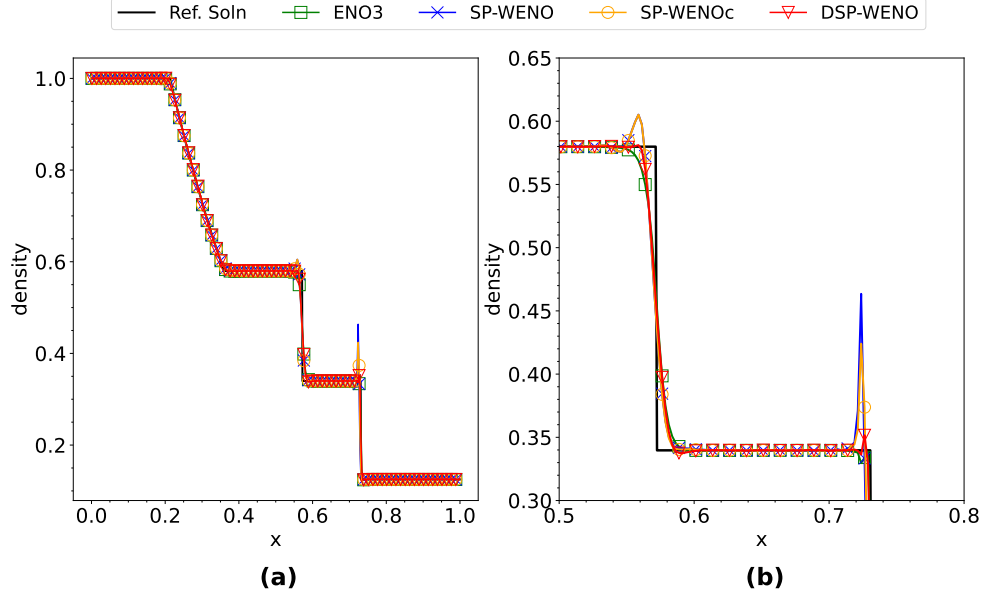


Figure 4.9: Modified Sod Shock Tube Test. Comparison of density profile at final time $T = 0.2$ for different reconstruction methods: **(a)** $x \in [0, 1]$, **(b)** $x \in [0.5, 0.8]$.

and Neumann boundary conditions. We solve on a mesh consisting of 400 cells, which is necessary to resolve the high frequency physical oscillations in the solution. This was one of the test cases presented in [89] where SP-WENOC significantly mitigates the overshoots near the shock as compared to SP-WENO, which is also what we observe in Figure 4.10. DSP-WENO is clearly the most dissipative of the methods (although comparable to ENO3) when focusing in the regions with smooth physical high-frequency oscillations. However, there is essentially no overshoot near the shock with DSP-WENO, which is a large improvement over SP-WENO and SP-WENOC.

1D Lax Test: The Lax shock tube problem [65] is described by the initial profile

$$(\rho, u, p) = (0.445, 0.698, 3.528)\mathcal{X}_{\{x < 0\}}(x) + (0.5, 0, 0.571)\mathcal{X}_{\{x \geq 0\}}(x)$$

on the domain $[-5, 5]$ with Neumann boundary conditions. The problem is solved on a mesh with

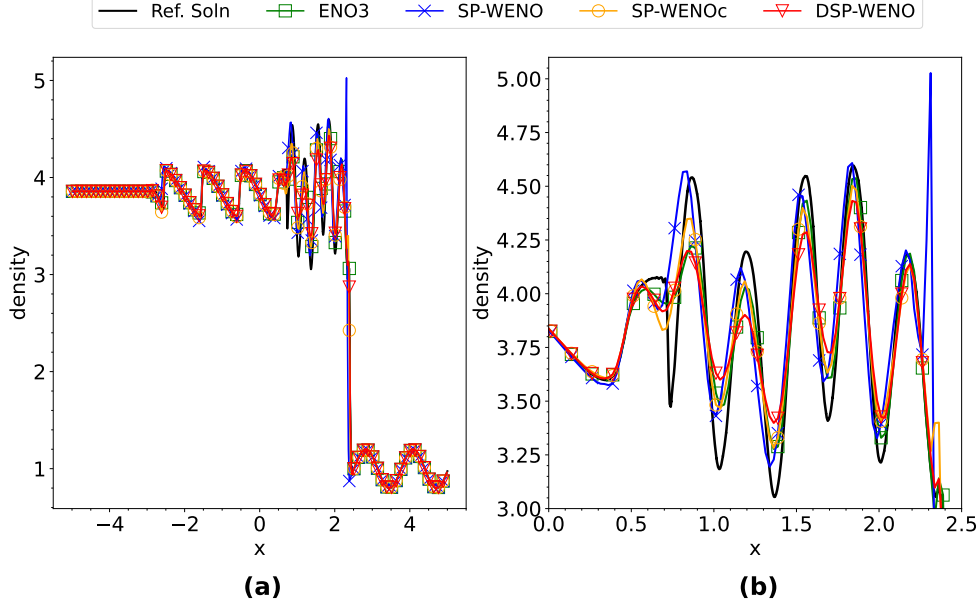


Figure 4.10: Shu-Osher Test: Comparison of density profile at final time $T = 1.8$ for different reconstruction methods: **(a)** $x \in [-5, 5]$, **(b)** $x \in [0, 2.5]$.

200 cells with $CFL = 0.4$ until a final time $T = 1.3$. Figure 4.11 shows that SP-WENO and SP-WENOc exhibit significant overshoots near the shocks. The solution obtained with DSP-WENO greatly minimizes the oscillatory behavior, similar to the previous test cases.

1D Double Rarefaction Test: For this problem consisting of two rarefaction waves [71], the domain is $[-1, 1]$, final time is $T = 0.6$, and $CFL = 0.2$, with initial profile

$$(\rho, u, p) = \begin{cases} (7, -1, 0.2) & \text{if } x < 0 \\ (7, 1, 0.2) & \text{if } x \geq 0 \end{cases},$$

and Neumann boundary conditions. This can be a challenging test case as the solution is close to vacuum near $x = 0$. We solve the problem on a mesh consisting of 128 cells. With the absence of a discontinuity (except in the initial condition), the solutions obtained using ENO3, SP-WENO, SP-WENOc, and DSP-WENO are very similar, as shown in Figure 4.12. We do

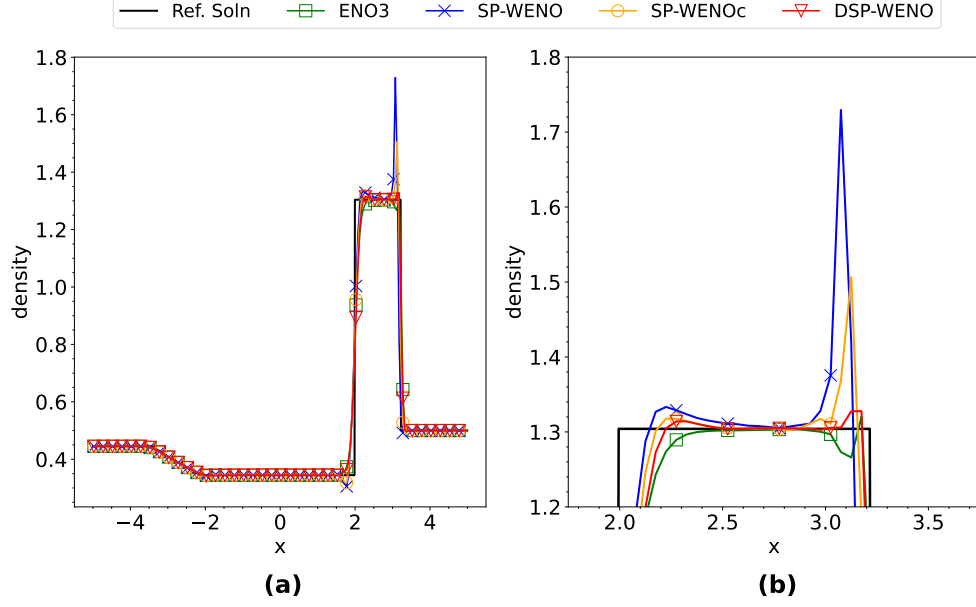


Figure 4.11: Lax Shock Tube Test. Comparison of density profile at final time $T = 1.3$ for different reconstruction methods: **(a)** $x \in [-5, 5]$, **(b)** $x \in [1.75, 3.75]$.

observe that DSP-WENO moderately mitigates the relatively minor overshooting behavior that the SP-WENO and SP-WENOc solutions exhibit.

2D Isentropic Vortex: We consider the advection of a smooth isentropic vortex and perform a mesh-refinement study. The domain is $[-5, 5] \times [-5, 5]$, final time is $T = 10$, and $\text{CFL} = 0.5$. The initial conditions are given by

$$u_1 = 1 - \frac{\Gamma y}{2\pi} \exp\left(\frac{1-r^2}{2}\right), \quad u_2 = \frac{\Gamma x}{2\pi} \exp\left(\frac{1-r^2}{2}\right), \quad \mathcal{T} = 1 - \frac{(\gamma-1)\Gamma^2}{8\gamma\pi^2} \exp(1-r^2),$$

where $\mathcal{T}$ is the fluid temperature field, $r^2 = x^2 + y^2$ and $\Gamma = 5$ (the vortex strength). Further, we have $\rho = \mathcal{T}^{1/(\gamma-1)}$ and $p = \mathcal{T}^{\gamma/(\gamma-1)}$ due to the isentropic conditions. Assuming periodic boundary condition, the vortex moves horizontal with unit velocity and completes one full periodic cycle at the final time. Table 4.4 shows the discrete L^1 errors and convergence rates for the density with various reconstructions. We observe that ENO3 is unable to achieve third-order con-

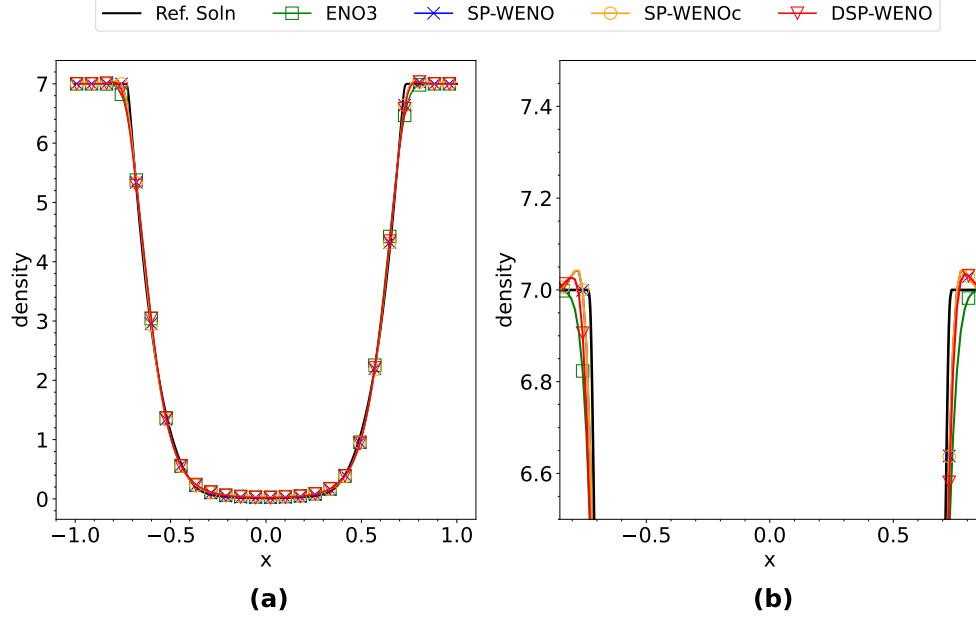


Figure 4.12: Double Rarefaction Test: Solution with TeCNO4 at time $T = 0.6$ for ρ , u , and p . Comparison of different reconstruction methods for $x \in [-1, 1]$ for ρ , u , and p .

vergence, which we hypothesize is due to the linear instabilities with ENO [93]. The SP-WENO variants lead to third-order accuracy, with the results using DSP-WENO being more dissipative as compared to SP-WENO and SP-WENOc. We reiterate that this behavior is due to SP-WENO and SP-WENOc having nearly zero reconstructed jumps in larger regions of the domain, which is not preferable in the presence of discontinuities.

N	ENO3		SP-WENO		SP-WENOc		DSP-WENO	
	Error	Rate	Error	Rate	Error	Rate	Error	Rate
50	1.17e-1	-	7.56e-2	-	7.51e-2	-	2.09e-1	-
100	1.75e-2	2.74	6.62e-3	3.51	6.73e-3	3.48	2.13e-2	3.30
150	7.13e-3	2.22	1.37e-3	3.88	1.40e-3	3.87	5.94e-3	3.15
200	3.60e-3	2.38	4.59e-4	3.80	4.69e-4	3.80	2.42e-3	3.13
300	1.48e-3	2.19	1.01e-4	3.73	1.03e-4	3.74	6.73e-4	3.15
400	7.86e-4	2.20	3.57e-5	3.61	3.64e-5	3.61	2.73e-4	3.14

Table 4.4: L^1 errors in density for advecting isentropic vortex with different reconstructions.

2D Riemann problem (configuration 12): We consider a two-dimensional Riemann prob-

lem for the Euler equations [63] whose initial conditions on the domain $[0, 1] \times [0, 1]$ are given by

$$(\rho, \mathbf{u}_1, \mathbf{u}_2, p) = (0.5313, 0, 0, 0.4)\mathcal{X}_{\mathcal{Q}_1}(x, y) + (1, 0.7276, 0, 1)\mathcal{X}_{\mathcal{Q}_2}(x, y) \\ + (0.8, 0, 0, 1)\mathcal{X}_{\mathcal{Q}_3}(x, y) + (1, 0, 0.7276, 1)\mathcal{X}_{\mathcal{Q}_4}(x, y)$$

where $\mathcal{Q}_1 = \{x > 0.5, y > 0.5\}$, $\mathcal{Q}_2 = \{x \leq 0.5, y > 0.5\}$, $\mathcal{Q}_3 = \{x \leq 0.5, y \leq 0.5\}$, and $\mathcal{Q}_4 = \{x > 0.5, y \leq 0.5\}$ are the four Cartesian quadrants. In this case, the evolved solution (with Neumann boundaries) comprises two shock waves and two contact discontinuities. The problem is solved on a mesh consisting of 400×400 cells until a final time $T = 0.25$ with $\text{CFL} = 0.5$. All methods sharply resolve the shocks and contact waves as shown in Figure 4.13. However, SP-WENO and SP-WENOC lead to significant overshoots, which can once again be seen in the one-dimensional slices shown in Figure 4.13. On the other hand, the overshoots are generally smaller with DSP-WENO and comparable to ENO3. The reference solution is generated using ENO3 by solving the problem on 1200×1200 mesh.

2D Riemann problem (configuration 3): We consider another two-dimensional Riemann problem for the Euler equations [63] whose initial conditions are

$$(\rho, \mathbf{u}_1, \mathbf{u}_2, p) = (1.5, 0, 0, 1.5)\mathcal{X}_{\mathcal{Q}_1}(x, y) + (0.5323, 1.206, 0, 0.3)\mathcal{X}_{\mathcal{Q}_2}(x, y) \\ + (0.138, 1.206, 1.206, 0.029)\mathcal{X}_{\mathcal{Q}_3}(x, y) + (0.5323, 0, 1.206, 0.3)\mathcal{X}_{\mathcal{Q}_4}(x, y).$$

When solved on the domain $[0, 1] \times [0, 1]$ with Neumann boundary conditions, the solution evolves into four interacting shock waves. We solve the problem on a mesh consisting of 400×400 cells

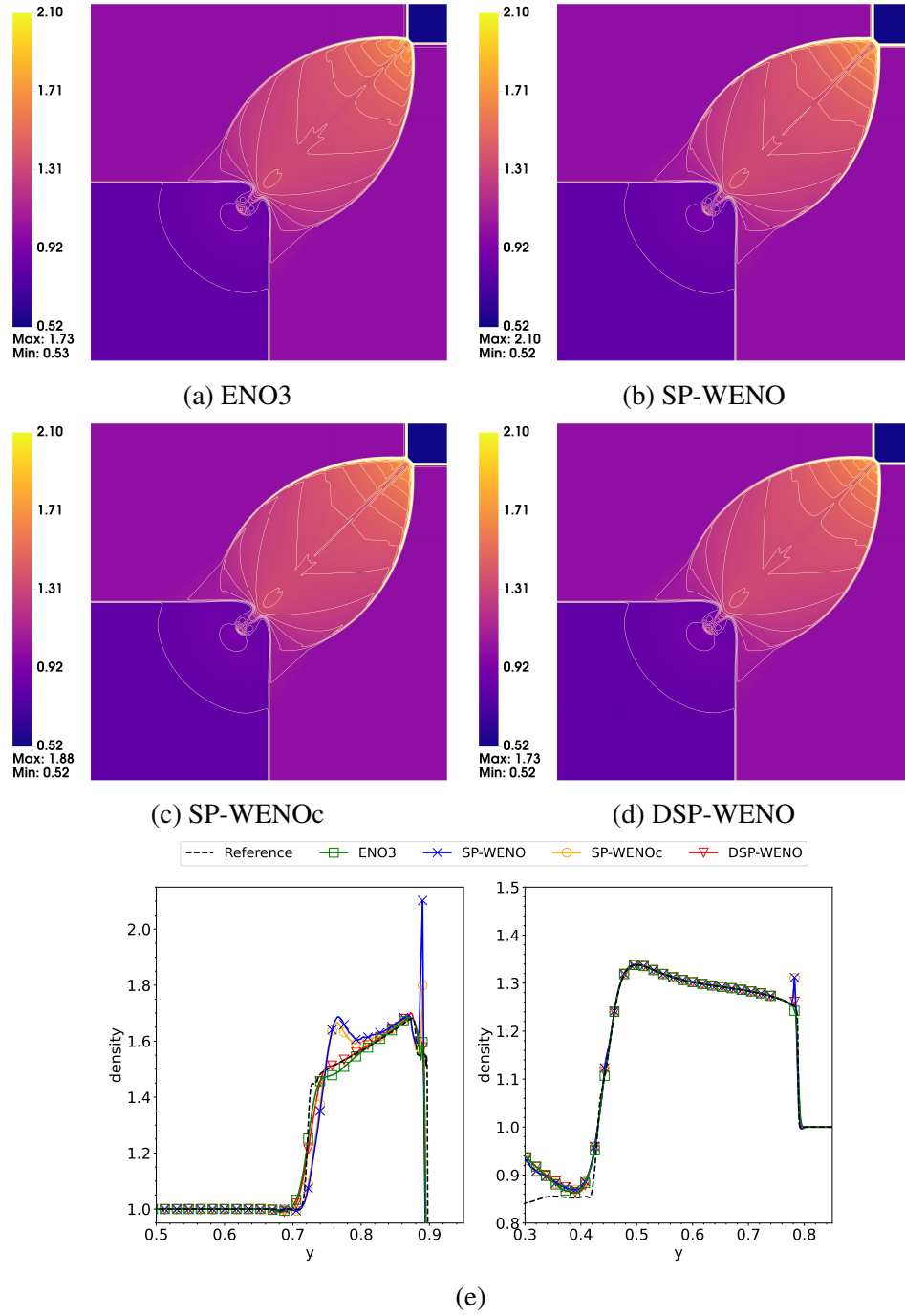


Figure 4.13: 2D Riemann problem (conf. 12): Density profiles at time $T = 0.25$ with 30 contour lines between 0.52 and 2.2. (a)-(d) Comparison of different reconstruction methods. (e) One-dimensional slices of the solution using each reconstruction methods through $x = 0.89$ (bottom left) and $x = 0.5$ (bottom right).

until a final time $T = 0.3$ with $\text{CFL} = 0.4$. We observe from the density profiles in Figure 4.14 that both SP-WENO and DSP-WENO lead to a protruding artifact in the bottom left region of the domain which resembles carbuncle-like phenomenon [86, 52]. This artifact is significantly more pronounced for SP-WENO. We remark here that entropy stability does not preclude the appearance of pathological behavior of this form. It was noted in [11, 84] that the carbuncle effect can be mitigated by introducing additional diffusion in the numerical scheme. In particular, Figure 4.15 demonstrates that the appearance of the carbuncle artifact is weakened by using a Rusanov-type dissipation operator of the form $\Lambda_{i+\frac{1}{2}} = (|u| + a)_{i+\frac{1}{2}} \mathbf{I}$ in (3.14). In particular, Figure 4.16 shows a zoomed-in perspective of the carbuncle-like artifact for SP-WENO and DSP-WENO with the Roe and Rusanov dissipation. It appears that the behavior is better mitigated in the DSP-WENO solution than that of SP-WENO when switching the dissipation operator.

Empirical results in literature [11] suggest that the carbuncle typically arises in numerical schemes that resolve contact discontinuities well. This merits further investigation to assess the properties of SP-WENO-type reconstruction schemes in this context.

Kelvin-Helmholtz Instability: Finally, we consider a more complex two-dimensional problem for the Euler equations. This is the well-known Kelvin-Helmholtz instability which describes a shear flow separating two fluid regions with differing density, leading to two-dimensional turbulence. The initial conditions [94] on the domain $[-0.5, 0.5] \times [-0.5, 0.5]$ are given by

$$\begin{aligned} (\rho, \mathbf{u}_1, \mathbf{u}_2, p) = & (1, 0.5, \epsilon, 2.5) \mathcal{X}_{\{y > 0.25\}}(x, y) + (2, -0.5, \epsilon, 2.5) \mathcal{X}_{\{|y| \leq 0.25\}}(x, y) \\ & + (1, 0.5, \epsilon, 2.5) \mathcal{X}_{\{y < -0.25\}}(x, y) \end{aligned}$$

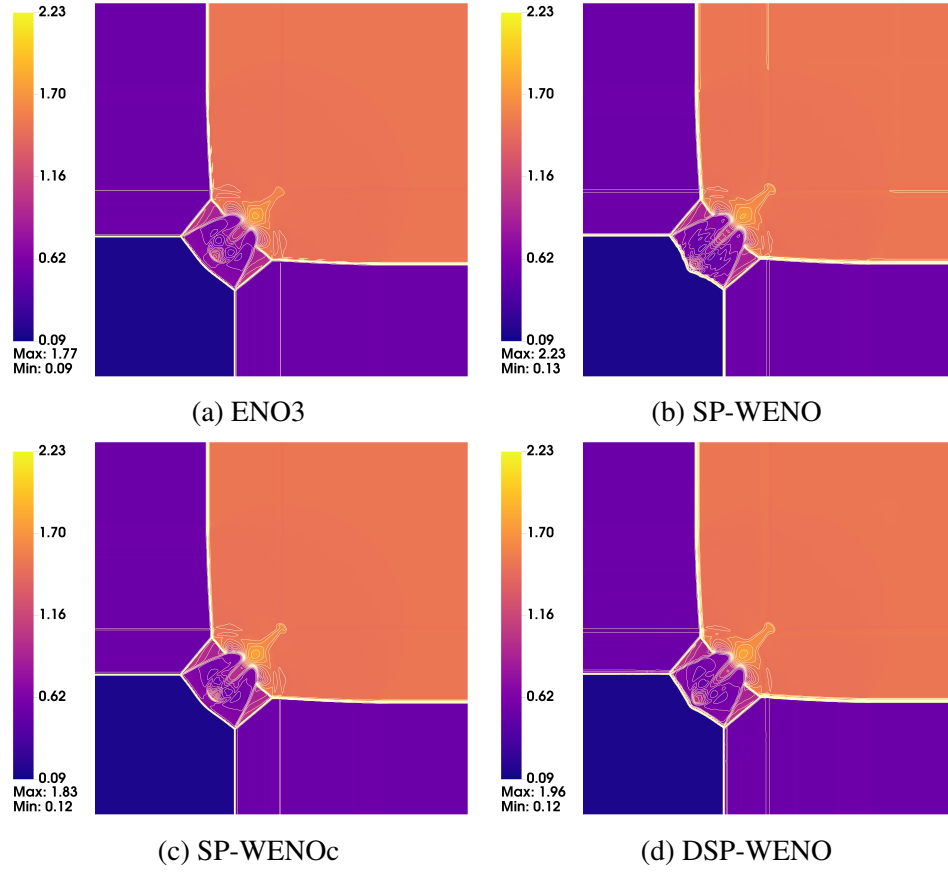


Figure 4.14: 2D Riemann problem (conf. 3): Density profiles at time $T = 0.3$ with 30 contour lines between 0.1 and 2.28 using Roe dissipation. Comparison of different reconstruction methods.

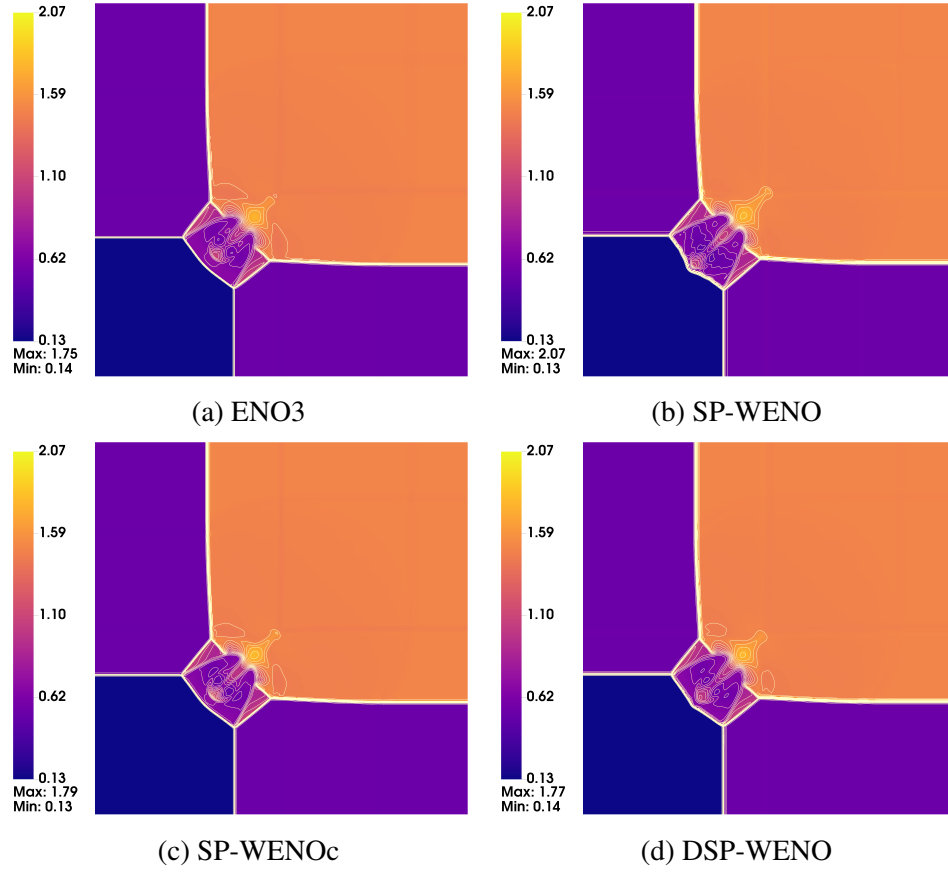
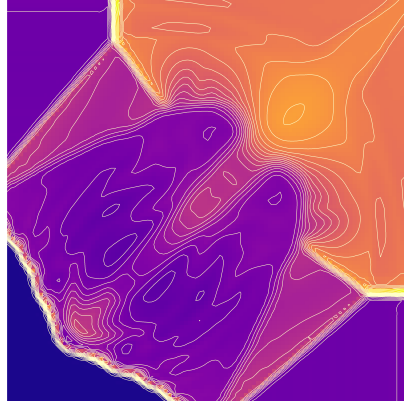
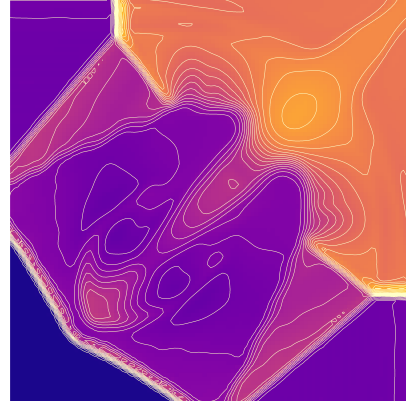


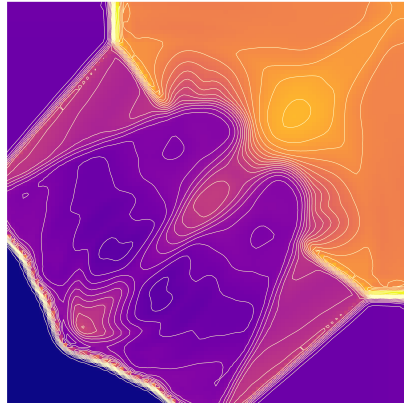
Figure 4.15: 2D Riemann problem (conf. 3): Density profiles at time $T = 0.3$ with 30 contour lines between 0.1 and 2.28 using Rusanov dissipation. Comparison of different reconstruction methods.



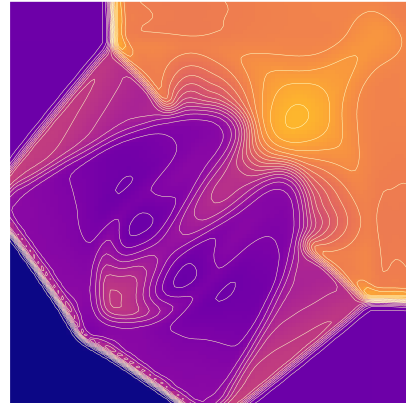
(a) SP-WENO Roe



(b) DSP-WENO Roe



(c) SP-WENO Rusanov



(d) DSP-WENO Rusanov

Figure 4.16: 2D Riemann problem (conf. 3): Zoom-in of the carbuncle-like phenomenon for SP-WENO and DSP-WENO with Roe and Rusanov dissipation.

with periodic boundary conditions. When $\epsilon = 0$, the initial conditions describe a stationary solution of the PDE system. However, when a small perturbation is introduced in the initial condition, which in our case is done by setting the vertical velocity to be $\epsilon = 0.01 \sin(2\pi x)$, the shearing leads to an instability generating small-scale vortical structures at the interfaces.

We solve this problem using ENO3, SP-WENO, SP-WENOC, and DSP-WENO in the TeCNO scheme on meshes consisting of 256×256 cells and 512×512 cells until a final time $T = 3$ with $\text{CFL} = 0.4$. For this problem, there is no convergence with mesh refinement [32], with finer structures appearing as the mesh is successively refined, which can be seen in Figures 4.17 and 4.18. Simulations with the various reconstruction methods on the same meshes exhibit similar behavior, although their corresponding solutions on the same mesh look very different. While the mesh convergence for a single deterministic sample does not exist, the convergence of the statistics of random samples (with randomized perturbations) does exist within the framework of measure-valued solutions [32] or statistical solutions [33].

4.5 High-order extensions

All of the numerical results presented in this chapter correspond to the use of our third-order reconstruction DSP-WENO. As developed, DSP-WENO is only third-order accurate, but it is likely possible to extend the framework to an SP-WENO reconstruction that is of arbitrary high-order accuracy because of the following theorem.

Theorem 4.1. *Consider a function v which we wish to interpolate to find the reconstructed jump*

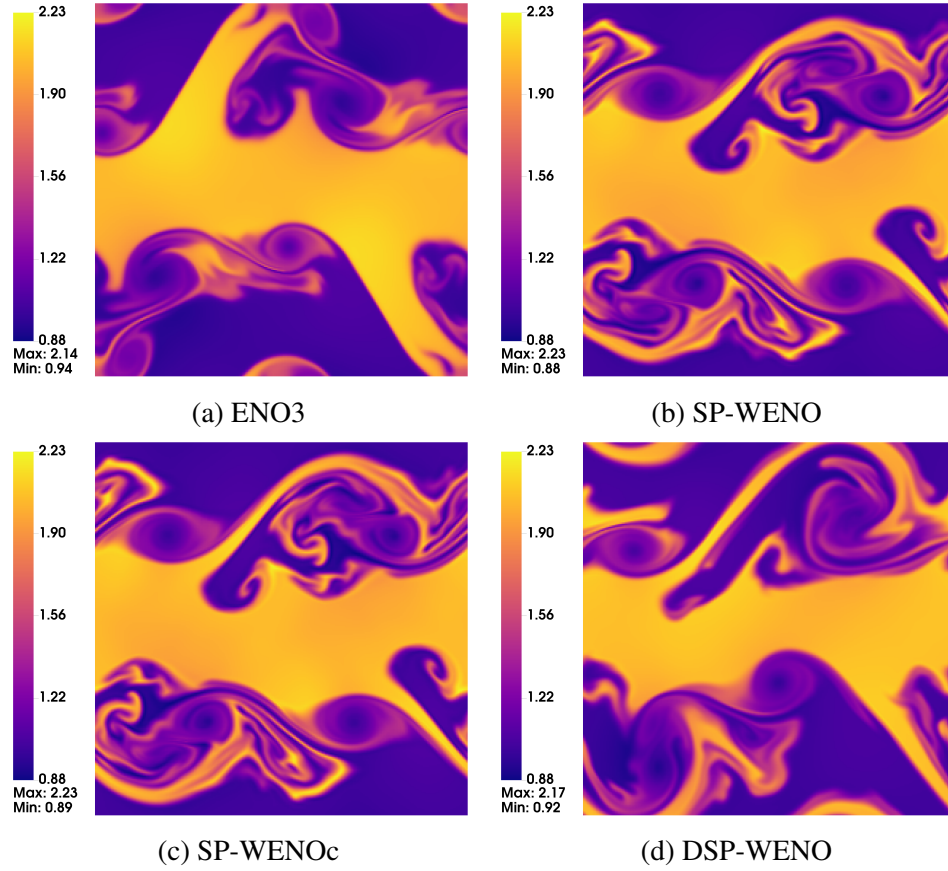


Figure 4.17: Kelvin-Helmholtz Instability: Density profiles at time $T = 3$. Comparison of different reconstruction methods for 256×256 mesh.

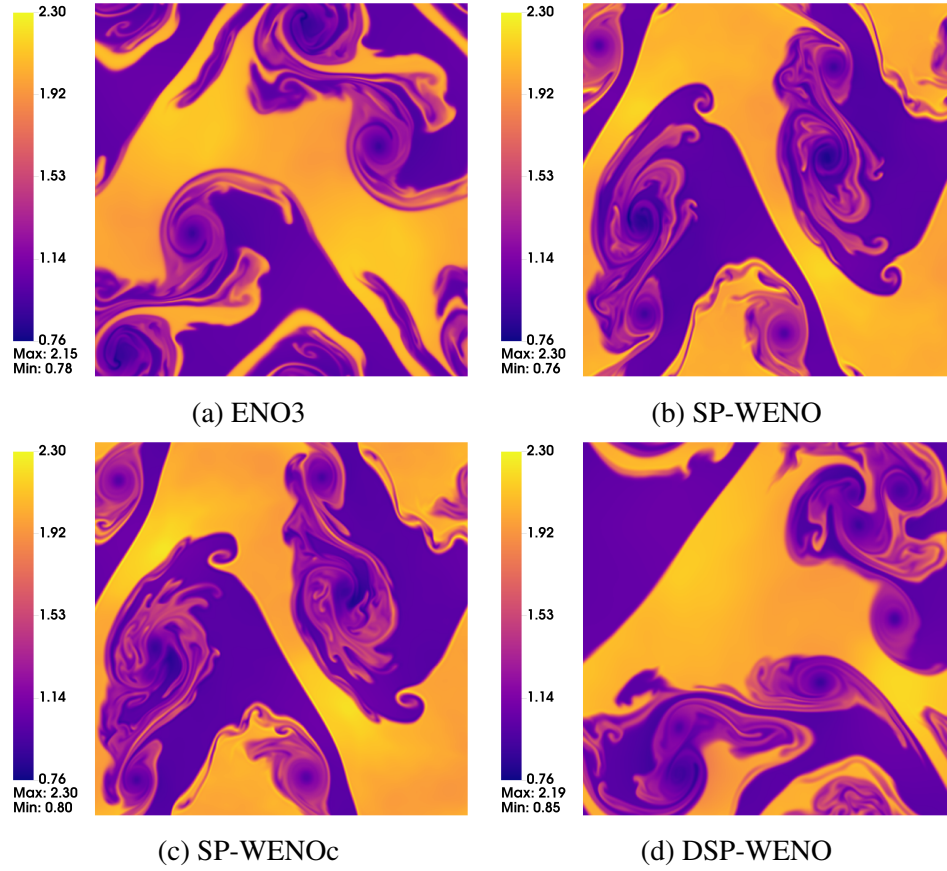


Figure 4.18: Kelvin-Helmholtz Instability: Density profiles at time $T = 3$. Comparison of different reconstruction methods for 512×512 mesh.

at cell interface $i + \frac{1}{2}$. Let us define the generalized jump ratio

$$\theta_{i,k} = \frac{\Delta v_{k+i+\frac{1}{2}}}{\Delta v_{i+\frac{1}{2}}}.$$

Assuming $\Delta v_{i+\frac{1}{2}} \neq 0$, a $(2k - 1)$ -th order WENO has the reconstructed jump

$$[[v]]_{i+\frac{1}{2}} = G_{i+\frac{1}{2}} \Delta v_{i+\frac{1}{2}} \quad (4.14)$$

where

$$\begin{aligned} G_{i+\frac{1}{2}} = & \tilde{w}_0 \sum_{j=1}^{k-1} \left[\sum_{q=0}^{j-1} \theta_{i,j-q} \prod_{\substack{l=i+1 \\ l \neq i+1+j}}^{i+k} \left(\frac{x - x_l}{x_{i+1+j} - x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \\ & - \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=0}^r \left[\sum_{q=0}^{r-j} \theta_{i,-r+j+q} \prod_{\substack{l=i-r \\ l \neq i-r+j}}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \right\} \\ & + \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=r+2}^{k-1} \left[\sum_{q=1}^{j-r-1} \theta_{i,-r+j-q} \prod_{\substack{l=i-r \\ l \neq i-r+j}}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \right\} \\ & + w_{k-1} \sum_{j=0}^{k-1} \left[\sum_{q=0}^{k-1-j} \theta_{i,-k+1+j+q} \prod_{\substack{l=i-k+1 \\ l \neq i-k+1+j}}^i \left(\frac{x - x_l}{x_{i-k+1+j} - x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right]. \quad (4.15) \end{aligned}$$

Thus, the resulting reconstruction will be sign-preserving if $G_{i+\frac{1}{2}} \geq 0$.

The proof of Theorem 4.1 can be found in Appendix C. The sign property for third-order SP-WENO (3.19) is a special case of this theorem when $k = 2$. Developing DSP-WENO in this case involved recasting the search for the WENO weights as a problem of finding optimal locations in two-dimensional convex regions. For the $k > 2$ case, it is not immediately obvious

that a similar strategy is feasible. Nonetheless, Theorem [4.1](#) serves as a starting point to develop arbitrarily high-order accurate SP-WENO_s. If other desirable properties are required, care must be taken to establish the necessary constraints.

Chapter 5: Conclusion for DSP-WENO

Part I of this dissertation demonstrates that it is possible to use deep learning tools to learn adaptive reconstruction algorithms constrained to satisfy critical physical properties, such as the sign property. We have designed a novel neural network-based third-order WENO scheme called DSP-WENO, which is guaranteed to satisfy the sign property. Thus, DSP-WENO can be used within the TeCNO framework to obtain high-order entropy stable finite difference schemes. The motivation behind the proposed approach was two-fold: (i) overcome the linear instability issues faced by ENO reconstruction (see Tables 4.3 and 4.4), and (ii) have better shock-capturing capabilities compared to existing WENO algorithms, i.e., SP-WENO and SP-WENOc, that satisfy the sign property.

A key element in the proposed strategy was to decouple the constraints guaranteeing the sign property and third-order accuracy (in smooth regions) from the learning process. A strong imposition of these constraints led to a convex polygonal feasible region from which the WENO weights need to be selected. Then a network was trained to adaptively choose the weights from the feasible region to ensure good reconstruction properties near discontinuities. In contrast, we could impose these constraints weakly by adding a penalty term to the loss functions (analogous to physics-informed neural networks [87]). However, this would lead to the following challenges:

- The weak imposition would not guarantee the satisfaction of the sign property in all sit-

uations, thus making it impossible to prove entropy stability in the TeCNO framework. Our early attempts to develop DSP-WENO involved this technique, and it never resulted in 100% satisfaction of the sign property.

- Neural networks trained on data extracted at a particular mesh resolution are rarely capable of demonstrating mesh convergence when tested on data from finer grids. In fact, the training (and test) errors typically plateau after a certain number of epochs, with the error values being several orders of magnitude larger than machine epsilon.

Thus, there are major benefits of decoupling such constraints from the learning process and imposing them strongly.

The data used to train DSP-WENO did not require solving conservation laws. Instead the data was generated from a library of functions with varying smoothness, which mimic the local structures typically arising in solutions to conservation laws. Thus, the cost of generating training data is insignificant, and the trained DSP-WENO is model agnostic, i.e., does not depend on a specific conservation law. In other words, the DSP-WENO needs to be trained once offline and can then be used for any conservation law. This strategy was based on similar ideas first considered in [90] for designing troubled-cell detectors.

When comparing the numerical solutions obtained using various reconstruction methods satisfying the sign property, DSP-WENO achieved third-order accuracy in smooth regions, while being more dissipative compared to SP-WENO and SP-WENOC. This was attributed to the fact that the reconstructed jump is mostly zero (or small) with SP-WENO and SP-WENOC. However, this had a negative impact near discontinuities where the solutions exhibited large spurious oscillations due to insufficient diffusion. DSP-WENO significantly mitigated these spurious os-

cillations without compromising the order of accuracy in smooth regions. Further, DSP-WENO did not suffer from linear instabilities faced by ENO3. It is important to note that the ENO3 stencil (corresponding to an interface) comprises six cells, while third-order SP-WENO and its variants (including DSP-WENO) have access to the solution on a stencil with just four cells.

Finally, we recognize that the proposed DSP-WENO has drawbacks. For one thing, we observe a carbuncle-like phenomenon in the 2D Riemann problem (configuration 3), which also appears with SP-WENO but surprisingly not SP-WENOc. We believe that this behavior is correlated by the ability of the schemes to resolve contact waves, which has also been numerically observed in the literature.

5.1 Future Directions

Building upon the current framework of DSP-WENO can be explored in a few future directions:

1. **Higher-order accurate DSP-WENO:** As discussed in Section 4.5, there is already a provided structure in which DSP-WENO can be extended to high-order (> 3) accurate reconstructions satisfying the sign property. To implement such a reconstruction, the sign property constraint must be formulated on a larger stencil and the constraints transformed into corresponding (possibly convex) polyhedral feasible regions in high-dimensional spaces for the WENO weights. Thus, instead of attempting to construct an explicit weight selection strategy by hand (which presents significant challenges in high dimensions), a neural network can learn the selection procedure from training data.
2. **Hybridization:** Since the improved performance of DSP-WENO comes at the expense of

additional computational cost, exploring hybridization techniques similar to [17, 19, 26] may allow us to dynamically switch between the reconstruction methods SP-WENO and DSP-WENO so as to enjoy better performance in smooth regions while retaining the improved shock-capturing ability of DSP-WENO. The actual method of smoothly switching between two methods is not clear, but this is a promising direction to explore.

3. **Unstructured grids:** Currently, DSP-WENO can only be deployed on problems with a uniform mesh. Similar sign-preserving WENO-type reconstructions can also be designed on unstructured grids.

Part II

Operator Learning with the Petrov-Galerkin formulation

Chapter 6: Variational Forms

PDEs have two associated forms, the strong and variational forms. Strong forms, such as those seen previously in the discussion of hyperbolic conservation laws describe the behavior of the corresponding system via differential equations. Their solutions directly determine the physical behavior. Variational forms are weaker in the sense that the solutions do not have to satisfy the strong form absolutely and in fact can be less regular. Moreover, these variational forms involve integrals where the PDE terms are multiplied with special test (or weighting) functions.

In this chapter, we focus on the variational forms associated with (second-order) linear elliptic PDEs. The PDE operator is denoted by $\mathcal{L}(\cdot, \mathbf{g})$ where $\mathbf{g}$ parameterizes the PDE (with a more formal definition in the following section). In this dissertation, we will consider elliptic problems with different boundary conditions.

6.1 Statement of the variational form

Let $\Omega \in \mathbb{R}^d$ be an open, bounded domain with piecewise smooth boundary Γ . The boundary is further split into the Dirichlet boundary Γ_D and natural boundary Γ_η , with $\Gamma = \Gamma_D \cup \Gamma_\eta$. Define the space $H_D^r(\Omega) = \{u \in H^r(\Omega) : u|_{\Gamma_D} = 0\}$. We consider the following scalar elliptic

boundary value problem

$$\begin{aligned}
\mathcal{L}(u(\mathbf{x}); \mathbf{g}(\mathbf{x})) &= f(\mathbf{x}) & \forall \mathbf{x} \in \Omega, \\
\mathcal{B}(u(\mathbf{x}); \mathbf{g}(\mathbf{x})) &= \eta(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_\eta, \\
u(\mathbf{x}) &= 0 & \forall \mathbf{x} \in \Gamma_D,
\end{aligned} \tag{6.1}$$

where $\mathcal{L}$ is a linear elliptic PDE operator and $\mathcal{B}$ is the natural boundary operator, both parametrized by a set of functions $\mathbf{g} \in \mathcal{G}$. Also, $f \in \mathcal{F} \subseteq L^2(\Omega)$ is the source term, and $\eta \in \mathcal{H} \subseteq L^2(\Gamma_\eta)$. The solution $u \in \mathcal{V} := H_D^r(\Omega)$, where r depends on the order of the operator $\mathcal{L}$ and the regularity of the PDE data.

A particular example of (6.1) is the steady advection-diffusion equation with

$$\begin{aligned}
-\nabla \cdot (\kappa(\mathbf{x}) \nabla u(\mathbf{x})) + \mathbf{c}(\mathbf{x}) \cdot \nabla u(\mathbf{x}) &= f(\mathbf{x}) & \forall \mathbf{x} \in \Omega, \\
\frac{|\mathbf{c} \cdot \mathbf{n}| - \mathbf{c} \cdot \mathbf{n}}{2} + \kappa(\mathbf{x}) \nabla u(\mathbf{x}) \cdot \mathbf{n} &= \eta(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_\eta, \\
u(\mathbf{x}) &= 0 & \forall \mathbf{x} \in \Gamma_D,
\end{aligned} \tag{6.2}$$

where $\mathcal{V} = H_D^1(\Omega)$, $\mathbf{n}$ is the unit outward normal on Γ_η and the set of parametrizing functions are $\mathbf{g} = [\kappa, \mathbf{c}]$. Here $\kappa \in L^\infty(\Omega) \cup \{\kappa \mid \kappa(\mathbf{x}) \geq \kappa_{\min} \text{ a.e. } \mathbf{x} \in \Omega\}$ for some (fixed) scalar $\kappa_{\min} > 0$ is the diffusion coefficient, while $\mathbf{c} \in H_{\text{div}}^1(\Omega) = \{\mathbf{c} \in [L^2(\Omega)]^2 \mid \nabla \cdot \mathbf{c} \in L^2(\Omega)\}$ is the velocity field. The advective term in the Neumann boundary condition acts as a stabilization term if there is backflow present [78] (note that it vanishes on outflow boundaries where $\mathbf{c} \cdot \mathbf{n} \geq 0$). We will use (6.2) for the remainder of this dissertation.

The corresponding variational formulation of (6.1) obtained with integration by parts is

given by: find $u \in \mathcal{V}$ such that

$$a(u, w; \mathbf{g}) = (f, w) + (\eta, w)_{\Gamma_\eta} \quad \forall w \in \mathcal{V}, \quad (6.3)$$

where $(\cdot, \cdot)$ is the $L^2(\Omega)$ inner product, $(\cdot, \cdot)_{\Gamma_\eta}$ is the $L^2(\Gamma_\eta)$ inner product, while $a(u, w; \mathbf{g})$ is the associated bilinear form parameterized by $\mathbf{g}$. We also assume that the bilinear form is coercive, which requires additional conditions on $\mathbf{g}$. With this assumption, a unique solution of (6.3) exists, as guaranteed by the Lax-Milgram theorem [10, 9].

For the particular case of the advection-diffusion equation, we have

$$a(u, w; \kappa, \mathbf{c}) := (\kappa \nabla u, \nabla w) + (\mathbf{c} \cdot \nabla u, w) + \left(\frac{|\mathbf{c} \cdot \mathbf{n}| - \mathbf{c} \cdot \mathbf{n}}{2} u, w \right)_{\Gamma_\eta} \quad (6.4)$$

where coercivity is guaranteed by assuming $\nabla \cdot \mathbf{c} = 0$, or alternately by assuming the bound $\|\mathbf{c}\|_{L^\infty} \leq C_\Omega \kappa_{\min}$ where C_Ω is the constant (depending on Ω) arising from the Poincaré inequality [1].

6.2 Finite-dimensional projection

Consider the finite-dimensional space $\bar{\mathcal{V}} \subset \mathcal{V}$ spanned by a *trial basis* $\{\phi_i(\mathbf{x})\}_{i=1}^N$. We are interested in the best finite-dimensional approximation $\bar{u} \in \bar{\mathcal{V}}$ of the true solution u of (6.3) as measured in the norm $\|\cdot\|_\dagger$ induce by a given inner product $(\cdot, \cdot)_\dagger$ on $\mathcal{V} \times \mathcal{V}$. More precisely, we solve the following projection problem: Find $\bar{u} \in \bar{\mathcal{V}}$ such that

$$\bar{u} = \arg \min_{\bar{w} \in \bar{\mathcal{V}}} \|u - \bar{w}\|_\dagger^2. \quad (6.5)$$

The optimality condition (by setting first variations to zero) associated with (6.5) leads to

$$(u - \bar{u}, \bar{w})_{\dagger} = 0 \quad \forall \bar{w} \in \bar{\mathcal{V}} \quad (6.6)$$

which implies that the projection error $e := u - \bar{u}$ is orthogonal to $\bar{\mathcal{V}}$.

Since $\bar{u} \in \bar{\mathcal{V}}$, we can express it as

$$\bar{u}(\mathbf{x}) = \sum_{j=1}^N \bar{u}_j \phi_j(\mathbf{x}) = \bar{\mathbf{u}}^\top \boldsymbol{\Phi}(\mathbf{x}), \quad (6.7)$$

where $\bar{\mathbf{u}} = [\bar{u}_1, \dots, \bar{u}_N]^\top \in \mathbb{R}^N$ is the coefficient vector while $\boldsymbol{\Phi} = [\phi_1, \dots, \phi_N]^\top \in \mathcal{V}^N$ is the vector of trial basis functions. Substituting this expansion into (6.6) and setting $\bar{w}$ as the trial basis functions leads to the following linear system of equations

$$\mathbf{M} \bar{\mathbf{u}} = \mathbf{r} \quad \text{where} \quad r_j = (u, \phi_j)_{\dagger}, \quad M_{ij} = (\phi_i, \phi_j)_{\dagger}, \quad \forall 1 \leq i, j \leq N. \quad (6.8)$$

Note that the mass matrix $\mathbf{M}$ is symmetric and positive definite. Solving (6.8) requires knowledge of the infinite-dimensional solution u of (6.3). However, our goal is to determine $\bar{u}$ directly without having to evaluate u . To describe a suitable framework, we first need to introduce a specific variational form.

6.3 Petrov-Galerkin formulation

We have the goal of recovering the weak solution's best approximation $\bar{u}$ on a given finite-dimensional functional space $\bar{\mathcal{V}} \subset \mathcal{V}$ as measured in a desired norm $\|\cdot\|_{\dagger}$. One can show that

$\bar{u}$ is equivalently the solution to a Petrov-Galerkin formulation of the discrete problem where the test space $\mathcal{V}^\dagger \subset \mathcal{V}$ is spanned by a set of optimal *weighting functions*. We remark that unlike a standard Galerkin formulation, the test space $\mathcal{V}^\dagger$ will typically be different than the trial space $\bar{\mathcal{V}}$. Once the weighting functions are determined, the discrete problem simplifies to a symmetric, positive-definite weak formulation determined by the norm $\|\cdot\|_\dagger$. The underlying theory of optimal Petrov-Galerkin methods is elegant and allows for the recovery of optimal convergence rates [74], even for cases where standard Galerkin methods fail.

6.3.1 Details

The notion of optimal weighting functions in variational methods for PDEs was first formalized for the advection-diffusion equation by Barrett and Morton [4], although an earlier instantiation of it occurred in the thesis of Hemker [45]. Keeping with the discussion of Section 6.1, we consider the advection-diffusion equation. With trial basis $\bar{\mathcal{V}} = \text{span}\{\phi_1, \dots, \phi_N\}$ and test basis $\mathcal{V}^\dagger = \text{span}\{\psi_1, \dots, \psi_N\}$, a Petrov-Galerkin approximation to (6.3) is: find $\bar{u} \in \bar{\mathcal{V}}$ such that

$$a(\bar{u}, w; \mathbf{g}) = (f, w) + (\eta, w)_{\Gamma_\eta} \quad \forall w \in \mathcal{V}^\dagger. \quad (6.9)$$

When advection is absent (i.e. $c = 0$), the PDE operator $\mathcal{L}$ is self-adjoint, and setting $\mathcal{V}^\dagger = \bar{\mathcal{V}}$ ensures the existence of a unique solution to (6.9). This reduces to the Galerkin formulation, and in fact this Galerkin approximation $\bar{u}$ is the best fit to u in the norm induced by the symmetric bilinear form $(\kappa \nabla u, \nabla w)$. This optimal approximation property of the Galerkin method does not hold when advection is present since the bilinear form is no longer symmetric. In practice, the numerical Galerkin approximation will exhibit significant oscillations due to the advective term.

Employing a Petrov-Galerkin formulation (6.9) where $\bar{\mathcal{V}}$ and $\mathcal{V}^\dagger$ are different allows for this to be remedied and provides accurate solutions [4].

6.3.2 Limitations of Galerkin methods

The optimal Petrov-Galerkin framework has been applied in the development of an adaptive characteristic fraction-step method [24], and also in the formulation for the Timoshenko beam problem that was insensitive to the ratio of thickness to length [74]. Regarding this latter problem, it was noted in [74] when this ratio becomes very small (i.e. in the limit of a thin beam), the trial/test space of standard Galerkin finite element formulations reduces to the space containing only the zero function (this is known as the *locking* pathology). Specifically, the bilinear form in the variational form of the problem is given as

$$a(\mathbf{u}, \mathbf{v}) = \int_0^1 u'_2 v'_2 dx + \frac{1}{\epsilon^2} \int_0^1 (u'_1 - u_2)(v'_1 - v_2) dx$$

where $\epsilon \geq 0$ is the dimensionless ratio of thickness to length. In the limit $\epsilon \rightarrow 0$, the only admissible solution is the trivial solution. Thus, the finite element solution is no longer able to capture the behavior of the true solution.

This locking behavior is avoided when using a Petrov-Galerkin formulation. However, the authors in [74] concluded that extending this formulation to the two-dimensional analog of the Timoshenko beam, namely the Reissner-Mindlin plate [49], would be a very difficult proposition. A similar conclusion may also be made for the advection-diffusion problem in higher dimensions. Although explicit constructions of optimal weighting functions have been carried out for simple problems in one-dimensions [4, 74, 23] and two-dimensions [3], such constructions for a general

multi-dimensional problem remained elusive. Thus, interest in this approach diminished as alternative finite element strategies such as Galerkin least squares [51], residual-free bubbles [34] and variational multiscale [50] gained attention.

6.4 Optimal Petrov-Galerkin

Now, we formulate the optimal Petrov-Galerkin framework. We begin by making the following assumption associated with the trial basis Φ and the given inner product $(\cdot, \cdot)_\dagger$.

Assumption 6.1. *Consider the parameterized form in (6.3). Then we assume that for each basis function ϕ_i for $1 \leq i \leq N$ the following variational formulation is well-defined*

$$a(w, \psi_i; \mathbf{g}) = (w, \phi_i)_\dagger \quad \forall w \in \mathcal{V}, \quad (6.10)$$

and possesses a unique solution $\psi_i \in \mathcal{V}$.

Note that (6.10) is equivalent to solving the variational problem associated with adjoint PDE operator $\mathcal{L}^*$ where $a(w, \psi; \mathbf{g}) = a^*(\psi, w; \mathbf{g})$. A sufficient condition for Assumption 6.1 to hold true is when $(\cdot, \cdot)_\dagger$ is a symmetric, continuous and coercive bilinear form on $\mathcal{V} \times \mathcal{V}$ [4]. As discussed in Section 6.3, $\{\psi_i(\mathbf{x})\}_{i=1}^N \subset \mathcal{V}$ are the optimal weighting functions and span the finite-dimensional test space $\mathcal{V}^\dagger \subset \mathcal{V}$.

The optimal Petrov-Galerkin formulation is discussed in Section 6.3. The solution to (6.9) is precisely the projected solution in (6.5) characterized by the optimality condition (6.6). This is outlined in the following result.

Lemma 6.1 (Optimality of $\bar{u}$). *Let u be the solution of the infinite-dimensional variational prob-*

lem (6.3) and Assumption 6.1 hold for the basis Φ and inner product $(\cdot, \cdot)_\dagger$. Then the solution $\bar{u}$ to (6.9) is optimal in $\bar{\mathcal{V}}$ in the sense that it satisfies the optimality condition (6.6).

Proof. Starting from (6.9) and setting $w = \psi_i$, we have

$$\begin{aligned}
& a(\bar{u}, \psi_i; \mathbf{g}) - (f, \psi_i) - (\eta, \psi_i)_{\Gamma_\eta} = 0 \quad \forall 1 \leq i \leq N \\
\implies & (\bar{u}, \phi_i)_\dagger - (f, \psi_i) - (\eta, \psi_i)_{\Gamma_\eta} = 0 \quad \forall 1 \leq i \leq N \quad (\text{from (6.10)}) \\
\implies & (\bar{u}, \phi_i)_\dagger - a(u, \psi_i; \mathbf{g}) = 0 \quad \forall 1 \leq i \leq N \quad (\text{from (6.3)}) \\
\implies & (\bar{u}, \phi_i)_\dagger - (u, \phi_i)_\dagger = 0 \quad \forall 1 \leq i \leq N \quad (\text{from (6.10)}) \\
\implies & (\bar{u} - u, \phi_i)_\dagger = 0 \quad \forall 1 \leq i \leq N \\
\implies & (\bar{u} - u, \bar{w})_\dagger = 0 \quad \forall \bar{w} \in \bar{\mathcal{V}} = \text{span}\{\phi_1, \dots, \phi_N\}.
\end{aligned}$$

□

In light of Lemma 6.1, the required finite-dimensional projection $\bar{u}$ can be obtained as the solution of the Petrov-Galerkin formulation (6.9) without solving for the infinite-dimensional solution u . However, solving (6.9) relies on the knowledge of the weighting functions $\{\psi_i\}_{i=1}^N$. This leads to the following challenges:

1. Given N trial basis functions $\{\phi_i\}_{i=1}^N$, we need to recover N weighting functions from (6.10). In the absence of a closed-form expression for $\psi_i(x)$, (6.10) needs to be approximately solved with a high-order numerical solver (finite element method, isogeometric analysis, etc.).
2. Since $\{\psi_i\}_{i=1}^N$ depend on $\mathbf{g}$, a new set of N adjoint problems needs to be solved to recover the weighting functions each time the PDE data $\mathbf{g}$ changes.

These two challenges together mean that the Petrov-Galerkin formulation quickly becomes infeasible as the PDE data varies across a large number of samples. Thus, we propose a mathematically sound deep operator learning approach to resolve these two challenges and make the Petrov-Galerkin formulation feasible.

Chapter 7: Operator Learning

An operator $\mathcal{S} : f \in \mathcal{F} \rightarrow u \in \mathcal{V}$ is a mapping between function spaces $\mathcal{F}$ and $\mathcal{V}$. While a PDE operator such as $\mathcal{L}$ in (6.1) is an example, a more important type of operator related to PDEs is the solution operator. For the solution operator of a PDE, the function space $\mathcal{F}$ represents the space of functions of PDE data and $\mathcal{V}$ is the function space consisting of solutions. Typically, a PDE solution operator is non-trivial or impossible to extract in a closed form. PDE solvers can also be seen as operators that map PDE data to an approximate solution. Generally, if there is an application that requires the PDE to be solved multiple times with varying PDE data, it can quickly become very computationally expensive if each single call to the PDE solver is costly.

In the search for computationally efficient *surrogate models*, interest in deep learning-based methods to learn PDE operators has exploded in recent years. The motivation is that once the PDE operator is learned, the neural network can be deployed and rapidly solve many instances of the PDE as the PDE data is varied.

Neural networks trained to learn an operator are termed *operator networks*. The first operator networks were shallow one-layer networks proposed in [16, 15]. Training an operator network involves using data for a specific PDE in order to approximately learn the PDE operator. Given enough training data, the network should be able to then generalize to unseen PDE data. Such operator networks come in many forms, and form a subset of the broader field of *operator*

learning, which encompasses various techniques for learning operators. In this chapter, we will discuss a few that are the most relevant to this dissertation.

7.1 DeepONet

Deep Operator Networks or *DeepONets* [75] are models that can be trained to learn PDE solution operators. Rather than being a single network, a DeepONet consists of at least two sub-networks, known as the trunk and branch, as shown in Figure 7.1. For simplicity, consider a PDE problem where the only PDE data to input is f . The trunk is a neural network whose output serves as an N -dimensional basis of scalar-valued functions, and the approximate PDE solution will exist in the space spanned by this basis. It can be represented as the following mapping: $\tau : \mathbb{R}^d \rightarrow \mathbb{R}^N$. N is called the latent dimension of the DeepONet.

The branch is a neural network that maps the PDE data to scalar coefficients corresponding to the trunk basis functions. The input is a discretized vector $\hat{\mathbf{F}}$ of the PDE data sampled at a fixed set of sensor nodes $\{\hat{\mathbf{x}}_j\}_{j=1}^{N_s}$. The branch can be represented as the following mapping: $\beta : \mathbb{R}^{N_s} \rightarrow \mathbb{R}^N$. These basis coefficients $\beta[\hat{\mathbf{F}}] = \{\beta_i\}_{i=1}^N$ represent the weightings corresponding to the basis functions. Generally, we write β where it is understood that the input is $\hat{\mathbf{F}}$. Hence, the DeepONet solution can be expressed as

$$u_\theta(\mathbf{x}) = \beta^\top \tau(\mathbf{x}). \quad (7.1)$$

Remark 7.1. *The set of sensor nodes $\{\hat{\mathbf{x}}_j\}_{j=1}^{N_s}$ need not be the same as the points at which the solution is evaluated. At least one extension of DeepONet has even explored varying sensor nodes across training samples [85].*

The idea to separate the input space of the problem such that PDE data is fed into the branch and the spatial data is fed into the trunk is simple yet powerful. This guarantees that the DeepONet solution will be a linear combination of a learned basis (whose functions can be extracted and visualized). In other words, $u_\theta \in \mathcal{V}_\theta = \text{span}\{\tau_1, \dots, \tau_N\}$. Moreover, it means that the PDE data corresponding to a particular instance of a problem can only influence how these basis functions are combined. Although the PDE data corresponding to the problem is evaluated at a fixed set of sensor nodes, in principle, the DeepONet solution can be computed at any $\mathbf{x}$.

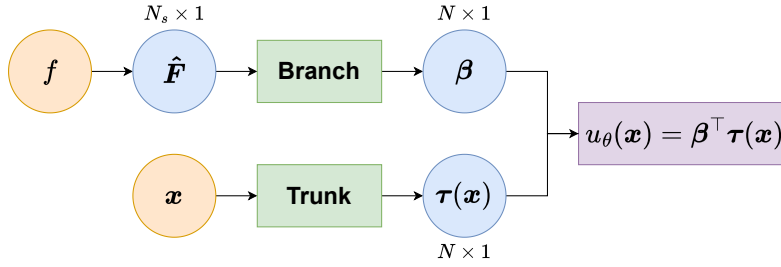


Figure 7.1: Architecture of a basic DeepONet.

The following theorem is a universal approximation theorem for the DeepONet operator, providing an existence theorem for DeepONets to approximate general PDE operators.

Theorem 7.1 ([64]). *Let Ω_X and Ω_Y be the domain of the input and output functions, respectively. Let $\mu \in \mathcal{P}(C(\Omega_X))$ be a probability measure on $C(\Omega_X)$. Let $\mathcal{S} : C(\Omega_X) \rightarrow L^2(\Omega_Y)$ be a Borel measurable mapping, with $\mathcal{S} \in L^2(\mu)$, then for every $\epsilon > 0$, there exists an operator network $\mathcal{S}_\theta$ such that*

$$\|\mathcal{S} - \mathcal{S}_\theta\|_{L^2(\mu)} = \left(\int_{\mathcal{F}} \|\mathcal{S}(u) - \mathcal{S}_\theta(u)\|_{L^2(\Omega_Y)}^2 d\mu(u) \right)^{\frac{1}{2}} < \epsilon.$$

Theorem 7.1 is a universal approximation theorem and represents a rigorous analysis of DeepONet that can be applied in realistic settings. See [64] for the proof and other generalization estimates.

The discussion for DeepONet thus far considers the problem of learning an operator defined on a single function space $\mathcal{F}$. MIONet [56] is an extension of DeepONet that learns multiple-input operators. The structure is similar to DeepONet except there are now multiple branch networks, one for each input function. There is still one trunk network to encode the spatial domain information.

Traditionally, DeepONets are constructed under the assumption that the operator in question can be approximated by a linear combination of basis functions, with the basis and linear coefficients represented by trainable networks. Since its inception, there have been several improvements and extensions to the DeepONet [100, 40, 36, 60, 103, 101, 41, 56, 47, 59].

7.2 VarMiON

Another type of operator network that has recently been proposed is the Variationally Mimetic Operator Network or *VarMiON* [81]. This architecture is intended to mimic the discrete variational form of the PDE and comprises multiple branches and a single trunk.

Specifically, the PDE considered in [81] is

$$\begin{aligned} -\nabla \cdot (\kappa(\mathbf{x}) \nabla u(\mathbf{x})) &= f(\mathbf{x}) & \forall \mathbf{x} \in \Omega, \\ \kappa(\mathbf{x}) \nabla u(\mathbf{x}) \cdot \mathbf{n} &= \eta(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_\eta, \\ u(\mathbf{x}) &= 0 & \forall \mathbf{x} \in \Gamma_D, \end{aligned} \tag{7.2}$$

where $f \in \mathcal{F} \subset L^2(\Omega)$, $\eta \in \mathcal{H} \subset L^2(\Gamma_\eta)$, $\kappa \in L^\infty(\Omega)$, and $\mathbf{n}$ is the unit outward normal on Γ_η .

The corresponding variational form is (6.3) where

$$a(u, w; \kappa, \mathbf{c}) := (\kappa \nabla u, \nabla w). \quad (7.3)$$

The variational form is discretized by projecting the solution space $\mathcal{V}$ and PDE data to a finite-dimensional space $\mathcal{V}^h$ spanned by a finite set of continuous basis functions $\{\phi_i(\mathbf{x})\}_{i=1}^N$

Thus, we have

$$\begin{aligned} v^h(\mathbf{x}) &= v_i \phi_i(\mathbf{x}) = \bar{\mathbf{V}}^\top \Phi(\mathbf{x}), \quad \bar{\mathbf{V}} = (v_1, \dots, v_N)^\top, \quad \Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_N(\mathbf{x}))^\top, \\ f^h(\mathbf{x}) &= \bar{\mathbf{F}}^\top \Phi(\mathbf{x}) \in \mathcal{V}^h, \quad \kappa^h(\mathbf{x}) = \bar{\mathbf{K}}^\top \Phi(\mathbf{x}) \in \mathcal{V}^h, \quad \eta^h(\mathbf{x}) = \bar{\mathbf{N}}^\top \Phi(\mathbf{x})|_{\Gamma_\eta} \in \mathcal{V}_\eta^h \end{aligned}$$

where $\mathcal{V}_\eta^h = \{v|_{\Gamma_\eta} : v \in \mathcal{V}^h\}$. If the approximate solution is expressed as $u^h(\mathbf{x}) = \bar{\mathbf{U}}^\top \Phi(\mathbf{x})$, then with these projections, the discrete variational form is expressed as

$$\Theta(\kappa^h) \bar{\mathbf{U}} = \mathbf{M}_1 \bar{\mathbf{F}} + \mathbf{M}_2 \bar{\mathbf{N}} \quad (7.4)$$

where

$$\Theta_{ij}(\kappa^h) = a(\phi_i, \phi_j; \kappa^h), \quad [\mathbf{M}_1]_{ij} = (\phi_i, \phi_j) \quad [\mathbf{M}_2]_{ij} = (\phi_i, \phi_j)_{\Gamma_\eta} \quad 1 \leq i, j \leq N. \quad (7.5)$$

With the assumption that $\Theta(\kappa^h)$ is invertible for all $\kappa^h \in \mathcal{V}^h$, the discrete solution operator is

$$\begin{aligned} \mathcal{S}^h : \mathcal{V}^h \times \mathcal{V}^h \times \mathcal{V}_\eta^h &\rightarrow \mathcal{V}^h, \\ \mathcal{S}^h(f^h, \kappa^h, \eta^h) &:= u^h(\mathbf{x}) = (\Theta^{-1}(\kappa^h) \mathbf{M}_1 \bar{\mathbf{F}} + \Theta^{-1}(\kappa^h) \mathbf{M}_2 \bar{\mathbf{N}})^\top \Phi(\mathbf{x}). \end{aligned} \quad (7.6)$$

VarMiON is constructed to mimic the structure of $\mathcal{S}^h$. Note that $\mathcal{S}^h$ is linear in $\bar{\mathbf{F}}$ and $\bar{\mathbf{N}}$ but can be non-linear in κ^h . This directly informs the structure of VarMiON.

As with DeepONet, the PDE data is sampled at fixed sensor nodes to generate discrete vectors for the branch inputs. In the formulation, PDE data (f, κ, η) is considered, and thus the following sensor nodes are employed: $\{\hat{\mathbf{x}}_i\}_{i=1}^{N_s}$ for f and κ and $\{\hat{\mathbf{x}}_i^b\}_{i=1}^{N_b}$ for η . Thus, the input vectors are

$$\hat{\mathbf{F}} = (f(\hat{\mathbf{x}}_1), \dots, f(\hat{\mathbf{x}}_{N_s}))^\top, \quad \hat{\mathbf{K}} = (\kappa(\hat{\mathbf{x}}_1), \dots, \kappa(\hat{\mathbf{x}}_{N_s}))^\top, \quad \hat{\mathbf{N}} = (\eta(\hat{\mathbf{x}}_1^b), \dots, \eta(\hat{\mathbf{x}}_{N_b}^b))^\top \quad (7.7)$$

The structure of VarMiON, depicted in Figure 7.2, comprises

- a non-linear branch, which transforms the input $\hat{\mathbf{K}} \in \mathbb{R}^{N_s}$ into a matrix output $\mathbf{D}(\hat{\mathbf{K}}) \in \mathbb{R}^{N \times N}$. Here, N is the latent dimension of the VarMiON. This mimics the non-linearity of $\Theta^{-1}(\kappa^h)$ in (7.6).
- a linear branch, which transforms the input $\hat{\mathbf{F}} \in \mathbb{R}^{N_s}$ to $\mathbf{A}\hat{\mathbf{F}}$, where $\mathbf{A} \in \mathbb{R}^{N \times N_s}$ is a learnable matrix. The output of this branch is acted upon by the matrix $\mathbf{D}$, giving $\beta(\hat{\mathbf{F}}, \hat{\mathbf{K}}) = \mathbf{D}(\hat{\mathbf{K}})\mathbf{A}\hat{\mathbf{F}} \in \mathbb{R}^N$.
- a linear branch, which transforms the input $\hat{\mathbf{N}} \in \mathbb{R}^{N_b}$ to $\tilde{\mathbf{A}}\hat{\mathbf{N}}$, where $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N_b}$ is a learnable matrix. The output of this branch is acted upon by the matrix $\mathbf{D}$, giving $\tilde{\beta}(\hat{\mathbf{N}}, \hat{\mathbf{K}}) = \mathbf{D}(\hat{\mathbf{K}})\tilde{\mathbf{A}}\hat{\mathbf{N}} \in \mathbb{R}^N$.
- a non-linear trunk, which transforms the input $\mathbf{x} \in \mathbb{R}^d$ to the output $\boldsymbol{\tau}(\mathbf{x}) = (\tau_1(\mathbf{x}), \dots, \tau_N(\mathbf{x}))^\top$.

Putting all of this together, the VarMiON operator is thus given by

$$\hat{\mathcal{S}} : \mathbb{R}^{N_s} \times \mathbb{R}^{N_s} \times \mathbb{R}^{N_b} \rightarrow \mathcal{V}^\tau, \quad \hat{\mathcal{S}}(\hat{\mathbf{F}}, \hat{\mathbf{K}}, \hat{\mathbf{N}}) = \left(\beta(\hat{\mathbf{F}}, \hat{\mathbf{K}}) + \tilde{\beta}(\hat{\mathbf{N}}, \hat{\mathbf{K}}) \right)^\top \boldsymbol{\tau}, \quad (7.8)$$

where $\mathcal{V}^\tau = \text{span}\{\tau_1, \dots, \tau_N\}$. Note that this operator is of the same form as the discrete solution operator (7.6).

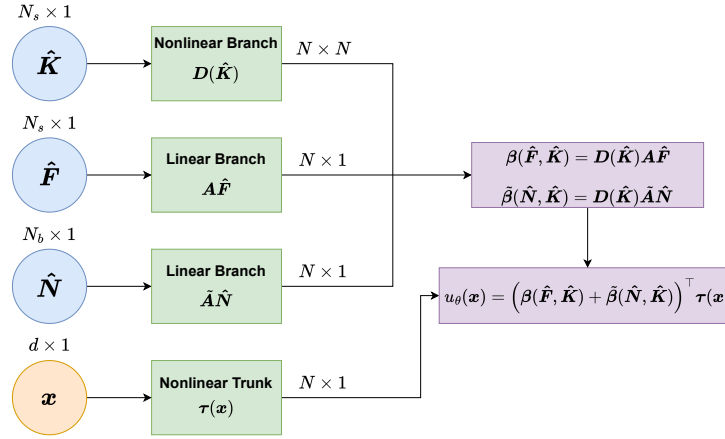


Figure 7.2: Architecture of VarMiON.

VarMiON does have an upper bound on its generalization error for any tuple of PDE data $(f, \kappa, \eta) \in \mathcal{F} \times \mathcal{K} \times \mathcal{H}$ (denoted $\mathcal{E}(f, \kappa, \eta)$) as stated in the following theorem.

Theorem 7.2 ([81]). *Under additional assumptions (see [81]), the following error estimate holds*

$$\mathcal{E}(f, \kappa, \eta) \leq C \left(\epsilon_h + \epsilon_s + \sqrt{\epsilon_t} + \frac{1}{M^{\alpha/2}} + \frac{1}{(M')^{\alpha'/2}} + \frac{1}{N_2^{\gamma/2}} \right), \quad (7.9)$$

where ϵ_h the error bound corresponding to the generation of the training data, ϵ_s is a measure of the maximum distance between any $(f, \kappa, \eta) \in \mathcal{F} \times \mathcal{K} \times \mathcal{H}$ and the set of N_1 samples in the training set, ϵ_t is the VarMiON training error, and $\alpha, \alpha', \gamma, M, M', N_2$ are associated with the quadrature used in discretizing the VarMiON input and output.

While the structure of VarMiON clearly mimics the variational form and has associated error estimates, there is more that can be done. In Chapter 8, we will explore an extension of VarMiON with even more information encoded into its structure.

7.3 FNO

Fourier Neural Operators or *FNOs* [68] represent an alternate framework for deep operator learning. In this methodology, the algorithm is considered in an infinite-dimensional setting and then discretized appropriately. Thus, rather than considering the input, output, and hidden layers of the network to be scalars and vectors, they are instead considered to be functions and vector-valued functions. The mappings between network layers, previously affine transformations, inevitably introduce an additional convolution term.

When discretizing the FNO, most of the architecture remains the same as if a discrete algorithm had been considered from the start. The only difference is now the mappings within the network that contain the convolution are discretized as well. This means that there is now an expensive operation present in each hidden layer which is a discrete sum approximating an integral. This costly component would make the algorithm impractical to use, but FNO evades this difficulty by making use of the fourier transform.

The convolution in the network can be effectively replaced by computing the fourier transforms of the features and kernels, multiplying them together, and finally computing the inverse fourier transform of the product. This is still a costly operation, but the use of the Fast Fourier Transform (FFT) reduces the cost. Thus, FNO becomes a feasible approach.

Performing the FFT generally requires measurements on a rectangular domain with uni-

form meshes. Then, FNO maps the vector of all measurements of each sample function $f(\boldsymbol{x})$ to the corresponding measurement vector of $u(\boldsymbol{x})$. The uniform mesh requirement can be alleviated by following the ideas in [70, 72] to include an analytical mapping from non-uniform mesh grids to uniform mesh grids. Regardless, the resultant FNO model will still use the same mesh grid for the input and output evaluations. This does pose a challenge in determining the approximate FNO solution at a given spatial coordinate, since the PDE data must be evaluated on a mesh that includes that same point in order to be fed as input into the FNO. This is unlike DeepONet and VarMiON, which can be evaluated at any given spatial point (the sensor nodes used to discretize the PDE data are separate from the evaluation points). Nevertheless, FNOs have been shown to provide accurate results and are comparable to operator networks. Rigorous analysis and derivation of error estimates for certain types of FNOs can be found in [61]. There has also been a lot of continuing work in extending the FNO framework [7, 69, 70, 72, 105].

Chapter 8: PG-VarMiON for PDE with fixed parameterization

In this chapter, we introduce the PG-VarMiON framework, a operator learning network that mimics the Petrov-Galerkin variational form. Specifically, the focus here is using PG-VarMiON to solve elliptic PDEs where the PDE parameterization data $\mathbf{g}$ is fixed across samples. Theoretical analysis and an error estimate for this case are developed as well to provide bounds on the performance. Most of the work in this chapter is available as a preprint in [13].

8.1 PG-VarMiON Formulation

The Petrov-Galerkin Variationally Mimetic Operator Network (PG-VarMiON) follows the structure of the Petrov-Galerkin formulation discussed in Section 6.4. The first key step in developing PG-VarMiON is combining (6.10) with (6.9) to obtain an alternate Petrov-Galerkin formulation: find $\bar{u} \in \bar{\mathcal{V}} = \text{span}\{\phi_1, \dots, \phi_N\}$ such that

$$(\bar{u}, \phi_i)_\dagger = (f, \psi_i) + (\eta, \psi_i)_{\Gamma_\eta} \quad \forall 1 \leq i \leq N. \quad (8.1)$$

The important property of this formulation is that explicit knowledge of the underlying PDE structure is not required. Note, however, that this information is encoded in the relationship between ψ_i and ϕ_i by Assumption 6.1. Thus, as long as the trial basis functions and their corre-

sponding optimal weighting functions are known, the specific PDE structure does not enter the framework. Motivated by (8.1), we propose an operator learning framework.

Assuming an explicit form of the trial basis $\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \dots, \phi_N(\mathbf{x})]^\top \in \mathbb{R}^N$, which spans $\bar{\mathcal{V}} \subset \mathcal{V}$, we have two objectives:

1. Given partial information about an $f \in \mathcal{F}$ and $\eta \in \mathcal{H}$, such as the value of f and η at a set of finite nodes, determine an accurate approximation $\hat{u}$ of the Petrov-Galerkin solution $\bar{u}$ solving (8.1).
2. Learn the optimal weighting functions $\{\psi_i\}_{i=1}^N$ in an unsupervised manner.

We begin by introducing a few useful notations that will allow us to express various solution frameworks in a compact form. Let $\{(\mathbf{x}_k, \gamma_k)\}_{k=1}^{N_s}$ be suitable quadrature nodes and weights for a quadrature rule in Ω , and $\{(\mathbf{x}_k^b, \gamma_k^b)\}_{k=1}^{N_b}$ be quadrature nodes and weights for a quadrature rule on boundary segment Γ_η . Using these quadrature rules, we define the following approximations for the L^2 inner products

$$\begin{aligned} (v, w)_h &:= \sum_{k=1}^{N_s} \gamma_k v(\mathbf{x}_k) w(\mathbf{x}_k) \approx (v, w) \quad \forall v, w \in L^2(\Omega), \\ (v, w)_{\Gamma_\eta, h} &:= \sum_{k=1}^{N_b} \gamma_k^b v(\mathbf{x}_k^b) w(\mathbf{x}_k^b) \approx (v, w)_{\Gamma_\eta} \quad \forall v, w \in L^2(\Gamma_\eta). \end{aligned} \tag{8.2}$$

Consider a generic vector of N functionals $\Upsilon(\mathbf{x}) = [\Upsilon_1(\mathbf{x}), \dots, \Upsilon_N(\mathbf{x})]^\top \in \mathbb{R}^N$. We introduce the vector $\ell_\Upsilon(f, \eta) \in \mathbb{R}^N$ to represent the exact evaluations:

$$[\ell_\Upsilon(f, \eta)]_i = (f, \Upsilon_i) + (\eta, \Upsilon_i)_{\Gamma_\eta} \quad 1 \leq i \leq N, \tag{8.3}$$

and the vector $\ell_{\Upsilon,h}(f, \eta) \in \mathbb{R}^N$ to denote the corresponding discretized inner product evaluations:

$$[\ell_{\Upsilon,h}(f, \eta)]_i = (f, \Upsilon_i)_h + (\eta, \Upsilon_i)_{\Gamma_{\eta,h}} \quad 1 \leq i \leq N. \quad (8.4)$$

For any constant matrix $B \in \mathbb{R}^{N \times N}$, the following holds true

$$\ell_{B\Upsilon}(f, \eta) = B\ell_{\Upsilon}(f, \eta), \quad \ell_{B\Upsilon,h}(f, \eta) = B\ell_{\Upsilon,h}(f, \eta). \quad (8.5)$$

For now, we assume that the PDE (6.1) is parameterized by data $\mathbf{g}$, which is given and fixed. Since $\bar{u} \in \bar{\mathcal{V}}$, we can express it as

$$\bar{u}(\mathbf{x}) = \sum_{i=1}^N \bar{u}_i \phi_i(\mathbf{x}) = \bar{\mathbf{u}}^\top \Phi(\mathbf{x}), \quad (8.6)$$

where $\bar{\mathbf{u}} = [\bar{u}_1(\mathbf{x}), \dots, \bar{u}_N(\mathbf{x})]^\top \in \mathbb{R}^N$ is the coefficient vector. Substituting this expansion into (8.1) leads to the following equivalent linear system of equations for the coefficients of the optimal Petrov-Galerkin solution

$$M\bar{\mathbf{u}} = \ell_{\Psi}(f, \eta). \quad (8.7)$$

Our goal is to construct an operator network which mimics the linear system (8.7). This requires the following components:

- We compute the mass matrix M in (8.7) and its inverse M^{-1} by using the known trial basis $\Phi(\mathbf{x})$. These matrices can be computed exactly when the inner products $(\phi_i, \phi_j)_{\dagger}$ have a closed-form expression, or using a very highly accurate quadrature rule. Note that these matrices only need to be computed once (offline).

- We choose the quadrature nodes $\{\mathbf{x}_k\}_{k=1}^{N_s} \subset \Omega$ as *sensor nodes* on which the source function f is sampled to create the vector $\mathbf{F} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_{N_s})]^\top \in \mathbb{R}^{N_s}$.
- We choose the boundary quadrature nodes $\{\mathbf{x}_k^b\}_{k=1}^{N_b} \subset \Gamma_\eta$ as *boundary sensor nodes* on which the boundary flux η is sampled to create the vector $\mathbf{N} = [\eta(\mathbf{x}_1^b), \dots, \eta(\mathbf{x}_{N_b}^b)]^\top \in \mathbb{R}^{N_b}$.
- We consider a network $\mathcal{N}(\cdot; \boldsymbol{\theta}) : \Omega \rightarrow \mathbb{R}^N$ with learnable parameters (weights and biases) $\boldsymbol{\theta}$. We assume that when transformed by the mass matrix, this network approximates the optimal Petrov-Galerkin weighting functions,

$$M\mathcal{N}(\mathbf{x}; \boldsymbol{\theta}) =: \widehat{\Psi}(\mathbf{x}; \boldsymbol{\theta}) \approx \Psi(\mathbf{x}) \quad (8.8)$$

with $\Psi(\mathbf{x}) = [\psi_1(\mathbf{x}), \dots, \psi_N(\mathbf{x})]^\top \in \mathbb{R}^N$ and $\widehat{\Psi}(\mathbf{x}; \boldsymbol{\theta}) = [\widehat{\psi}_1(\mathbf{x}; \boldsymbol{\theta}), \dots, \widehat{\psi}_N(\mathbf{x}; \boldsymbol{\theta})]^\top \in \mathbb{R}^N$.

- Using the above network and (8.2), we consider the approximations

$$\begin{aligned} (\mathcal{N}_i, f) &\approx (\mathcal{N}_i, f)_h = \sum_{k=1}^{N_s} \gamma_k \mathcal{N}_i(\mathbf{x}_k; \boldsymbol{\theta}) f(\mathbf{x}_k) \quad \forall 1 \leq i \leq N, \\ (\mathcal{N}_i, \eta)_{\Gamma_\eta} &\approx (\mathcal{N}_i, \eta)_{\Gamma_\eta, h} = \sum_{k=1}^{N_b} \gamma_k^b \mathcal{N}_i(\mathbf{x}_k^b; \boldsymbol{\theta}) \eta(\mathbf{x}_k^b) \quad \forall 1 \leq i \leq N. \end{aligned}$$

With $\mathbf{Q} = \text{diag}(\gamma_1, \dots, \gamma_{N_s})$ and $\mathbf{Q}^b = \text{diag}(\gamma_1^b, \dots, \gamma_{N_b}^b)$, we define the vector $\boldsymbol{\beta} \in \mathbb{R}^N$

as

$$\boldsymbol{\beta} = \ell_{\mathcal{N},h}(f, \eta) = \mathbf{A}\mathbf{Q}\mathbf{F} + \mathbf{A}^b\mathbf{Q}^b\mathbf{N} \quad (8.9)$$

$$\text{where} \quad A_{ij} = \mathcal{N}_i(\mathbf{x}_j; \boldsymbol{\theta}) \quad \forall 1 \leq i \leq N, 1 \leq j \leq N_s$$

$$A_{ij}^b = \mathcal{N}_i(\mathbf{x}_j^b; \boldsymbol{\theta}) \quad \forall 1 \leq i \leq N, 1 \leq j \leq N_b$$

Using the above components, the approximate solution given by the operator network is expressed as

$$\hat{u}(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\beta}^\top \boldsymbol{\Phi}(\mathbf{x}). \quad (8.10)$$

The schematic of PG-VarMiON is shown in Figure 8.1. Note that while the mass matrix $\mathbf{M}$ is not explicitly used in PG-VarMiON, it is needed to recover $\widehat{\boldsymbol{\Psi}}$ from $\mathcal{N}$.

We remark here that the PG-VarMiON solution mimics the formulation in (8.7). To see this, note that by using (8.9) with the notation in (8.8) and the property (8.5) we have

$$\mathbf{M}\boldsymbol{\beta} = \mathbf{M}\ell_{\mathcal{N},h}(f, \eta) = \ell_{\mathbf{M}\mathcal{N},h}(f, \eta) = \ell_{\widehat{\boldsymbol{\Psi}},h}(f, \eta) \approx \ell_{\boldsymbol{\Psi}}(f, \eta) \quad (8.11)$$

which implies $\boldsymbol{\beta} \approx \bar{\mathbf{u}}$. We will make these approximation more precise in Section 8.3. Note that if $\boldsymbol{\Phi}$ is an orthonormal basis with respect to $\|\cdot\|_+$, then $\mathbf{M} = \mathbf{I}$ and $\mathcal{N} = \widehat{\boldsymbol{\Psi}}$.

8.2 Training

PG-VarMiON, like other operator networks discussed in Chapter 7, is trained in a supervised manner. Thus, a labeled training dataset is required, which is generated according to the

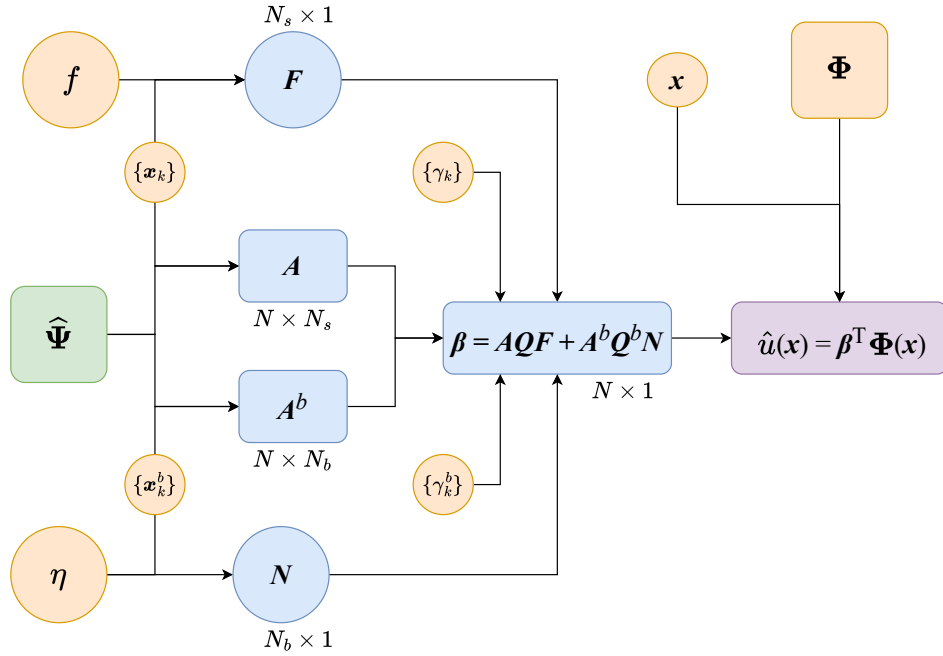


Figure 8.1: Schematic of the PG-VarMiON algorithm for fixed PDE parameterization data $\mathbf{g}$ where the only trainable component is the block in green.

following procedure:

1. Select a suitable $\mathbf{g} \in \mathcal{G}$. This will remain fixed.
2. Choose $\{(f^{(j)}, \eta^{(j)})\}_{j=1}^{N_d} \subset \mathcal{F} \times \mathcal{H}$ sampled independently based on a suitable probability measure on $\mathcal{F} \times \mathcal{H}$.
3. For each $1 \leq j \leq N_d$ find the solution $u^{(j)} \in \mathcal{V}$ satisfying (6.3) with the PDE data $(f^{(j)}, \eta^{(j)}, \mathbf{g})$. In the absence of a closed-form expression, the solution can be recovered using a high-order accurate numerical solver.
4. Generate the input vectors $\mathbf{F}^{(j)} \in \mathbb{R}^{N_s}$, $\mathbf{N}^{(j)} \in \mathbb{R}^{N_b}$ by evaluating each $f^{(j)}$ and $\eta^{(j)}$ at the sensor nodes $\{\mathbf{x}_i\}_{i=1}^{N_s}$ and $\{\mathbf{x}_i^b\}_{i=1}^{N_b}$, respectively.

5. Select a set of *output nodes* $\{\widehat{\mathbf{x}}_l\}_{l=1}^{N_o}$. For each $1 \leq j \leq N_d$, sample the exact/reference solution at these nodes as $u_l^{(j)} = u^{(j)}(\widehat{\mathbf{x}}_l)$.
6. Collect all the inputs and output labels to form the training set of N_d tuples

$$\mathbb{S} = \left\{ \left(\mathbf{F}^{(j)}, \mathbf{N}^{(j)}, \{\widehat{\mathbf{x}}_l\}_{l=1}^{N_o}, \{u_l^{(j)}\}_{l=1}^{N_o} \right) : 1 \leq j \leq N_d \right\}. \quad (8.12)$$

The loss/objective function is based on the optimal norm $\|\cdot\|_{\dagger}$ in (6.5). We treat the output nodes used in the data generation algorithm as quadrature nodes in Ω with associated quadrature weights $\{\hat{\gamma}_l\}_{l=1}^{N_o}$. We then denote the discrete optimal norm by $\|\cdot\|_{\dagger,h}$. For example, if $\|\cdot\|_{\dagger} = \|\cdot\|_{L^2(\Omega)}$, then

$$\|u\|_{\dagger,h}^2 = \sum_{l=1}^{N_o} \hat{\gamma}_l u(\widehat{\mathbf{x}}_l)^2 \approx \|u\|_{\dagger}^2. \quad (8.13)$$

We denote the PG-VarMiON solution corresponding to $(f^{(j)}, \eta^{(j)})$ as $\hat{u}^{(j)}(\mathbf{x}; \boldsymbol{\theta})$. Then the operator network is trained by solving the following optimization problem

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg \min_{\boldsymbol{\theta}} \Pi_h(\boldsymbol{\theta}) \\ \text{where } \Pi_h(\boldsymbol{\theta}) &= \frac{1}{N_d} \sum_{j=1}^{N_d} \|u^{(j)} - \hat{u}^{(j)}(\cdot; \boldsymbol{\theta})\|_{\dagger,h}^2. \end{aligned} \quad (8.14)$$

8.3 Theoretical Analysis

This section can be skipped if the reader wants to directly see the numerical results in Section 8.4.

To make the dependence on f, η explicit, let us denote by $\bar{u}(\mathbf{x}; f, \eta)$ the optimal Petrov-

Galerkin solution $\bar{u} \in \bar{\mathcal{V}}$ corresponding to the norm $\|\cdot\|_{\dagger}$ for a given $f \in \mathcal{F}$ and $\eta \in \mathcal{H}$ and the pre-selected trial basis $\Phi(\mathbf{x})$ spanning $\bar{\mathcal{V}}$. We denote the corresponding PG-VarMiON solution as $\hat{u}(\mathbf{x}; \boldsymbol{\theta}, f, \eta)$. We recall (see (8.6), (8.7), (8.10), (8.11)) that these solutions can be expressed in compact form as

$$\begin{aligned}\bar{u}(\mathbf{x}; f, \eta) &= (\mathbf{M}^{-1} \boldsymbol{\ell}_{\Psi}(f, \eta))^{\top} \Phi(\mathbf{x}), \\ \hat{u}(\mathbf{x}; \boldsymbol{\theta}, f, \eta) &= \boldsymbol{\beta}^{\top} \Phi(\mathbf{x}) = (\mathbf{M}^{-1} \boldsymbol{\ell}_{\hat{\Psi}, h}(f, \eta))^{\top} \Phi(\mathbf{x}).\end{aligned}\tag{8.15}$$

We define the error between the exact (or reference) solution $u(\mathbf{x}; f, \eta)$ and the PG-VarMiON solution as $\mathcal{E}(\boldsymbol{\theta}, f, \eta) := \|u(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}$. Similarly, we define the error due to the finite-dimensional projection of the exact solution into $\bar{\mathcal{V}}$ as $\mathcal{E}_{\Phi}(f, \eta) := \|u(\cdot; f, \eta) - \bar{u}(\cdot; f, \eta)\|_{\dagger}$, and the error between the projected solution and the PG-VarMiON solution as $\mathcal{E}_{\Psi}(\boldsymbol{\theta}, f, \eta) := \|\bar{u}(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}$. These three errors are related by a decomposition rule as described in the following lemma.

Lemma 8.1. *Given any $(f, \eta) \in \mathcal{F} \times \mathcal{H}$, the PG-VarMiON error can be decomposed as*

$$\mathcal{E}^2(\boldsymbol{\theta}, f, \eta) = \mathcal{E}_{\Phi}^2(f, \eta) + \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta).\tag{8.16}$$

Proof. We can express

$$\begin{aligned}
\mathcal{E}^2(\boldsymbol{\theta}, f, \eta) &= \|u(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}^2 \\
&= \|u(\cdot; f, \eta) - \bar{u}(\cdot; f, \eta) + \bar{u}(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}^2 \\
&= \|u(\cdot; f, \eta) - \bar{u}(\cdot; f, \eta)\|_{\dagger}^2 + \|\bar{u}(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}^2 \\
&\quad + 2\left(u(\cdot; f, \eta) - \bar{u}(\cdot; f, \eta), \bar{u}(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\right)_{\dagger} \\
&= \mathcal{E}_{\Phi}^2(f, \eta) + \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta)
\end{aligned}$$

where the final expression is recovered using (6.6) and noting that $\bar{u}, \hat{u} \in \bar{\mathcal{V}}$. □

Note that the above result implies that the error $\mathcal{E}(\boldsymbol{\theta}, f, \eta)$ is bounded from below by the projection error $\mathcal{E}_{\Phi}(f, \eta)$. In other words, PG-VarMiON can never do better than the orthogonal projection of the solution into $\bar{\mathcal{V}}$.

Remark 8.1. *In the case of varying PDE parameterization data $\mathbf{g}$ (to be considered in Chapter 9), the errors will become $\mathcal{E}(\boldsymbol{\theta}, f, \eta, \mathbf{g})$, $\mathcal{E}_{\Phi}(f, \eta, \mathbf{g})$, and $\mathcal{E}_{\Psi}(\boldsymbol{\theta}, f, \eta, \mathbf{g})$.*

We also require the following result about the action of the mass matrix $\mathbf{M}$.

Lemma 8.2. *Let $\mathbf{v} \in \mathbb{R}^N$. Then the following holds for the mass matrix $\mathbf{M}$ defined in (8.7)*

$$\|\mathbf{v}^{\top} \mathbf{M}^{-1} \Phi\|_{\dagger}^2 = (\mathbf{v}^{\top} \mathbf{M}^{-1} \Phi, \mathbf{v}^{\top} \mathbf{M}^{-1} \Phi)_{\dagger} = \mathbf{v}^{\top} \mathbf{M}^{-1} \mathbf{v}.$$

Proof. We have,

$$\begin{aligned}
\|\mathbf{v}^\top \mathbf{M}^{-1} \Phi\|_{\dagger}^2 &= (\mathbf{v}^\top \mathbf{M}^{-1} \Phi, \mathbf{v}^\top \mathbf{M}^{-1} \Phi)_{\dagger} \\
&= \left(\sum_{i,j=1}^N v_i M_{ij}^{-1} \phi_j, \sum_{k,l=1}^N v_k M_{kl}^{-1} \phi_l \right)_{\dagger} \\
&= \sum_{i,j=1}^N \sum_{k,l=1}^N v_i M_{ij}^{-1} \underbrace{(\phi_j, \phi_l)_{\dagger}}_{M_{jl}} M_{kl}^{-1} v_k \\
&= \sum_{i,k=1}^N v_i \left[\sum_{j,l=1}^N M_{ij}^{-1} M_{jl} M_{kl}^{-1} \right] v_k \\
&= \sum_{i,k=1}^N v_i [\mathbf{M}^{-1} \mathbf{M} \mathbf{M}^{-T}]_{ik} v_k \\
&= \sum_{i,k=1}^N v_i M_{ik}^{-T} v_k = \mathbf{v}^\top \mathbf{M}^{-1} \mathbf{v}
\end{aligned}$$

where we used the symmetry of $\mathbf{M}^{-1}$ to get the final expression. $\square$

As a consequence of Lemma 8.2, we can obtain the following bounds on $\mathcal{E}_{\Psi}(\boldsymbol{\theta}, f, \eta)$.

Lemma 8.3. *Let $\lambda_{\min}$ and $\lambda_{\max}$ be respectively the smallest and largest (positive) eigenvalues of the symmetric positive definite mass matrix $\mathbf{M}$. Given any $(f, \eta) \in \mathcal{F} \times \mathcal{H}$, the following bounds hold*

$$\frac{\|\ell_{\Psi}(f, \eta) - \ell_{\hat{\Psi},h}(f, \eta)\|_2^2}{\lambda_{\max}} \leq \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta) \leq \frac{\|\ell_{\Psi}(f, \eta) - \ell_{\hat{\Psi},h}(f, \eta)\|_2^2}{\lambda_{\min}} \quad (8.17)$$

where $\|\cdot\|_2$ is the usual Euclidean norm.

Proof. Using (8.15) and Lemma 8.2 we have

$$\begin{aligned}
\mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta) &= \|\bar{u}(\cdot; f, \eta) - \hat{u}(\cdot; \boldsymbol{\theta}, f, \eta)\|_{\dagger}^2 \\
&= \left\| \left(\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta) \right)^{\top} \mathbf{M}^{-1} \boldsymbol{\Phi}(\cdot) \right\|_{\dagger}^2 \\
&= \left(\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta) \right)^{\top} \mathbf{M}^{-1} \left(\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta) \right).
\end{aligned}$$

Since $\mathbf{M}$ is symmetric positive definite with $\lambda_{\min}$ and $\lambda_{\max}$ being respectively the smallest and largest (positive) eigenvalues, $\mathbf{M}^{-1}$ is also symmetric positive definite with the smallest and largest eigenvalues $1/\lambda_{\max}$ and $1/\lambda_{\min}$, respectively. Using the Rayleigh quotient for $\mathbf{M}^{-1}$ gives us

$$\frac{\|\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta)\|_2^2}{\lambda_{\max}} \leq \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta) \leq \frac{\|\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta)\|_2^2}{\lambda_{\min}}. \quad (8.18)$$

□

Now, since $\boldsymbol{\ell}_{\widehat{\Psi},h}(f, \eta)$ involves approximations of the inner products with suitable quadrature rules, we write the following for any $(f, \eta) \in \mathcal{F} \times \mathcal{H}$

$$(\widehat{\psi}_i, f) = (\widehat{\psi}_i, f)_h + \epsilon_{Q,i}[f], \quad (\widehat{\psi}_i, \eta)_{\Gamma_{\eta}} = (\widehat{\psi}_i, \eta)_{\Gamma_{\eta},h} + \epsilon_{Q,i}^b[\eta] \quad \forall 1 \leq i \leq N \quad (8.19)$$

where $\epsilon_{Q,i}[f]$ and $\epsilon_{Q,i}^b[\eta]$ are the interior and boundary quadrature errors, respectively.

Moving forward, we assume that the N_d PDE data samples used to train the PG-VarMiON are stratified in the following manner

$$\{(f^{(1)}, 0), (f^{(2)}, 0), \dots, (f^{(N_f)}, 0), (0, \eta^{(1)}), (0, \eta^{(2)}), \dots, (0, \eta^{(N_{\eta})})\} \quad (8.20)$$

with $N_d = N_f + N_\eta$ for some suitable choice of N_f, N_η quantifying how much training data is required to control the error. This will be made more precise in Theorem 8.1. We define the total training error (for the stratified data) as

$$\epsilon_T := \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) + \sum_{j=1}^{N_\eta} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}). \quad (8.21)$$

For simplicity, we have assumed an exact evaluation of the (squared) optimal norm $\|\cdot\|_{\dagger}^2$ when defining ϵ_T . If we were to use an approximation of the form (8.13), then additional quadrature error terms would also need to be included in the definition of ϵ_T .

We also define the total quadrature error associated with the training data when evaluating (8.19)

$$\epsilon_Q = \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2 + \sum_{j=1}^{N_\eta} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2. \quad (8.22)$$

We now demonstrate how the error in approximating Ψ with $\widehat{\Psi}$ can be bounded in terms of the total training error (8.21), the total quadrature error (8.22) and a truncation error associated with the training set.

Theorem 8.1. *Let us assume that $\psi_i, \widehat{\psi}_i \in \mathcal{V} \subset L^2(\Omega)$ and $\psi_i|_{\Gamma_\eta}, \widehat{\psi}_i|_{\Gamma_\eta} \in L^2(\Gamma_\eta)$ for all $1 \leq i \leq N$. Let the training data be stratified according to (8.20). Assume that $\{f^{(j)}\}_{j=1}^{N_f}$ and $\{\eta^{(j)}\}_{j=1}^{N_\eta}$ form orthonormal sets in $L^2(\Omega)$ and $L^2(\Gamma_\eta)$, respectively. Let $\lambda_{\max}$ be the largest (positive) eigenvalue of the symmetric positive definite mass matrix $\mathbf{M}$. Then given any $\epsilon > 0$, we can find a large enough training dataset (i.e., large enough integers N_f, N_η) such that the following*

interior and boundary estimates hold true

$$\sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) + 2 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2, \quad (8.23)$$

$$\sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_\eta)}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_\eta} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}) + 2 \sum_{j=1}^{N_\eta} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2, \quad (8.24)$$

where the quadrature errors are as defined in (8.19). Here, ϵ can be seen as a control on the truncation error associated with the training set.

Proof. Using the assumed orthonormality of $\{f^{(j)}\}_{j=1}^{N_f}$ and $\{\eta^{(j)}\}_{j=1}^{N_\eta}$, these sets can be extended to form countable orthonormal bases $\{f^{(j)}\}_{j=1}^\infty$ and $\{\eta^{(j)}\}_{j=1}^\infty$ in $L^2(\Omega)$ and $L^2(\Gamma_\eta)$, respectively.

Using these bases, we can express for each $1 \leq i \leq N$

$$\psi_i - \widehat{\psi}_i = \sum_{j=1}^{\infty} (\psi_i - \widehat{\psi}_i, f^{(j)}) f^{(j)} \quad \text{on } \Omega \quad \text{with } \|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 = \sum_{j=1}^{\infty} (\psi_i - \widehat{\psi}_i, f^{(j)})^2 < \infty, \quad (8.25)$$

$$\psi_i - \widehat{\psi}_i = \sum_{j=1}^{\infty} (\psi_i - \widehat{\psi}_i, \eta^{(j)}) \eta^{(j)} \quad \text{on } \Gamma_\eta \quad \text{with } \|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_\eta)}^2 = \sum_{j=1}^{\infty} (\psi_i - \widehat{\psi}_i, \eta^{(j)})^2 < \infty. \quad (8.26)$$

Thus, given any $\epsilon > 0$, there exist positive integers N_f^c and N_η^c such that

$$\sum_{j=N_f^c+1}^{\infty} (\psi_i - \widehat{\psi}_i, f^{(j)})^2 < \frac{\epsilon}{N}, \quad \sum_{j=N_\eta^c+1}^{\infty} (\psi_i - \widehat{\psi}_i, \eta^{(j)})^2 < \frac{\epsilon}{N} \quad \forall 1 \leq i \leq N. \quad (8.27)$$

We assume that the size of the training data is chosen such that $N_f \geq N_f^c$ and $N_\eta \geq N_\eta^c$. Using

(8.25) and (8.26), we arrive at the truncation error estimates

$$\|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 < \sum_{j=1}^{N_f} (\psi_i - \widehat{\psi}_i, f^{(j)})^2 + \frac{\epsilon}{N} \quad \forall 1 \leq i \leq N, \quad (8.28)$$

$$\|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_\eta)}^2 < \sum_{j=1}^{N_\eta} (\psi_i - \widehat{\psi}_i, \eta^{(j)})^2 + \frac{\epsilon}{N} \quad \forall 1 \leq i \leq N. \quad (8.29)$$

We now estimate the finite summation terms in the above expressions.

Using (8.17) and (8.22), we have

$$\begin{aligned} \lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) &\geq \sum_{j=1}^{N_f} \|\boldsymbol{\ell}_{\Psi}(f^{(j)}, 0) - \boldsymbol{\ell}_{\widehat{\Psi},h}(f^{(j)}, 0)\|_2^2 \\ &= \sum_{j=1}^{N_f} \sum_{i=1}^N \left((\psi_i, f^{(j)}) - (\widehat{\psi}_i, f^{(j)})_h \right)^2 \\ &= \sum_{j=1}^{N_f} \sum_{i=1}^N \left((\psi_i - \widehat{\psi}_i, f^{(j)}) + \epsilon_{Q,i}[f^{(j)}] \right)^2. \end{aligned}$$

Taking a square-root on both sides and using the reverse triangle inequality, we have

$$\begin{aligned} \sqrt{\lambda_{\max}} \left(\sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) \right)^{1/2} &\geq \left(\sum_{j=1}^{N_f} \sum_{i=1}^N \left((\psi_i - \widehat{\psi}_i, f^{(j)}) + \epsilon_{Q,i}[f^{(j)}] \right)^2 \right)^{1/2} \\ &\geq \left(\sum_{j=1}^{N_f} \sum_{i=1}^N \left((\psi_i - \widehat{\psi}_i, f^{(j)}) \right)^2 \right)^{1/2} - \left(\sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2 \right)^{1/2} \end{aligned}$$

which further implies

$$2\lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) + 2 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2 \geq \sum_{j=1}^{N_f} \sum_{i=1}^N \left((\psi_i - \widehat{\psi}_i, f^{(j)}) \right)^2. \quad (8.30)$$

Combining (8.28) with (8.30), we get the required interior estimate

$$\sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0) + 2 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2.$$

Using a similar approach, we can also derive the following estimates on the boundary

$$2\lambda_{\max} \sum_{j=1}^{N_{\eta}} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}) + 2 \sum_{j=1}^{N_{\eta}} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2 \geq \sum_{j=1}^{N_{\eta}} \sum_{i=1}^N \left((\psi_i - \widehat{\psi}_i, \eta^{(j)})_{\Gamma_{\eta}} \right)^2 \quad (8.31)$$

and

$$\sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_{\eta})}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_{\eta}} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}) + 2 \sum_{j=1}^{N_{\eta}} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2.$$

□

Remark 8.2. We can weaken the orthonormality assumption on the training PDE data, by working with orthonormal bases of $\text{Span}(\{f^{(j)}\}_{j=1}^{N_f})$ and $\text{Span}(\{\eta^{(j)}\}_{j=1}^{N_{\eta}})$, assuming that the dimensions of $\text{Span}(\{f^{(j)}\}_{j=1}^{N_f})$ and $\text{Span}(\{\eta^{(j)}\}_{j=1}^{N_{\eta}})$ are large enough to obtain estimates similar to (8.28) and (8.29) up to a scaling constant.

We now state the main result for estimating the generalization error of PG-VarMiON.

Theorem 8.2. Assume Ω is a bounded domain with Lipschitz boundary. Let $f \in \mathcal{F}$, $\eta \in \mathcal{H}$, and $\lambda_{\min}$ and $\lambda_{\max}$ be the smallest and largest (positive) eigenvalues of the symmetric positive definite mass matrix $\mathbf{M}$. Also, assume that the conditions of Theorem 8.1 hold. Then given any $\epsilon > 0$,

we can find a large enough training dataset (in accordance to Theorem 8.1) such that

$$\begin{aligned} \mathcal{E}^2(\boldsymbol{\theta}, f, \eta) \leq \mathcal{E}_{\Phi}^2(f, \eta) + \frac{3}{\lambda_{\min}} & \left(2 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} (\epsilon + \lambda_{\max} \epsilon_T + \epsilon_Q) \right. \\ & \left. + \sum_{i=1}^N |\epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta]|^2 \right) \end{aligned} \quad (8.32)$$

where ϵ_T and ϵ_Q are respectively the total training error (8.21) and the total quadrature error (8.22) associated with the training set.

Proof. Using Lemma 8.1 and Lemma 8.3, we can write

$$\mathcal{E}^2(\boldsymbol{\theta}, f, \eta) = \mathcal{E}_{\Phi}^2(f, \eta) + \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta) \leq \mathcal{E}_{\Phi}^2(f, \eta) + \frac{\|\boldsymbol{\ell}_{\Psi}(f, \eta) - \boldsymbol{\ell}_{\hat{\Psi},h}(f, \eta)\|_2^2}{\lambda_{\min}}. \quad (8.33)$$

Next, using the quadrature approximations (8.19) we can express

$$[\boldsymbol{\ell}_{\hat{\Psi},h}(f, \eta)]_i = (\hat{\psi}_i, f)_h + (\hat{\psi}_i, \eta)_{\Gamma_{\eta,h}} = (\hat{\psi}_i, f) + (\hat{\psi}_i, \eta)_{\Gamma_{\eta}} - \epsilon_{Q,i}[f] - \epsilon_{Q,i}^b[\eta] \quad \forall 1 \leq i \leq N. \quad (8.34)$$

Thus,

$$\begin{aligned}
\|\ell_{\Psi}(f, \eta) - \ell_{\widehat{\Psi},h}(f, \eta)\|_2^2 &= \sum_{i=1}^N \left| [\ell_{\Psi}(f, \eta)]_i - [\ell_{\widehat{\Psi},h}(f, \eta)]_i \right|^2 \\
&= \sum_{i=1}^N \left| (\psi_i - \widehat{\psi}_i, f) + (\psi_i - \widehat{\psi}_i, \eta)_{\Gamma_\eta} + \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2 \\
&\leq 3 \sum_{i=1}^N \left(\left| (\psi_i - \widehat{\psi}_i, f) \right|^2 + \left| (\psi_i - \widehat{\psi}_i, \eta)_{\Gamma_\eta} \right|^2 + \left| \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2 \right) \\
&\leq 3 \sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 \|f\|_{L^2(\Omega)}^2 + 3 \sum_{i=1}^N \|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_\eta)}^2 \|\eta\|_{L^2(\Gamma_\eta)}^2 \\
&\quad + 3 \sum_{i=1}^N \left| \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2 \\
&\leq 3 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} \sum_{i=1}^N \left(\|\psi_i - \widehat{\psi}_i\|_{L^2(\Omega)}^2 + \|\psi_i - \widehat{\psi}_i\|_{L^2(\Gamma_\eta)}^2 \right) \\
&\quad + 3 \sum_{i=1}^N \left| \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2. \tag{8.35}
\end{aligned}$$

If the training set is chosen to be large enough (as per Theorem 8.1), we can use (8.21), (8.22), (8.23) and (8.24) with (8.35) to get

$$\begin{aligned}
\|\ell_{\Psi}(f, \eta) - \ell_{\widehat{\Psi},h}(f, \eta)\|_2^2 &\leq 6 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} (\epsilon + \lambda_{\max} \epsilon_T + \epsilon_Q) \\
&\quad + 3 \sum_{i=1}^N \left| \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2. \tag{8.36}
\end{aligned}$$

Finally, combining (8.33) and (8.36) gives us the required result (8.32). $\square$

Remark 8.3. From the above analysis, the generalization error in approximating the true solution with PG-VarMiON is bounded from below by the projection error $\mathcal{E}_{\Phi}(f, \eta)$. Thus, choosing a good set of trial basis functions Φ can facilitate in lowering the overall approximation error. The approximation properties of finite element spaces in Sobolev norms began with the classic

Bramble-Hilbert lemma [8]. The generalization of the Bramble-Hilbert lemma in isogeometric analysis is presented in [5]. The L^2 and H^1 results for finite elements are summarized in the following estimates for [2] mesh length h and polynomial degree p : If $u \in H^r(\Omega)$, $1 < r \leq p + 1$, the interpolation error $u - \bar{u}$ satisfies:

$$\|u - \bar{u}\|_{L^2}^2 \leq C(h/p)^{2r} |u|_{H^r}^2. \quad (8.37)$$

$$\|u - \bar{u}\|_{H^1}^2 \leq C(h/p)^{2r-2} |u|_{H^r}^2. \quad (8.38)$$

Remark 8.4. While Theorem 8.2 provides an estimate for the generalization error, the final result can obscure that the generalization error is bounded by the discrepancy in Ψ as shown in (8.35). Thus, how well Ψ is approximated by the PG-VarMiON's $\hat{\Psi}$ directly impacts its performance.

8.4 Numerical Results

In this section, we train operator networks to learn the solution operator for the diffusion problem (in 1D) as well as the advection-diffusion problem (in 1D and 2D) assuming purely homogeneous Dirichlet boundary conditions (i.e., $\Gamma_D = \Gamma$) as well as mixed boundary conditions (i.e., $\Gamma = \Gamma_\eta \cup \Gamma_D$). Thus, we are interested in learning the mapping between the source f (and η if applicable) to the solution u . Through these numerical experiments, we aim to demonstrate: (i) the accuracy of PG-VarMiON on unseen data which includes out-of-distribution (OOD) data, and (ii) the ability of the proposed method to learn the optimal weighting functions Ψ in an unsupervised manner. We also compare the results of the PG-VarMiON with those obtained using Fourier Neural Operators (FNOs) and variants of DeepONets. We remark here that FNOs

and DeepONets will only be used to compare the test accuracy, as they do not imitate the Petrov-Galerkin structure to allow the recovery of the optimal weighting functions. In most experiments, we choose the optimal norm to be the L^2 norm, i.e., $\|\cdot\|_{\dagger} = \|\cdot\|_{L^2(\Omega)}$. Further, we always use an orthonormal trial basis Φ which ensures that $M = I$ and $\mathcal{N} = \hat{\Psi}$ (refer to (8.8)).

8.4.1 Training and testing data

In the absence of closed-form expressions of the PDE solutions, high-resolution reference solutions are generated using a Streamline Upwind Petrov-Galerkin (SUPG) solver [35] in Nutils [108]. Depending on the problem, appropriate boundary conditions are selected within Nutils.

1D problems: We take the domain to be $\Omega = [0, 1]$. The training dataset is constructed by choosing f of the form

$$f(\mathbf{x}) = D \sum_{j=1}^{10} a_j \sin(j\pi x + b_j), \quad a_j \sim \mathcal{U}([-2, 2]), \quad b_j \sim \mathcal{U}([-1, 1]) \quad (8.39)$$

where D is a scaling that normalizes f to take values in $[-1, 1]$. The corresponding reference solution is determined with Nutils using 1502 cubic B-splines basis functions. The input vector F is constructed by evaluating f at $N_s = 40$ nodes corresponding to the Gauss-Legendre quadrature in Ω . The solutions are evaluated and saved at $N_o = 200$ Gauss-Legendre nodes in Ω . Note that for the FNO training/testing, we need to evaluate f and u on the same mesh. Thus, we also save a high-resolution input (i.e., f) vector evaluated on the 200 Gauss-Legendre nodes to deal with the FNO evaluations. The training set comprises 4000 independent samples of f .

We construct three different test datasets. The first one, referred to as DATASET 1, comprises in-distribution samples generated by f 's of the form (8.39). The remaining two are

OOD datasets called DATASET 2 and DATASET 3, where f is sampled from Gaussian random fields with length scales 0.1 and 0.05, respectively. The f are once again normalized so that $f(\mathbf{x}) \in [-1, 1]$. Each test dataset is constructed using 2000 independent samples of f . Note that DATASET 3 contains the roughest source functions among the three, while DATASET 2 contains smoother source functions.

2D problem: We take the domain to be $\Omega = [0, 1] \times [0, 1]$. The training dataset is constructed by choosing f to be of the form

$$f(\mathbf{x}) = D \sum_{j=1}^{10} \sum_{k=1}^{10} a_{jk} \sin(j\pi x + b_{jk}) \sin(k\pi y + c_{jk}), \quad a_{jk} \sim \mathcal{U}([-2, 2]), \quad b_{jk}, c_{jk} \sim \mathcal{U}([-1, 1]) \quad (8.40)$$

where as earlier D normalizes f to take values in $[-1, 1]$. The reference data is generated using 52×52 tensorized cubic B-spline basis functions in 2D. We remark here that the computational cost and memory requirements (both for data generation and training the networks) are significantly larger for the 2D problem as compared to the 1D simulations. Thus, we have chosen a lower resolution to generate the data for the 2D setup.

The input vector $\mathbf{F}$ is constructed by evaluating f at $N_s = 40 \times 40$ tensorized nodes corresponding to the Gauss-Legendre quadrature in Ω . The solutions are evaluated and saved at $N_o = 100 \times 100$ tensorized Gauss-Legendre nodes in Ω . The training set comprises 4000 independent samples of f .

As in 1D, we construct three different test datasets. DATASET 1 comprises in-distribution samples generated by f 's of the form (8.40). The remaining two datasets are OOD datasets DATASET 2 and DATASET 3, where f is sampled from two-dimensional Gaussian random fields with length scales 0.1 and 0.05 respectively. The f are normalized so that $f(\mathbf{x}) \in [-1, 1]$.

Each test dataset is constructed using 2000 independent samples of f .

The preceding description exactly determines the data generated for the problem with homogeneous Dirichlet boundary. For problems including Neumann boundary conditions, we must also specify the flux η . For a given Neumann boundary, we need to define a 1D function. For the training set and DATASET 1, we define η on any Neumann boundary to be of the form (8.39). For DATASET 2 and DATASET 3, we select η by sampling one-dimensional Gaussian random fields with length scales 0.1 and 0.05, respectively, which are scaled such that $\eta \in [-0.05, 0.05]$.

For all datasets, the input vector $\mathbf{N}$ is constructed by evaluating each η at 40 nodes corresponding to the one-dimensional Gauss-Legendre quadrature. This is done for each Neumann boundary, and thus, we have $N_b = n_b \times 40$ boundary nodes, where n_b is the number of Neumann boundaries.

8.4.2 Network architectures

We describe the network architecture for the various operator networks considered in the numerical experiments. The hyperparameters corresponding to each operator network are specified when discussing each problem.

Remark 8.5. *In training these operator networks, we use a learning rate (LR) scheduler to decay the LR every K epochs. This is captured in the following expression*

$$LR = \gamma^{\text{floor}(\frac{t}{K})} LR_0$$

where t is the current epoch and LR_0 is the initial LR.

PG-VarMiON: The overall structure of this surrogate model is depicted in Figure 8.1. The

only trainable component is the green block which produces the $\widehat{\Psi}$. We use a simple feedforward multi-layer perceptron (MLP). The activation function σ in the hidden layers of the MLP depends on the problem and is taken to either be ELU or the hat function, which can be written using a combination of three ReLU functions

$$\sigma(z) = \text{ReLU}(z) - \text{ReLU}(2z - 2) + \text{ReLU}(z - 2). \quad (8.41)$$

The hat function has been observed to overcome spectral bias in networks [46] by acting as a high-pass filter and leading to faster training; see also [107] for its use in finite element deep learning applications. To enforce homogeneous Dirichlet boundary conditions, we multiply each component of the network's output by the cut-off function

$$g(\mathbf{x}) = 1 + \frac{e^{px} + e^{-p(x-1)}}{1 - e^p}, \quad (8.42)$$

with a very large value for p . In two-dimensional problems, the network's output is multiplied by the cut-off function for x and y separately if necessary. Notice that this function enforces zero boundary conditions at opposite sides. For the problems we consider, this makes sense since we will never have a situation where, for instance, there is a Dirichlet boundary at $x = 0$ and a Neumann boundary at $x = 1$. Of course, we can modify the cut-off function to handle such a case. Figure 8.2 shows the cut-off function for $p = 100$ and $p = 400$.

L-DeepONet: This is a variant of the DeepONet with a trainable nonlinear trunk $\tau : \mathbb{R}^d \rightarrow \mathbb{R}^q$ but a learnable linear branch, since the PDE solution operators we are learning are linear in f . This is meant to provide a structural advantage to the network. Thus, the architecture consists of

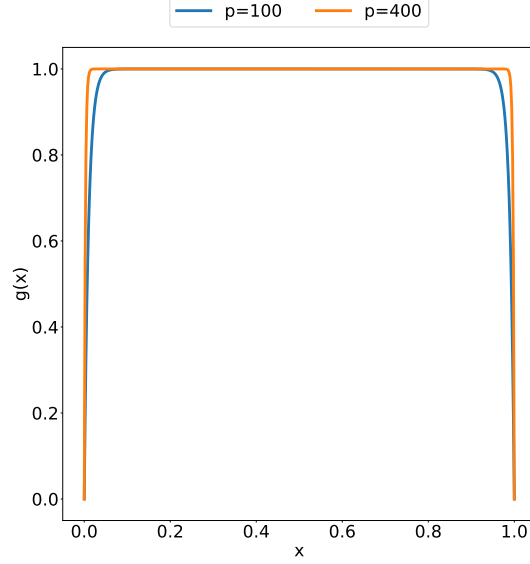


Figure 8.2: Cut-off function used to enforce the homogeneous Dirichlet boundary conditions.

an input vector $\mathbf{F} \in \mathbb{R}^{N_s}$ that is acted on by a learnable matrix $\mathbf{B}$ to produce coefficients for the trunk functions. Thus, the L-DeepONet solution approximation is given by $\hat{u}(\mathbf{x}) = (\mathbf{B}\mathbf{F})^\top \boldsymbol{\tau}(\mathbf{x})$. If Neumann boundary conditions are present, there is a second learnable matrix $\mathbf{B}^b$ that acts on the input vector $\mathbf{N} \in \mathbb{R}^{N_b}$. In this case, the L-DeepONet solution approximation is given by $\hat{u}(\mathbf{x}) = (\mathbf{B}\mathbf{F} + \mathbf{B}^b\mathbf{N})^\top \boldsymbol{\tau}(\mathbf{x})$. Note that the shape of $\mathbf{B}$ is fixed as $q \times N_s$, where q is the latent dimension of the DeepONet, and $\mathbf{B}^b$ has a fixed shape of $q \times N_b$. For our experiments, we choose the $\boldsymbol{\tau}$ to have the same structure as the learnable $\hat{\Psi}$ in PG-VarMiON, i.e., an MLP with the hat activation function (8.41) in the hidden layers and homogeneous Dirichlet boundary conditions enforced using (8.42) at the end of the network. Further, we pick $q = N$ for all experiments, where we recall that N is the number of trial basis functions used in the PG-VarMiON.

BNet: This network is a simplification of PG-VarMiON in that the trial basis Φ is pre-determined and fixed. However, learnable components are replaced by a single trainable matrix $\mathbf{B}$, with the final solution given as $\hat{u}(\mathbf{x}) = (\mathbf{B}\mathbf{F})^\top \Phi(\mathbf{x})$. With Neumann boundary conditions,

a second trainable matrix $\mathbf{B}^b$ is present, with the final solution $\hat{u}(\mathbf{x}) = (\mathbf{B}\mathbf{F} + \mathbf{B}^b\mathbf{N})^\top \boldsymbol{\tau}(\mathbf{x})$. When compared to the PG-VarMiON (see Figure 8.1), we note that $\mathbf{B}$ and $\mathbf{B}^b$ replace $\mathbf{A}\mathbf{Q}$ and $\mathbf{A}^b\mathbf{Q}$, respectively, without having the specialized structure of $\mathbf{A}$ in terms of $\hat{\Psi}$. When compared to L-DeepONet, BNet replaces the learnable trunk $\boldsymbol{\tau}$ with the fixed trial basis Φ . We note that the number of trainable parameters of the BNet is strictly determined by the number of sensor locations (and boundary sensor locations if applicable) and the size of the trial basis. This leads to a significantly small network for 1D problems (see Tables 8.1 and 8.4) compared to the other operator networks but a significantly larger network for the 2D problem (see Table 8.6).

FNO: FNO, described in Section 7.3, is originally designed as a function-to-function mapping via integral operators. In particular, an L -layer NO has the following form:

$$\mathcal{Q} \circ \mathcal{J}_L \circ \cdots \circ \mathcal{J}_1 \circ \mathcal{P}[f](\mathbf{x}) \approx u(\mathbf{x}), \quad (8.43)$$

where $\mathcal{P}$, $\mathcal{Q}$ are shallow-layer neural networks that map a low-dimensional vector into a high-dimensional vector and vice versa. Each intermediate layer, $\mathcal{J}_l$, consists of a local linear transformation operator, an integral (nonlocal) kernel operator, and an activation function σ :

$$\mathcal{J}^l[\mathbf{h}](\mathbf{x}) = \sigma \left(\mathbf{W}^l \mathbf{h}(\mathbf{x}) + \mathbf{b}^l + \mathcal{F}^{-1}[\mathcal{F}[\mathbf{k}(\cdot; \boldsymbol{\theta}^l)] \cdot \mathcal{F}[\mathbf{h}(\cdot)]](\mathbf{x}) \right),$$

where $\mathbf{W}^l \in \mathbb{R}^{d_h \times d_h}$ and $\mathbf{b}^l \in \mathbb{R}^{d_h}$ are learnable tensors at the l -th layer, and $\mathbf{k} \in \mathbb{R}^{d_h \times d_h}$ is a tensor kernel function with parameters $\boldsymbol{\theta}^l$. $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier transform and its inverse, respectively, which are computed using the FFT algorithm to each component of $\mathbf{h}$ separately.

To perform the above FFT calculation, FNO generally requires measurements on a rectangular domain with uniform meshes, then it maps the vector of all measurements of each sample function $f(\boldsymbol{x})$ to the corresponding measurement vector of $u(\boldsymbol{x})$. To alleviate the uniform mesh requirement, here we follow the ideas in [70, 72] and include an analytical mapping from non-uniform mesh grids to uniform mesh grids. As such, the input and output functions are both evaluated on the same Gauss-Legendre quadrature nodes at which the solution is known. However, we point out that the resultant FNO model still takes the measurements on all points in f as input and maps it to the measurements on all points in u .

When Neumann boundary conditions are present, the f and η inputs are stacked to serve as a combined input into the FNO. However, η is only defined on the Neumann boundaries so it must be transformed into a suitable 2D input. To this end, we create a tensor of the same size as the mesh and add the η boundary values to each Neumann boundary locations. Then, we repeat the boundary values for the interior nodes. Since in our Neumann 2D problem, we will consider Neumann flux at $y = 0$ and $y = 1$, the resulting input is a tensor with half η values from $y = 0$ and half from $y = 1$.

Remark 8.6. *Introducing Neumann boundary conditions to a problem changes the number of learnable parameters in BNet, L-DeepONet, and FNO. Due to the architecture of PG-VarMiON, the number of parameters is invariant to the boundary conditions. The construction of PG-VarMiON's $\mathbf{A}$ and $\mathbf{A}^b$ depend on evaluating the same learned optimal weighting functions $\hat{\Psi}$, though the $\mathbf{A}^b$ depends solely on the Neumann boundary evaluations. For FNO, while there is not a specific structure to account for the η input, slightly more parameters are required for the additional input channel (as compared to an identical FNO structure lacking η input).*

Remark 8.7. *The number of parameters in PG-VarMiON additionally does not depend on the number of sensor nodes used to sample f and η . More sensor nodes only means that the neural network approximating the optimal weighting functions is evaluated at more nodes. It has no impact on the structure of the network.*

8.4.3 1D Dirichlet Problems

The first set of problems we test the operator networks on are one-dimensional with homogeneous Dirichlet boundary conditions.

8.4.3.1 Diffusion equation

We consider the pure diffusion equation on $\Omega = [0, 1]$ by setting $c = 0$ in (6.2) and taking $\kappa = 0.01$. We choose the Petrov-Galerkin trial basis of size $N = 10$ as $\Phi = \{\sqrt{2} \sin(j\pi x)\}_{j=1}^{10}$ for which the expressions of optimal weighting functions can be explicitly computed as

$$\psi_j(\mathbf{x}) = \frac{\sqrt{2}}{j^2 \pi^2 \kappa} \sin(j\pi x), \quad 1 \leq j \leq 10.$$

Note that Φ are also the (scaled) eigenfunctions for the diffusion problem with homogeneous Dirichlet boundary conditions. We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 8.4.2 with 3 hidden layers of widths [10,20,30] and train with the hyperparameters stated in Table 8.1. The trained network is then used on the three test datasets described in Section 8.4.1. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 8.3, where we also compare with the relative finite-dimensional projection error. Theoretically, if PG-VarMiON learned the exact Ψ , its error would match the projection error. In practice $\hat{\Psi} \approx \Psi$, and thus it is

	PG-VarMiON	L-DeepONet	BNet	FNO
Activation Function	Hat		None	GeLU
Cut-off function (8.42)	Yes		No	
p in (8.42)	100		N/A	
No. of Parameters	1180	1580	400	1154
Optimizer	AdamW			Adam
AdamW β_1	0.5			N/A
AdamW β_2	0.9			N/A
Weight decay	0			10^{-7}
Epochs	1000			
LR Scheduler	Yes			
Initial LR	10^{-3}			10^{-2}
LR Scheduler step	100			
LR Scheduler γ	0.75			0.7
Batch size	200			

Table 8.1: Summary of training settings for all networks in the 1D pure diffusion problem.

hard to beat the projection error (as also explained in Section 8.3). However, as can be observed from the distribution, the mean relative errors with the PG-VarMiON (see numbers in the figure legend) are very near the projection error, even on the OOD datasets. In Figures 8.4, 8.5, and 8.6, we show the f and corresponding PG-VarMiON solutions for 4 samples in DATASETS 1, 2, and 3, respectively. The PG-VarMiON solutions are indistinguishable from the reference solutions.

The PG-VarMiON is designed to implicitly learn the optimal weighting functions. We plot the true Ψ and the PG-VarMiON approximation $\hat{\Psi}$ in Figure 8.7. We can observe that the low frequency modes are captured much better by PG-VarMiON, while the high frequency modes are qualitatively well approximated. Since the PG-VarMiON trained for the current problem is able to learn the structure of Ψ , it performs well on the OOD datasets, as noted in Remark 8.4.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO. The L-DeepONet has a linear branch taking $\mathbf{F} \in \mathbb{R}^{40}$ as input, while the learnable trunk has exactly the same architecture as $\mathcal{N}$ in our PG-VarMiON. The FNO is composed with linear lifting layer, 3 iterative layers, and a shallow MLP for the projection layer. Here, the input and output functions are both evaluated on the same 200 Gauss-Legendre quadrature nodes at which the solution is known. As a result, the FNO is shown significantly more information about f as compared to PG-VarMiON, L-DeepONet, and BNet. See Table 8.1 for additional details on the selected hyperparameters for all of the networks.

Table 8.2 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON yields the best performance on all three datasets. Note that the L-DeepONet performs much worse on DATASET 3 in comparison to the other two methods, indicating its poor generalization to OOD data. Interestingly, BNet performs comparably to PG-VarMiON on DATASET 1 and 2. This could be attributed to the fact that both BNet and PG-VarMiON are supplied with a good pre-determined trial basis for this problem. However, the performance of BNet severely deteriorates on the challenging OOD DATASET 3. This clearly indicates that the BNet has a tendency to overfit the training data distribution, while the specialized structure of the matrix $\mathcal{A}$ in PG-VarMiON based on $\hat{\Psi}$ (as dictated by the Petrov-Galerkin formulation) plays a critical role in ensuring better generalization of the operator network.

We claim that emulating the Petrov-Galerkin structure of the PDE can also significantly reduce the data complexity with PG-VarMiON. To demonstrate this, we train all of the operator networks on datasets of different sizes (characterized by the number of f samples used). The mean relative test errors of these models are shown in Figure 8.8. The PG-VarMiON error (across all test datasets) remains mostly unchanged as the number of training samples is varied, with a



Figure 8.3: 1D diffusion problem: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

Method	No. of Parameters	DATASET 1	DATASET 2	DATASET 3
Projection	-	0.61 (0.63)	0.31 (0.29)	0.99 (1.06)
PG-VarMiON	1180	0.69 (0.67)	0.38 (0.33)	1.19 (1.21)
FNO	1154	1.41 (1.10)	1.06 (0.77)	1.61 (1.34)
L-DeepONet	1580	2.41 (2.16)	1.17 (0.80)	16.72 (21.56)
BNet	400	0.62 (0.64)	0.51 (0.48)	31.91 (33.74)

Table 8.2: 1D diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.

slight deterioration when the training set is constructed using closer to 100 samples of f . The same robustness does not appear to hold for FNO or L-DeepONet, which require many more training samples to lower the test error. While BNet seems robust on the first two (easier datasets), its approximation is very poor on DATASET 3.

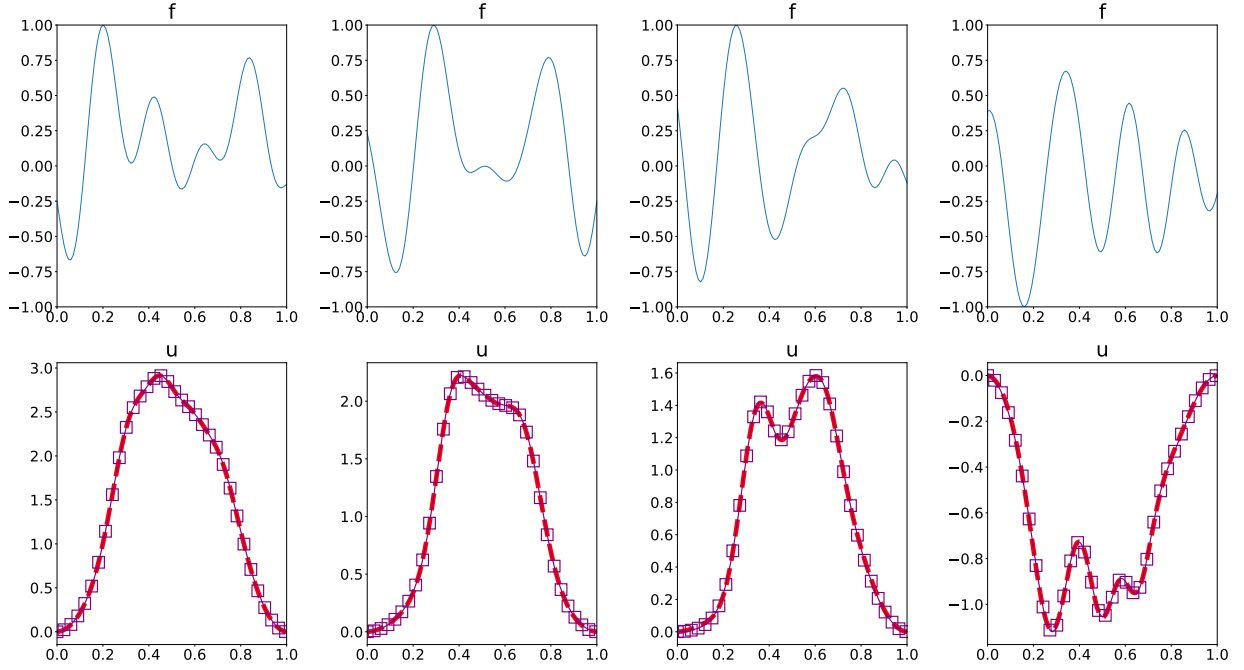


Figure 8.4: 1D diffusion problem: 4 samples from DATASET 1, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

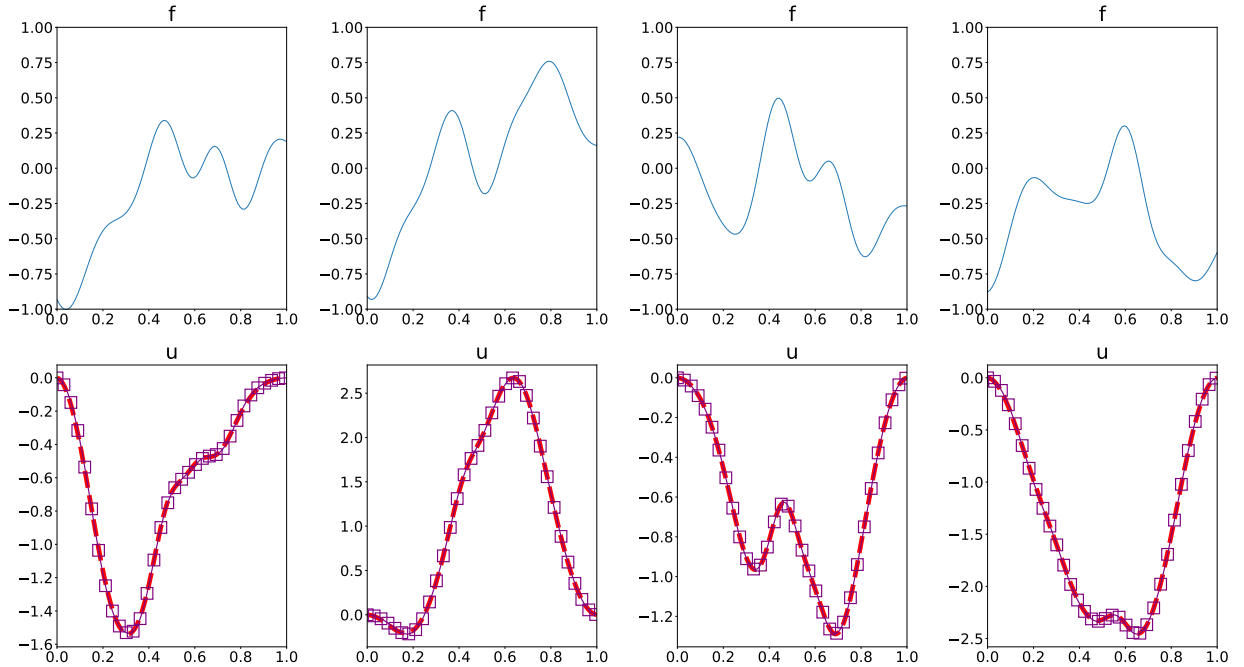


Figure 8.5: 1D diffusion problem: 4 samples from DATASET 2, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

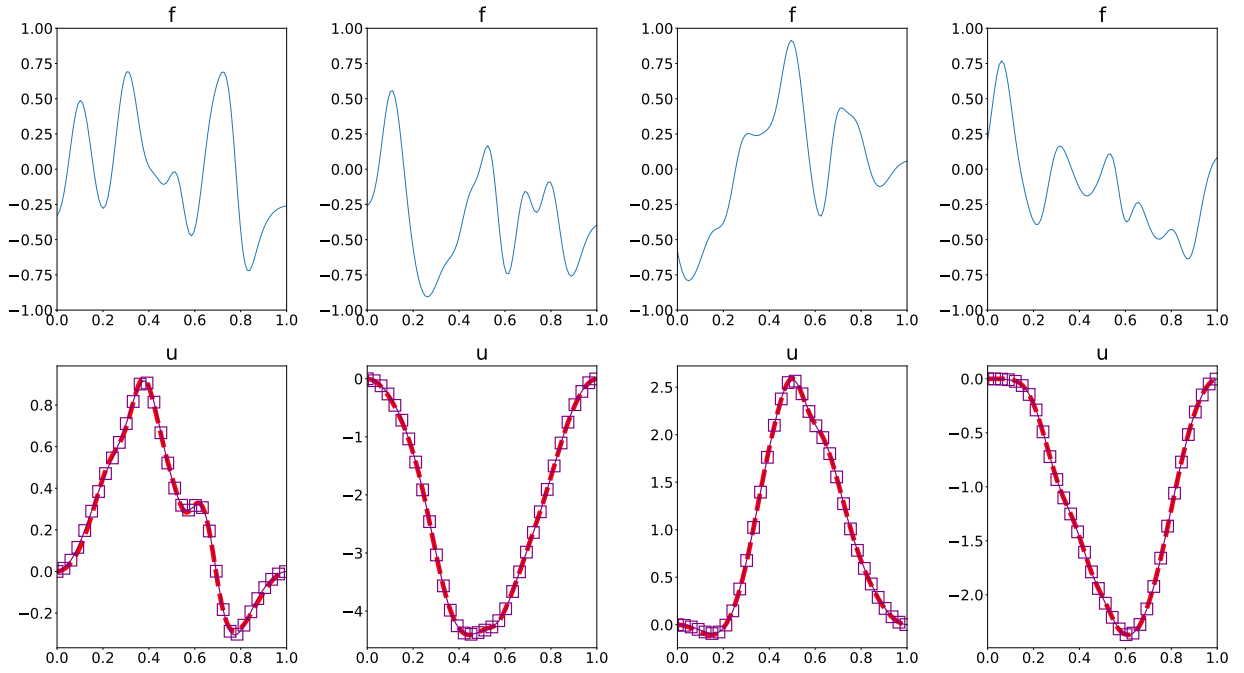


Figure 8.6: 1D diffusion problem: 4 samples from DATASET 3, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

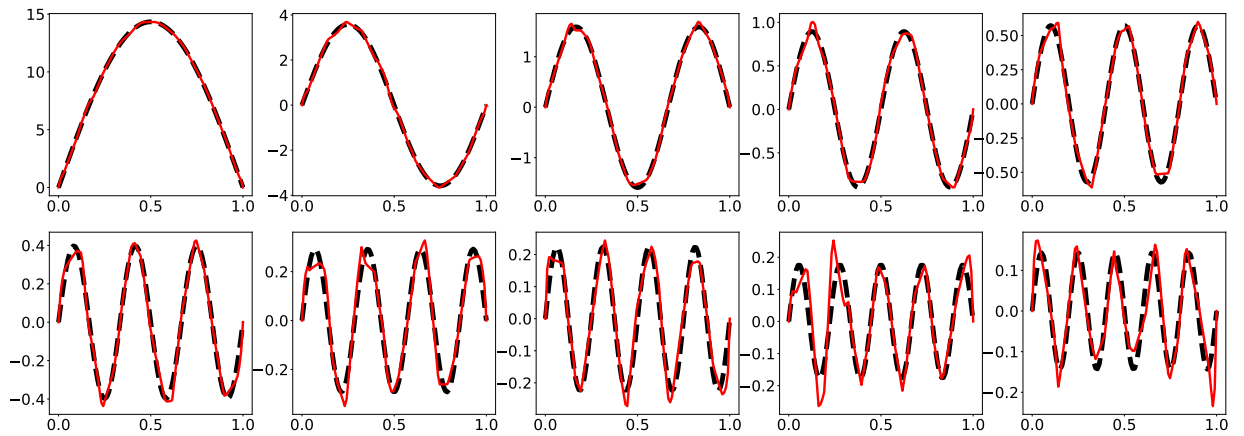


Figure 8.7: 1D diffusion problem: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red).

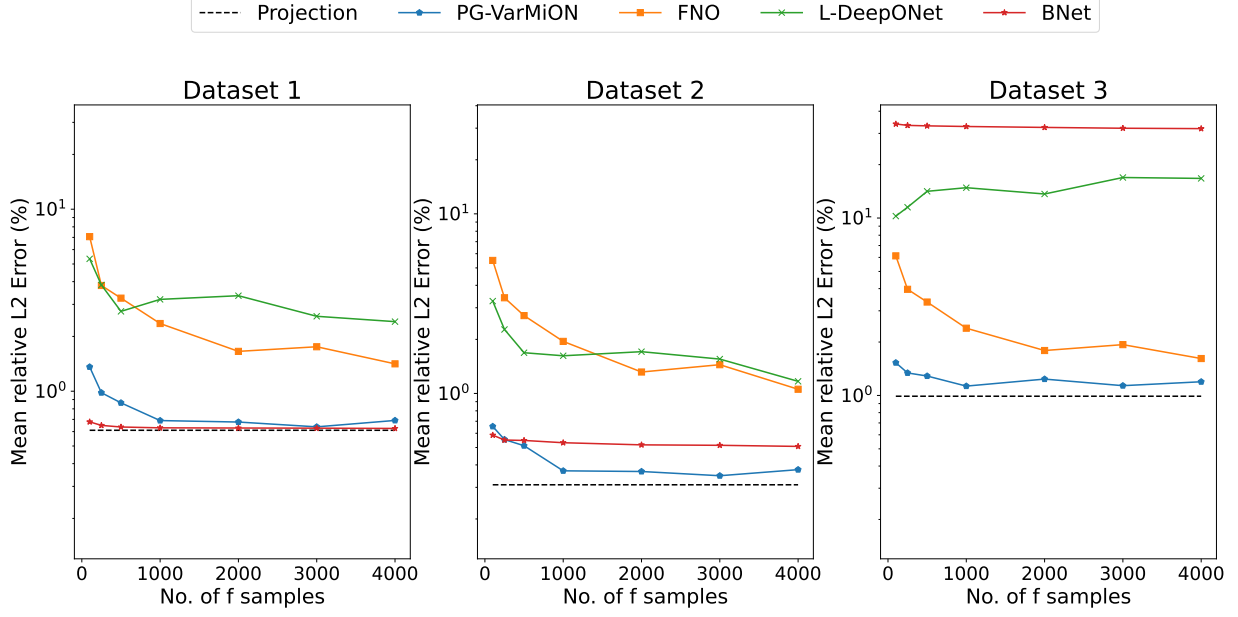


Figure 8.8: 1D diffusion problem: Mean relative test error (in %) with the various operator models as the number of f samples in the training set is varied. The projection error for each dataset is shown in black.

8.4.3.2 Advection-diffusion equation

We consider the advection-dominated problem on $\Omega = [0, 1]$ by setting $c = 0.1$ and $\kappa = 10^{-4}$ in (6.2), which leads to a global Peclet number of 1000. The training and test datasets are constructed by selecting forcing functions f as described in Section 8.4.1. The strong advection here results in a boundary layer at the right boundary, which is difficult to resolve. If we use a trial basis consisting of sine functions as in the pure diffusion problem, we would require more than 100 sine functions of increasing frequency to bring the projection error (which lower bounds the PG-VarMiON error) down to acceptable values, as shown in Table 8.3. It is unlikely that PG-VarMiON will adequately resolve such high frequencies, and in attempting to train it with this basis, the results are generally poor. Instead, we opt for a trial basis with a relatively small dimension, which can lead to good approximations by accounting for boundary layers.

Dimension	DATASET 1	DATASET 2	DATASET 3
10	17.41	16.73	19.05
15	13.36	13.48	14.69
20	11.35	11.49	12.43
30	8.93	9.05	9.79
100	3.67	3.73	4.03

Table 8.3: 1D advection-diffusion problem: Mean relative L^2 projection error (in %) on the test set for the pure sine trial basis with varying dimension size.

To this end, we construct the Petrov-Galerkin trial basis by starting with $\Phi = \{\sqrt{2} \sin(j\pi x)\}_{j=1}^5$ and augmenting the basis with the ten functions given by,

$$\Phi_n(\mathbf{x}) = \sqrt{2} \frac{(\pi \kappa n \sin(\pi n x) - c \cos(\pi n x) + h(x, n))}{\pi n ((\pi n \kappa)^2 + c^2)}, \quad 1 \leq n \leq 10 \quad (8.44)$$

where
$$h(x, n) = \frac{(-1)^{n-1} \left(e^{\frac{-c}{\kappa}(1-x)} - e^{\frac{-c}{\kappa}} \right) c - \left(1 - e^{\frac{-c}{\kappa}(1-x)} \right) c}{e^{\frac{-c}{\kappa}} - 1}.$$

This basis thus has dimension 15. Note that the mass matrix corresponding to this basis is severely ill-conditioned, so we use the Gram-Schmidt process to transform the basis into an orthonormal trial basis. We have observed that this helps maintain the stability of PG-VarMiON. The resulting trial basis can be seen in Figure 8.9.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 8.4.2 with 5 hidden layers of widths [10,20,30,40,30] and train with the hyperparameters stated in Table 8.4. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 8.10, where we also compare with the relative finite-dimensional projection error. Note that the (mean) projection error on DATASET 3 is much larger as compared to DATASET 1 and 2. We observe that the performance of PG-VarMiON is close to the projection, while struggling a bit on DATASET 3. This is not surprising as the current

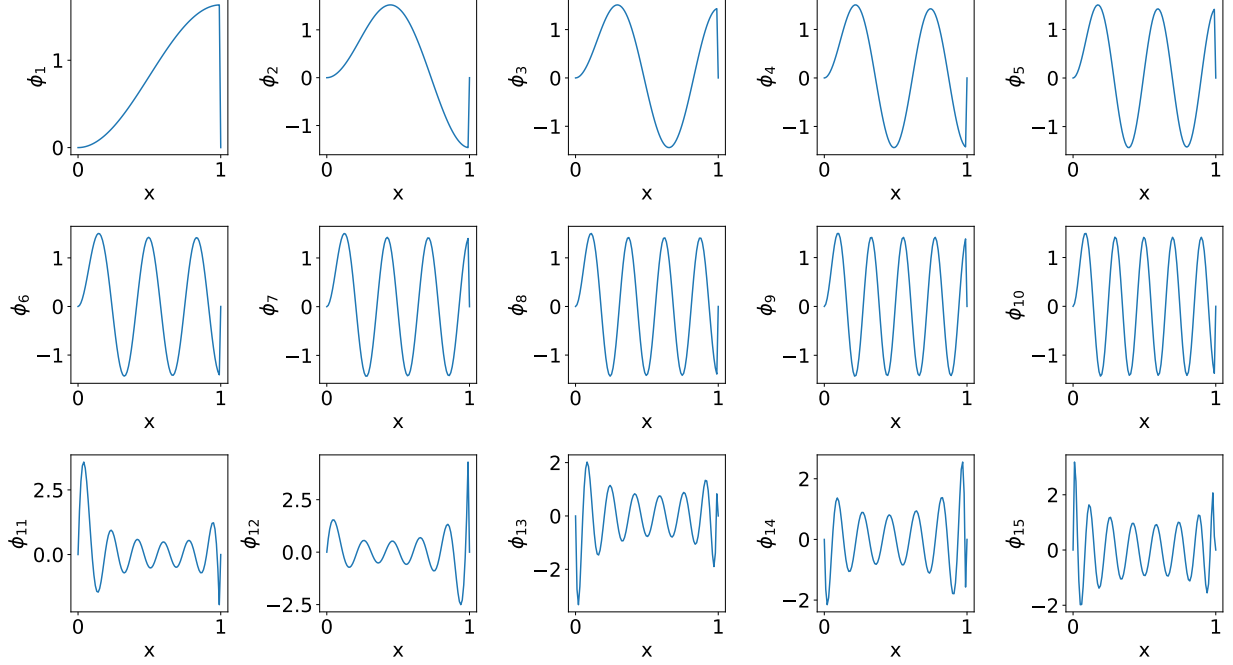


Figure 8.9: The trial basis comprised of fifteen orthonormalized functions used for the 1D advection-diffusion problem. Note the presence of boundary layers near the right boundary.

problem is much more challenging than the diffusion problem, especially with the existence of a boundary layer and the imposition of homogeneous Dirichlet boundary conditions. In Figures 8.11, 8.12, and 8.13, we show the f and corresponding PG-VarMiON solutions for 4 samples in DATASETS 1, 2, and 3, respectively. The PG-VarMiON solutions are indistinguishable from the reference solutions.

We plot the true Ψ and the PG-VarMiON approximation $\hat{\Psi}$ in Figure 8.14. Recall that these basis functions are not included in the training objective, i.e., Ψ are learned implicitly. We can observe that the low frequency modes are captured well by PG-VarMiON, while the high frequency modes are qualitatively well approximated with the exception of two, which are less accurately resolved. Since most of the optimal weighting functions are correctly captured, the PG-VarMiON performs well on the OOD datasets.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet,

	PG-VarMiON	L-DeepONet	BNet	FNO
Activation Function	Hat		None	GeLU
Cut-off function (8.42)	Yes		No	
p in (8.42)	400		N/A	
No. of Parameters	3805	4405	600	3953
Optimizer	AdamW			Adam
AdamW β_1	0.5			N/A
AdamW β_2	0.9			N/A
Weight decay	0			10^{-7}
Epochs	2000			
LR Scheduler	Yes			
Initial LR	10^{-3}			10^{-2}
LR Scheduler step	100			
LR Scheduler γ	0.75			0.9
Batch size	200			

Table 8.4: Summary of training settings for all networks in the 1D advection-diffusion problem with homogeneous Dirichlet boundary conditions.



Figure 8.10: 1D advection-diffusion problem: Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

and FNO. Table 8.5 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON and FNO yield similar performance. L-DeepONet performs poorly on all 3 datasets, but we note that the errors on Dataset 1 and 2 are lower than those of the projection error with a 100 dimensional pure sine basis (refer to Table 8.3). This indicates that L-DeepONet may outperform PG-VarMiON on a poorly chosen trial basis, thus choosing a trial basis for PG-VarMiON with low projection errors is imperative. Similar to the 1D diffusion problem, the BNet performs as well as PG-VarMiON on the first two datasets, but is the worst performer on the challenging DATASET 3.

When we train the operator networks on datasets of different sizes (characterized by the number of f training samples used), we once again observe (see Figure 8.15) that PG-VarMiON error is fairly robust to the number of training samples. Surprisingly, we note that the FNO is also fairly robust on test DATASET 3, although this is not true for the other two datasets.

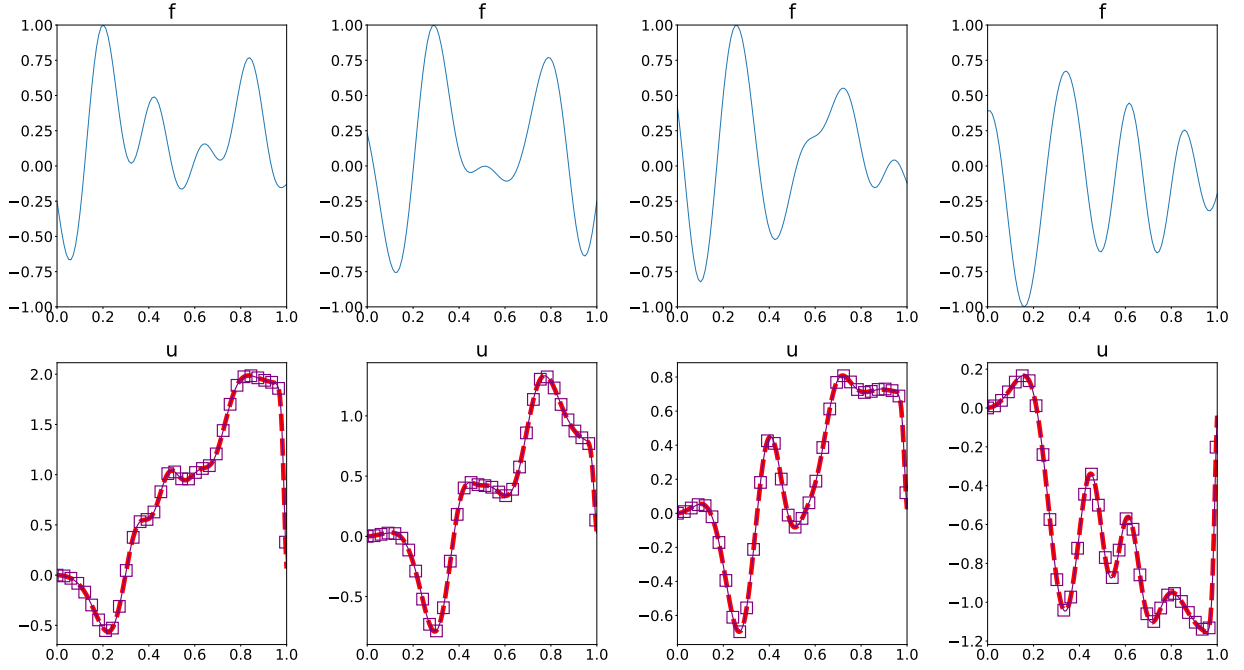


Figure 8.11: 1D advection-diffusion problem: 4 samples from DATASET 1, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

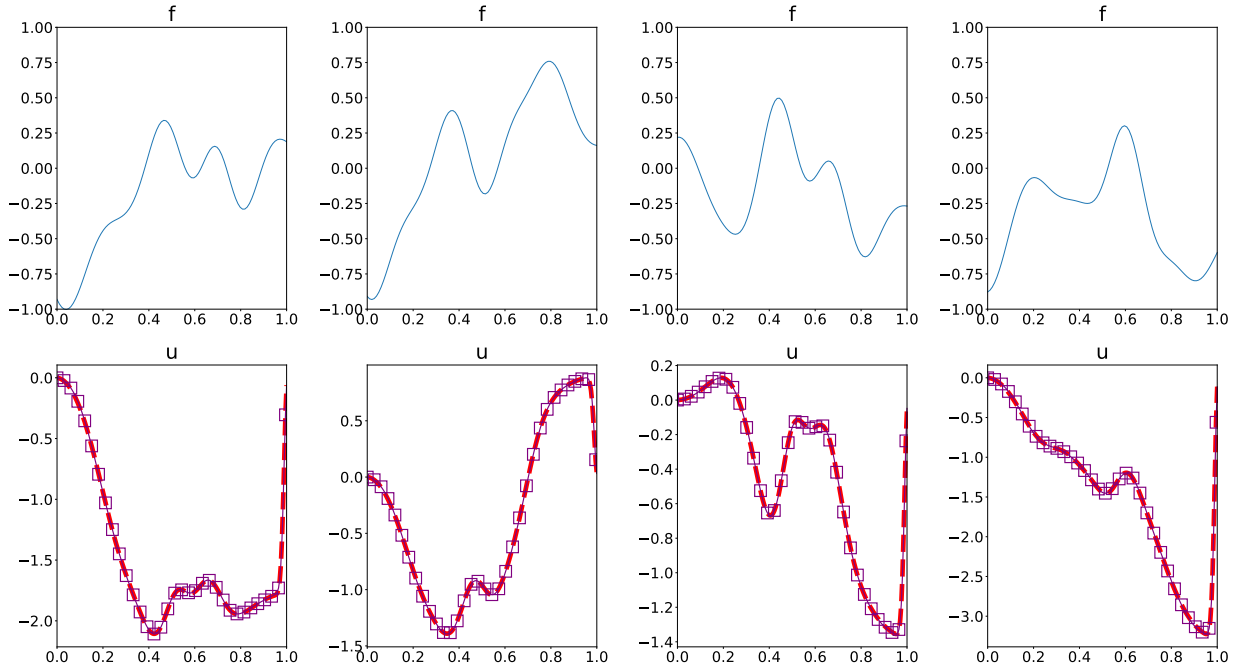


Figure 8.12: 1D advection-diffusion problem: 4 samples from DATASET 2, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

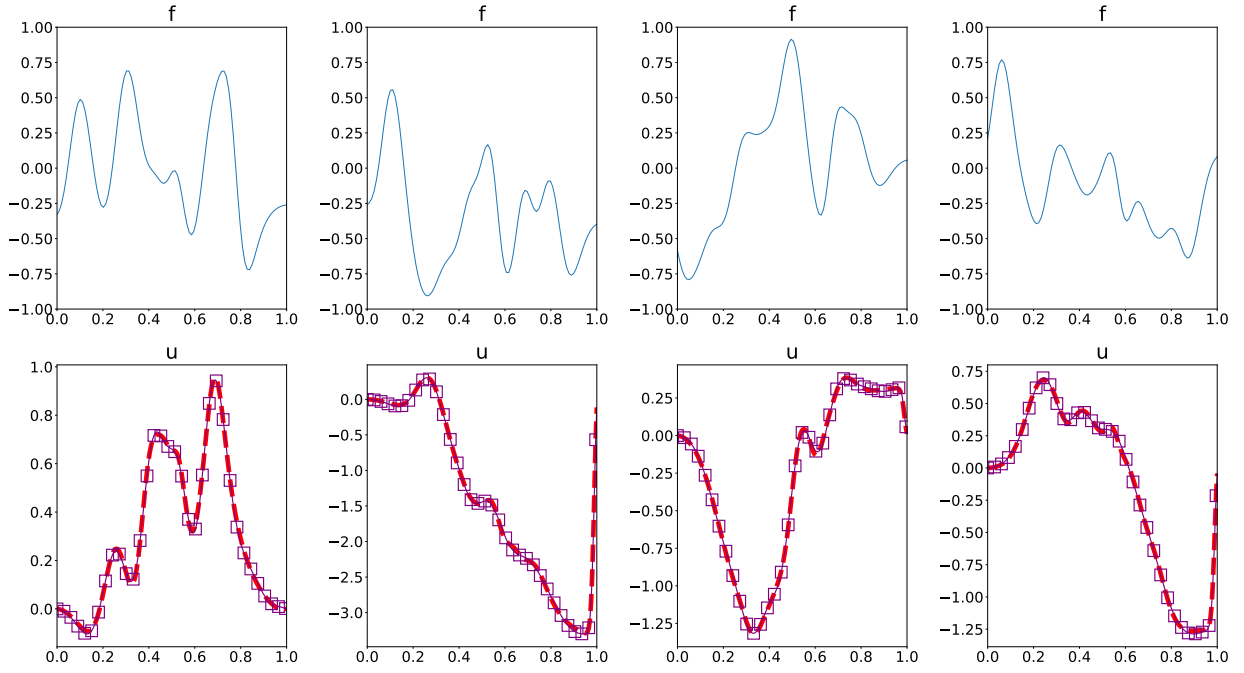


Figure 8.13: 1D advection-diffusion problem: 4 samples from DATASET 3, with the forcing functions f plotted in the first row. The corresponding reference solutions (red dashed) and the PG-VarMiON approximations (purple squares) are shown in the second row.

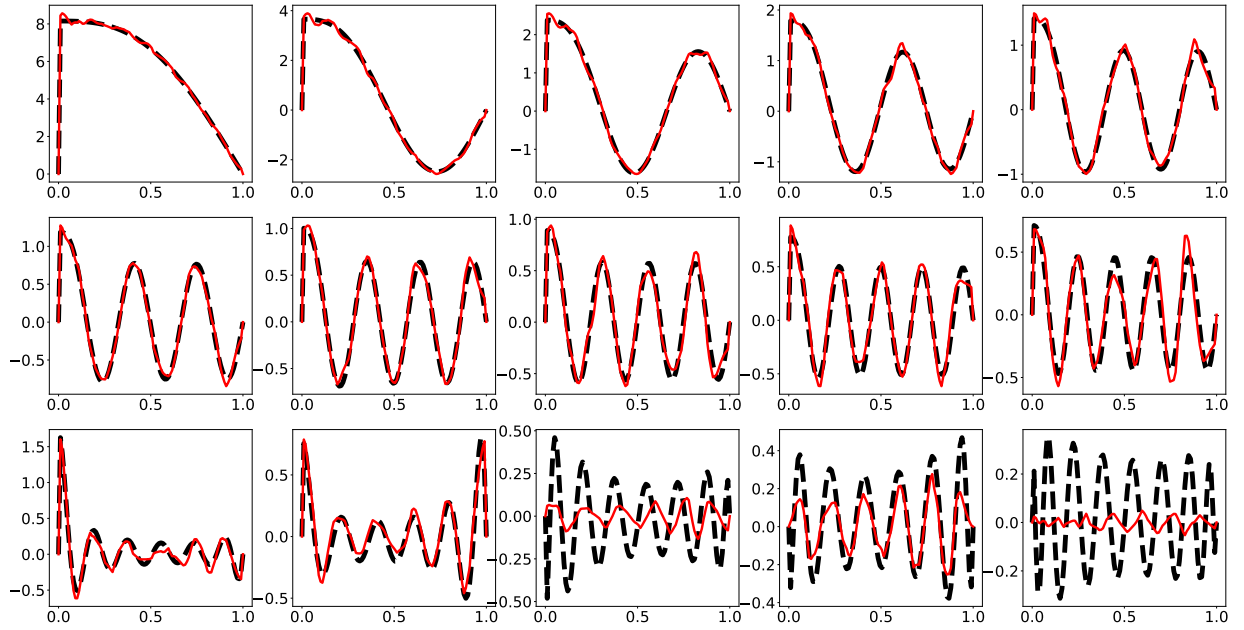


Figure 8.14: 1D advection-diffusion problem: Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red).

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	0.33 (0.35)	0.10 (0.09)	2.60 (2.15)
PG-VarMiON	3805	0.48 (0.40)	0.28 (0.20)	3.53 (2.82)
FNO	3953	0.34 (0.16)	0.36 (0.23)	3.26 (2.62)
L-DeepONet	4405	2.36 (0.84)	2.04 (0.91)	17.59 (16.24)
BNet	600	0.36 (0.34)	0.54 (0.55)	47.57 (40.31)

Table 8.5: 1D advection-diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.

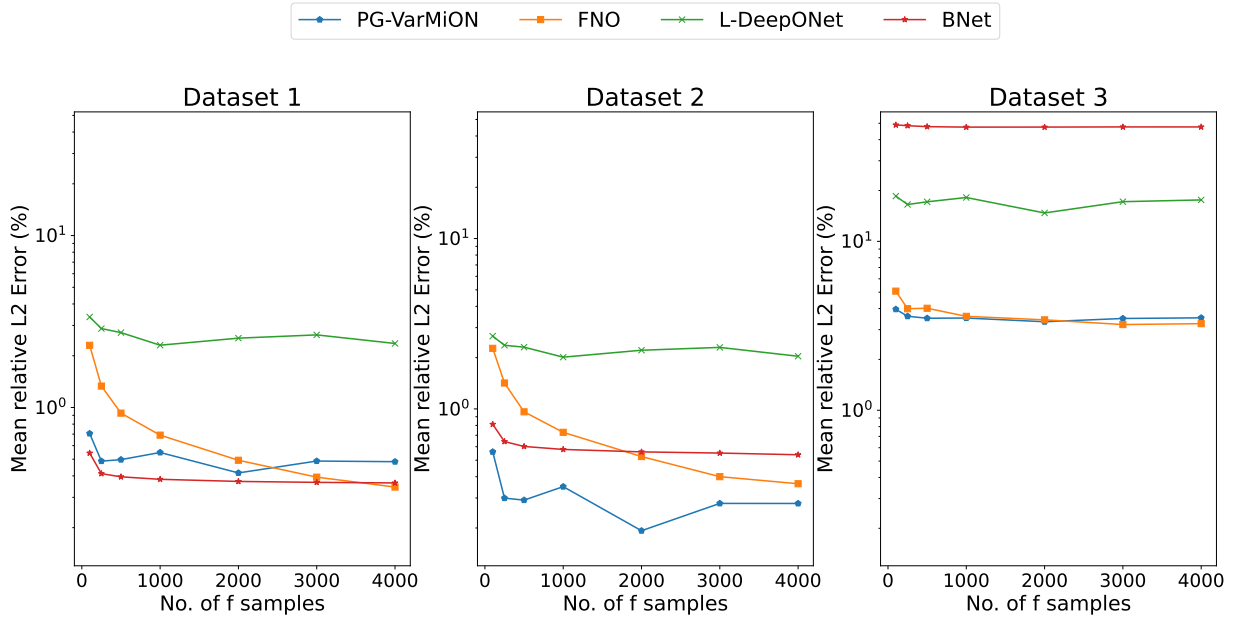


Figure 8.15: 1D advection-diffusion problem: Comparison of mean relative L^2 test error (in %) with the various methods.

8.4.4 2D Problems

With the results of various one-dimensional problems described, we now turn our attention to higher dimensions, namely two-dimensional problems. It is feasible to extend the PG-VarMiON and other operator networks to two dimensions as nothing inherent to the architecture needs to change. It is just a matter of changing the implementation to accomodate the new input (see Section 8.4.1 for details on the datasets and quadrature). In order to focus on the more interesting results, we choose to omit the diffusion equation and move directly to the advection-diffusion equation.

8.4.4.1 Trial basis selection

One aspect of the PG-VarMiON architecture that requires a bit more care is the selection of the fixed trial basis. In the one-dimensional problems, it is feasible to construct a trial basis, which has a low dimensionality and low projection error for the solution samples. Unfortunately, this is not necessarily the case in two dimensions, as it is not obvious how to select a trial basis by hand that can adequately represent 2D solution features.

To determine a trial basis for any given PDE, we consider a more systematic strategy that consists of constructing a basis of tensorized cubic B-splines. The strategy is as follows:

1. Choose a random sampling of solutions in the training dataset.
2. For a number of epochs, minimize the with respect to the optimal norm $\|\cdot\|_{\dagger}$ by tuning the basis function parameters.

B-splines are collections of piecewise polynomials where the higher degree functions are con-

structed recursively from the lower degree functions by the Cox-de Boor formula:

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise,} \end{cases}$$

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) - \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t)$$

where p is the degree of the B-spline. The knot vector $\mathbf{t} = \{t_i\}_{i=0}^m$ consists of the values at which the piecewise polynomials meet. Note that for a B-spline basis, these knots are the only parameters. Thus, a tensorized B-spline basis, which is a tensorized product of the x and y 1D B-spline bases, only has two knot vectors as parameters.

Figure 8.16 shows the projection error of the B-spline basis for the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions with respect to the number of samples used in the optimization process. The errors in this figure are the mean relative errors for the entire test set. Clearly, the B-spline trial basis can be optimized to yield low projection errors even from a sparse amount of data. Thus, we choose to use 200 random samples from the training dataset to determine the B-spline trial bases for all problems. Once the basis is determined, we can then use the Gram-Schmidt process to transform the basis into an orthonormal trial basis.

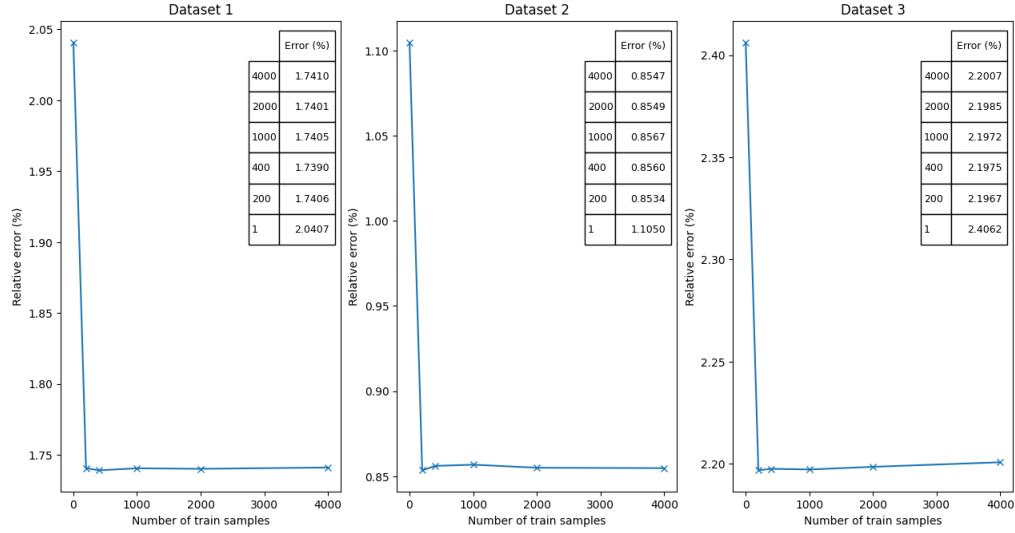


Figure 8.16: The mean relative projection errors for the tensorized B-spline basis against the number of samples used in the optimization. These results are computed on the three test sets for the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.

8.4.4.2 Advection-diffusion (Homogeneous Dirichlet)

We consider the operator defined by the solution to the advection-diffusion problem given by (6.2) where $\kappa = 10^{-3}$ and c is a vortex centered on $(0.75, 0.75)$ expressed as

$$\begin{aligned} c_1 &= -5(y - 0.75) \exp \left(\frac{1 - (5(x - 0.75))^2 - (5(y - 0.75))^2}{2} \right), \\ c_2 &= 5(x - 0.75) \exp \left(\frac{1 - (5(x - 0.75))^2 - (5(y - 0.75))^2}{2} \right). \end{aligned} \quad (8.45)$$

Although c is spatially varying, we fix this advective field across all samples. The velocity field for this problem is shown in Figure 8.17. The training and test datasets are constructed by selecting forcing functions f as described in Section 8.4.1.

For the trial basis, we opt to generate a 225-dimensional tensorized cubic B-spline basis, as described in Section 8.4.4.1, and use the Gram-Schmidt process to transform it into an orthonor-

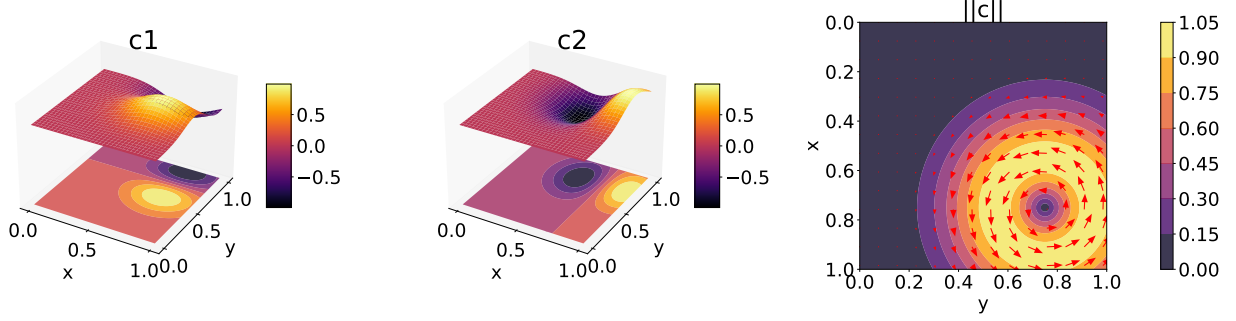


Figure 8.17: The velocity field for the 2D advection-diffusion solution sample. c_1 and c_2 are the x and y components of the velocity field, respectively. $\|c\|$ is the magnitude of the velocity. The red arrows indicate the direction of the velocity.

mal trial basis. Figure 8.18 depicts the first sixteen basis functions of the trial basis. Note that these functions are local since they are given by B-splines.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 8.4.2 with 6 hidden layers of widths [10,40,60,80,100,200] and train with the hyperparameters stated in Table 8.6. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 8.19, where we also compare with the relative finite-dimensional projection error. Note that the deviation from the projection error is larger as compared to the one-dimensional problem. However, this is not unexpected given that we continue to maintain the same resolution in terms of the number of basis functions (per dimension) as the one-dimensional problem, but are now dealing with a more complex velocity field. While not shown here, we have observed that the PG-VarMiON error can be further reduced by choosing a larger trial basis.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO (see Table 8.6 for training details). Table 8.7 compares the mean relative L^2 error with all the methods on each test dataset. We remark that FNO performs the best on DATASET 1, where it has a lower error than the projection error. This is possible because it does not

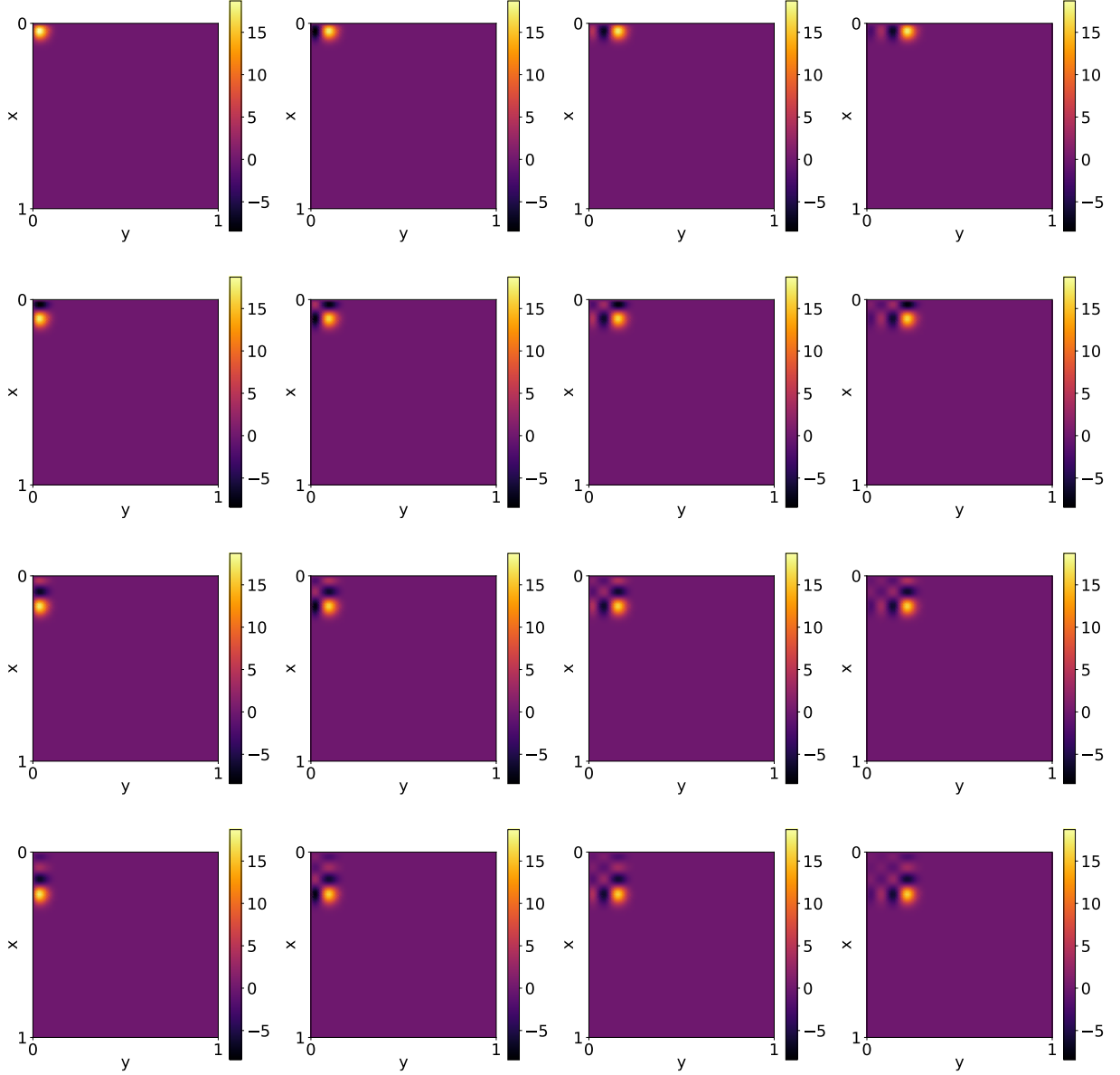


Figure 8.18: The first sixteen functions of the trial basis used for the 2D advection-diffusion problem (homogeneous Dirichlet BC). These are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$.

incorporate the trial basis Φ in its structure and will not have the projection error as a lower bound. On the other datasets, FNO does poorly on DATASET 2 compared to the other methods and performs similarly to PG-VarMiON on DATASET 3. We observe that PG-VarMiON and L-DeepONet (which is learning a 225-dimensional trunk) perform similarly on all three datasets,

	PG-VarMiON	L-DeepONet	BNet	FNO
Activation Function	ELU	Hat	None	GELU
Cut-off function (8.42)	Yes		No	
p in (8.42)	100		N/A	
No. of Parameters (Neumann)	81335	441335 (459335)	360000 (378000)	81249 (81259)
Optimizer	AdamW			Adam
AdamW β_1	0.5			N/A
AdamW β_2	0.9			N/A
Weight decay	0			10^{-8}
Epochs	2000			
LR Scheduler	Yes			
Initial LR	10^{-3}			10^{-2}
LR Scheduler step	100			
LR Scheduler γ	0.9		0.75	0.9
Batch size	200			

Table 8.6: Summary of training settings for all networks in the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.

with PG-VarMiON leading to marginally lower errors. However, note that the PG-VarMiON uses one-fifth the number of trainable parameters compared to L-DeepONet, which has at least as many parameters as BNet because of the $\mathbf{B}$ matrix in its architecture. Similar to the 1D problems, the BNet performs very well on the first two datasets but is essentially useless on the challenging DATASET 3 where it cannot provide any meaningful results.

Next, we take a closer look at 3 test samples from DATASET 3, whose forcing functions are shown in Figure 8.20, with the corresponding solutions compared in Figure 8.21. From these contour plots, the reference, projection, and PG-VarMiON solutions look very similar. The BNet solutions are omitted because they essentially appear as noise (note the mean error with BNet

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	1.74 (0.48)	0.85 (0.33)	2.20 (0.81)
PG-VarMiON	81335	2.47 (0.63)	1.42 (0.35)	4.44 (1.77)
FNO	81249	1.58* (0.40)	3.69 (1.71)	4.11 (1.39)
L-DeepONet	441335	3.32 (0.86)	1.49 (0.38)	5.76 (2.39)
BNet	360000	2.00 (0.54)	2.06 (0.77)	177.24 (75.26)

Table 8.7: 2D advection-diffusion problem (homogeneous Dirichlet BC): Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *.

on DATASET 3 listed in Table 8.7). To accentuate the similarities (and differences) between the various methods, we also plot the pointwise errors in Figure 8.22, which clearly shows that PG-VarMiON leads to lower errors compared to L-DeepONet. We also plot the solutions extracted along 1D slices in Figure 8.23. Generally, all methods capture the shape of the reference solution in these 1D slices. However, we observe that there are some areas where PG-VarMiON, L-DeepONet, and FNO have noticeable errors, clearly showing the challenge of this OOD dataset. Furthermore, we notice that the L-DeepONet and FNO solutions may contain low magnitude spurious oscillation (see Sample 3 slice at $y = 0.51$ in Figure 8.23).

Finally, we depict expected weighting functions (computed by solving the adjoint problem using nutils) corresponding to the first sixteen trial basis functions and the PG-VarMiON approximations in Figure 8.24. We observe that the PG-VarMiON is able to qualitatively learn the optimal weighting functions, despite not being shown the true Ψ while training.

8.4.4.3 Advection-diffusion (Mixed boundary conditions)

We consider the operator defined by the solution to the advection-diffusion problem given by (6.2) where $\kappa = 10^{-3}$ and $\mathbf{c}$ is a vortex as defined by (8.45). This problem is identical to that



Figure 8.19: 2D advection-diffusion problem (homogeneous Dirichlet BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

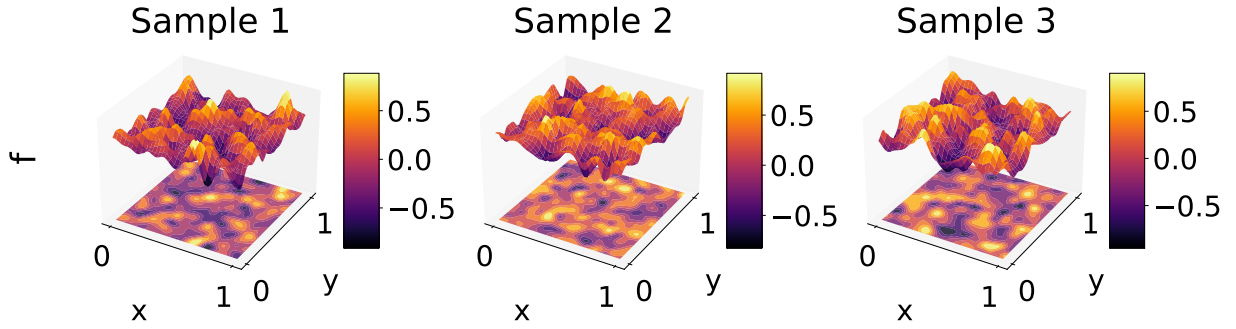


Figure 8.20: 3D plots with contours of the forcing functions for 3 test samples.

described in Section 8.4.4.2 except there are now Neumann boundary conditions at $y = 0$ and $y = 1$ (homogeneous Dirichlet boundary conditions are maintained for $x = 0$ and $x = 1$). The training and test datasets are constructed by selecting forcing functions f as described in Section 8.4.1.

For the trial basis, we opt to generate a 225-dimensional tensorized cubic B-spline basis, as described in Section 8.4.4.1, and use the Gram-Schmidt process to transform it into an orthonormal trial basis. Figure 8.25 depicts the first sixteen basis functions of the trial basis. Note

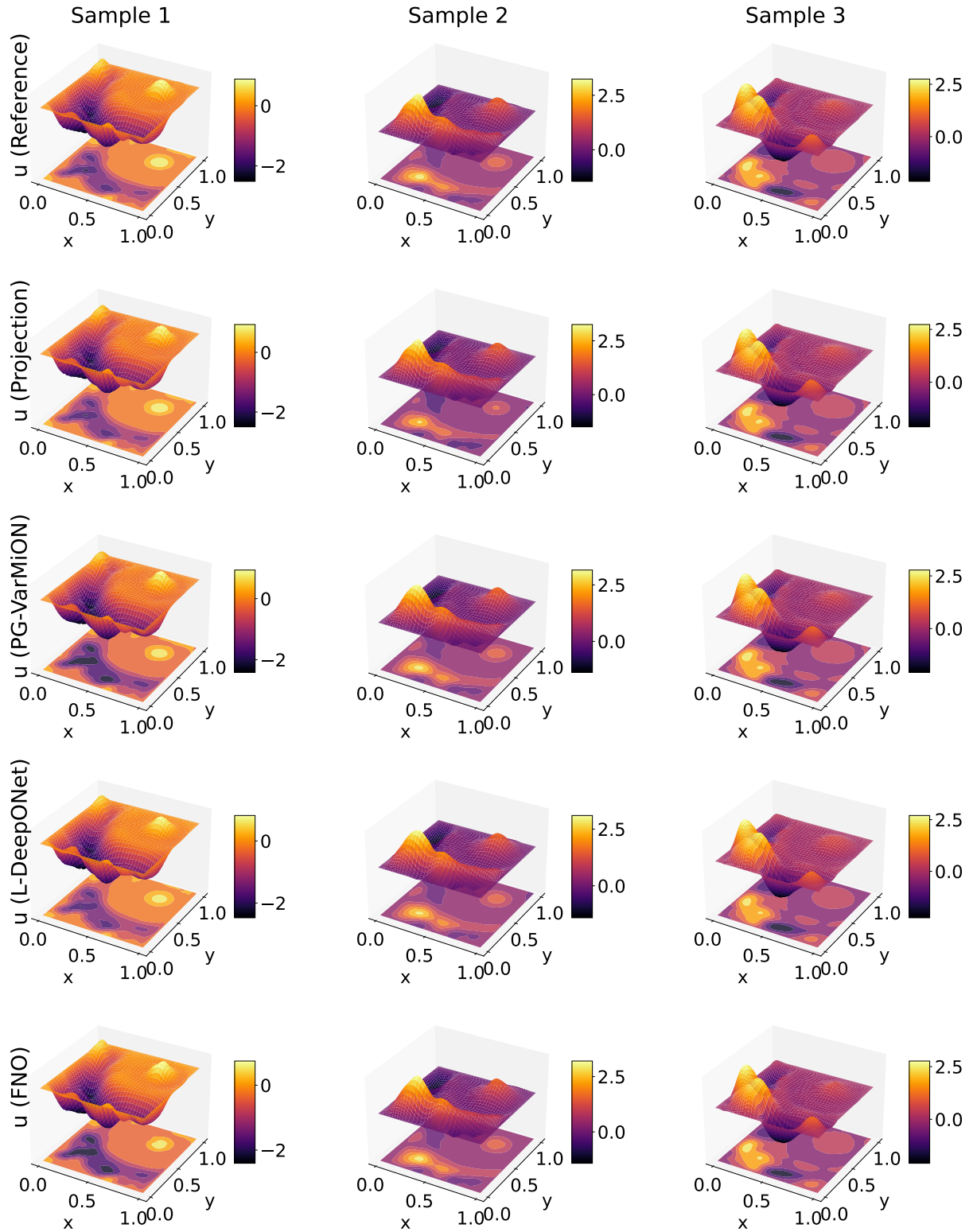


Figure 8.21: 3D plots with contours of the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations for 3 test samples.

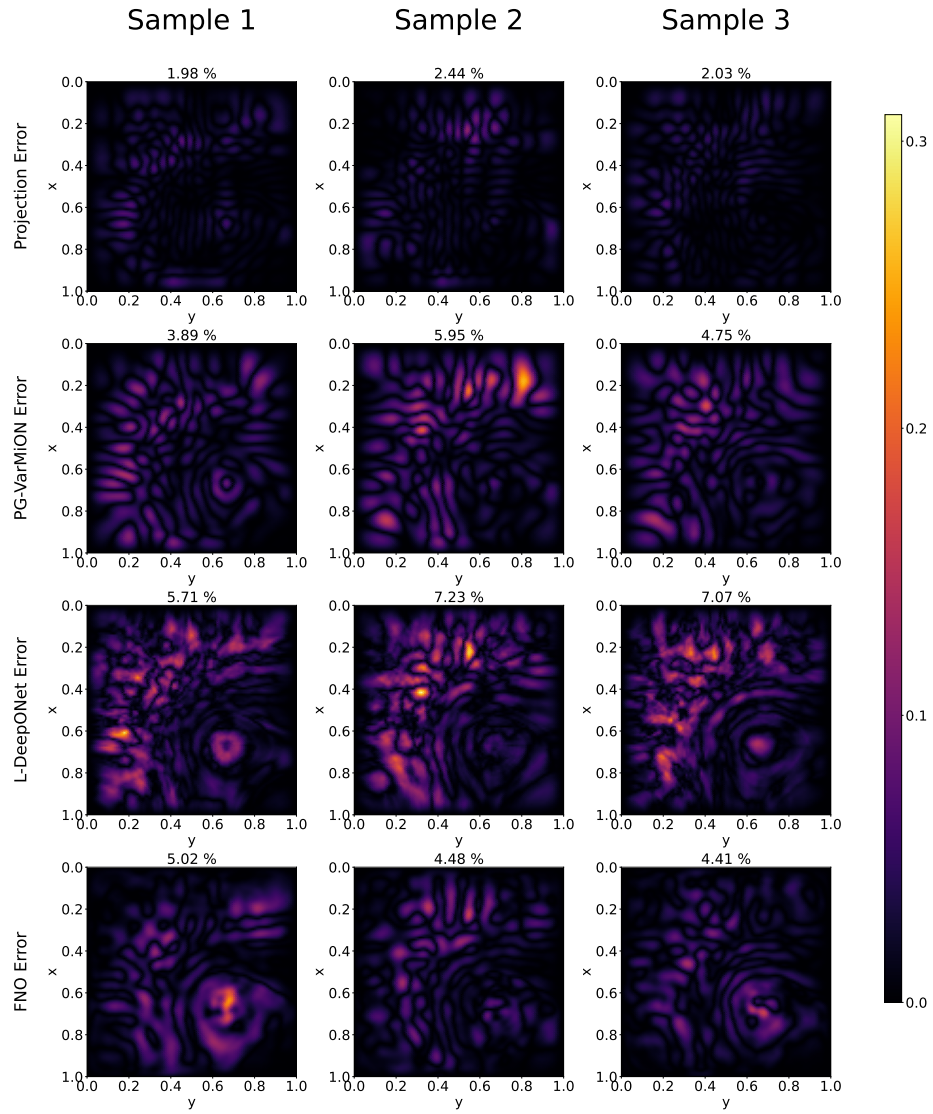


Figure 8.22: Plots of the errors of the 3 test samples for the projection, PG-VarMiON, L-DeepONet, and FNO approximations.

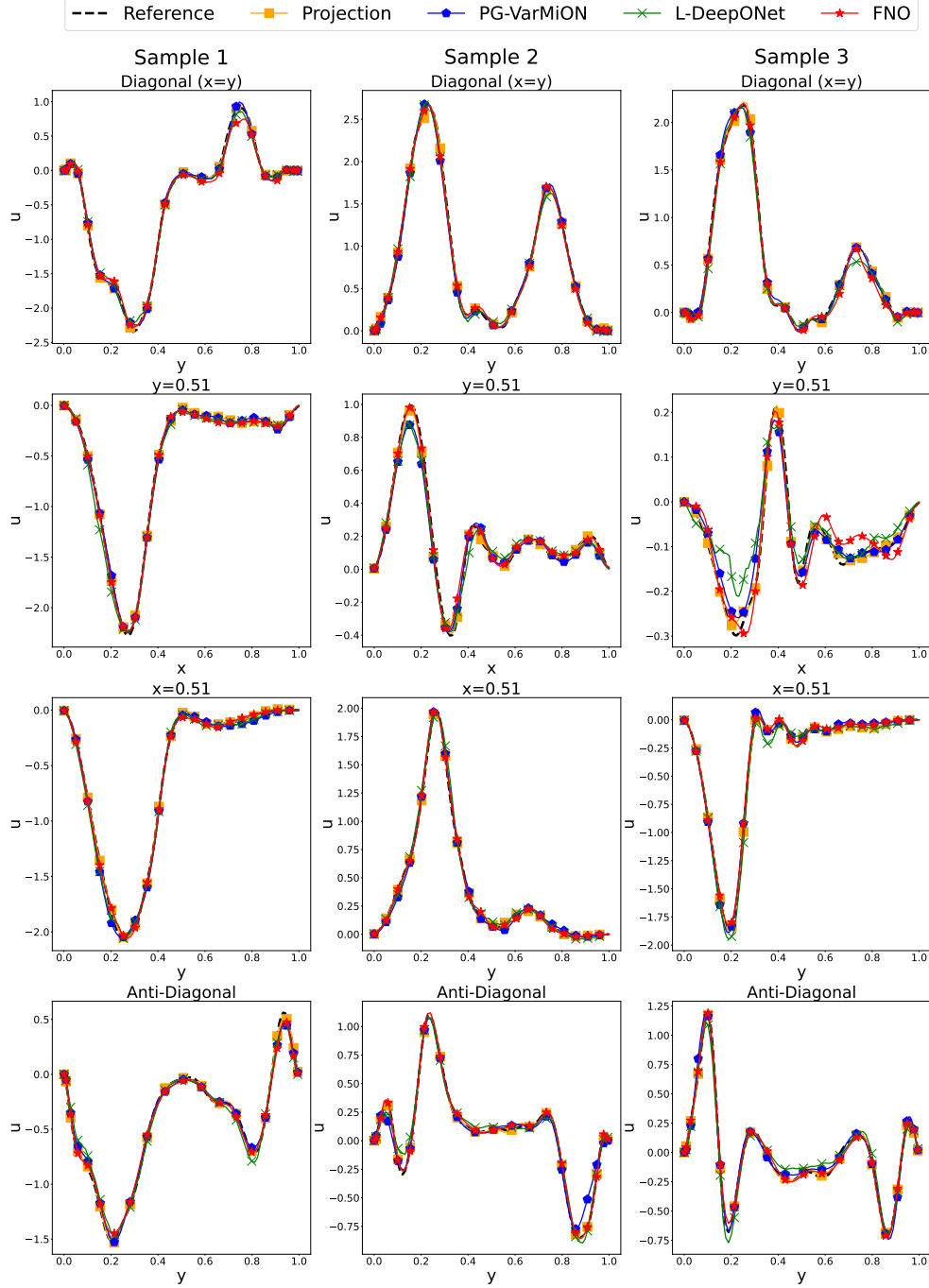


Figure 8.23: 1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51$, $x = 0.51$.

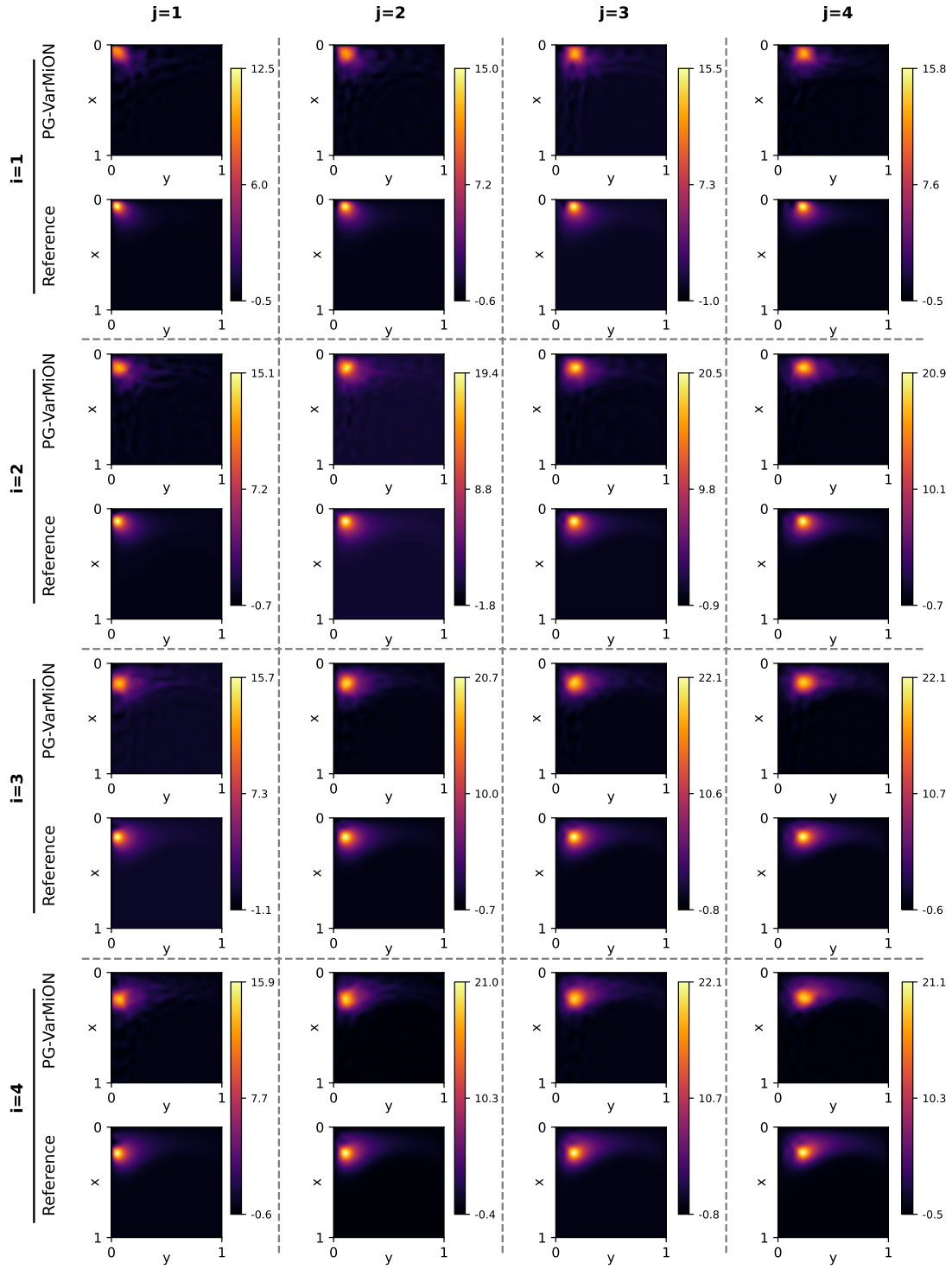


Figure 8.24: Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$.

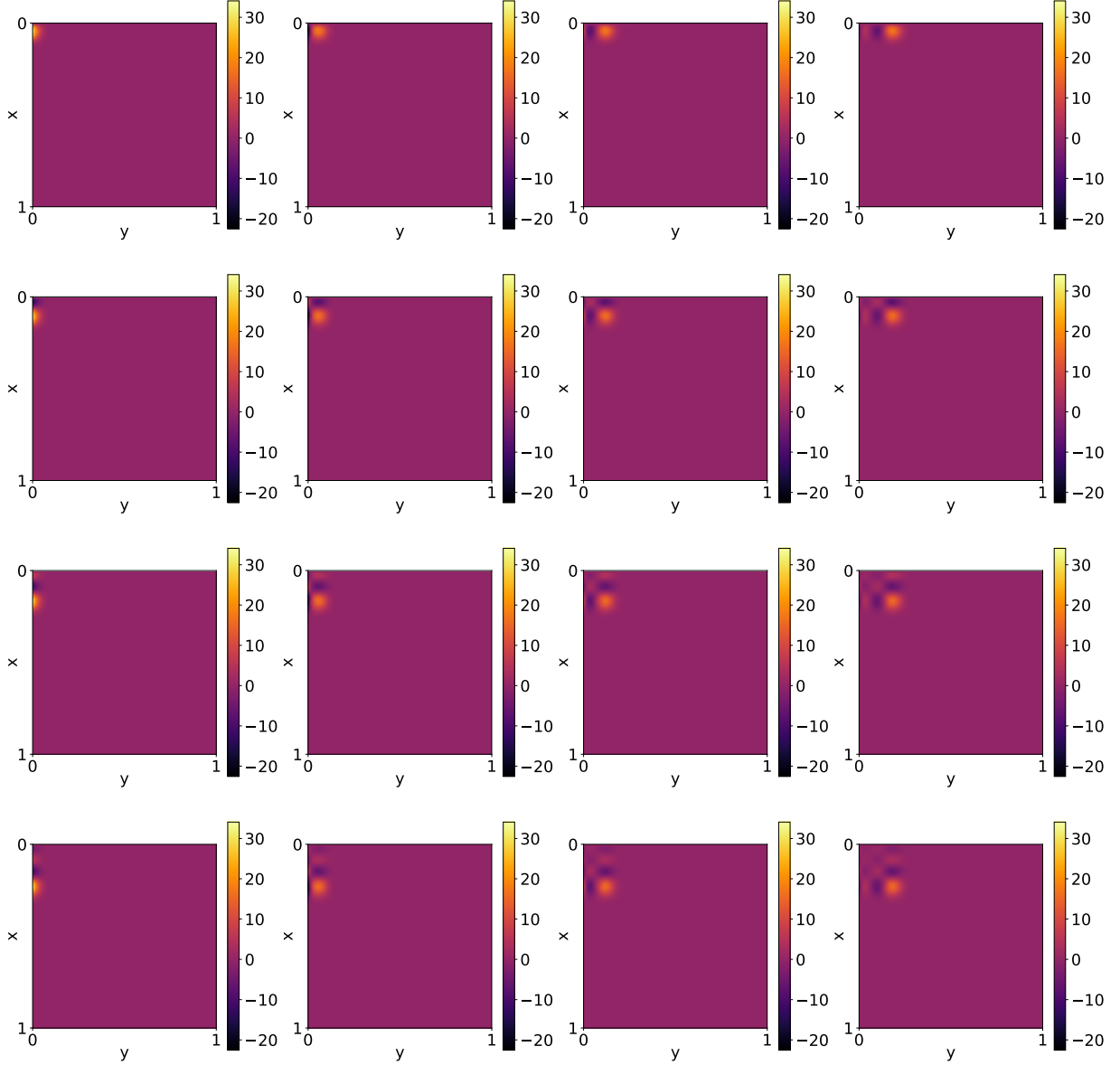


Figure 8.25: The first sixteen functions of the trial basis used for the 2D advection-diffusion problem (mixed BC). These are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$.

that unlike the homogeneous Dirichlet problem, basis functions can have non-zero values at the Neumann boundaries $y = 0$ and $y = 1$.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 8.4.2 with 6 hidden layers of widths [10,40,60,80,100,200] and train with the hyperparameters stated in Table 8.6 (same as those used for the homogeneous Dirichlet problem). The trained network is then used on the

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	1.76 (0.58)	0.69 (0.29)	1.80 (0.74)
PG-VarMiON	81335	2.57 (0.79)	1.21 (0.36)	3.47 (1.47)
FNO	81259	1.90 (0.53)	3.94 (1.73)	4.05 (1.44)
L-DeepONet	459335	3.03 (0.92)	1.33 (0.41)	5.68 (2.61)
BNet	378000	1.87 (0.61)	1.68 (0.72)	112.09 (50.63)

Table 8.8: 2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.

three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 8.26, where we also compare with the relative finite-dimensional projection error. Overall, the projection errors and performance of PG-VarMiON are similar to that of the homogeneous Dirichlet problem.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO. Table 8.8 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON and L-DeepONet perform similarly on all three datasets, with PG-VarMiON leading to marginally lower errors (though the performance gap is significantly wider on DATASET 3). FNO has a similar performance trend to the homogeneous Dirichlet problem with poor performance on DATASET 2 compared to the other methods. However, it is never lower than the projection error here, likely due to the added complexity with the Neumann boundaries. As has been observed in the previous problems, the BNet performs very well on the first two datasets but struggles on DATASET 3.

Next, we take a closer look at 3 test samples from DATASET 3, whose forcing functions and Neumann boundary fluxes are shown in Figure 8.27. The corresponding solutions for the operator networks (excluding BNet) are compared in Figure 8.28. From these contour plots, the reference, projection and PG-VarMiON solutions look very similar. To accentuate the similarities

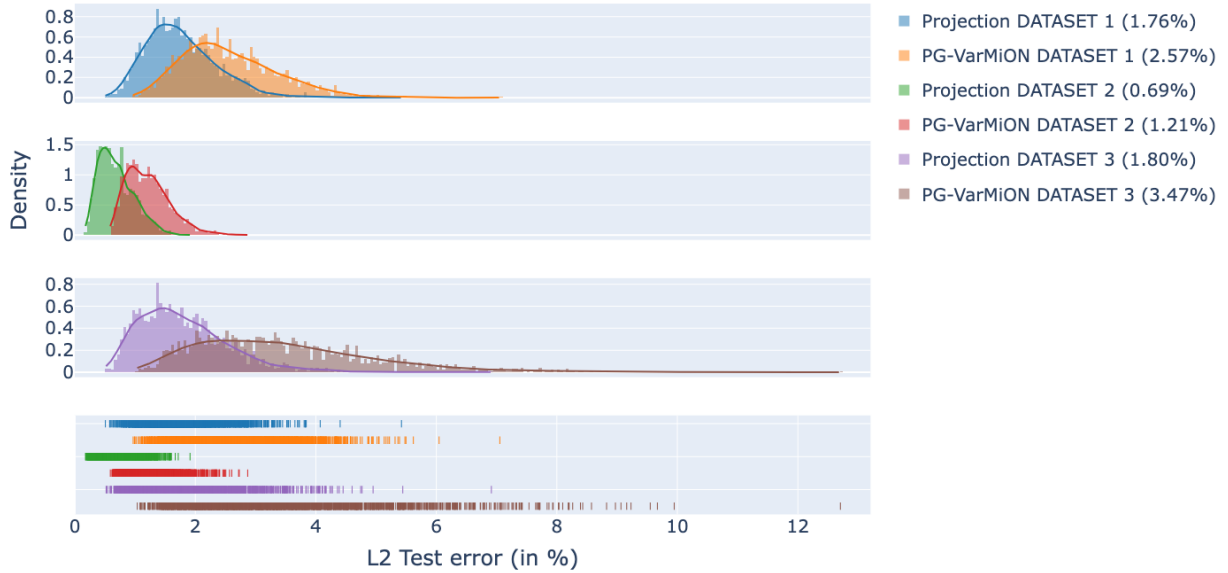


Figure 8.26: 2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

(and differences) between the various methods, we also plot the pointwise errors in Figure 8.29, which clearly shows that PG-VarMiON leads to lower errors compared to L-DeepONet. We also plot the solutions extracted along 1D slices in Figure 8.30. Generally, all methods capture the shape of the reference solution in these 1D slices. Once again, we observe that there are some areas where PG-VarMiON, L-DeepONet, and FNO have noticeable errors, clearly showing the challenge of this OOD dataset (for instance, see Sample 1 slice at $x = 0.51$ in Figure 8.30). Overall, PG-VarMiON appears to capture the solutions better than L-DeepONet and similarly to FNO.

Finally, we depict expected weighting functions (computed by solving the adjoint problem using nutils) corresponding to the first sixteen trial basis functions and the PG-VarMiON approximations in Figure 8.31. We observe that the PG-VarMiON is able to qualitatively learn

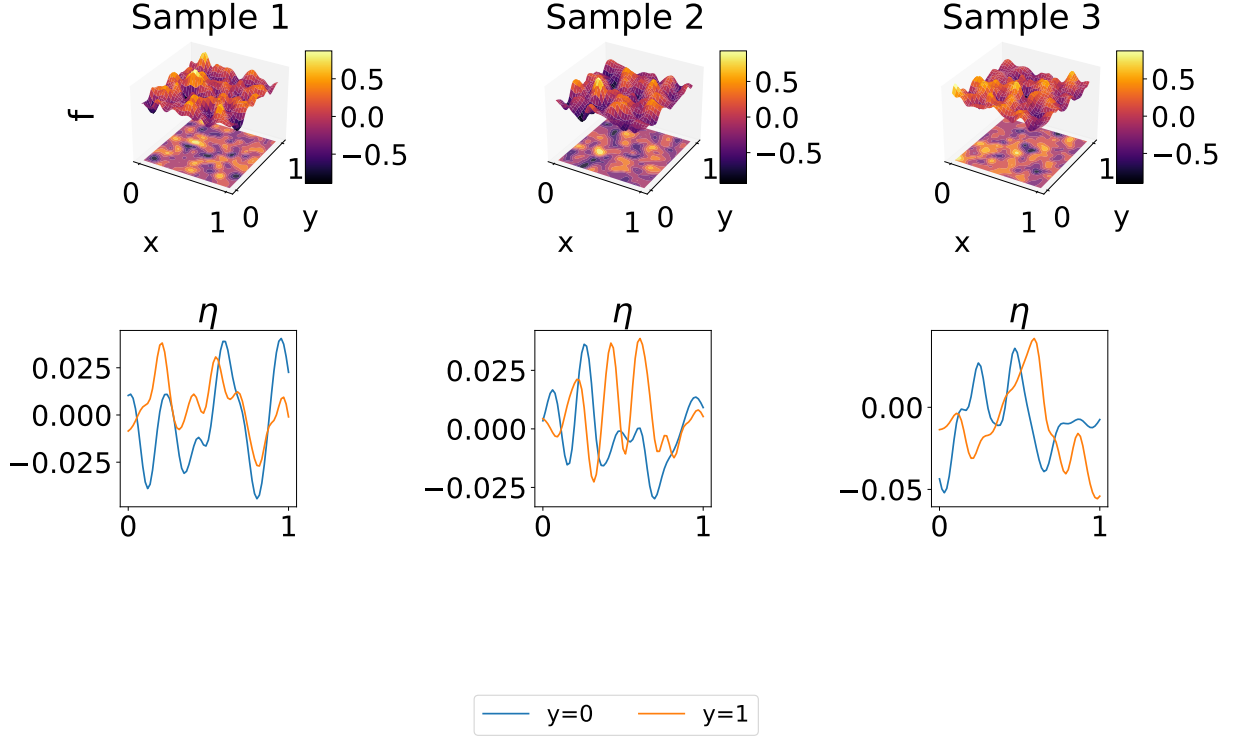


Figure 8.27: 3D plots with contours of the forcing functions and plots of η for 3 test samples.

the optimal weighting functions, despite not being shown the true Ψ while training.

8.4.4.4 PG-VarMiON trained with zero Neumann

In the previous problem, we trained and tested PG-VarMiON on solution samples where the Neumann flux was non-zero. If we now consider the same problem but with zero Neumann flux on those same boundaries, we can take an interesting approach where we train PG-VarMiON on samples with zero Neumann flux and then test on the same samples from before where Neumann data varies. As noted in Remark 8.6, there are no specific parameters in PG-VarMiON associated with the Neumann boundary conditions. In principle, A^b should be recovered well enough to provide adequate performance even in the absence of non-zero Neumann data. This test allows for the generalization capabilities of PG-VarMiON to become clear in contrast to the similar

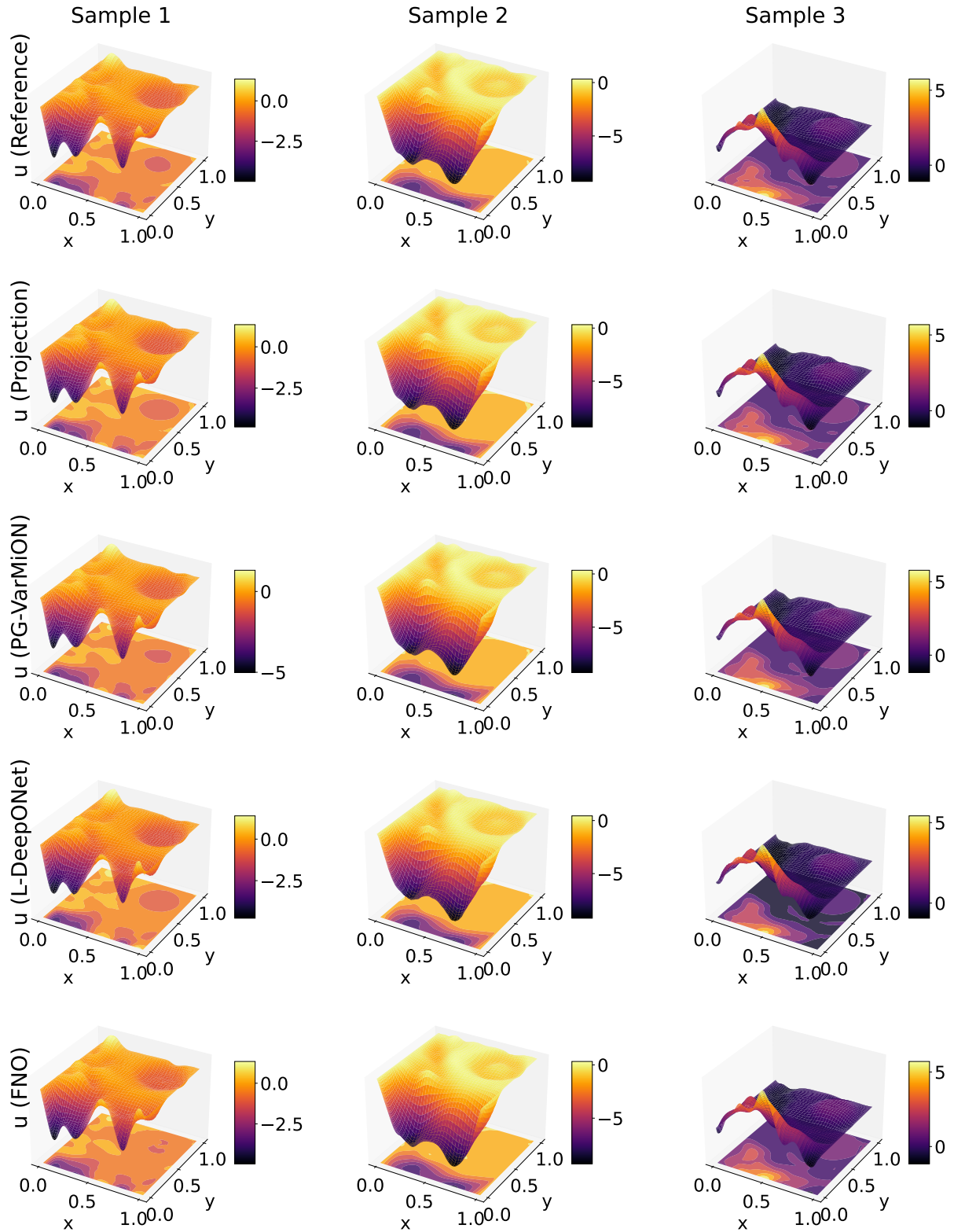


Figure 8.28: 3D plots with contours of the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations for 3 test samples.

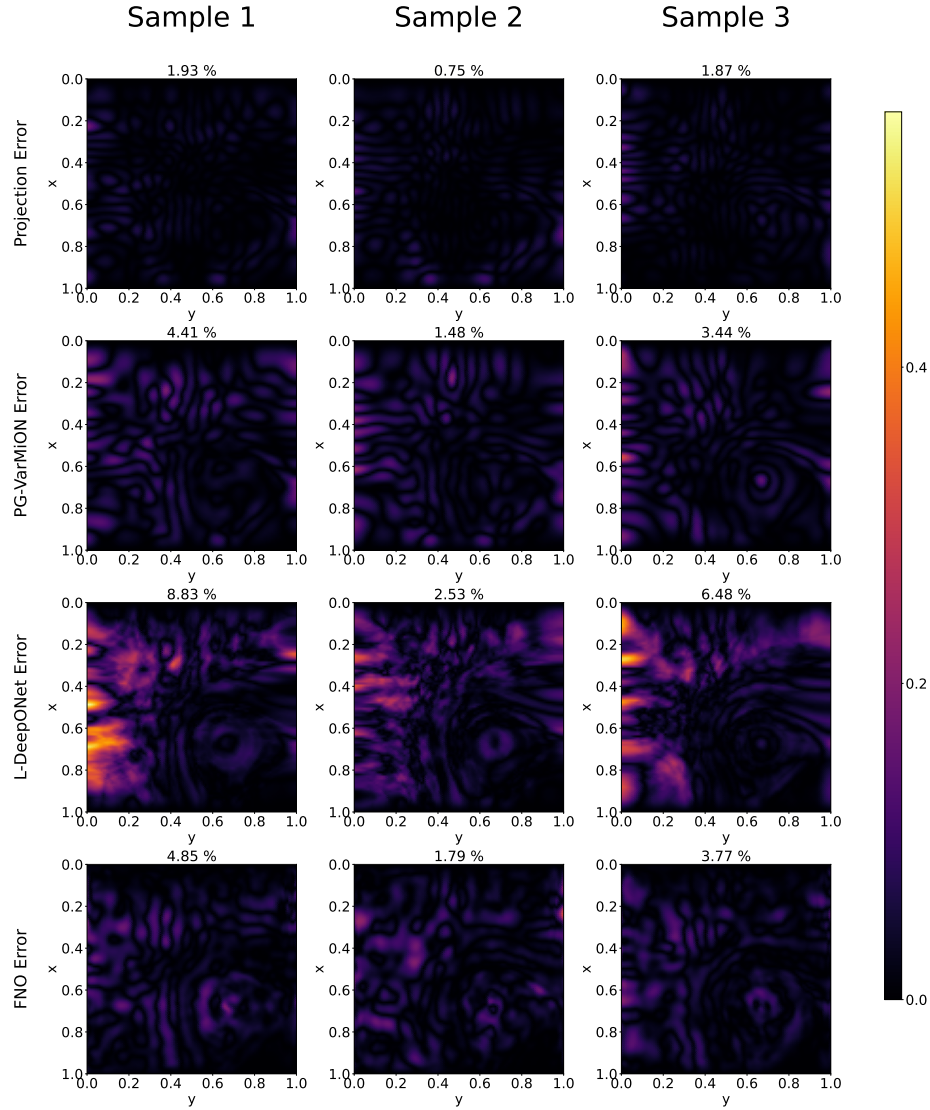


Figure 8.29: Plots of the errors of the 3 test samples for the projection, PG-VarMiON, L-DeepONet, and FNO approximations.

models such as BNet and L-DeepONet as well as FNO, which will not be able to learn how to handle non-zero Neumann data.

Using the same architecture and hyperparameters as the previous problem, PG-VarMiON is trained on the same data as described 8.4.1 except η is zero on all Neumann boundaries. The trained network is then used on the three test datasets with zero and non-zero Neumann data. The histograms (with rug plots) of the relative L^2 test errors are shown in Figures 8.32 and 8.33, where

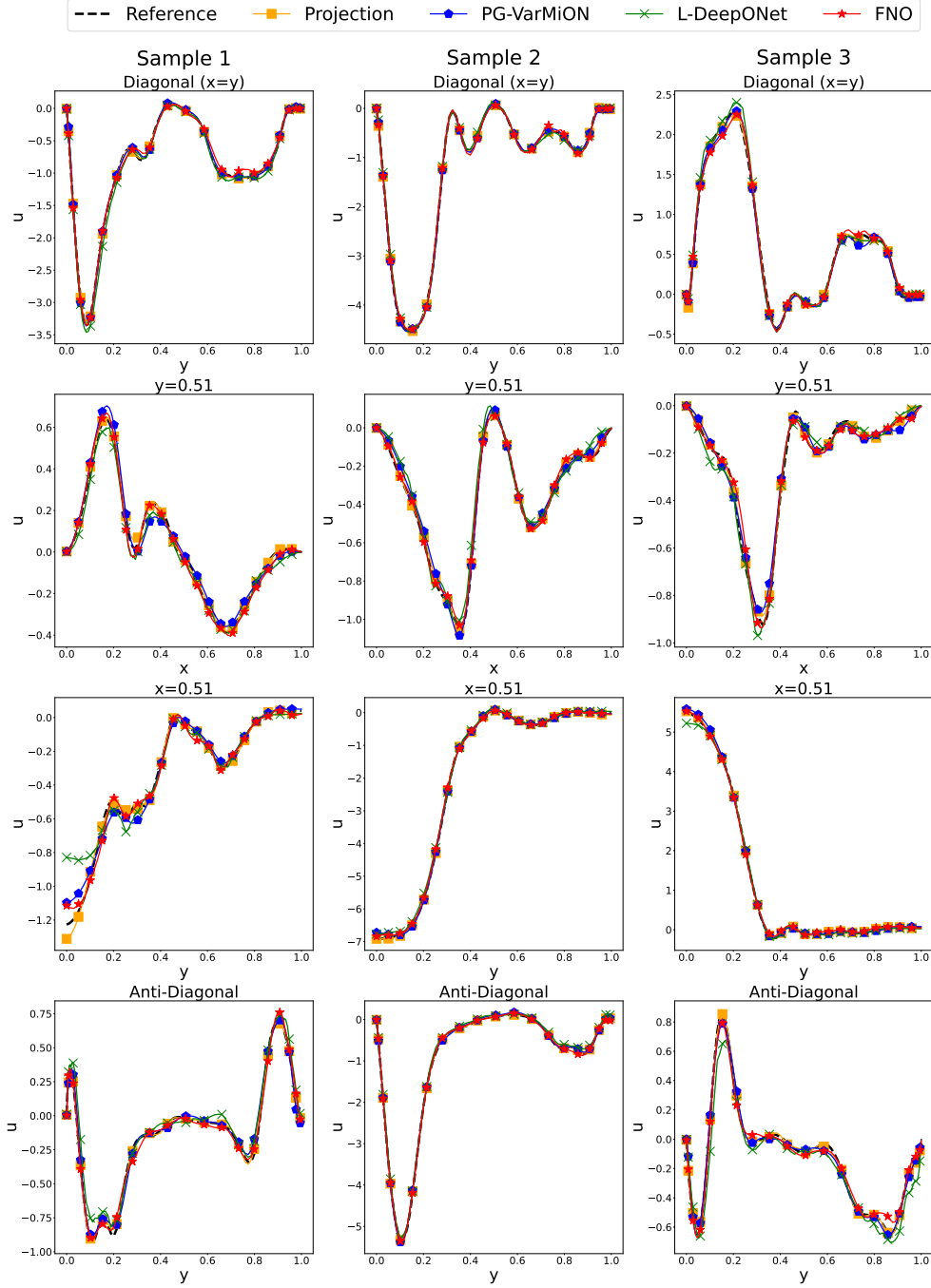


Figure 8.30: 1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, L-DeepONet, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51$, $x = 0.51$.

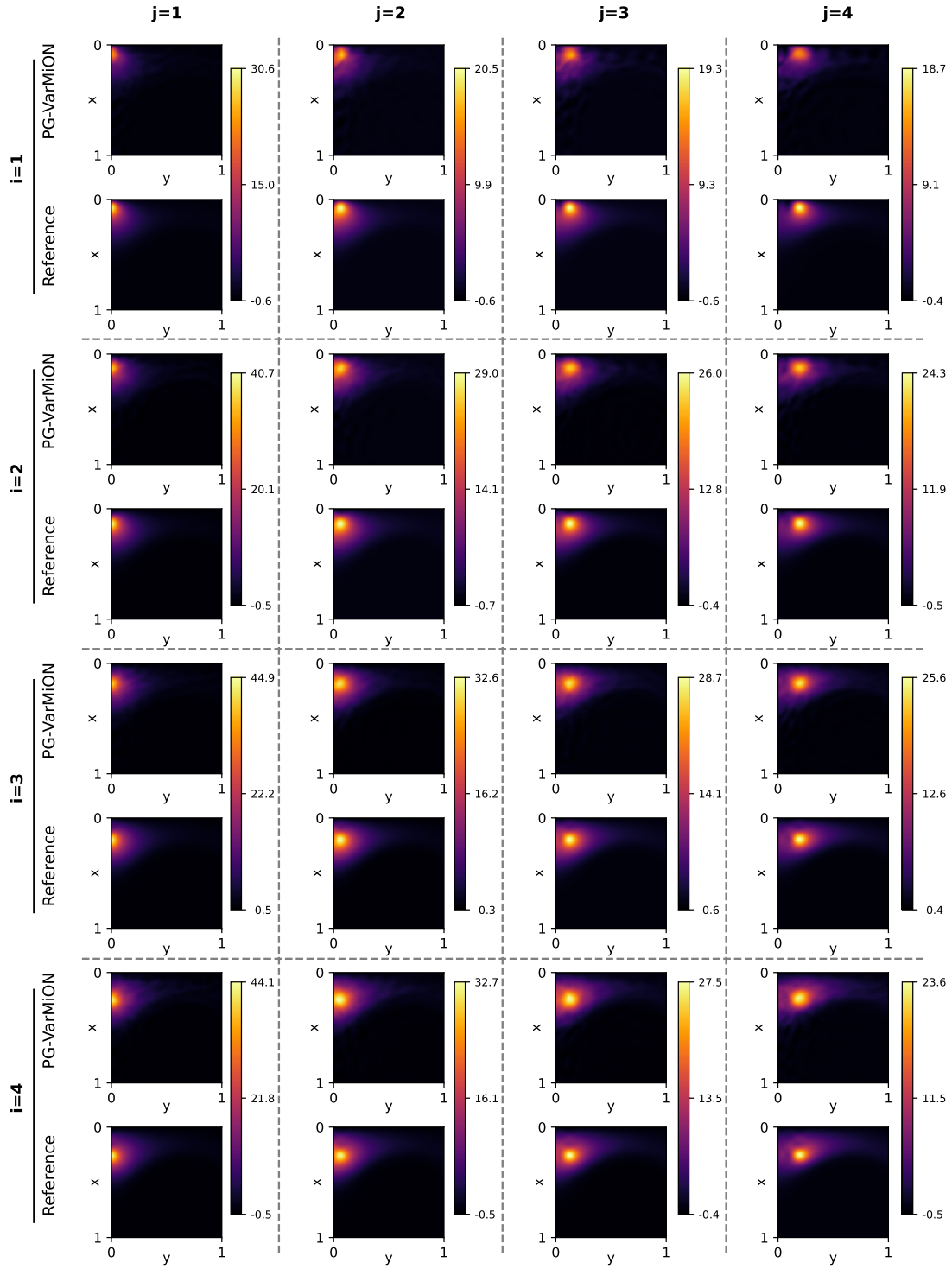


Figure 8.31: Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$.

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
		Zero Neumann		
Projection	-	1.83 (0.61)	0.68 (0.29)	1.79 (0.77)
PG-VarMiON	81335	2.51 (0.76)	1.19 (0.34)	3.39 (1.49)
FNO	81259	1.77* (0.49)	4.60 (2.39)	4.12 (1.43)
L-DeepONet	459335	2.62 (0.81)	1.09 (0.34)	4.35 (2.03)
BNet	378000	1.96 (0.64)	1.72 (0.76)	118.08 (55.66)
		Non-zero Neumann		
Projection	-	1.76 (0.58)	0.69 (0.29)	1.80 (0.74)
PG-VarMiON	81335	7.13 (2.32)	3.05 (1.29)	5.03 (2.16)
FNO	81259	52.62 (21.69)	27.61 (15.40)	31.29 (17.24)
L-DeepONet	459335	52.62 (21.66)	27.02 (15.64)	31.27 (17.40)
BNet	378000	62.51 (22.94)	30.12 (16.26)	118.18 (52.63)

Table 8.9: 2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 test error (in %) with the various methods trained on zero Neumann data. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *.

we also compare with the relative finite-dimensional projection error. It is clear that the errors on the test sets with zero Neumann data are comparable to the performance observed previously with PG-VarMiON. However, when tested on the test sets with non-zero Neumann data, this PG-VarMiON performs significantly worse. That said, it is still reasonable performance overall since the non-zero Neumann data is completely OOD in this scenario.

Attempting to train BNet, L-DeepONet, and FNO on the zero Neumann data and then testing them on the non-zero Neumann data provides exceedingly poor performance, as shown in Table 8.9. In the case of BNet and L-DeepONet, this is because there is no way to learn the $\mathbf{A}^b$ matrix as it only acts on a zero vector during training. Thus, it can only generate noise when it acts on non-zero Neumann data. For FNO, while there is not a set structure for the Neumann data, the parameters that directly handle this input cannot be trained either with zero Neumann training data.

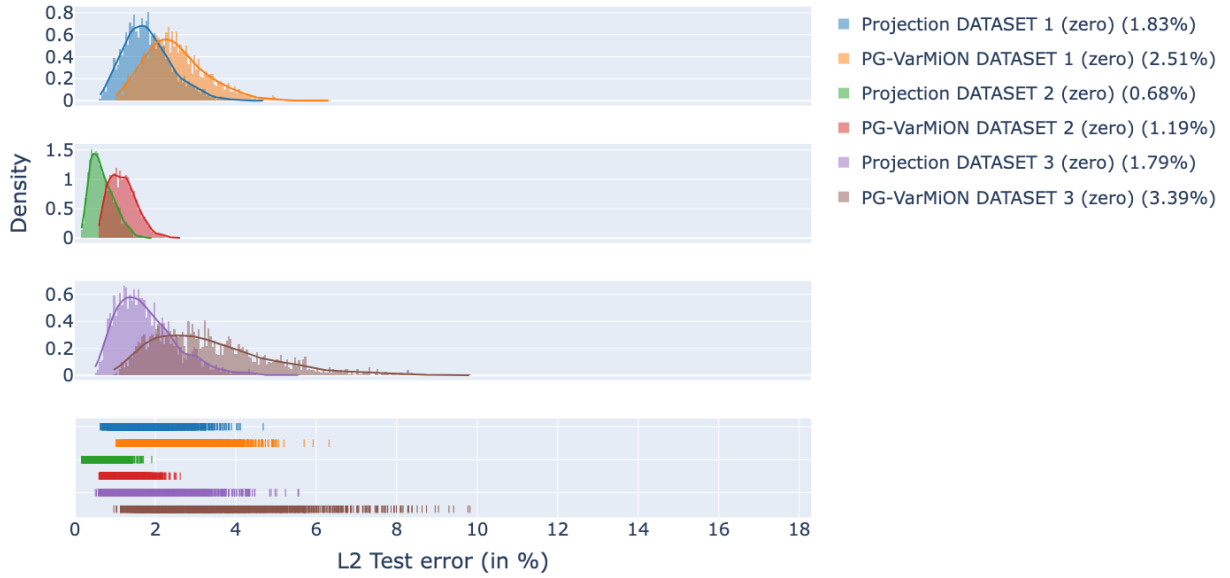


Figure 8.32: 2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses. The test sets consist of samples with zero Neumann data.

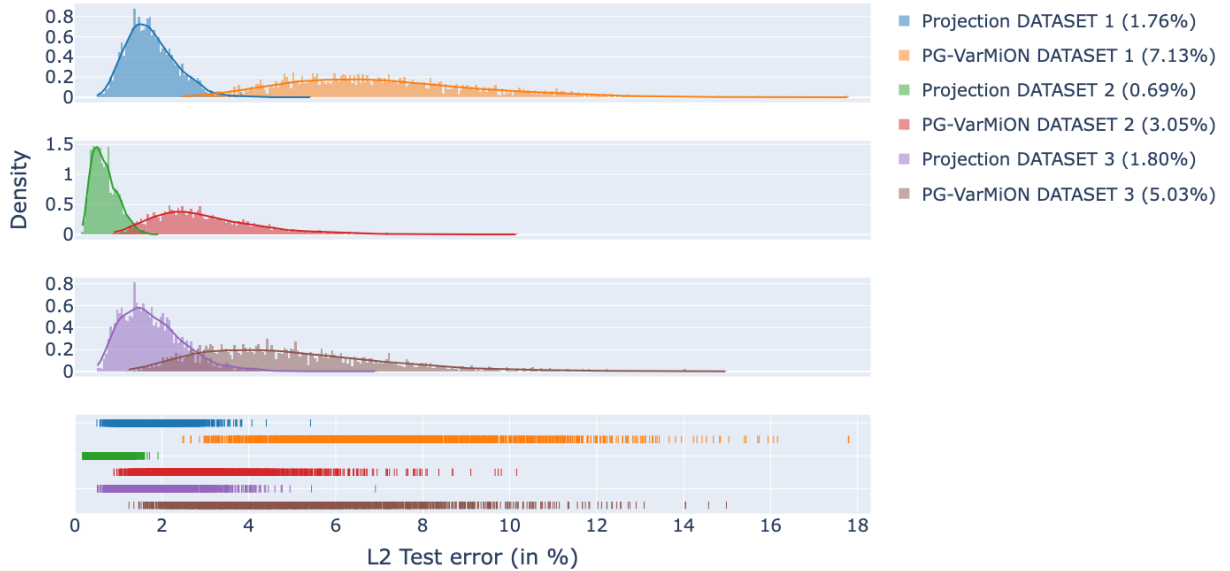


Figure 8.33: 2D advection-diffusion problem (mixed BC): Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses. The test sets consist of samples with non-zero Neumann data.

Method	Training Norm	Dataset 1	Dataset 2	Dataset 3
		L^2 error		
Projection	-	1.76 (0.58)	0.69 (0.29)	1.80 (0.74)
PG-VarMiON	L^2	2.57 (0.79)	1.21 (0.36)	3.47 (1.47)
	H^1 seminorm	3.71 (1.19)	1.73 (0.54)	4.34 (1.81)
	H^1	3.71 (1.04)	2.33 (0.72)	4.67 (1.54)
		H^1 seminorm error		
Projection	-	11.04 (1.89)	8.14 (2.87)	12.69 (2.44)
PG-VarMiON	L^2	12.67 (1.96)	9.31 (2.70)	17.58 (3.50)
	H^1 seminorm	12.46 (1.88)	8.81 (2.46)	17.17 (3.40)
	H^1	12.24 (1.87)	8.62 (2.48)	16.99 (3.42)
		H^1 error		
Projection	-	10.35 (1.85)	7.40 (2.64)	11.68 (2.37)
PG-VarMiON	L^2	12.13 (1.94)	8.72 (2.54)	16.63 (3.47)
	H^1 seminorm	11.81 (1.87)	8.10 (2.28)	15.99 (3.38)
	H^1	11.60 (1.85)	8.00 (2.29)	15.86 (3.37)

Table 8.10: 2D advection-diffusion problem (mixed BC): Comparison of mean relative L^2 , H^1 seminorm, and H^1 test errors (in %) with PG-VarMiON trained with respect to each norm. The standard deviation of error is in parentheses.

8.4.4.5 Training with other norms

As one final test, we train the PG-VarMiON twice more on the 2D advection-diffusion problem with mixed boundary conditions. This time, however, we select the H^1 seminorm and H^1 norm as the optimal norms in which to minimize the error instead of the L^2 norm. We use the same hyperparameters and the same B-spline trial basis as before, but the B-spline basis is orthonormalized with respect to the H^1 seminorm and H^1 norm. We compare the three PG-VarMiON's performances via the L^2 , H^1 seminorm, and H^1 errors on the three non-zero Neumann datasets in Table 8.10.

Clearly, the norm we choose to optimize with respect to will be the norm that PG-VarMiON

tends to minimize. The H^1 errors (especially in two dimensions) have a much higher error than the L^2 norm's error. PG-VarMiON's performance is limited by the projection error, so we cannot expect excellent performance with respect to the H^1 seminorm and H^1 norm. However, we observe that training with the H^1 norm provides marginally better performance in approximating the gradients than training with the H^1 seminorm or L^2 norm. In the big picture, there is not a significant difference in the performance of the three models, but this demonstrates that PG-VarMiON can be trained with respect to an arbitrary (but suitable) norm.

Chapter 9: PG-VarMiON for PDE with varying parameterization

In this chapter, we augment the PG-VarMiON framework to solve PDEs where the parameterization data $\mathbf{g}$ can vary. The development of this material is similar to that of Chapter 8. Theoretical analysis and an error estimate for this case are developed as well to provide bounds on the performance.

9.1 PG-VarMiON Formulation

With varying parameterization data, the Petrov-Galerkin structure that we seek to mimic does not fundamentally change. We are still seeking the coefficients of the optimal solution as dictated by (8.7). Of course, the main difference is that now the optimal weighting functions will vary across samples since $\mathbf{g}$ varies, and thus PG-VarMiON must be able to approximate these functions by considering $\mathbf{g}$ directly.

We maintain the same goal of constructing an operator network which mimics the linear system (8.7). This requires the same components as described in Section 8.1 with a key addition:

- The selected quadrature nodes $\{\mathbf{x}_k\}_{k=1}^{N_s} \subset \Omega$ used to sample the source function f are also used to sample the PDE parameterization data $\mathbf{g}$ to create input vector(s) for the PG-VarMiON (e.g. with varying $\mathbf{g} = [\mathbf{c}, \kappa]$, we obtain two vectors $\mathbf{C} = [\mathbf{c}(\mathbf{x}_1), \dots, \mathbf{c}(\mathbf{x}_{N_s})]^\top \in \mathbb{R}^{N_s}$ and $\mathbf{K} = [\kappa(\mathbf{x}_1), \dots, \kappa(\mathbf{x}_{N_s})]^\top \in \mathbb{R}^{N_s}$).

The updated schematic of PG-VarMiON with varying $\mathbf{g}$ is shown in Figure 9.1, where N_g is the number of varying features in the PDE parameterization data $\mathbf{g} = \{\mathbf{g}_1, \dots, \mathbf{g}_{N_g}\}$.

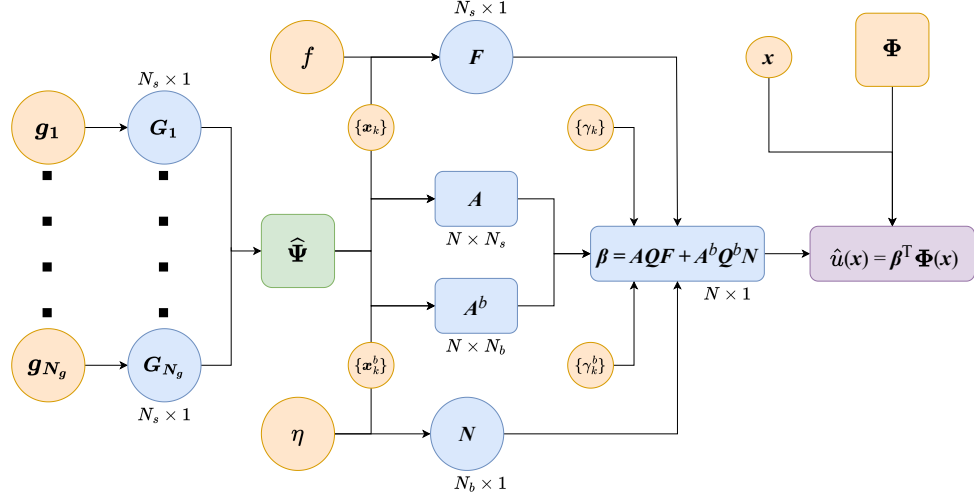


Figure 9.1: Schematic of the PG-VarMiON algorithm for varying PDE parameterization data $\mathbf{g}$ where the only trainable component is the block in green.

9.2 Training

As in Chapter 8, we train PG-VarMiON through supervised learning and thus require a labeled training dataset. To this end, we employ the following procedure:

1. Choose $\{(f^{(j)}, \eta^{(j)}, \mathbf{g}^{(j)})\}_{j=1}^{N_d} \subset \mathcal{F} \times \mathcal{H} \times \mathcal{G}$ sampled independently based on a suitable probability measure on $\mathcal{F} \times \mathcal{H} \times \mathcal{G}$.
2. For each $1 \leq j \leq N_d$ find the solution $u^{(j)} \in \mathcal{V}$ satisfying (6.3) with the PDE data $(f^{(j)}, \eta^{(j)}, \mathbf{g}^{(j)})$. In the absence of a closed-form expression, the solution can be recovered using a high-order accurate numerical solver.
3. Generate the input vectors $\mathbf{F}^{(j)} \in \mathbb{R}^{N_s}$ and $\mathbf{G}^{(j)} \in \mathbb{R}^{N_s \times N_g}$ by evaluating each $f^{(j)}$ and

$\mathbf{g}^{(j)}$ at the sensor nodes $\{\mathbf{x}_i\}_{i=1}^{N_s}$. Generate $\mathbf{N}^{(j)} \in \mathbb{R}^{N_b}$ by evaluating $\eta^{(j)}$ at the boundary sensor nodes $\{\mathbf{x}_i^b\}_{i=1}^{N_b}$.

4. Select a set of output nodes $\{\hat{\mathbf{x}}_l\}_{l=1}^{N_o}$. For each $1 \leq j \leq N_d$, sample the exact/reference solution at these nodes as $u_l^{(j)} = u^{(j)}(\hat{\mathbf{x}}_l)$.
5. Collect all the inputs and output labels to form the training set of N_d tuples

$$\mathbb{S} = \left\{ \left(\mathbf{F}^{(j)}, \mathbf{N}^{(j)}, \mathbf{G}^{(j)}, \{\hat{\mathbf{x}}_l\}_{l=1}^{N_o}, \{u_l^{(j)}\}_{l=1}^{N_o} \right) : 1 \leq j \leq N_d \right\}. \quad (9.1)$$

With this dataset, we train PG-VarMiON just as described in Chapter 8 by selecting an optimal norm to optimize with respect to. The optimization problem (8.14) remains the same.

9.3 Theoretical Analysis

To make the dependence on $f, \eta, \mathbf{g}$ explicit, let us denote by $\bar{u}(\mathbf{x}; f, \eta, \mathbf{g})$ the optimal Petrov-Galerkin solution $\bar{u} \in \bar{\mathcal{V}}$ corresponding to the norm $\|\cdot\|_{\dagger}$ for a given $f \in \mathcal{F}$, $\eta \in \mathcal{H}$, and $\mathbf{g} \in \mathcal{G}$ and the pre-selected trial basis $\Phi(\mathbf{x})$ spanning $\bar{\mathcal{V}}$. We denote the corresponding PG-VarMiON solution as $\hat{u}(\mathbf{x}; \boldsymbol{\theta}, f, \eta, \mathbf{g})$. These solutions can be expressed as

$$\begin{aligned} \bar{u}(\mathbf{x}; f, \eta, \mathbf{g}) &= \left(\mathbf{M}^{-1} \boldsymbol{\ell}_{\Psi(\cdot, \mathbf{g})}(f, \eta) \right)^\top \Phi(\mathbf{x}), \\ \hat{u}(\mathbf{x}; \boldsymbol{\theta}, f, \eta, \mathbf{g}) &= \boldsymbol{\beta}^\top \Phi(\mathbf{x}) = \left(\mathbf{M}^{-1} \boldsymbol{\ell}_{\hat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta) \right)^\top \Phi(\mathbf{x}). \end{aligned} \quad (9.2)$$

Note that the we often move the dependence on $\mathbf{g}$ to Ψ and $\hat{\Psi}$ when working with the solutions $\bar{u}$ and $\hat{u}$, respectively. As in the fixed PDE data $\mathbf{g}$ case, the following lemmas still hold true (with the errors redefined as in Remark 8.1).

Lemma 9.1. *Given any $(f, \eta, \mathbf{g}) \in \mathcal{F} \times \mathcal{H} \times \mathcal{G}$, the PG-VarMiON error can be decomposed as*

$$\mathcal{E}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}) = \mathcal{E}_{\Phi}^2(f, \eta, \mathbf{g}) + \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}). \quad (9.3)$$

Lemma 9.2. *Let $\lambda_{\min}$ and $\lambda_{\max}$ be respectively the smallest and largest (positive) eigenvalues of the symmetric positive definite mass matrix $\mathbf{M}$. Given any $(f, \eta, \mathbf{g}) \in \mathcal{F} \times \mathcal{H} \times \mathcal{G}$, the following bounds hold*

$$\frac{\|\ell_{\Psi(\cdot, \mathbf{g})}(f, \eta) - \ell_{\hat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta)\|_2^2}{\lambda_{\max}} \leq \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}) \leq \frac{\|\ell_{\Psi(\cdot, \mathbf{g})}(f, \eta) - \ell_{\hat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta)\|_2^2}{\lambda_{\min}} \quad (9.4)$$

where $\|\cdot\|_2$ is the usual Euclidean norm.

In order to extend the error analysis to the case where $\mathbf{g}$ is allowed to vary (across samples), we need the following assumptions:

Assumption 9.1.

1. *The solution operator corresponding to the adjoint PDE is stable with respect to the PDE parameterization data $\mathbf{g}$. More precisely, the following type of estimate holds*

$$\|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}')\|_{H^1(\Omega)} \leq C_{stab} \|\mathbf{g} - \mathbf{g}'\|_{\mathcal{G}} \quad \forall 1 \leq i \leq N \quad (9.5)$$

where C_{stab} may depend on Ω, Γ_{η} .

2. *The non-linear neural network of the trained PG-VarMiON that approximates the optimal*

weighting functions is Lipschitz as a function from $\mathcal{G}$ to $H_D^1(\Omega)$

$$\|\widehat{\psi}_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g}')\|_{H^1(\Omega)} \leq L_D \|\mathbf{g} - \mathbf{g}'\|_{\mathcal{G}} \quad \forall 1 \leq i \leq N \quad (9.6)$$

with Lipschitz constant L_D .

3. We additionally make the following covering estimate assumption, which will be necessary in the proofs that follow:

Given any $\epsilon_s > 0$, there exists a set $W := \{\mathbf{g}_j\}_{j=1}^J \in \mathcal{G}$ such that for any $\mathbf{g} \in \mathcal{G}$

$$\|\mathbf{g} - \mathbf{g}_j\|_{\mathcal{G}} < \epsilon_s \text{ for some } 1 \leq j \leq J. \quad (9.7)$$

These assumptions are not unreasonable. The first assumption is essential to ensure that the weighting function recovery process is stable with respect to the PDE data $\mathbf{g}$. The stability would depend on the choice of the optimal $(\cdot, \cdot)_{\dagger}$ inner-product, which defines the right hand side (see (6.10)). The second assumption requires a similar control from the learned $\widehat{\Psi}$, which hinges on the Lipschitz-stability of the neural network with respect to its inputs. It is possible to introduce such a control in neural networks by using gradient penalty terms [43] or using spectral normalization in the network layers [80]. The final assumption implies that the training data covers the input function space $\mathcal{G}$ well enough. Such a condition would be true for instance if $\mathcal{G}$ is compact.

Remark 9.1. We have opted to use the notation $\|\cdot\|_{\mathcal{G}}$ to denote a suitable norm of $\mathbf{g}$ in Ω , since $\mathcal{G}$ can incorporate multiple spaces. Depending on what PDE parametrization data can vary, this norm will change. For example, if $\mathbf{g} = \mathbf{c}$, then $\|\cdot\|_{\mathcal{G}} = \|\cdot\|_{H_{div}^1(\Omega)}$.

Similar to the data stratification (8.20), we again assume that the N_d PDE data samples used to train the PG-VarMiON are stratified in the following manner

$$\{(f^{(1)}, 0, \mathbf{g}^{(1)}), \dots, (f^{(N_f)}, 0, \mathbf{g}^{(N_f)}), (0, \eta^{(1)}, \mathbf{g}^{(N_f+1)}), \dots, (0, \eta^{(N_\eta)}, \mathbf{g}^{(N_d)})\} \quad (9.8)$$

with the addition of varying $\mathbf{g}$ samples that do not need to be specifically separated to samples with zero f and η .

We now state the following theorem.

Theorem 9.1. *Let us assume that Assumption 9.1 holds and that $\psi_i(\cdot, \mathbf{g}), \widehat{\psi}_i(\cdot, \mathbf{g}) \in \mathcal{V} \subset H_D^1(\Omega)$ and $\psi_i(\cdot, \mathbf{g})|_{\Gamma_\eta}, \widehat{\psi}_i(\cdot, \mathbf{g})|_{\Gamma_\eta} \in L^2(\Gamma_\eta)$ for all $1 \leq i \leq N$. Let the training data be stratified according to (9.8). Assume that $\{f^{(j)}\}_{j=1}^{N_f}$ and $\{\eta^{(j)}\}_{j=1}^{N_\eta}$ form orthonormal sets in $L^2(\Omega)$ and $L^2(\Gamma_\eta)$, respectively. With further assumptions as in Theorem 8.1, the following interior and boundary estimates hold true*

$$\begin{aligned} \sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)}^2 &< 2N(C_{stab} + L_D)^2 \epsilon_s^2 + 2\epsilon + 4\lambda_{max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0, \mathbf{g}) \\ &+ 4 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2, \end{aligned} \quad (9.9)$$

$$\begin{aligned} \sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)}^2 &< 2NC_\Omega^2(C_{stab} + L_D)^2 \epsilon_s^2 + 2\epsilon + 4\lambda_{max} \sum_{j=1}^{N_\eta} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}, \mathbf{g}) \\ &+ 4 \sum_{j=1}^{N_\eta} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2. \end{aligned} \quad (9.10)$$

Proof. Given any $\mathbf{g} \in \mathcal{G}$, let $\mathbf{g}_j$ be such that (9.7) holds. Following the proof of Theorem 8.1, we

arrive at the estimates on the error $\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)$

$$\sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0, \mathbf{g}) + 2 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2 \quad (9.11)$$

$$\sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)}^2 < \epsilon + 2\lambda_{\max} \sum_{j=1}^{N_\eta} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}, \mathbf{g}) + 2 \sum_{j=1}^{N_\eta} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2 \quad (9.12)$$

From Assumption 9.1, we have the following

$$\begin{aligned} \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)} &= \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j) + \psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j) + \widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)} \\ &\leq \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)} \\ &\quad + \|\widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)} \\ &\leq \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{H^1(\Omega)} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)} \\ &\quad + \|\widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{H^1(\Omega)} \\ &\leq (C_{stab} + L_D) \|\mathbf{g} - \mathbf{g}_j\|_{\mathcal{G}} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)} \\ &\leq (C_{stab} + L_D) \epsilon_s + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)}. \end{aligned}$$

We can perform a similar analysis

$$\begin{aligned}
\|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)} &= \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j) + \psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j) + \widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)} \\
&\leq \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)} \\
&\quad + \|\widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)} \\
&\leq C_\Omega \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{H^1(\Omega)} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)} \\
&\quad + C_\Omega \|\widehat{\psi}_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{H^1(\Omega)} \\
&\leq C_\Omega (C_{stab} + L_D) \|\mathbf{g} - \mathbf{g}_j\|_{\mathcal{G}} + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)} \\
&< C_\Omega (C_{stab} + L_D) \epsilon_s + \|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)}.
\end{aligned}$$

where we have used a trace inequality for functions in $H_D^1(\Omega)$:

$$\|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)} = \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma)} \leq C_\Omega \|\psi_i(\cdot, \mathbf{g}) - \psi_i(\cdot, \mathbf{g}_j)\|_{H^1(\Omega)}. \quad (9.13)$$

Thus,

$$\|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)}^2 \leq 2(C_{stab} + L_D)^2 \epsilon_s^2 + 2\|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Omega)}^2, \quad (9.14)$$

$$\|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)}^2 \leq 2C_\Omega^2 (C_{stab} + L_D)^2 \epsilon_s^2 + 2\|\psi_i(\cdot, \mathbf{g}_j) - \widehat{\psi}_i(\cdot, \mathbf{g}_j)\|_{L^2(\Gamma_\eta)}^2. \quad (9.15)$$

Combining these estimates with (9.11) and (9.12), we arrive at the following estimates

$$\begin{aligned}
\sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)}^2 &< 2N(C_{stab} + L_D)^2 \epsilon_s^2 + 2\epsilon + 4\lambda_{\max} \sum_{j=1}^{N_f} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f^{(j)}, 0, \mathbf{g}) \\
&\quad + 4 \sum_{j=1}^{N_f} \sum_{i=1}^N \left(\epsilon_{Q,i}[f^{(j)}] \right)^2, \\
\sum_{i=1}^N \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)}^2 &< 2NC_\Omega^2(C_{stab} + L_D)^2 \epsilon_s^2 + 2\epsilon + 4\lambda_{\max} \sum_{j=1}^{N_\eta} \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, 0, \eta^{(j)}, \mathbf{g}) \\
&\quad + 4 \sum_{j=1}^{N_\eta} \sum_{i=1}^N \left(\epsilon_{Q,i}^b[\eta^{(j)}] \right)^2.
\end{aligned}$$

□

Remark 9.2. We used a general trace inequality (9.13) in Theorem 9.1. It can be shown that the trace constant C_Ω typically scales inversely with the length scale of the domain. This constant and the trace inequality can be described more precisely for specific types of domain, especially in the context finite element and isogeometric analysis [25].

Theorem 9.2. Assume Ω is a bounded domain with Lipschitz boundary. Let $f \in \mathcal{F}$, $\eta \in \mathcal{H}$, $\mathbf{g} \in \mathcal{G}$, and $\lambda_{\min}$ and $\lambda_{\max}$ be the smallest and largest (positive) eigenvalues of the symmetric positive definite mass matrix $\mathbf{M}$. Also, assume that the conditions of Theorem 9.1 hold. Then given any $\epsilon > 0$, we can find a large enough training dataset (in accordance to Theorem 9.1) such that

$$\begin{aligned}
\mathcal{E}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}) &\leq \mathcal{E}_{\Phi}^2(f, \eta, \mathbf{g}) + \frac{3}{\lambda_{\min}} \left(2 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} (N(C_{stab} + L_D)^2 \epsilon_s^2 (1 + C_\Omega^2)) \right. \\
&\quad \left. + 2\epsilon + 2\lambda_{\max} \epsilon_T + 2\epsilon_Q \right) + \sum_{i=1}^N \left| \epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta] \right|^2
\end{aligned}$$

where ϵ_T and ϵ_Q are respectively the total training error (8.21) and the total quadrature error (8.22) associated with the training set.

Proof. Following the proof of Theorem 8.2, we obtain

$$\mathcal{E}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}) = \mathcal{E}_{\Phi}^2(f, \eta, \mathbf{g}) + \mathcal{E}_{\Psi}^2(\boldsymbol{\theta}, f, \eta, \mathbf{g}) \leq \mathcal{E}_{\Phi}^2(f, \eta, \mathbf{g}) + \frac{\|\ell_{\Psi(\cdot, \mathbf{g})}(f, \eta) - \ell_{\widehat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta)\|_2^2}{\lambda_{\min}} \quad (9.16)$$

and

$$\begin{aligned} \|\ell_{\Psi(\cdot, \mathbf{g})}(f, \eta) - \ell_{\widehat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta)\|_2^2 &\leq 3 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} \sum_{i=1}^N \left(\|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Omega)}^2 \right. \\ &\quad \left. + \|\psi_i(\cdot, \mathbf{g}) - \widehat{\psi}_i(\cdot, \mathbf{g})\|_{L^2(\Gamma_\eta)}^2 \right) + 3 \sum_{i=1}^N |\epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta]|^2. \end{aligned} \quad (9.17)$$

If the training set is chosen to be large enough (as per Theorem 9.1), we can use (8.21), (8.22), (9.9) and (9.10) with (9.17) to get

$$\begin{aligned} \|\ell_{\Psi(\cdot, \mathbf{g})}(f, \eta) - \ell_{\widehat{\Psi}(\cdot, \mathbf{g}), h}(f, \eta)\|_2^2 &\leq 6 \max\{\|f\|_{L^2(\Omega)}^2, \|\eta\|_{L^2(\Gamma_\eta)}^2\} (N(C_{stab} + L_D)^2 \epsilon_s^2 (1 + C_\Omega^2) \\ &\quad + 2\epsilon + 2\lambda_{\max} \epsilon_T + 2\epsilon_Q) + 3 \sum_{i=1}^N |\epsilon_{Q,i}[f] + \epsilon_{Q,i}^b[\eta]|^2. \end{aligned} \quad (9.18)$$

Finally, combining (9.16) and (9.18) gives us the required result (9.16). $\square$

9.4 Numerical Results

In this section, we train operator networks to learn the solution operator for advection-diffusion problems where the PDE parameterization data varies across samples. Thus, it is the

mapping between the source f and parameterization data $\mathbf{g}$ to the solution u that we are interested in learning (it is also possible to vary η , but we do not consider Neumann boundaries for this set of problems). Through these numerical experiments, we especially demonstrate the ability of PG-VarMiON to learn the optimal weighting functions Ψ (which depend on $\mathbf{g}$) in an unsupervised manner. In all experiments, we choose the optimal norm to be the L^2 norm, i.e., $\|\cdot\|_{\dagger} = \|\cdot\|_{L^2(\Omega)}$.

9.4.1 Training and testing data

The data is generated in the manner of 8.4.1 with the exception that for each sample, the PDE parameterization data $\mathbf{g}$ must be set for each training and test sample. The distribution of $\mathbf{g}$ remains the same for the training and testing data, so we specify $\mathbf{g}$ in the description of each problem.

Generally, $\mathbf{g}$ is selected along with f such that reference data can be generated. To sample $\mathbf{g}$ to construct the vector $\mathbf{G}$, we use the sensor nodes identical to those used to sample f . Thus, in 1D, we use $N_s = 40$ Gauss-Legendre nodes, and in 2D, we use $N_s = 40 \times 40$ tensorized Gauss-Legendre nodes. We note that it is possible to select different nodes for f and $\mathbf{g}$ if necessary.

9.4.2 Network architectures

We describe the network architecture for the various operator networks considered in the numerical experiments. The basic structures are similar to that described in Section 8.4.2 with some changes and additional augmentations for improved performance necessary for these more difficult problems.

PG-VarMiON: Figure 9.1 depicts the structure of this surrogate model. As in Chapter 8,

we use MLPs as the starting point for the architecture of the $\hat{\Psi}$ network. Additionally, the ELU and hat functions are considered as activation functions, but we also introduce a more expressive activation function, which seems to perform well.

This activation function is a B-spline and is inspired by their use in Kolmogorov-Arnold Networks (KANs) [73]. It was demonstrated in [106] that in many machine learning problems, B-spline activation can provide superior performance when used in MLPs. Unlike most other activation functions, B-spline activation itself has learnable parameters. It is a more expensive operation, so rather than every neuron in a hidden layer having its own learnable B-spline basis, we assign one B-spline basis per layer which is shared across all of the neurons. Each neuron, however, is assigned its own set of trainable weights which are used to obtain a linear combination of B-spline evaluations of the neuron's output. This is depicted in Figure 9.2.

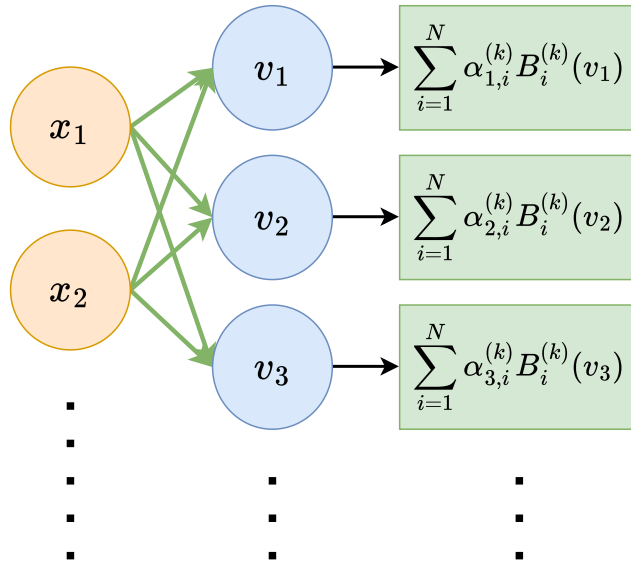


Figure 9.2: The B-spline activation function in the k -th hidden layer of an MLP. Notice that each neuron shares the same B-spline basis, but the linear combination weights are unique to each neuron.

Since the learnable $\hat{\Psi}$ depends on $\mathbf{g}$ instead of just $\mathbf{x}$, we require a balancing between these

two types of input. The spatial component, while important, only has as many features as there are dimensions of the domain. On the other hand, $\mathbf{g}$ will have $N_s \times N_g$ features. To achieve a balance, we use a Fourier encoding to expand the $\mathbf{x}$ input to a vector of sines and cosines of various frequencies evaluated at that point [98]. Then, this encoding is concatenated with the $\mathbf{g}$ data input. The rest of the $\hat{\Psi}$ network is an MLP which produces the approximate optimal weighting functions from this intermediate values.

For the 2D problem we consider, each function in $\mathbf{g}$ sampled on the 2D mesh of sensor nodes can be seen as an image. Thus, it is natural to use convolutions to process the input, but to keep the implementation simple, we exploit the specific structure of $\mathbf{g}$ for the problem to encode the $\mathbf{g}$ data (see Section 9.4.4 for details).

L-DeepONet: The structure is similar to that described in Section 8.4.2. However, with varying PDE parameterization data, we must allow the learnable $\mathbf{B}$ matrix to depend on the data $\mathbf{g}$. Rather than learning a single matrix, we consider an MLP that outputs $\mathbf{B}$ given $\mathbf{g}$ input. As with PG-VarMiON, we consider the more expressive B-spline activation function. Although we have omitted Neumann boundary conditions for this set of problems, their presence means there is a second MLP that produces $\mathbf{B}^b$. We only consider 1D problems at present for L-DeepONet, since the learnable $\mathbf{B}$ matrix is of size $N \times N_s$. This means for a 2D problem with a mesh of sensor nodes of 40×40 and a trial basis dimension of 225, this is a 225×1600 matrix. The resulting networks are either very large or have long training times, and at present, we have not determined the best way to implement this model.

BNet: The structure is similar to that described in Section 8.4.2. Once again, the trial basis Φ is pre-determined and fixed to be the same as that of PG-VarMiON. With varying PDE parameterization data $\mathbf{g}$, we replace the learnable $\mathbf{B}$ matrix with an MLP. With Neumann boundary

conditions, a second MLP that produces $\mathbf{B}^b$ is present. As with L-DeepONet, we only consider BNet for 1D problems.

FNO: The structure is identical to that described in Section 9.4.2 except the varying PDE parameterization data $\mathbf{g}$ must be input as additional channels to the FNO. The rest of the structure does not change. This is the case for the 1D problems and the 2D problem we consider.

9.4.3 1D Problems

The first set of problems we test the operator networks on are one-dimensional with homogeneous Dirichlet boundary conditions. For all of these problems, we choose the $\mathbf{g}$ data that varies across samples to be $\mathbf{c}$ (with the exception of the last problem where κ also varies). We choose $\mathbf{c}$ to be of the form

$$\mathbf{c}(\mathbf{x}) = \frac{1}{30} (\tanh(a(x - b)) + 2), \quad a \sim \mathcal{U}([4, 10]), b \sim \mathcal{U}([0.3, 0.7]) \quad (9.19)$$

or

$$\mathbf{c}(\mathbf{x}) = \frac{(-1)^n}{30} (\tanh(a(-1)^n(x - b)) + 2), \quad n = \{0, 1\}, a \sim \mathcal{U}([4, 10]), b \sim \mathcal{U}([0.3, 0.7]). \quad (9.20)$$

Samples with (9.19) will have boundary layers at the right boundary (and never at the left) since $\mathbf{c} > 0$. Any given sample with (9.20) can have a boundary layer at the right or left boundary since we now have $\mathbf{c} > 0$ or $\mathbf{c} < 0$. Unless otherwise stated, we take $\kappa = 10^{-4}$.

	PG-VarMiON	L-DeepONet	BNet	FNO
Activation Function	B-spline	Φ : Hat B : B-spline	B-spline	GeLU
Cut-off function (8.42)	Yes		No	
p in (8.42)	400		N/A	
Optimizer	AdamW			Adam
AdamW β_1	0.5			N/A
AdamW β_2	0.9			N/A
Weight decay	0			10^{-7}
Epochs	2000			
LR Scheduler	Yes			
Initial LR	10^{-3}			$5 \cdot 10^{-3}$
LR Scheduler step	100			
LR Scheduler γ	0.75			0.9
Batch size	400			

Table 9.1: Summary of training settings for all networks in the 1D advection-diffusion problem with varying parameterization data $\mathbf{g}$.

9.4.3.1 Varying c (right boundary layers)

We consider the advection-dominated problem on $\Omega = [0, 1]$ with c sampled according to (9.19). The training and test datasets are constructed by selecting forcing functions f as described in Section 9.4.1 (along with the varying c). To account for the right boundary layers, we select the orthonormalized 15-dimensional basis described in Section 8.4.3.2 as the Petrov-Galerkin trial basis.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 9.4.2 with 5 hidden layers of widths [230,250,150,100,60] and train with the hyperparameters stated in Table 9.1. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 9.3, where we also compare with the relative finite-dimensional projection error. Generally, the PG-VarMiON's performance is very close to the projection error even with the additional c input. While omitted, other architectures for PG-VarMiON were considered, but this provides the best result. The B-spline activation function in particular significantly boosts performance. In Figure 9.4, we show the f , c , and corresponding PG-VarMiON solutions for 3 samples in DATASET 3. The PG-VarMiON solutions are indistinguishable from the reference solutions.

We plot the true Ψ and the PG-VarMiON approximation $\hat{\Psi}$ for 2 samples in Figures 9.5 and 9.6 (corresponding to the first two rows of Figure 9.4). Recall that the true Ψ basis functions are not included in the training objective, i.e., $\hat{\Psi}$ are learned implicitly. We can observe that the low frequency modes are captured well by PG-VarMiON, while the high frequency modes are qualitatively well approximated with the exception of the very last basis function, which is less accurately resolved. Since most of the optimal weighting functions are correctly captured, the

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	0.63 (0.63)	0.15 (0.15)	2.56 (2.12)
PG-VarMiON	139440	0.65 (0.63)	0.18 (0.16)	3.14 (2.55)
FNO	137119	0.51* (0.50)	0.87 (1.52)	2.96 (2.29)
FNO-UN	137119	0.49* (0.33)	0.71 (0.78)	2.90 (2.27)
L-DeepONet	258070	1.47 (0.59)	1.27 (0.49)	4.95 (3.78)
BNet	140135	0.64 (0.63)	0.23 (0.22)	7.12 (5.82)

Table 9.2: 1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *. FNO-UN denotes FNO trained and tested on un-normalized input data.

PG-VarMiON performs well on the OOD datasets (with respect to f). Further, this demonstrates how PG-VarMiON is able to approximately capture the optimal weighting functions for various samples even as g changes.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO (trained according to Table 9.1). The FNO is trained and tested on normalized and un-normalized input data (in Chapter 8, we always normalize the FNO input, but here we consider both). Table 9.2 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON and BNet yield similar performance on the first two datasets, but BNet does significantly worse on DATASET 3. Both FNO configurations give lower errors than the projection error on DATASET 1 and do significantly worse on DATASET 2. The performance of the FNOs on DATASET 3 is similar to PG-VarMiON. Interestingly, the FNO performs marginally better with un-normalized data. L-DeepONet does not perform as well on the first two datasets and is between PG-VarMiON and BNet on DATASET 3.

PG-VarMiON is not intended to work well if g is OOD, unlike f (and η). This is reflected in Theorem 9.2. See Appendix D for an additional test demonstrating this behavior.

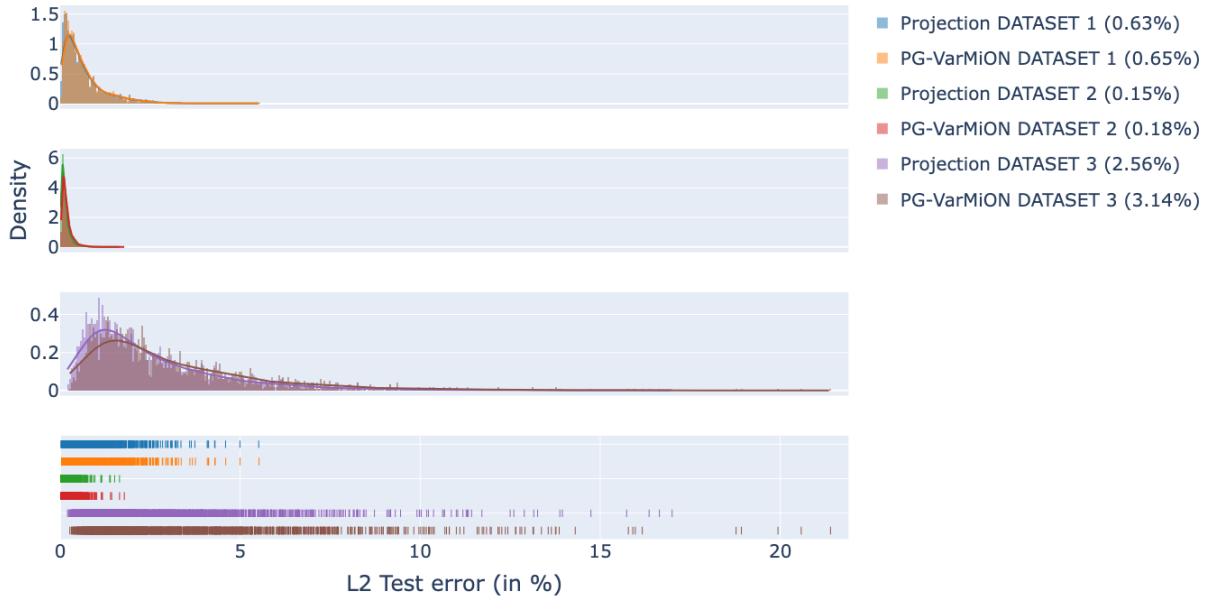


Figure 9.3: 1D advection-diffusion problem with varying c : Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

9.4.3.2 Varying c (left/right boundary layers)

We now consider the same problem but choose c to be of the form (9.20). Since there can now be a boundary layer on either side, we need to augment the previously described basis with a function that has a left boundary layer. We add the function

$$\Phi(x) = \sqrt{2} \frac{(\pi\kappa \sin(\pi x) + c \cos(\pi x) + h(x))}{\pi((\pi\kappa)^2 + c^2)}, \quad (9.21)$$

$$\text{where } h(x) = \frac{-\left(e^{\frac{-c}{\kappa}x} - 1\right)c + \left(e^{\frac{-c}{\kappa}} - e^{\frac{-c}{\kappa}x}\right)c}{1 - e^{\frac{-c}{\kappa}}},$$

and then perform the Gram-Schmidt process to obtain a 16-dimensional orthonormal trial basis.

The resulting trial basis can be seen in Figure 9.7.

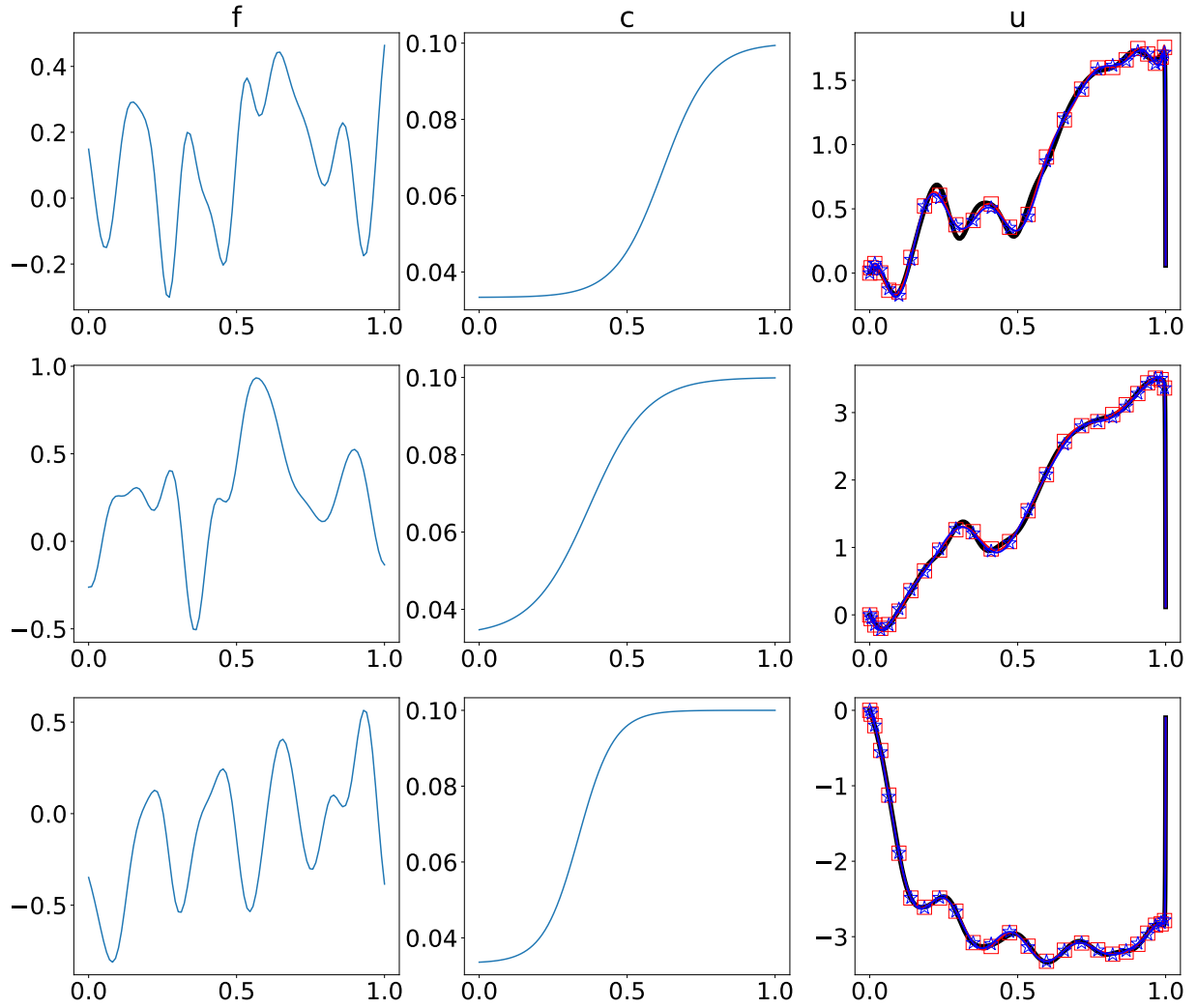


Figure 9.4: 1D advection-diffusion problem with varying c : 3 samples from DATASET 3, with the forcing functions f and velocities c plotted in the first and second columns. The corresponding reference solutions (black lines) are plotted in the third column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).

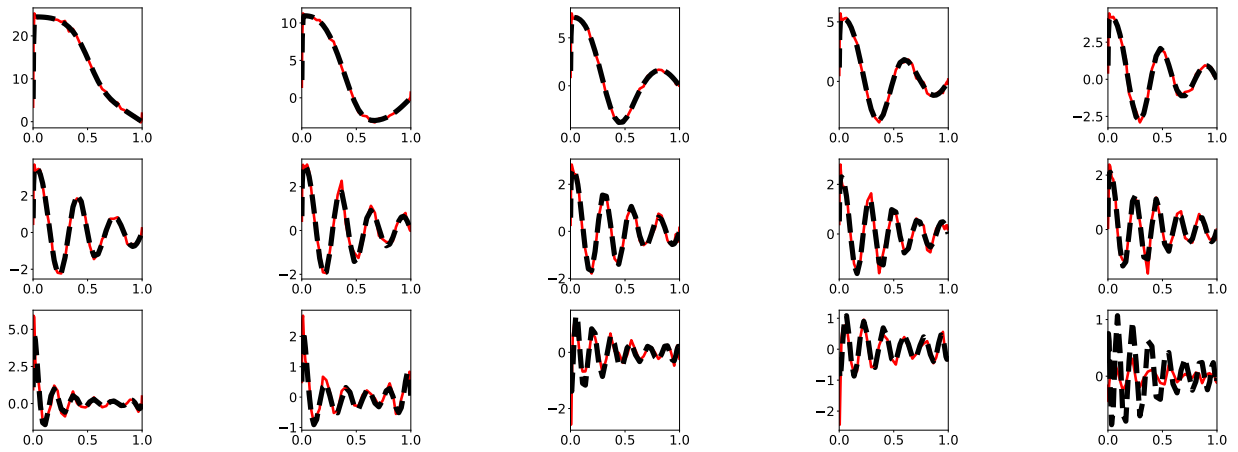


Figure 9.5: 1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.4.

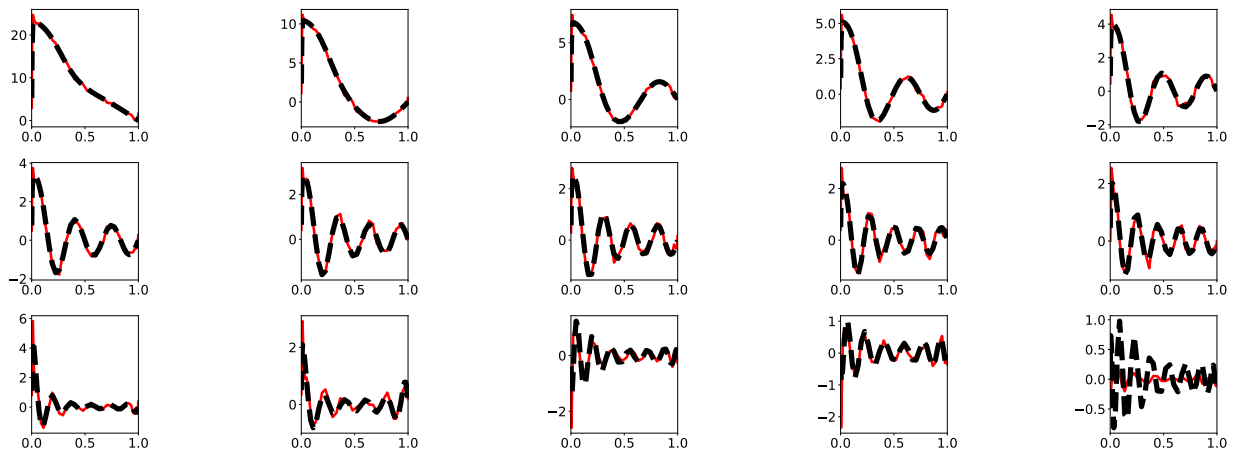


Figure 9.6: 1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.4.

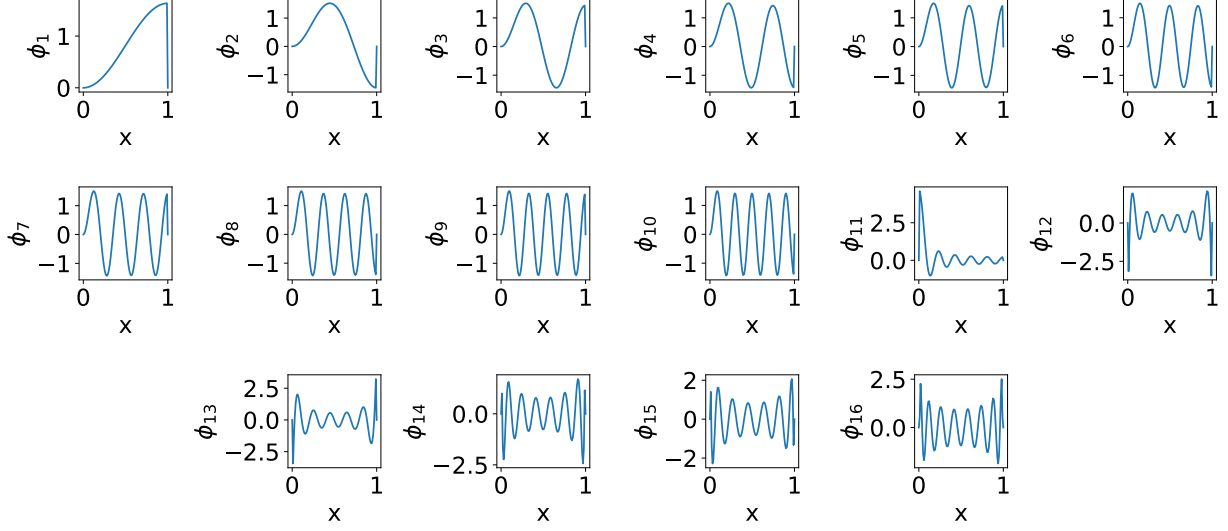


Figure 9.7: The trial basis comprised of sixteen orthonormalized functions used for the 1D advection-diffusion problem. Note the presence of boundary layers near the left and right boundaries.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 9.4.2 with 5 hidden layers of widths [250,300,250,150,80] and train with the hyperparameters stated in Table 9.1. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 9.8, where we also compare with the relative finite-dimensional projection error. Generally, the PG-VarMiON's performance is very close to the projection error even with the added complication of having c such that there are two possible locations of boundary layers. In Figure 9.9, we show the f , c and corresponding PG-VarMiON solutions for 3 samples in DATASET 3. The PG-VarMiON solutions are indistinguishable from the reference solutions.

We plot the true Ψ and the PG-VarMiON approximation $\hat{\Psi}$ for 2 samples in Figures 9.10 and 9.11 (corresponding to the first two rows of Figure 9.9). As in the previous problem, most of the functions are captured well by PG-VarMiON, with the exception of the very last few basis functions, which are less accurately resolved. Since most of the optimal weighting functions

are correctly captured, the PG-VarMiON performs well on the OOD datasets (with respect to f). These samples are particularly interesting because they demonstrate how the appearance of the optimal weighting functions can significantly change when the PDE parameterization data changes. Here, we have the sample in Figure 9.10 that corresponds to a positive velocity, and in Figure 9.11, the velocity is negative. Regardless, PG-VarMiON captures the basis functions very well.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO (trained according to Table 9.1 except FNO uses an LR scheduler γ of 0.75 in this problem). The FNO is trained and tested on normalized and un-normalized input data. Table 9.3 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON and BNet yield similar performance on the first two datasets, but BNet does significantly worse on DATASET 3. The FNOs in this case are generally out-performed by PG-VarMiON except on DATASET 3, where the one with normalized input performs slightly better than PG-VarMiON. Note that in this problem, it is likely that FNO does not beat the projection errors like in the previous problem because the velocity fields are more varied with left and right boundary layers. L-DeepONet does not perform as well on the first two datasets and is similar to BNet on DATASET 3.

9.4.3.3 Varying c , κ

We now consider the same problem where c is of the form (9.19) and κ is also allowed to vary. For each sample, we construct κ by generating a Gaussian random field $\tilde{\kappa}$ with length scale

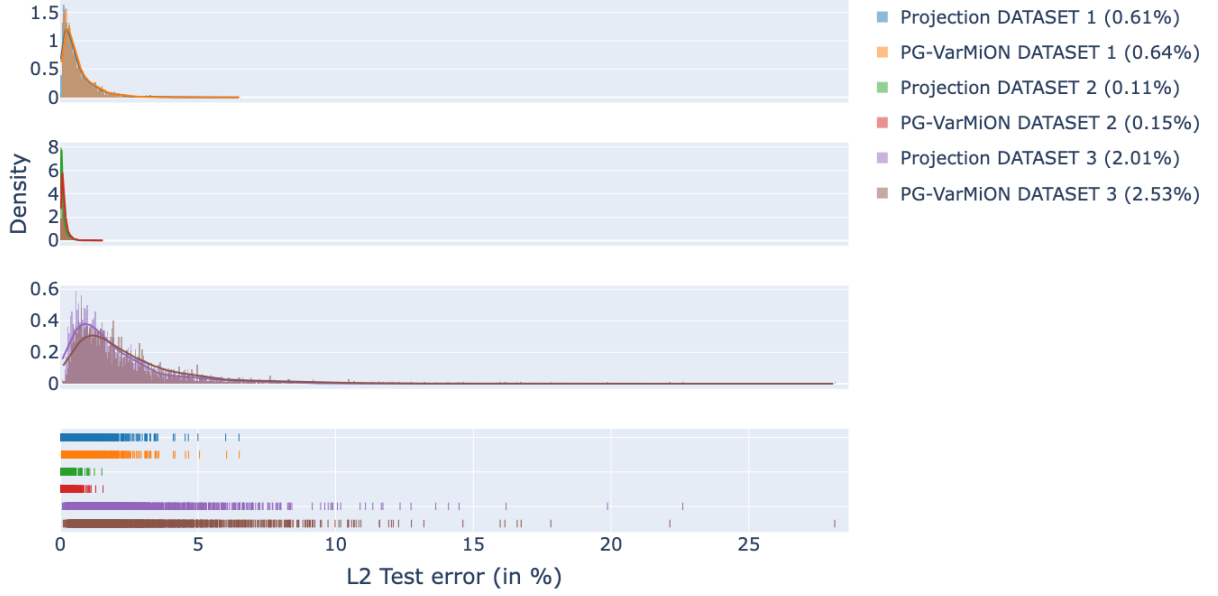


Figure 9.8: 1D advection-diffusion problem with varying c : Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	0.61 (0.63)	0.11 (0.12)	2.01 (1.92)
PG-VarMiON	227161	0.64 (0.63)	0.15 (0.14)	2.53 (2.29)
FNO	234965	0.70 (0.48)	0.72 (0.59)	2.37 (2.09)
FNO-UN	234965	1.42 (1.07)	1.33 (0.96)	3.30 (2.55)
L-DeepONet	432211	1.35 (0.65)	0.93 (0.38)	4.94 (4.13)
BNet	230135	0.63 (0.65)	0.16 (0.22)	5.84 (4.96)

Table 9.3: 1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded. FNO-UN denotes FNO trained and tested on un-normalized input data.

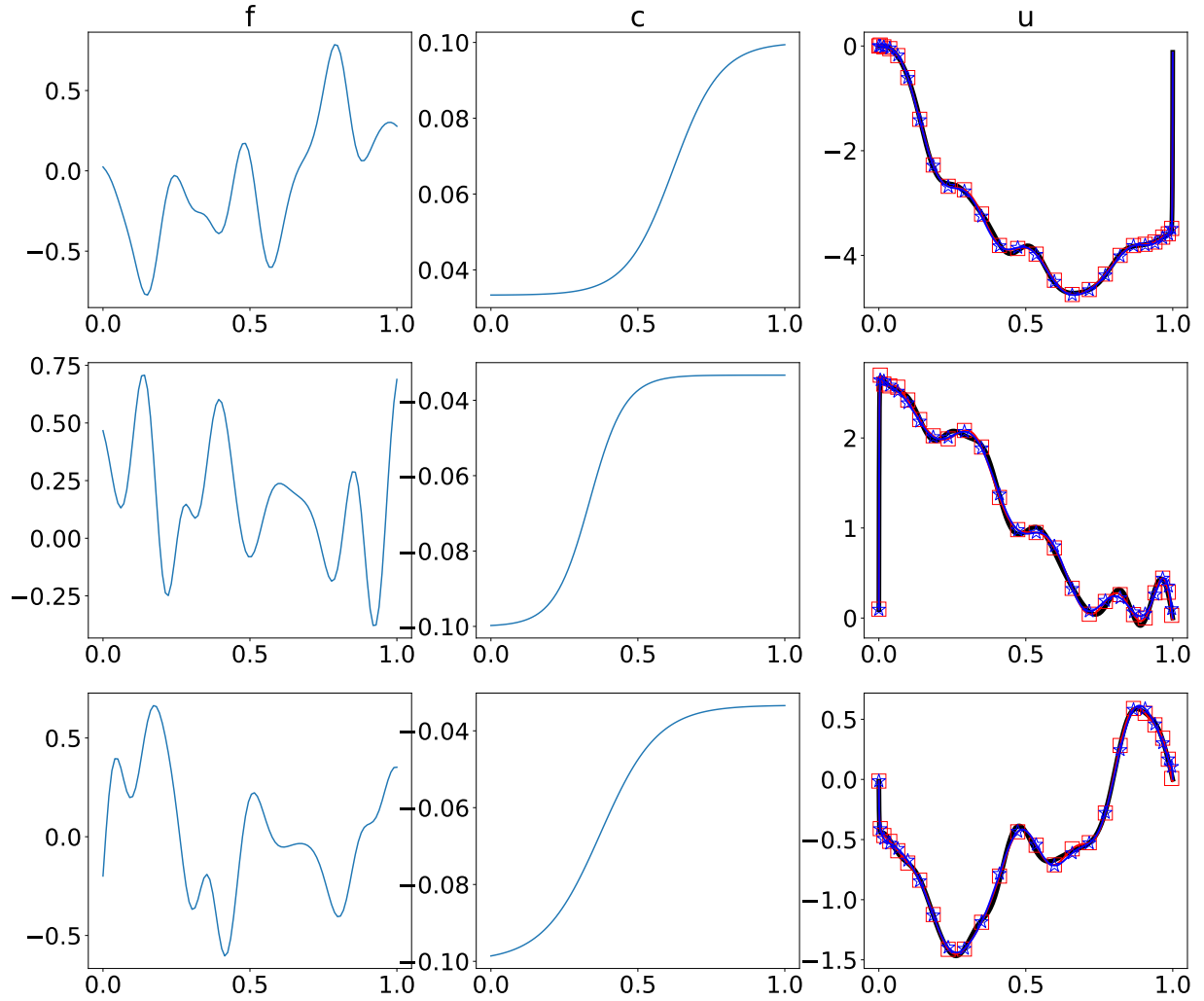


Figure 9.9: 1D advection-diffusion problem with varying c : 3 samples from DATASET 3, with the forcing functions f and velocities c plotted in the first and second columns. The corresponding reference solutions (black lines) are plotted in the third column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).

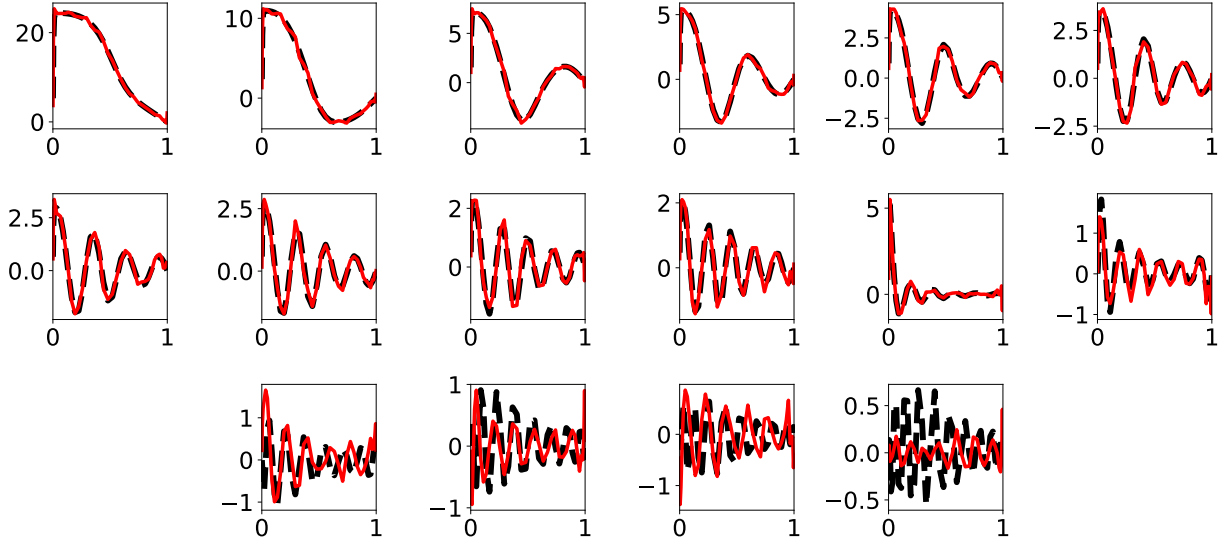


Figure 9.10: 1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.9.

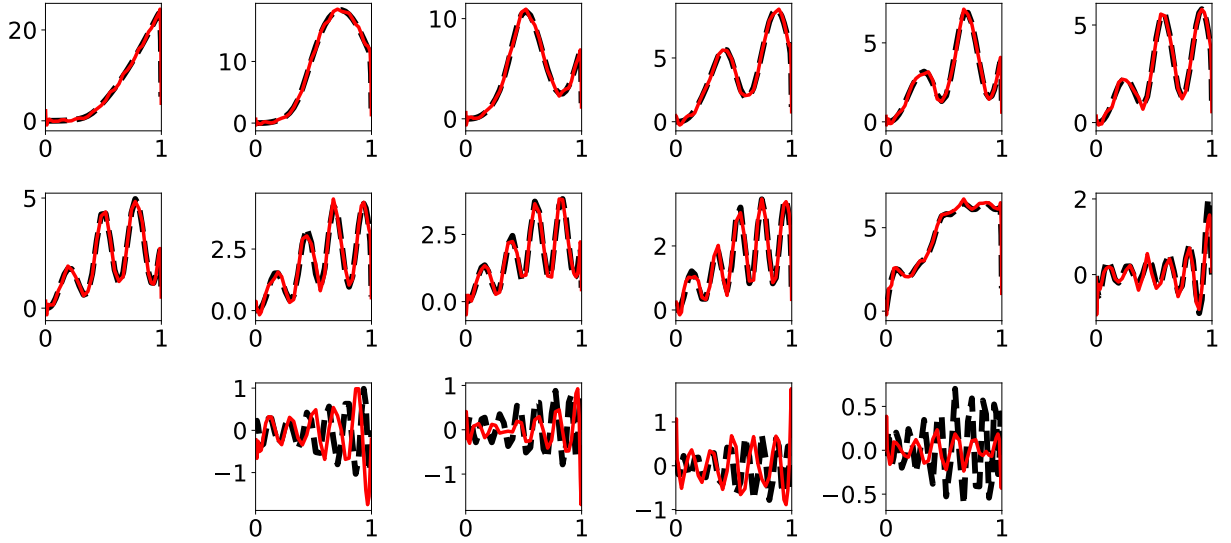


Figure 9.11: 1D advection-diffusion problem with varying c : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.9.

0.1 scaled to be in the range $[0, 1]$. Then, we threshold the function such that

$$\kappa = \begin{cases} 2 \cdot 10^{-4} & \text{if } \tilde{\kappa} > 0.5 \\ 10^{-4} & \text{otherwise} \end{cases}$$

This results in a discontinuous κ . We select the usual 15-dimensional basis as the trial basis since we only have right boundary layers.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 9.4.2 with 5 hidden layers of widths $[250, 300, 250, 150, 80]$ and train with the hyperparameters stated in Table 9.1. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 9.12, where we also compare with the relative finite-dimensional projection error. Generally, the PG-VarMiON's performance is close to the projection error but the performance is worse than the two previous problems. This is due to the fact that now there are two varying fields of PDE parameterization data on which the solution depends, thus making it a harder problem. In Figure 9.13, we show the f , κ , c , and corresponding PG-VarMiON solutions for 3 samples in DATASET 3. The PG-VarMiON solutions are indistinguishable from the reference solutions.

We plot the true Ψ and the PG-VarMiON approximation $\hat{\Psi}$ for 2 samples in Figures 9.14 and 9.15 (corresponding to the first two rows of Figure 9.13). Once again, PG-VarMiON is able to capture most of the optimal weighting functions for different samples. In this problem, we demonstrate that this is possible even with multiple fields of PDE parameterization data.

Next, we compare the performance of PG-VarMiON with a suitable L-DeepONet, BNet, and FNO (trained according to Table 9.1 except FNO uses an initial LR of 10^{-2} in this prob-

lem). The FNO is trained and tested on normalized and un-normalized input data. Table 9.4 compares the mean relative L^2 error with all the methods on each test dataset. We observe that PG-VarMiON and BNet yield similar performance on the first two datasets, but BNet does significantly worse on DATASET 3. L-DeepONet does not perform as well on the first two datasets and is similar to BNet on DATASET 3. The FNO with normalized inputs does the best on all three datasets, beating the projection error on all except DATASET 2. The lacking performance of PG-VarMiON here is likely due to the decreased efficacy of the trial basis when used for this problem. The trial basis is originally designed for constant c and constant κ , so by varying c and κ , the projection error has significantly increased.

However, we do observe that FNO is not as robust as PG-VarMiON when the amount of training samples is reduced. Figure 9.16 shows that the FNO error in particular significantly increases as the training set shrinks. PG-VarMiON's performance, on the other hand, remains much steadier and generally beats FNO when using only 2000 or less training samples. The results also indicate that adding more training samples from the chosen training distribution may not help to close the gap between the PG-VarMiON and projection errors. A better network architecture or augmenting the training dataset with different types of samples (perhaps OOD) may help.

9.4.4 2D Problem

We now consider the two-dimensional advection-diffusion equation with homogeneous Dirichlet boundary conditions where $\kappa = 10^{-3}$ and the velocity field c varies across samples. As in Section 8.4.4, we consider a vortex, but this time, the location of the vortex's center will

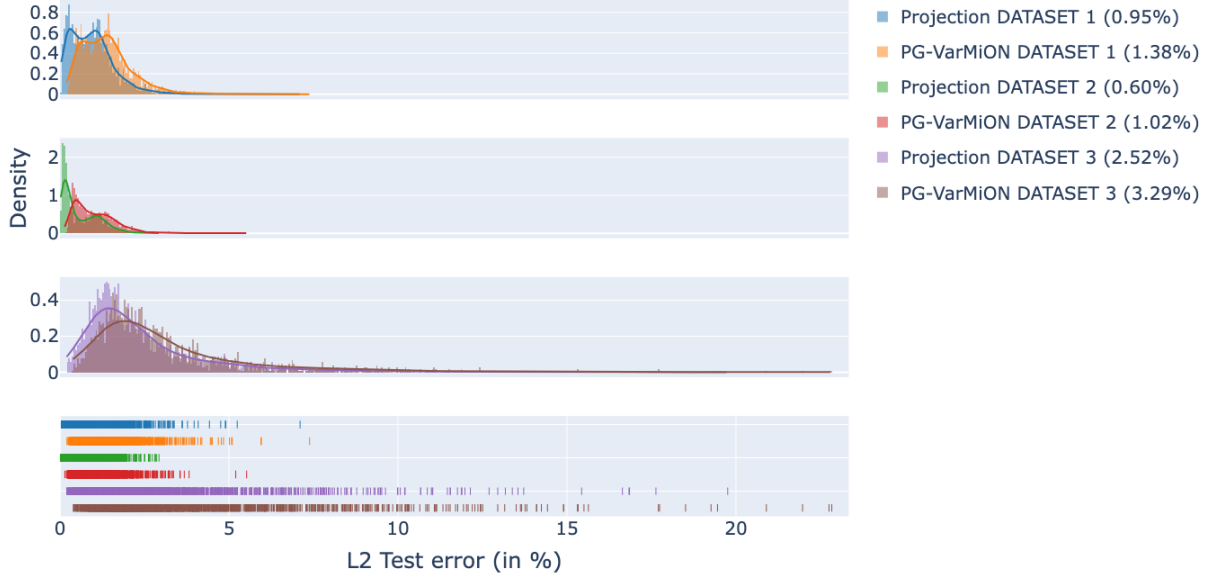


Figure 9.12: 1D advection-diffusion problem with varying c and κ : Histograms with rug plots showing the relative L^2 error (in %) with the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	0.95 (0.67)	0.60 (0.55)	2.52 (2.11)
PG-VarMiON	237080	1.38 (0.76)	1.02 (0.61)	3.29 (2.66)
FNO	239697	0.74* (0.61)	0.83 (1.04)	2.18* (1.91)
FNO-UN	239697	2.23 (1.32)	2.09 (1.61)	4.89 (4.03)
L-DeepONet	441970	1.96 (1.07)	1.39 (0.70)	4.78 (4.01)
BNet	239975	1.73 (1.07)	1.25 (0.74)	6.70 (5.91)

Table 9.4: 1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 test error (in %) with the various methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded, and errors lower than the projection error are marked with a *. FNO-UN denotes FNO trained and tested on un-normalized input data.

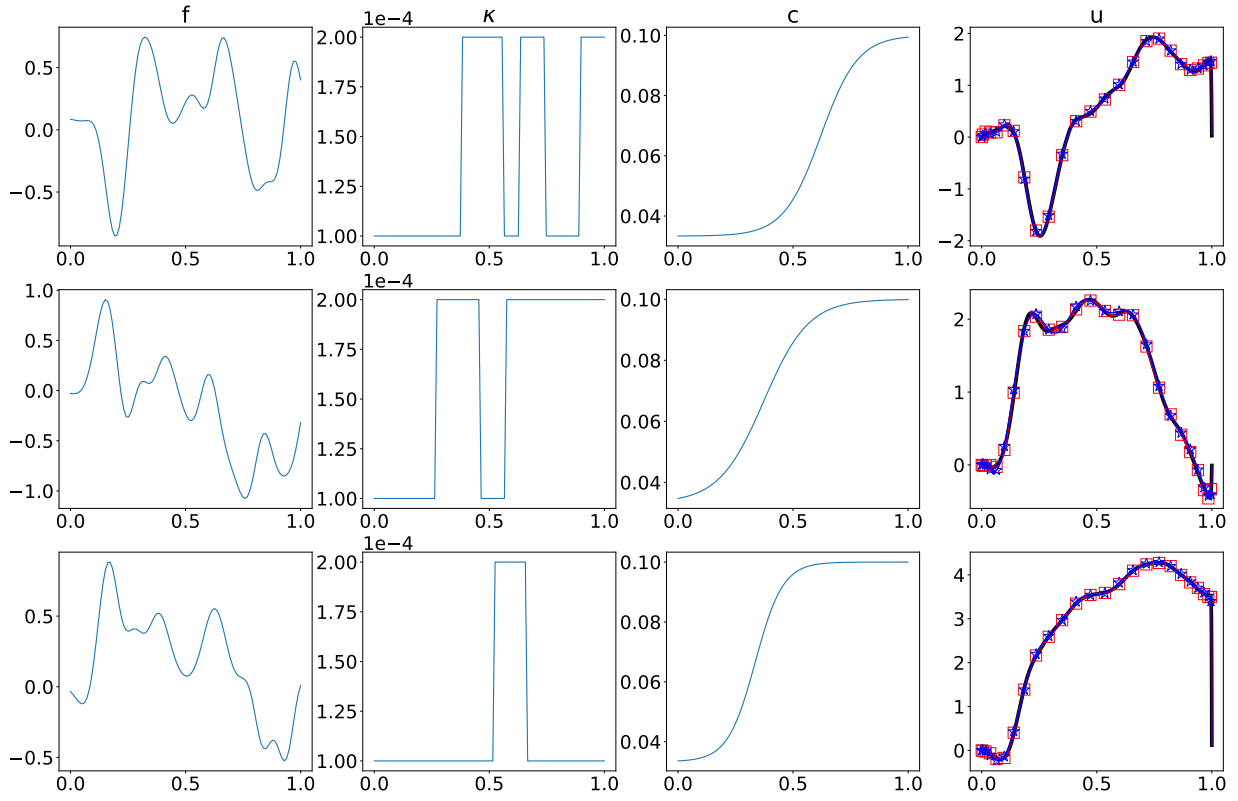


Figure 9.13: 1D advection-diffusion problem with varying c and κ : 3 samples from DATASET 3, with the forcing functions f , κ , and velocities c plotted in the first, second, and third columns. The corresponding reference solutions (black lines) are plotted in the fourth column along with the projected solutions (red squares) and the PG-VarMiON approximations (blue stars).

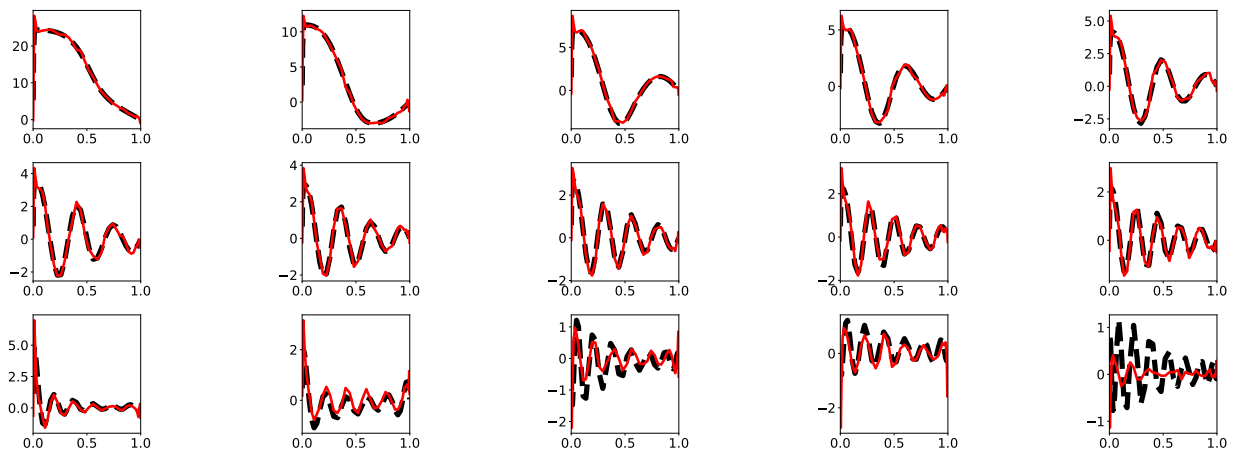


Figure 9.14: 1D advection-diffusion problem with varying c and κ : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the first sample in Figure 9.13.

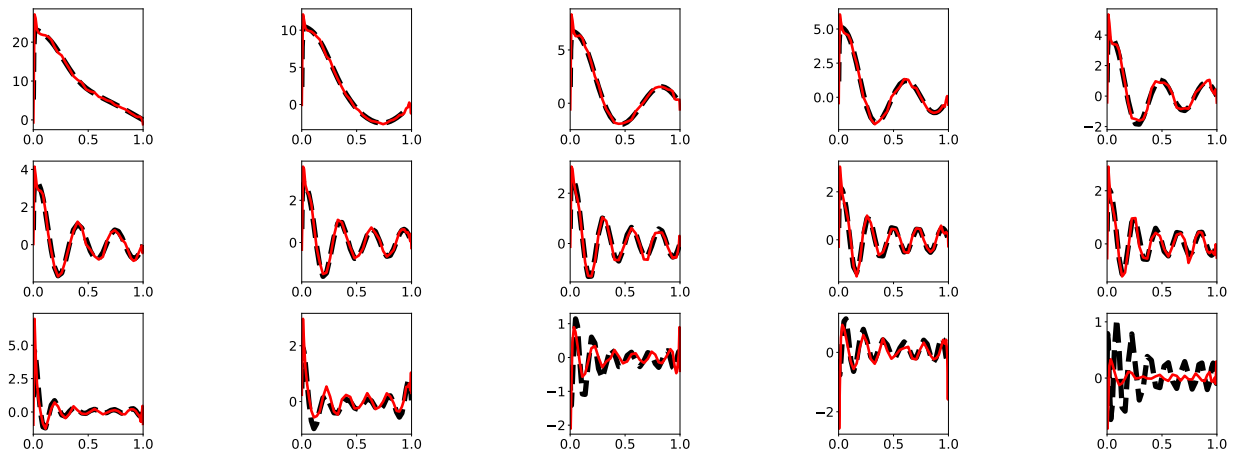


Figure 9.15: 1D advection-diffusion problem with varying c and κ : Comparison of the exact weighting functions Ψ (black) and the approximations $\hat{\Psi}$ yielded by PG-VarMiON (red) corresponding to the second sample in Figure 9.13.

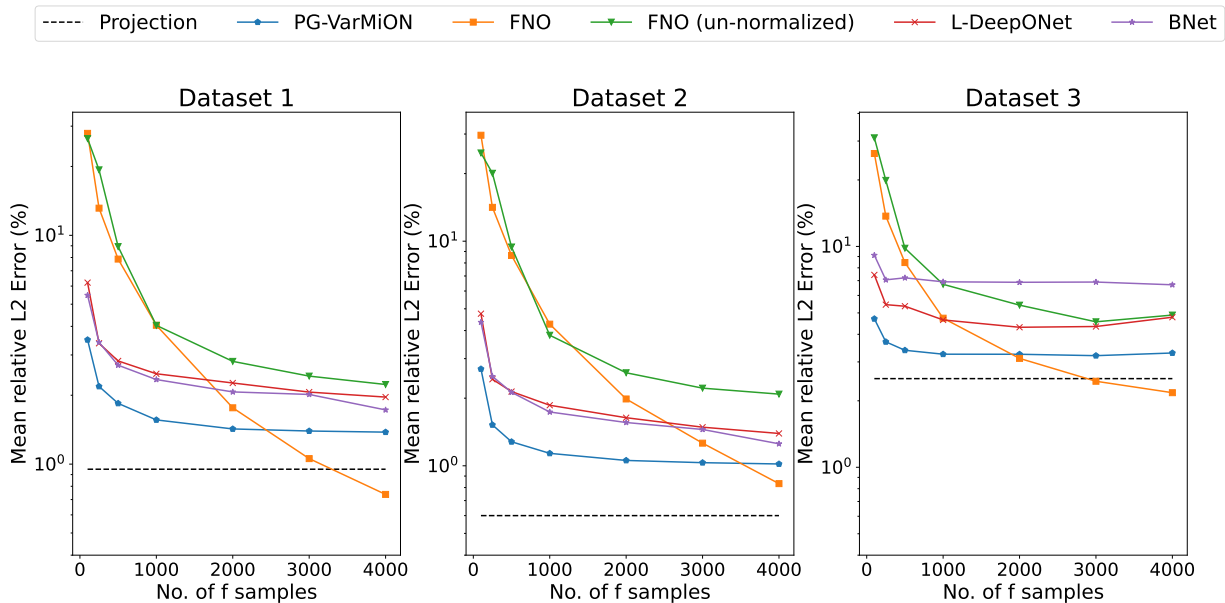


Figure 9.16: 1D advection-diffusion problem with varying c and κ : Mean relative test error (in %) with the various operator models as the number of f samples in the training set is varied. The projection error for each dataset is shown in black.

differ for each sample. Thus, $\mathbf{c}$ for a given sample is expressed as

$$\begin{aligned} c_1 &= -5(y - b) \exp\left(\frac{1 - (5(x - a))^2 - (5(y - b))^2}{2}\right), \\ c_2 &= 5(x - a) \exp\left(\frac{1 - (5(x - a))^2 - (5(y - b))^2}{2}\right), \end{aligned} \quad (9.22)$$

$a, b \sim \mathcal{U}([0.25, 0.75])$

where we restrict the vortex centers such that they do not appear too close to the boundaries. Note that the velocity's magnitude does not change from sample to sample; all that varies is the location of the vortex. As mentioned in Section 9.4.2, we observe that the structure of the $\mathbf{c}$ data can be encoded. In particular, the information of $\mathbf{c}$ as prescribed in (9.22) can be reduced to two features, the coordinates of the vortex center. Thus, we can use an MLP to process the input. For the trial basis, we again opt to generate a 225-dimensional tensorized cubic B-spline basis (see Section 8.4.4.1) and orthonormalize it with the Gram-Schmidt process.

We construct a PG-VarMiON where $\mathcal{N}$ is as described in Section 9.4.2 with 6 hidden layers of widths [60,120,200,200,200,200] and train with the hyperparameters stated in Table 9.5. The trained network is then used on the three test datasets. The histograms (with rug plots) of the relative L^2 test errors are shown in Figure 9.17, where we also compare with the relative finite-dimensional projection error. Surprisingly, PG-VarMiON has the weakest performance on the in-distribution data with lower errors on the OOD data. This is a much more challenging problem than previously considered because it is multi-dimensional and the velocity field $\mathbf{c}$ is varying. We stress that these are preliminary results and that with better implementations, the PG-VarMiON performance can be improved.

We also train an FNO according to Table 9.5 with normalized input data. As mentioned in Section 9.4.2, we omit L-DeepONet and BNet. Table 9.6 compares the mean relative L^2 error

	PG-VarMiON	FNO
Activation Function	ELU	GELU
Cut-off function (8.42)	Yes	No
p in (8.42)	100	N/A
No. of Parameters	197645	196455
Optimizer	AdamW	Adam
AdamW β_1	0.5	N/A
AdamW β_2	0.9	N/A
Weight decay	0	10^{-8}
Epochs	4000	
LR Scheduler	Yes	
Initial LR	10^{-3}	10^{-2}
LR Scheduler step	200	100
LR Scheduler γ	0.9	
Batch size	50	

Table 9.5: Summary of training settings for PG-VarMiON and FNO in the 2D advection-diffusion problem with homogeneous Dirichlet boundary conditions.

with all considered methods on each test dataset. We observe that PG-VarMiON has significantly lower error than FNO on all three datasets. We remark that as with PG-VarMiON, it is likely the FNO error can be improved as well with additional tuning for this challenging problem.

Next, we take a closer look at 3 test samples from DATASET 3, whose forcing functions and velocity fields are shown in Figure 9.18, with the corresponding solutions compared in 9.19. From these contour plots, the reference, projection, PG-VarMiON, and FNO solutions look very similar. We also plot the pointwise errors in Figure 9.20 and the solutions extracted along 1D slices in Figure 9.21. We observe in these figures that PG-VarMiON capture the general shape of the reference solution in these 1D slices. However, we do observe that there are areas with high error due to the fact that introducing varying g data makes this a much more challenging

Method	No. of Parameters	Dataset 1	Dataset 2	Dataset 3
Projection	-	2.17 (0.97)	1.32 (0.68)	2.23 (1.20)
PG-VarMiON	197645	7.21 (3.62)	3.97 (2.20)	5.82 (3.52)
FNO	196455	10.74 (5.75)	15.73 (13.72)	11.35 (6.46)

Table 9.6: 2D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) with the methods. The standard deviation of error is in parentheses. The lowest operator network errors for each test set are bolded.

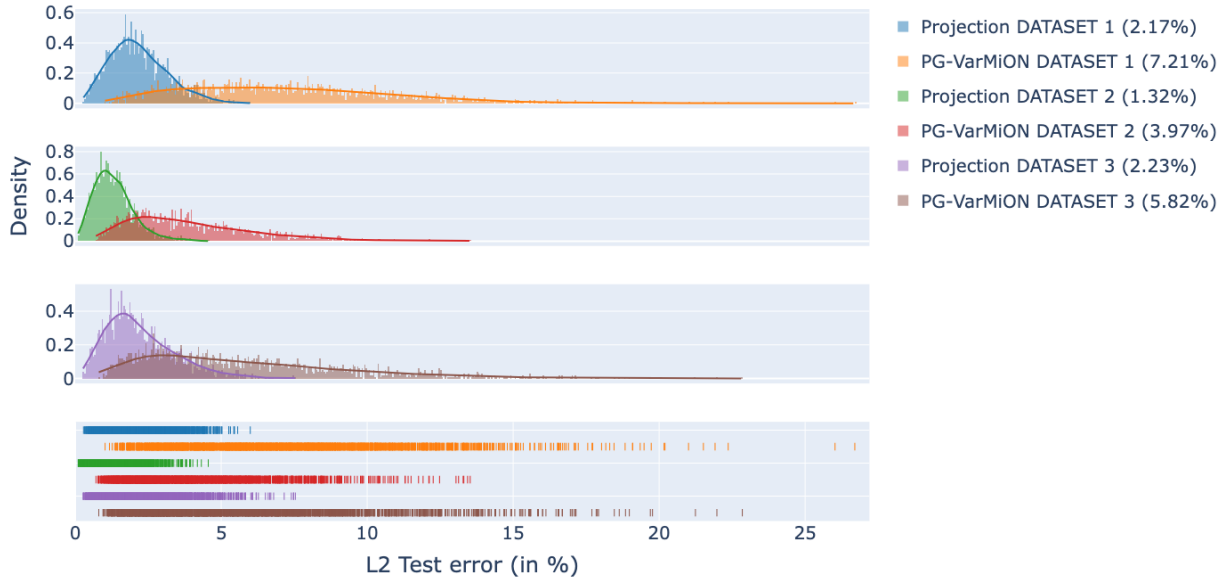


Figure 9.17: 2D advection-diffusion problem with varying c : Histograms with rug plots showing the relative L^2 error for the projected solution $\bar{u}$ and PG-VarMiON solution $\hat{u}$. The average error for each dataset is shown in parentheses.

problem. FNO appears to not perform quite as well as PG-VarMiON.

We depict the expected weighting functions corresponding to the first sixteen trial basis functions and the PG-VarMiON approximations for 2 samples in Figures 9.22 and 9.23 (corresponding to the first two columns of Figure 9.18). We observe that the PG-VarMiON is able to qualitatively learn the optimal weighting functions, despite not being shown the true Ψ while training. This demonstrates PG-VarMiON is able to approximately capture two-dimensional optimal weighting functions for various samples even as g changes.

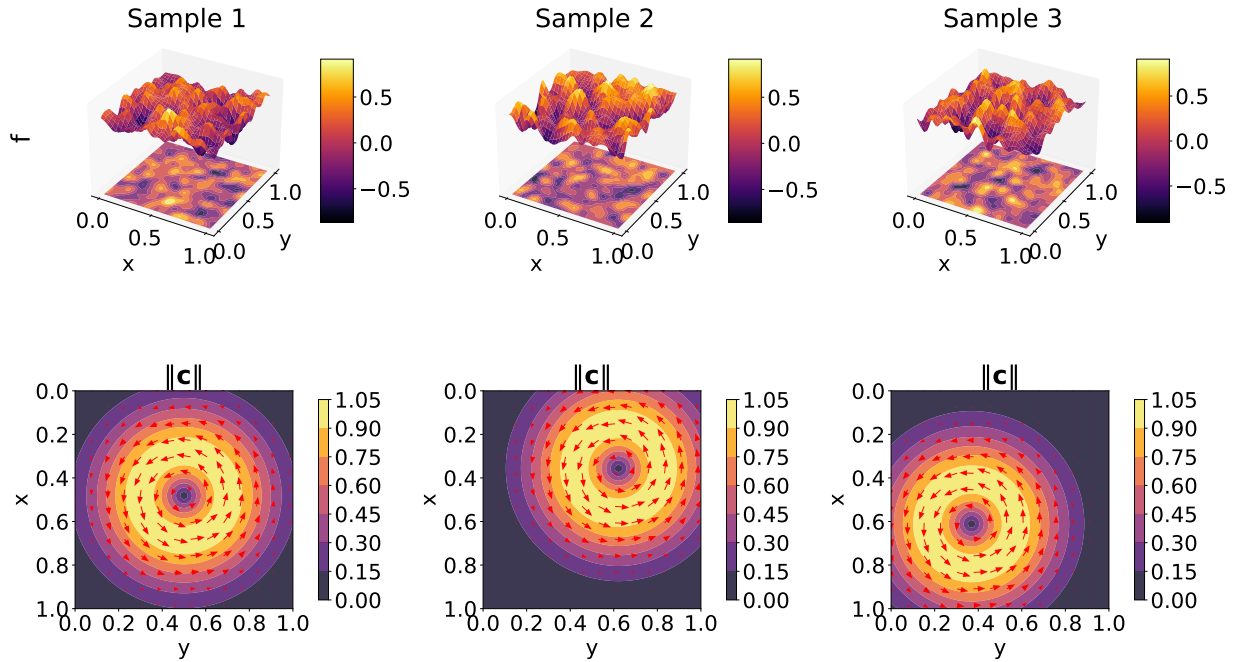


Figure 9.18: 3D plots with contours of the forcing functions for 3 test samples as well as the magnitude of the velocity. The red arrows indicate the direction of the velocity.

9.5 Computational resources

Generating the necessary large quantities of data and actually training the operator networks are expensive processes. It is generally not advisable to run these routines on a local machine unless it is sufficiently powerful. Thus, we have elected to use a high-performance computing cluster (HPC) to perform both of these tasks. In particular, the training of the networks is performed with an A100 GPU such that CUDA can be used for optimal performance. This significantly reduces the training time and makes it feasible to train many configurations of the models to find suitable hyperparameters. Specifically, we have used the UMD Zartan and the UMD Math Department’s HPC clusters. Data generation in particular can be time-consuming with two-dimensional datasets taking on the order of 20-30 hours to fully generate.

Table 9.7 shows the approximate computational runtimes for the operator networks for

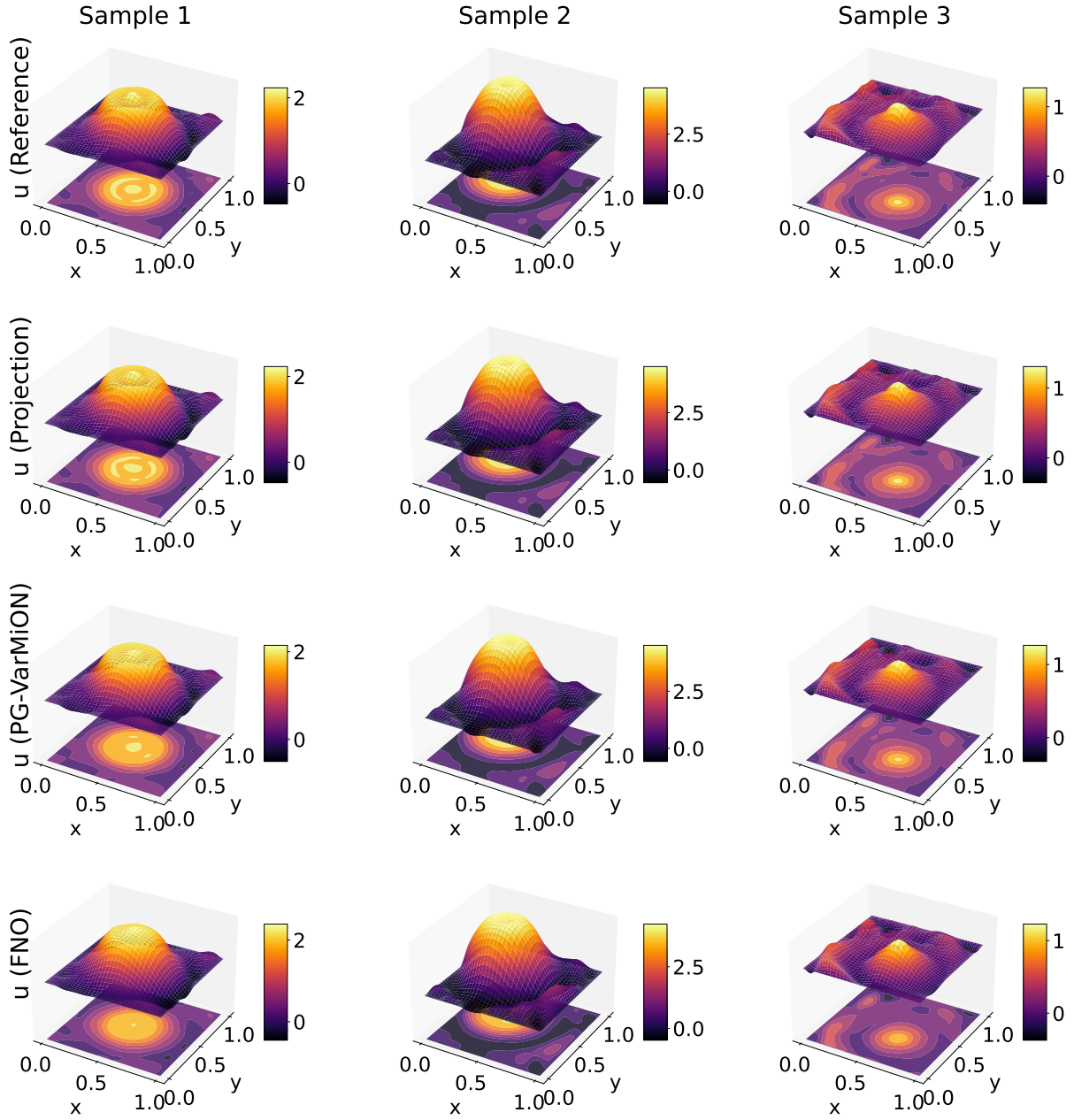


Figure 9.19: 3D plots with contours of the reference solutions, projection, PG-VarMiON, and FNO approximations for 3 test samples.

three problems. Generally, PG-VarMiON has to construct the $\mathbf{A}$ (and $\mathbf{A}^b$) matrix, which involves evaluating all of the approximate optimal weighting functions at the quadrature nodes. This can be an expensive process which the other operator networks do not perform. Thus, it makes sense that PG-VarMiON can be slower to evaluate than other operator networks. This occurs to a much

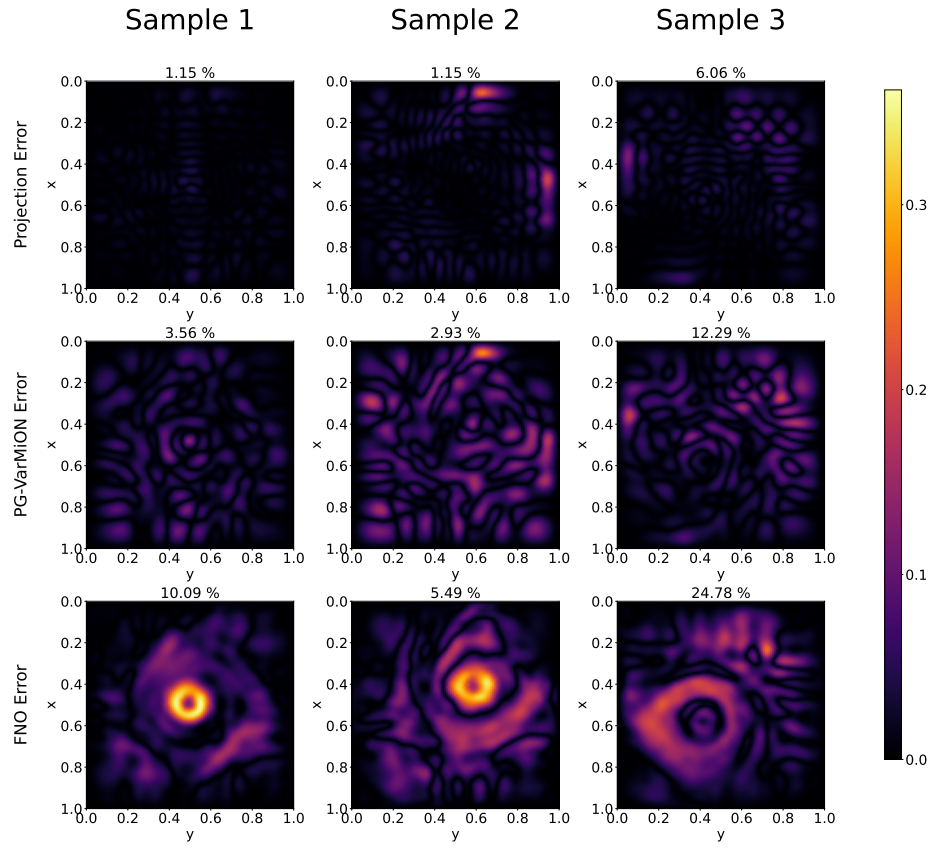


Figure 9.20: Plots of the errors of the 3 test samples for the projection, PG-VarMiON, and FNO approximations.

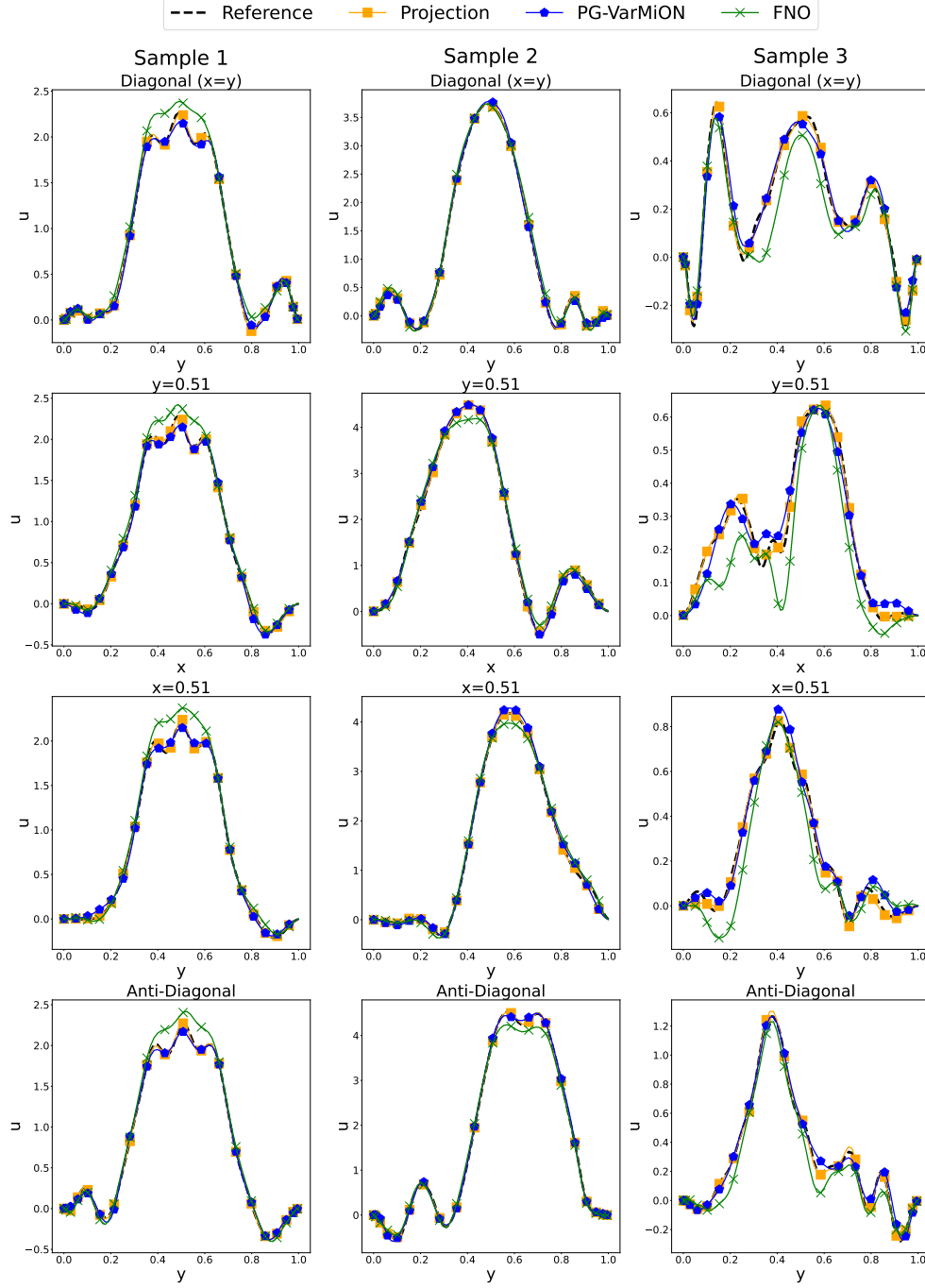


Figure 9.21: 1D slices of the 3 test samples for the reference solutions, projection, PG-VarMiON, and FNO approximations. Slices are for the diagonal $x = y$ and anti-diagonal $x = 1 - y$ and the lines $y = 0.51$, $x = 0.51$.

greater extent as the neural network approximating the optimal weighting function becomes more complicated, as seen with the training time for the problem with varying PDE parameterization

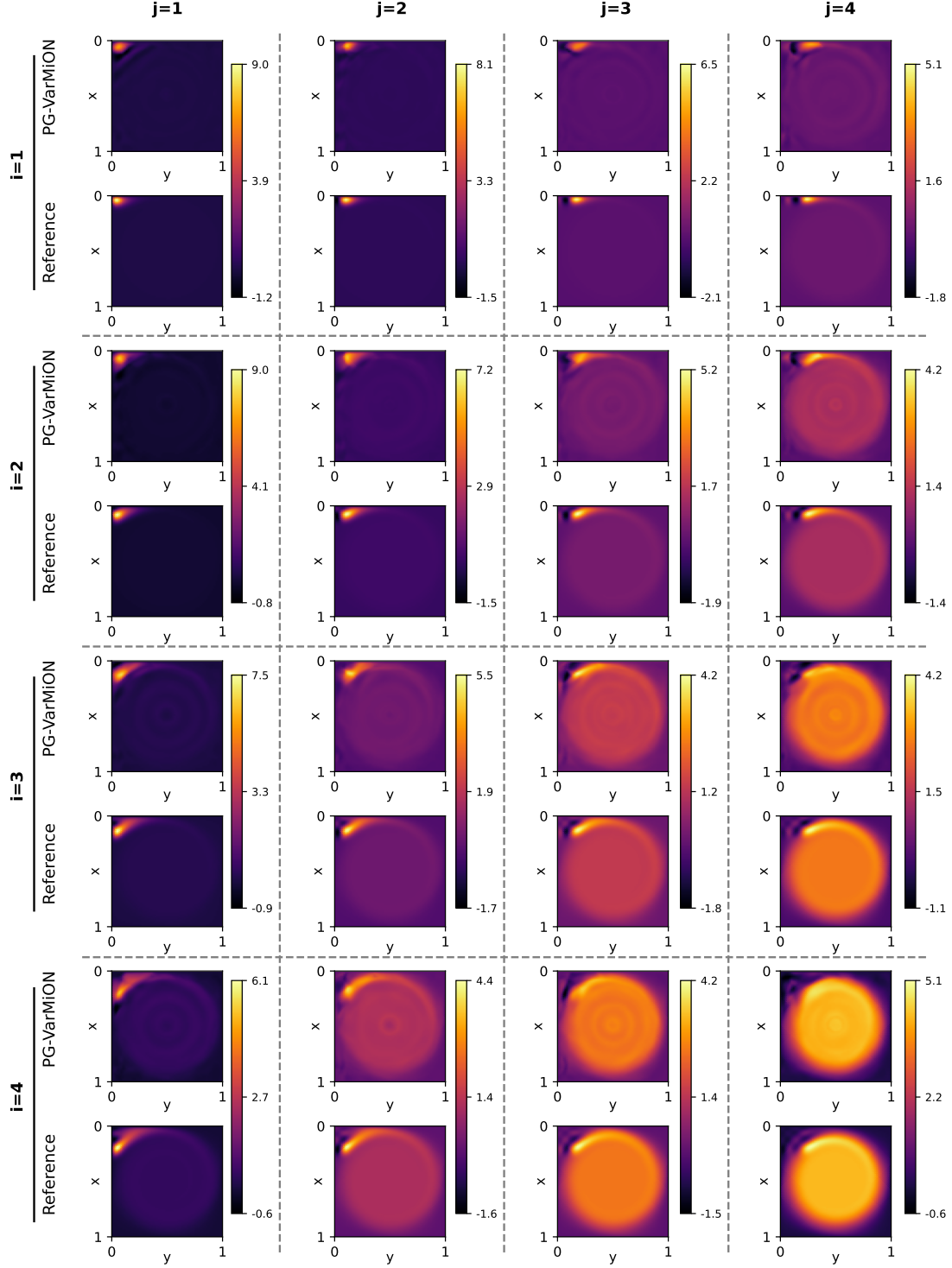


Figure 9.22: Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$. These correspond to the first sample in Figure 9.18.

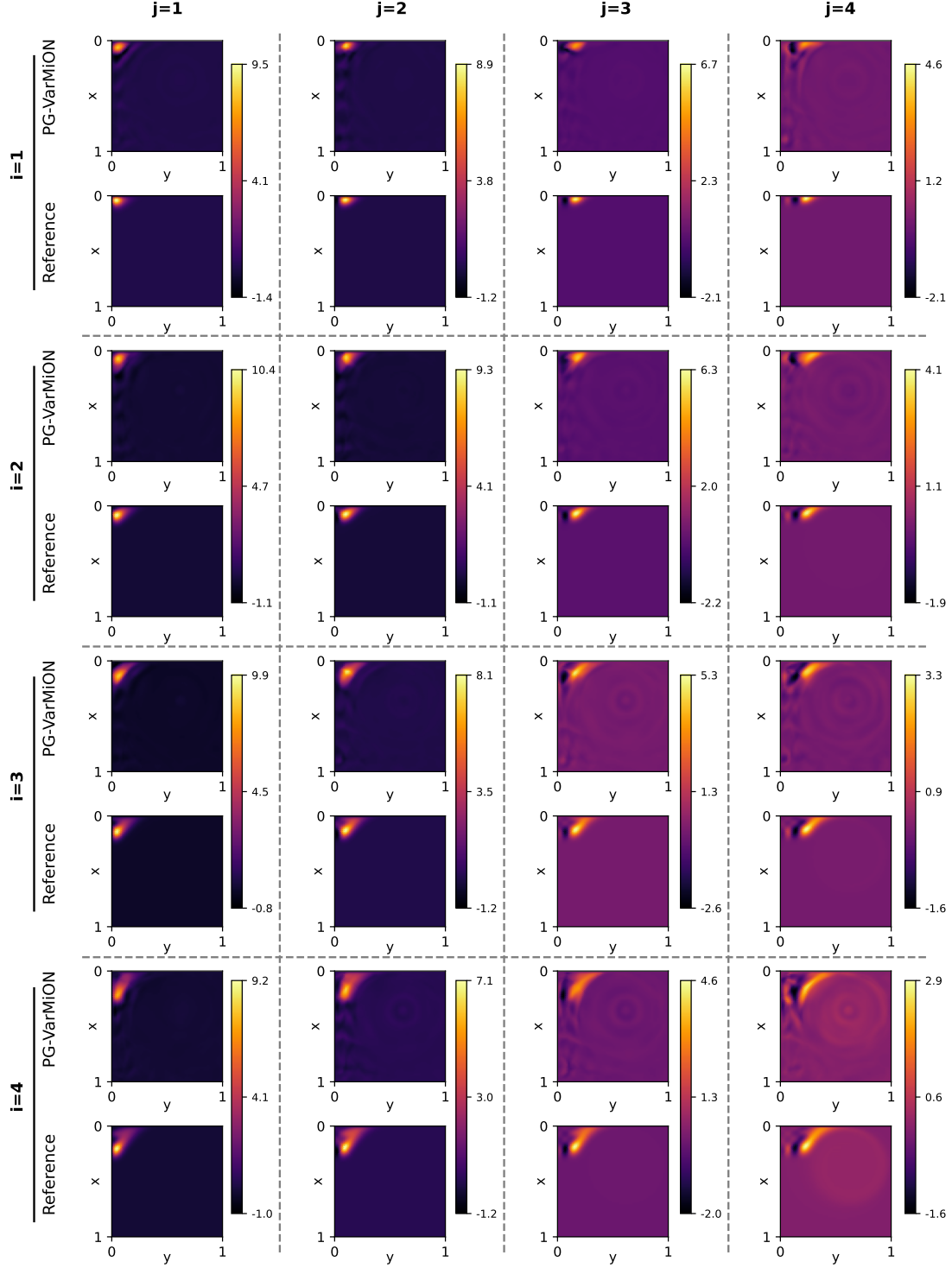


Figure 9.23: Expected and PG-VarMiON approximations for some of the optimal weighting functions, for the functions that are tensor products of the first four 1D B-spline bases $1 \leq i, j \leq 4$. These correspond to the second sample in Figure 9.18.

Method	1D Advection-Diffusion Problem (Section 8.4.3.2)	2D Mixed BC Problem (Section 8.4.4.3)	1D varying c, κ (Section 9.4.3.3)
PG-VarMiON	280	2200	5790
BNet	180	252	455
L-DeepONet	275	630	470
FNO	220	3000	180

Table 9.7: Approximate computational times (in seconds) for training the various operator networks for selected problems.

data $\mathbf{g}$. In contrast, BNet and L-DeepONet have more straight-forward architectures and so do not have such long training times for these problems. FNO can be trained rapidly for 1D problems, but in 2D, there is a significant increase in the size of the input data, resulting in an increase in the training cost.

Although there is an increased training time when using PG-VarMiON, the numerical results shown previously indicate that this structure can allow for better generalization even when minimal training data is used, thus giving it an advantage over other operator network frameworks.

9.6 Comparison to spline-based FEM

As an additional study, we compare the accuracy and inference time of PG-VarMiON with a standard method for solving PDEs by specifically comparing with Nutils, a spline-based solver. We will perform these comparisons by using Nutils in two different ways: (i) directly solve for the solution and (ii) within the Petrov-Galerkin formulation to approximate Ψ .

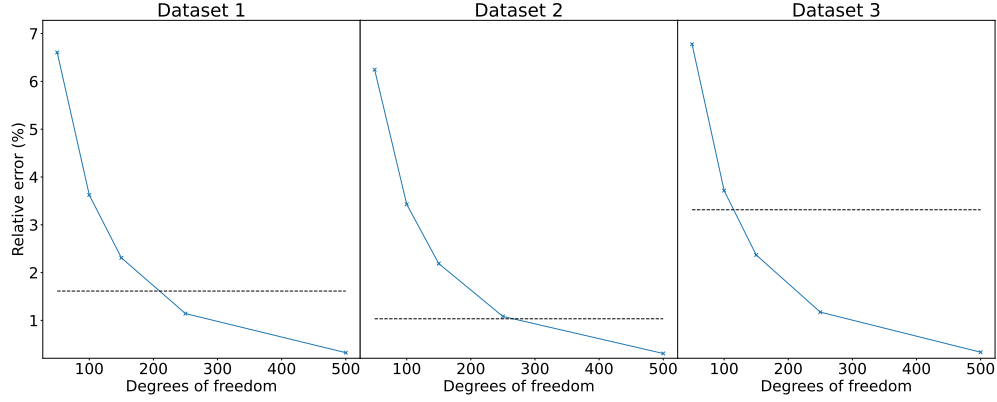


Figure 9.24: 1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 error (in %) for the Nutils solution as the number of degrees of freedom increases for 50 samples in each dataset. The mean PG-VarMiON performance is shown in black.

9.6.1 Nutils: direct computation

Figure 9.24 shows the performance of the Nutils solver to directly approximate the solution for each dataset in the advection-diffusion problem with varying c , κ (see Section 9.4.3.3) as the number of degrees of freedom changes (the number of degrees of freedom for one dimension is the sum of the number of knots and degree of the B-splines). For this study, we use cubic B-splines and vary the number of degrees of freedom by increasing the number of knots. The Nutils solver has a significantly larger solve time than the inference time of operator networks. Table 9.8 shows the time to obtain a single solution with the various methods for samples from Dataset 1. Clearly, the Nutils solver is significantly slower than the surrogate models. Moreover, the operator networks offer an efficient way to solve many samples simultaneously by batching and are faster when computational resources are limited. However, if sufficient computational resources are available, a standard solver (like Nutils) is expected to outperform operator networks.

We also perform the same comparison for a two-dimensional problem where $\mathbf{g}$ is not varying. Figure 9.25 shows the performance of the Nutils solver for each dataset in the 2D advection-

Nutils d.o.f.	Inference Time (s)
50	0.154
100	0.163
150	0.165
250	0.175
500	0.202

(a)

Operator Network	Inference Time (s)
PG-VarMiON	0.011
L-DeepONet	0.003
BNet	0.003
FNO	0.0008

(b)

Table 9.8: 1D advection-diffusion problem with varying c and κ : Inference times (in seconds) for a single Dataset 1 sample for (a) Nutils and (b) various operator networks.

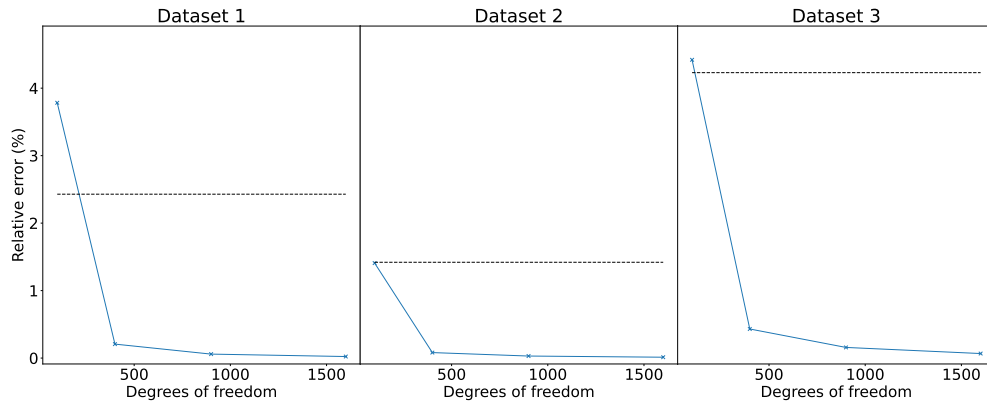


Figure 9.25: 2D advection-diffusion problem: Comparison of mean relative L^2 error (in %) for the Nutils solution as the number of degrees of freedom increases for 50 samples in each dataset. The mean PG-VarMiON performance is shown in black.

diffusion problem with homogeneous Dirichlet boundary conditions (see Section 8.4.4.2) as the number of degrees of freedom changes. We observe that solutions more accurate than PG-VarMiON can be obtained with relatively few degrees of freedom on some datasets (due to the decreased accuracy of PG-VarMiON on the two-dimensional problem). However, the Nutils solver has a significantly slower solve time than the inference time of operator networks. Table 9.9 shows the time to obtain a single solution with the various methods for samples from Dataset 2. Clearly, the Nutils solver is significantly slower than the surrogate models.

Nutils d.o.f.	Inference Time (s)
100	0.178
400	0.236
900	0.342
1600	0.503

(a)

Operator Network	Inference Time (s)
PG-VarMiON	0.023
L-DeepONet	0.023
BNet	0.019
FNO	0.005

(b)

Table 9.9: 2D advection-diffusion problem: Inference times (in seconds) for a single Dataset 2 sample for (a) Nutils and (b) various operator networks.

9.6.2 Nutils: Petrov-Galerkin formulation

We can also examine the cost and accuracy of the Petrov-Galerkin formulation when we compute the optimal weighting functions with Nutils. First, we examine our one-dimensional problem. Figure 9.26 shows the average computational time of the 15 optimal weighting functions in Nutils for different numbers of degrees of freedom as well as the accuracy of using these functions to compute the Petrov-Galerkin solution. Here, 50 samples from Dataset 1 are used. It is clear that Nutils can generate more accurate solutions than PG-VarMiON without too many degrees of freedom, but we note that it is over 200 times more expensive. Here, the size of the basis of optimal weighting functions is 15, and we can expect the cost will become more expensive as the basis size increases.

We perform the same comparison for our two-dimensional problem. Figure 9.27 shows the average computational time of the optimal weighting functions in Nutils for different numbers of degrees of freedom as well as the accuracy of using these functions to compute the Petrov-Galerkin solution. Here, 50 samples from Dataset 2 are used. It is clear that Nutils can generate more accurate solutions than PG-VarMiON without too many degrees of freedom, but we note that it is over 20000 times more expensive. Here, the size of the basis of optimal weighting

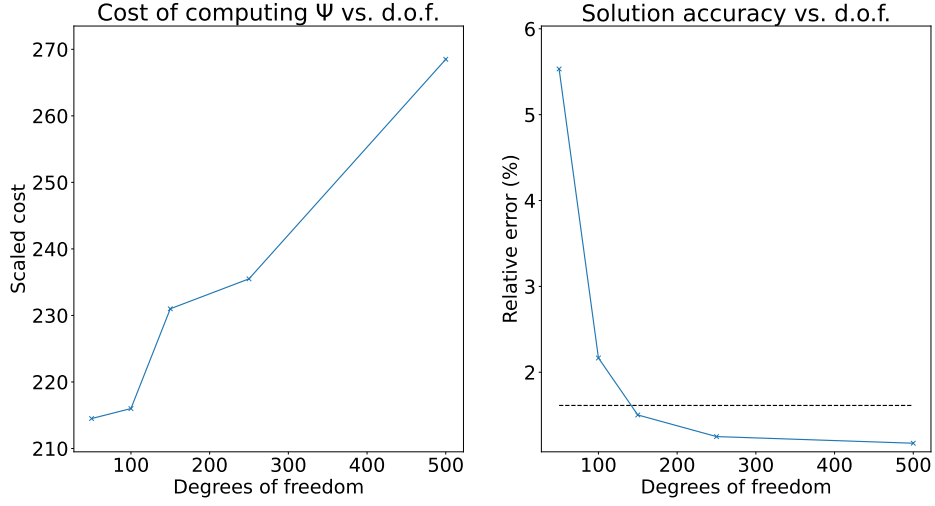


Figure 9.26: 1D advection-diffusion problem with varying c and κ : Comparison of computation time for the optimal weighting functions with Nutils for varying degrees of freedom and accuracy of the Petrov-Galerkin formulation using these computed weighting functions. The Nutils cost is scaled by the cost of PG-VarMiON (actual cost of PG-VarMiON: 0.01 s).

functions is 225, which is why the cost is much more significant than for the one-dimensional problem. We note that for this problem without varying $\mathbf{g}$ that these computations would only occur once since the optimal weighting functions are the same for each sample. However, if $\mathbf{g}$ is allowed to vary, this cost must be borne by the Nutils solver for each new $\mathbf{g}$.

9.7 Error Estimate Numerics

As a final study, we examine what the general error terms look like in the error estimate Theorem 9.2 for the advection-diffusion problem with varying c , κ (see Section 9.4.3.3). Table 9.10 shows some of the important terms in the error estimate for this problem. We observe that the total training error ϵ_T is the dominant term while the total quadrature error ϵ_Q is negligible. To ascertain an estimate for ϵ_s , we approximate the smallest distance from the training dataset for each of 100000 randomly generated in-distribution $\mathbf{g}$ samples for both c and κ in their suitable

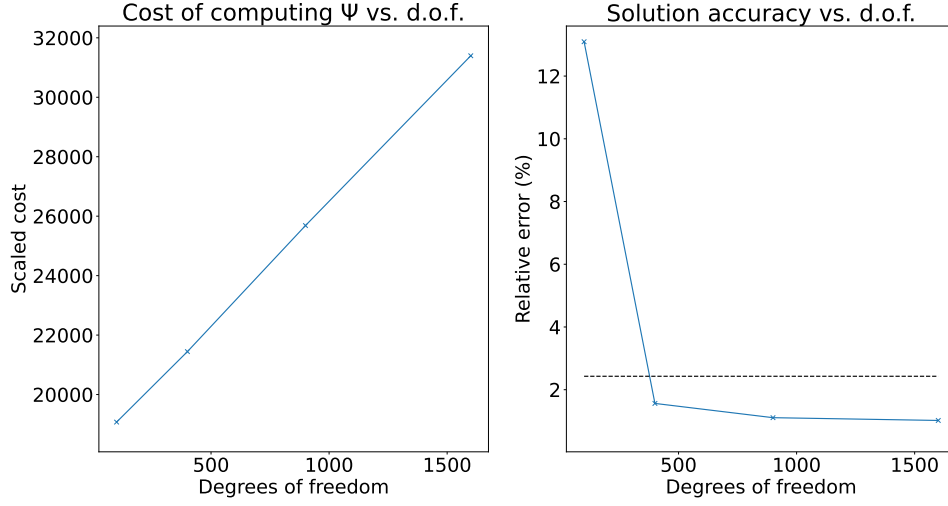


Figure 9.27: 2D advection-diffusion problem: Comparison of computation time for the optimal weighting functions with Nutils for varying degrees of freedom and accuracy of the Petrov-Galerkin formulation using these computed weighting functions. The Nutils cost is scaled by the cost of PG-VarMiON (actual cost of PG-VarMiON: 0.01 s).

ϵ_T	1.10
ϵ_Q	9.4e-3
ϵ_s for c in $L^2([0, 1])$ norm	5.7e-4
ϵ_s for κ $L^\infty([0, 1])$ norm	1.0e-4
ϵ_s for κ in $L^2([0, 1])$ norm	4.5e-5

Table 9.10: 1D advection-diffusion problem with varying c and κ : Sources of error in Theorem 9.2.

norms. Then, we select the largest of these minimal distances.

An interesting implication of Theorem 9.2 is that it seems to indicate that as the size of the trial basis N increases, the gap between the projection and PG-VarMiON errors increases. In fact, as can be seen in Figure 9.28, we observe that as N increases the gap in performance between the projection and PG-VarMiON marginally expands. This indicates a subtle point that while the PG-VarMiON error decreases with an increasing trial basis size, the gap between it and the projection error (the lower bound of the error estimate) may increase. To potentially remedy this, a better training set can be chosen to reduce ϵ_s , but this is to be investigated in the future.

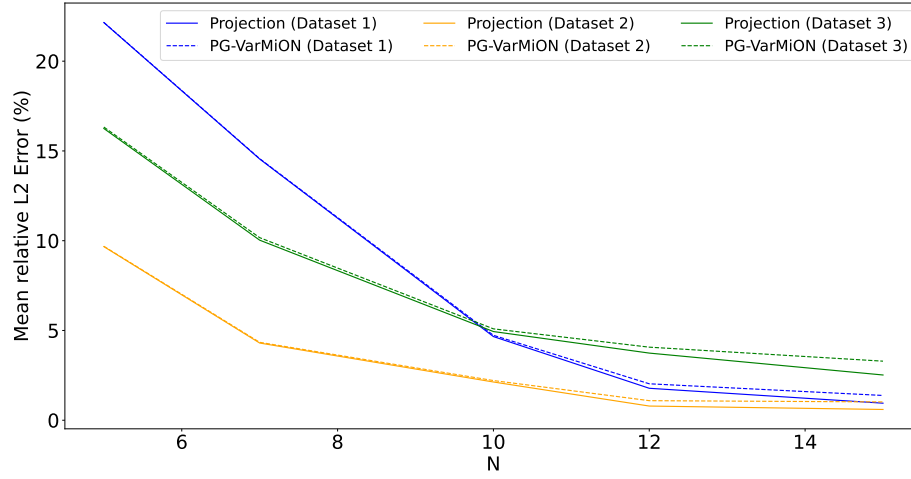


Figure 9.28: 1D advection-diffusion problem with varying c and κ : Comparison of mean relative L^2 test error (in %) between projection and PG-VarMiON for different trial basis sizes for the three test datasets.

Remark 9.3. Since ϵ_T and ϵ_Q are sums over all of the training samples, these terms can only get bigger as the training dataset increases. However, the error estimate is derived with a special choice of orthonormal f and η samples, which is not considered in this numerical example. Thus, there is a need to sharpen the error estimates to accommodate for this, which will be investigated in the future.

Chapter 10: PG-VarMiON Conclusion

We have proposed a novel framework to design operator networks for linear elliptic PDEs by emulating the optimal Petrov-Galerkin variational form. The solution of this variational form is the projection of the infinite-dimensional weak solution of the PDE onto a given finite-dimensional function space, consequently the best approximation in the norm inducing the projector. This optimal solution can be recovered if we are able to explicitly construct the set of optimal weighting functions. However, with the exception of simple problems, the constructions of these weighting functions are unavailable.

The proposed PG-VarMiON emulates the symmetrized Petrov-Galerkin formulation of the PDE, and recovers the optimal solution given a source function f and boundary flux η . We can also consider varying PDE parameterization data $\mathbf{g}$ and have this as input. Further, by training on a suitable dataset of solutions paired with relevant data, the PG-VarMiON is also able to learn the structure of the optimal basis functions in an unsupervised manner. Thus, the PG-VarMiON is capable of generalizing to out-of-distribution data far beyond existing operator learning frameworks.

We also derive explicit estimates for the generalization error, which is bounded from below by the projection error, and from above by the sum of the projection error, the quadrature error (due to finite-dimensional network input), and the error in approximating the weighting functions.

In the case of varying data $\mathbf{g}$, the error is additionally bounded from above by the covering estimate for $\mathbf{g}$ in the training dataset.

Taking the advection-diffusion equation as a canonical but non-trivial example, we present detailed numerical results to demonstrate the efficacy of our approach. In particular, we show that:

- Given a good trial basis $\Phi(\mathbf{x})$, the PG-VarMiON is able to approximate the optimal (projected) solution accurately.
- Since the PG-VarMiON learns the structure of the weighting functions, it is capable of generalizing to out-of-distribution samples, in contrast to other popular operator network frameworks.
- By embedding the Petrov-Galerkin structure in the network, we are able to significantly reduce the data-complexity when it comes to training the operator network. Furthermore, the specialized structure allows us to control the overall size of the network by allocating all the trainable weights to learn the weighting functions. The size advantage is particularly clear when considering 2D problems with static $\mathbf{g}$ data.
- Unlike the other operator networks shown for comparison, the number of sensor nodes used to sample f and η (which the PDE solution depends on linearly) does not impact the number of parameters.

10.1 Future Directions

There are several extensions possible using the current PG-VarMiON framework as a starting point:

1. **Non-linear PDEs:** We would like to design PG-VarMiON-type operator networks for non-linear PDEs. As one possibility, we envision using an iterative approach incorporating the linearized problem as in a Newton-Raphson method, with the linear problem inheriting the PG-VarMiON structure.
2. **Time-dependent PDEs:** We can also consider time-dependent problems where the design challenge will be determining how the optimal weighting functions should evolve within a time-stepping scheme.
3. **Arbitrary geometries:** Currently, the only two-dimensional problems PG-VarMiON is used for are defined on unit square domains. We would like to extend the capabilities of PG-VarMiON by making its use compatible with more difficult domains, such as those that are curved or include holes. The use of B-splines to define the trial basis should make it easy to handle all sorts of geometries, but the exact details remain to be investigated.

Chapter 11: Final Conclusions

Whether we are considering hyperbolic or elliptic PDEs, there is a place for deep learning to be used to augment existing numerical schemes or develop new ones heavily inspired by mathematical theory. We have shown the advantages of approaching the synergy between numerical PDEs and deep learning in the manner presented in this dissertation.

In the case of DSP-WENO, we have shown that with a sufficiently constrained neural network, we can make use of an existing entropy stable solver by replacing a single algorithm with the trained network. This retains the theoretical properties of the numerical solver while allowing for an improvement in shock-capturing capabilities by directly learning from relevant data.

In the case of PG-VarMiON, we have demonstrated the advantage of strongly encoding the Petrov-Galerkin form into an operator network. Namely, we have shown increased generalization capabilities for OOD data, including the case of generalizing to non-zero Neumann data when the network is only trained with zero Neumann data. We have also demonstrated that the performance is not significantly diminished as the size of the training dataset used to train PG-VarMiON is decreased. Further, we have shown the (unsupervised) approximate recovery of the Petrov-Galerkin optimal weighting functions.

Through these methods, we have demonstrated reliable ways to integrate deep learning to

the solving of PDEs. However, there are still several directions to be carefully explored to extend these techniques to more general settings. Thus, the present dissertation serves as a starting point to undertake such explorations.

Appendix A: Logarithmic average

To construct the KEPEC flux and the dissipation operator for the Euler equations, we compute the averaged values at the cell interface with the logarithmic mean

$$\hat{z} = \frac{z_L - z_R}{\ln(z_L) - \ln(z_R)},$$

which is well-defined for strictly positive quantities. In the above equation, z_L and z_R are two states of the quantity of interest. When z_L and z_R are nearly equal, this is not numerically well-posed. [53] provides a stable algorithm to compute the logarithmic mean.

Let $\zeta = \frac{z_L}{z_R}$ such that

$$\hat{z} = \frac{z_L + z_R}{\ln(\zeta)} \left(\frac{\zeta - 1}{\zeta + 1} \right),$$

where we use the series expansion of $\ln(\zeta)$

$$\ln(\zeta) = 2 \left(\frac{\zeta - 1}{\zeta + 1} + \frac{1}{3} \frac{(\zeta - 1)^3}{(\zeta + 1)^3} + \frac{1}{5} \frac{(\zeta - 1)^5}{(\zeta + 1)^5} + \frac{1}{7} \frac{(\zeta - 1)^7}{(\zeta + 1)^7} + \mathcal{O}(\zeta^9) \right),$$

to obtain a numerically well-formed logarithmic mean. Thus, the algorithm is as given in Algorithm A.1.

Algorithm A.1 Numerically stable algorithm for evaluating logarithmic mean

$$\epsilon = 10^{-2}$$

$$\zeta = \frac{z_L}{z_R}$$

$$f = \frac{\zeta - 1}{\zeta + 1}$$

$$u = f^2$$

if $u < \epsilon$ **then**

$$F = 1 + \frac{u}{3} + \frac{u^2}{5} + \frac{u^3}{7}$$

else

$$F = \frac{\ln(\zeta)}{2f}$$

end if

$$\hat{z} = \frac{z_L + z_R}{2F}$$

Appendix B: Additional DSP-WENO errors

Tables [B.1](#) and [B.2](#) show the errors (measured in the discrete L^2 and L^∞ norm, respectively) with various reconstructions for both smooth linear advection test cases. As with L^1 , we observe third-order accuracy in L^2 in both test cases (except for ENO3 in Test case 2). There seems to be slight deterioration in the L^∞ -order of convergence (which is much more severe for ENO3 in Test case 2).

	N	ENO3		SP-WENO		SP-WENOC		DSP-WENO	
		Error	Rate	Error	Rate	Error	Rate	Error	Rate
Test 1	100	1.43e-5	-	6.08e-5	-	5.69e-5	-	5.77e-5	-
	200	1.79e-6	3.00	8.40e-6	2.86	7.91e-6	2.85	5.79e-6	3.32
	400	2.24e-7	3.00	1.17e-6	2.85	1.13e-6	2.81	6.89e-7	3.07
	600	6.63e-8	3.00	3.64e-7	2.87	3.60e-7	2.82	2.05e-7	2.99
	800	2.80e-8	3.00	1.59e-7	2.87	1.57e-7	2.88	8.74e-8	2.96
	1000	1.43e-8	3.00	8.40e-8	2.86	8.33e-8	2.84	4.47e-8	3.01
Test 2	100	7.02e-4	-	9.88e-4	-	9.71e-4	-	1.14e-3	-
	200	9.44e-5	2.90	1.35e-4	2.87	1.34e-4	2.86	1.37e-4	3.05
	400	1.24e-5	2.93	1.83e-5	2.89	1.81e-5	2.89	1.68e-5	3.03
	600	4.04e-6	2.76	5.78e-6	2.84	5.76e-6	2.82	5.38e-6	2.80
	800	2.51e-6	1.66	2.50e-6	2.92	2.48e-6	2.94	2.49e-6	2.67
	1000	2.15e-6	0.68	1.33e-6	2.83	1.31e-6	2.84	1.32e-6	2.85

Table B.1: L^2 errors and convergence rates for linear advection smooth tests 1 and 2.

	N	ENO3		SP-WENO		SP-WENOC		DSP-WENO	
		Error	Rate	Error	Rate	Error	Rate	Error	Rate
Test 1	100	9.20e-6	-	1.05e-4	-	9.77e-5	-	1.13e-4	-
	200	1.10e-6	3.06	2.02e-5	2.38	1.91e-5	2.36	1.11e-5	3.35
	400	1.42e-7	2.96	3.56e-6	2.50	3.47e-6	2.46	1.71e-6	2.69
	600	4.21e-8	2.99	1.24e-6	2.60	1.23e-6	2.56	5.94e-7	2.61
	800	1.73e-8	3.10	6.42e-7	2.30	6.33e-7	2.31	2.98e-7	2.39
	1000	8.88e-9	2.98	3.83e-7	2.31	3.80e-7	2.28	1.70e-7	2.51
Test 2	100	6.38e-4	-	1.33e-3	-	1.30e-3	-	1.75e-3	-
	200	9.22e-5	2.79	2.21e-4	2.59	2.19e-4	2.57	2.07e-4	3.08
	400	1.29e-5	2.84	4.16e-5	2.41	4.14e-5	2.40	2.82e-5	2.88
	600	4.44e-6	2.62	1.45e-5	2.59	1.45e-5	2.59	1.15e-5	2.21
	800	3.56e-6	0.76	7.62e-6	2.24	7.49e-6	2.29	6.08e-6	2.21
	1000	3.24e-6	0.43	4.45e-6	2.41	4.37e-6	2.41	3.84e-6	2.06

Table B.2: L^∞ errors and convergence rates for linear advection smooth tests 1 and 2.

Appendix C: Proof of Theorem 4.1

Proof. Consider WENO-($2k - 1$) for $k \geq 2$. We have the following reconstructions at the interface $x_{i+\frac{1}{2}}$.

Left Reconstruction: If $p^{(r)}(x)$ is the polynomial interpolating the values $\{v_{i-r}, \dots, v_{i-r+k-1}\}$, then

$$v_{i+\frac{1}{2}}^- = \sum_{r=0}^{k-1} w_r p^{(r)}(x_{i+\frac{1}{2}}). \quad (\text{C.1})$$

Right Reconstruction: If $\tilde{p}^{(r)}(x)$ is the polynomial interpolating the values $\{v_{i+1-r}, \dots, v_{i-r+k}\}$, then

$$v_{i+\frac{1}{2}}^+ = \sum_{r=0}^{k-1} \tilde{w}_r \tilde{p}^{(r)}(x_{i+\frac{1}{2}}). \quad (\text{C.2})$$

Note that

$$p^{(r)} = \tilde{p}^{(r+1)} \quad r = 0, \dots, k-2. \quad (\text{C.3})$$

Thus, we have

$$\begin{aligned} \llbracket v \rrbracket_{i+\frac{1}{2}} &= \sum_{r=0}^{k-1} \tilde{w}_r \tilde{p}^{(r)}(x_{i+\frac{1}{2}}) - \sum_{r=0}^{k-1} w_r p^{(r)}(x_{i+\frac{1}{2}}) \\ &= \tilde{w}_0 \tilde{p}^{(0)}(x_{i+\frac{1}{2}}) + \sum_{r=0}^{k-2} (\tilde{w}_{r+1} - w_r) p^{(r)}(x_{i+\frac{1}{2}}) - w_{k-1} p^{(k-1)}(x_{i+\frac{1}{2}}) \end{aligned} \quad (\text{C.4})$$

Note that the interpolatory polynomials can be expressed as

$$\begin{aligned} p^{(r)}(x) &= \sum_{j=0}^{k-1} \left[v_{i-r+j} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right], \\ \tilde{p}^{(r)}(x) &= \sum_{j=0}^{k-1} \left[v_{i+1-r+j} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right], \end{aligned} \quad (\text{C.5})$$

where

$$\prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) = \prod_{\substack{l=i-r \\ l \neq i-r+j}}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right)$$

defines the product with all terms in the range for which the denominator does not vanish. Also, we write the point values in terms of adjacent jumps

$$v_{i-r+j} = \begin{cases} -\sum_{q=0}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} + v_{i+1} & \text{if } j \leq r \\ \sum_{q=1}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} + v_{i+1} & \text{if } j > r+1 \\ v_{i+1} & \text{if } j = r+1. \end{cases} \quad (\text{C.6})$$

Thus,

$$\begin{aligned}
p^{(r)}(x) &= \sum_{j=0}^{k-1} \left[v_{i-r+j} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right], \\
&= - \sum_{j=0}^r \left[\sum_{q=0}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=0}^r \left[\prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=r+1}^{r+1} \left[\prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&\quad + \sum_{j=r+2}^{k-1} \left[\sum_{q=1}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=r+2}^{k-1} \left[\prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&= v_{i+1} - \sum_{j=0}^r \left[\sum_{q=0}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right] \\
&\quad + \sum_{j=r+2}^{k-1} \left[\sum_{q=1}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right].
\end{aligned} \tag{C.7}$$

Similarly,

$$v_{i+1-r+j} = \begin{cases} - \sum_{q=1}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} + v_{i+1} & \text{if } j \leq r-1 \\ \sum_{q=0}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} + v_{i+1} & \text{if } j > r \\ v_{i+1} & \text{if } j = r. \end{cases} \tag{C.8}$$

Thus,

$$\begin{aligned}
\tilde{p}^{(r)}(x) &= \sum_{j=0}^{k-1} \left[v_{i+1-r+j} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right], \\
&= - \sum_{j=0}^{r-1} \left[\sum_{q=1}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=0}^{r-1} \left[\prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=r}^r \left[\prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&\quad + \sum_{j=r+1}^{k-1} \left[\sum_{q=0}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&\quad + v_{i+1} \sum_{j=r+1}^{k-1} \left[\prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&= v_{i+1} - \sum_{j=0}^{r-1} \left[\sum_{q=1}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right] \\
&\quad + \sum_{j=r+1}^{k-1} \left[\sum_{q=0}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i+1-r}^{i-r+k} \left(\frac{x - x_l}{x_{i+1-r+j} - x_l} \right) \right].
\end{aligned} \tag{C.9}$$

Hence,

$$\begin{aligned}
\llbracket v \rrbracket_{i+\frac{1}{2}} &= \tilde{w}_0 \tilde{p}^{(0)}(x_{i+\frac{1}{2}}) + \sum_{r=0}^{k-2} (\tilde{w}_{r+1} - w_r) p^{(r)}(x_{i+\frac{1}{2}}) - w_{k-1} p^{(k-1)}(x_{i+\frac{1}{2}}) \\
&= \underbrace{\tilde{w}_0 v_{i+1} - \tilde{w}_0 \sum_{j=0}^{-1} \left[\sum_{q=1}^{-j} \Delta v_{i+j+q+\frac{1}{2}} \prod_{l=i+1}^{i-r+k} \left(\frac{x - x_l}{x_{i+1+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}}}_{=0} \\
&\quad + \tilde{w}_0 \sum_{j=1}^{k-1} \left[\sum_{q=0}^{j-1} \Delta v_{i+j-q+\frac{1}{2}} \prod_{l=i+1}^{i+k} \left(\frac{x - x_l}{x_{i+1+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}} \\
&\quad + \sum_{r=0}^{k-2} (\tilde{w}_{r+1} - w_r) v_{i+1} \\
&\quad - \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=0}^r \left[\sum_{q=0}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}} \right\} \\
&\quad + \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=r+2}^{k-1} \left[\sum_{q=1}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1} \left(\frac{x - x_l}{x_{i-r+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}} \right\} \\
&\quad - w_{k-1} v_{i+1} \\
&\quad + w_{k-1} \sum_{j=0}^{k-1} \left[\sum_{q=0}^{k-1-j} \Delta v_{i-k+1+j+q+\frac{1}{2}} \prod_{l=i-k+1}^i \left(\frac{x - x_l}{x_{i-k+1+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}} \\
&\quad - \underbrace{w_{k-1} \sum_{j=k+1}^{k-1} \left[\sum_{q=1}^{j-k+1-1} \Delta v_{i-k+1+j-q+\frac{1}{2}} \prod_{l=i-k+1}^i \left(\frac{x - x_l}{x_{i-k+1+j} - x_l} \right) \right]_{x=x_{i+\frac{1}{2}}}}_{=0}
\end{aligned}$$

$$\begin{aligned}
&= \tilde{w}_0 \sum_{j=1}^{k-1} \left[\sum_{q=0}^{j-1} \Delta v_{i+j-q+\frac{1}{2}} \prod_{l=i+1}^{i+k}{}' \left(\frac{x-x_l}{x_{i+1+j}-x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \\
&\quad - \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=0}^r \left[\sum_{q=0}^{r-j} \Delta v_{i-r+j+q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1}{}' \left(\frac{x-x_l}{x_{i-r+j}-x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \right\} \\
&\quad + \sum_{r=0}^{k-2} \left\{ (\tilde{w}_{r+1} - w_r) \sum_{j=r+2}^{k-1} \left[\sum_{q=1}^{j-r-1} \Delta v_{i-r+j-q+\frac{1}{2}} \prod_{l=i-r}^{i-r+k-1}{}' \left(\frac{x-x_l}{x_{i-r+j}-x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right] \right\} \\
&\quad + w_{k-1} \sum_{j=0}^{k-1} \left[\sum_{q=0}^{k-1-j} \Delta v_{i-k+1+j+q+\frac{1}{2}} \prod_{l=i-k+1}^i{}' \left(\frac{x-x_l}{x_{i-k+1+j}-x_l} \right) \Big|_{x=x_{i+\frac{1}{2}}} \right]. \tag{C.10}
\end{aligned}$$

We define the following jump ratios

$$\theta_{i,k} = \frac{\Delta v_{k+i+\frac{1}{2}}}{\Delta v_{i+\frac{1}{2}}}. \tag{C.11}$$

With these, and assuming $\Delta v_{i+\frac{1}{2}} \neq 0$, we obtain (4.15) from (C.10). The reconstruction will be sign-preserving so long as $G_{i+\frac{1}{2}} \geq 0$.

□

Appendix D: PG-VarMiON with OOD parameterization data

We perform an additional test with PG-VarMiON. We construct a test set similar to DATASET 1 but with quadratic velocities of the form

$$c(x) = ax^2 + bx + \frac{1}{30}, \quad y \sim \mathcal{U}\left(\left[\frac{1}{20}, \frac{1}{12}\right]\right), \quad b = 4\left(y - \frac{1}{20}\right), \quad a = \frac{1}{15} - b$$

With this test set, the PDE parameterization data is OOD. According to the error estimate we have derived, we expect that PG-VarMiON will not generalize well to this. This is clearly shown in Table D.1. Thus, while PG-VarMiON can generalize to OOD f (and η), it is necessary to include sufficient data g to generalize.

Method	Tanh c	Quadratic c
Projection	0.63 (0.63)	0.32 (0.37)
PG-VarMiON	0.65 (0.63)	5.62 (5.09)

Table D.1: 1D advection-diffusion problem with varying c : Comparison of mean relative L^2 test error (in %) for projection and PG-VarMiON for in-distribution and OOD velocities. The standard deviation of error is in parentheses.

Bibliography

- [1] Gabriel Acosta and Ricardo Durán. “An optimal Poincaré inequality in L^1 for convex domains.” In: *Proceedings of the american mathematical society* 132.1 (2004), pp. 195–202.
- [2] Ivo Babuška and Manil Suri. “The p- and h-p versions of the finite element method, an overview.” In: *Computer Methods in Applied Mechanics and Engineering* 80.1 (1990), pp. 5–26. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(90\)90011-A](https://doi.org/10.1016/0045-7825(90)90011-A). URL: <https://www.sciencedirect.com/science/article/pii/004578259090011A>.
- [3] Paul E. Barbone and Isaac Harari. “Nearly H1-optimal finite element methods.” In: *Computer Methods in Applied Mechanics and Engineering* 190.43 (2001), pp. 5679–5690. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/S0045-7825\(01\)00191-8](https://doi.org/10.1016/S0045-7825(01)00191-8). URL: <https://www.sciencedirect.com/science/article/pii/S0045782501001918>.
- [4] J.W. Barrett and K.W. Morton. “Approximate symmetrization and Petrov-Galerkin methods for diffusion-convection problems.” In: *Computer Methods in Applied Mechanics and Engineering* 45.1 (1984), pp. 97–122. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(84\)90152-X](https://doi.org/10.1016/0045-7825(84)90152-X). URL: <https://www.sciencedirect.com/science/article/pii/004578258490152X>.
- [5] Yuri Bazilevs et al. “Isogeometric analysis: approximation, stability and error estimates for h-refined meshes.” In: *Mathematical Models and Methods in Applied Sciences* 16.07 (2006), pp. 1031–1090.
- [6] Andrea Beck, David Flad, and Claus-Dieter Munz. “Deep neural networks for data-driven LES closure models.” In: *Journal of Computational Physics* 398 (2019), p. 108910. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2019.108910>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999119306151>.
- [7] Boris Bonev et al. “Spherical fourier neural operators: Learning stable dynamics on the sphere.” In: *International conference on machine learning*. PMLR. 2023, pp. 2806–2823.
- [8] James H Bramble and SR Hilbert. “Estimation of linear functionals on Sobolev spaces with application to Fourier transforms and spline interpolation.” In: *SIAM Journal on Numerical Analysis* 7.1 (1970), pp. 112–124.
- [9] S. Brenner and L.R. Scott. *The Mathematical Theory of Finite Element Methods*. Texts in Applied Mathematics. Springer New York, 2002. ISBN: 9780387954516.

- [10] H Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer Science & Business Media, 2011.
- [11] Praveen Chandrashekar. “Kinetic energy preserving and entropy stable finite volume schemes for compressible Euler and Navier-Stokes equations.” In: *Communications in Computational Physics* 14.5 (2013), pp. 1252–1286.
- [12] Philip Charles and Deep Ray. “Learning WENO for Entropy Stable Schemes to Solve Conservation Laws.” In: *SIAM Journal on Scientific Computing* 47.6 (2025), pp. C1196–C1222. DOI: 10.1137/24M1697566. eprint: <https://doi.org/10.1137/24M1697566>. URL: <https://doi.org/10.1137/24M1697566>.
- [13] Philip Charles et al. “An optimal Petrov-Galerkin framework for operator networks.” In: *arXiv preprint arXiv:2503.04024* (2025).
- [14] Tianheng Chen and Chi-Wang Shu. “Entropy stable high order discontinuous Galerkin methods with suitable quadrature rules for hyperbolic conservation laws.” In: *Journal of Computational Physics* 345 (2017), pp. 427–461.
- [15] Tianping Chen and Hong Chen. “Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks.” In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 904–910. DOI: 10.1109/72.392252.
- [16] Tianping Chen and Hong Chen. “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems.” In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 911–917.
- [17] Alina Chertock, Shaoshuai Chu, and Alexander Kurganov. “Adaptive high-order A-WENO schemes based on a new local smoothness indicator.” In: *arXiv preprint arXiv:2211.07099* (2022).
- [18] Elisabetta Chiodaroli, Camillo De Lellis, and Ondřej Kreml. “Global Ill-Posedness of the Isentropic System of Gas Dynamics.” In: *Communications on Pure and Applied Mathematics* 68.7 (2015), pp. 1157–1190. DOI: <https://doi.org/10.1002/cpa.21537>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.21537>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.21537>.
- [19] Stéphane Clain, Steven Diot, and Raphaël Loubère. “A high-order finite volume method for systems of conservation laws—Multi-dimensional Optimal Order Detection (MOOD).” In: *Journal of computational Physics* 230.10 (2011), pp. 4028–4050.
- [20] Salvatore Cuomo et al. “Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What’s Next.” In: *Journal of Scientific Computing* 92.3 (2022), p. 88. DOI: 10.1007/s10915-022-01939-z. URL: <https://doi.org/10.1007/s10915-022-01939-z>.
- [21] Constantine M Dafermos and Constantine M Dafermos. *Hyperbolic conservation laws in continuum physics*. Vol. 3. Springer, 2005.

- [22] Agnimitra Dasgupta et al. “Conditional score-based generative models for solving physics-based inverse problems.” In: *NeurIPS 2023 Workshop on Deep Learning and Inverse Problems*. 2023.
- [23] L Demkowicz and J.T Oden. “An adaptive characteristic Petrov-Galerkin finite element method for convection-dominated linear and nonlinear parabolic problems in one space variable.” In: *Journal of Computational Physics* 67.1 (1986), pp. 188–213. ISSN: 0021-9991. DOI: [https://doi.org/10.1016/0021-9991\(86\)90121-X](https://doi.org/10.1016/0021-9991(86)90121-X). URL: <https://www.sciencedirect.com/science/article/pii/002199918690121X>.
- [24] L. Demkowicz and J.T. Oden. “An adaptive characteristic petrov-galerkin finite element method for convection-dominated linear and nonlinear parabolic problems in two space variables.” In: *Computer Methods in Applied Mechanics and Engineering* 55.1 (1986), pp. 63–87. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(86\)90086-1](https://doi.org/10.1016/0045-7825(86)90086-1). URL: <https://www.sciencedirect.com/science/article/pii/0045782586900861>.
- [25] John A Evans and Thomas JR Hughes. “Explicit trace inequalities for isogeometric analysis and parametric hexahedral finite elements.” In: *Numerische Mathematik* 123 (2013), pp. 259–290.
- [26] Pericles S Farmakis, Panagiotis Tsoutsanis, and Xesús Nogueira. “WENO schemes on unstructured meshes using a relaxed a posteriori MOOD limiting approach.” In: *Computer Methods in Applied Mechanics and Engineering* 363 (2020), p. 112921.
- [27] Travis C Fisher and Mark H Carpenter. “High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains.” In: *Journal of Computational Physics* 252 (2013), pp. 518–557.
- [28] Ulrik S Fjordholm, Siddhartha Mishra, and Eitan Tadmor. “Arbitrarily high-order accurate entropy stable essentially nonoscillatory schemes for systems of conservation laws.” In: *SIAM Journal on Numerical Analysis* 50.2 (2012), pp. 544–573.
- [29] Ulrik S Fjordholm, Siddhartha Mishra, and Eitan Tadmor. “Energy preserving and energy stable schemes for the shallow water equations.” In: *Foundations of computational mathematics, Hong Kong* 363 (2008), pp. 93–139.
- [30] Ulrik S Fjordholm, Siddhartha Mishra, and Eitan Tadmor. “ENO reconstruction and ENO interpolation are stable.” In: *Foundations of Computational Mathematics* 13 (2013), pp. 139–159.
- [31] Ulrik S Fjordholm and Deep Ray. “A sign preserving WENO reconstruction method.” In: *Journal of Scientific Computing* 68 (2016), pp. 42–63.
- [32] Ulrik S Fjordholm et al. “Construction of approximate entropy measure-valued solutions for hyperbolic systems of conservation laws.” In: *Foundations of Computational Mathematics* 17 (2017), pp. 763–827.
- [33] Ulrik S Fjordholm et al. “Statistical solutions of hyperbolic systems of conservation laws: numerical approximation.” In: *Mathematical Models and Methods in Applied Sciences* 30.03 (2020), pp. 539–609.

- [34] Leopoldo P. Franca and Alessandro Russo. “Unlocking with residual-free bubbles.” In: *Computer Methods in Applied Mechanics and Engineering* 142.3 (1997), pp. 361–364. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/S0045-7825\(96\)01138-3](https://doi.org/10.1016/S0045-7825(96)01138-3). URL: <https://www.sciencedirect.com/science/article/pii/S0045782596011383>.
- [35] Leopoldo P Franca, Sergio L Frey, and Thomas JR Hughes. “Stabilized finite element methods: I. Application to the advective-diffusive model.” In: *Computer Methods in Applied Mechanics and Engineering* 95.2 (1992), pp. 253–276.
- [36] Shailesh Garg and Souvik Chakraborty. “VB-DeepONet: A Bayesian operator learning framework for uncertainty quantification.” In: *Engineering Applications of Artificial Intelligence* 118 (2023), p. 105685. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2022.105685>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197622006753>.
- [37] Edwige Godlewski and Pierre-Arnaud Raviart. “Hyperbolic systems of conservation laws.” In: 3-4. Ellipses, 1991. Chap. 1, 2.
- [38] Hwan Goh et al. “Solving Bayesian Inverse Problems via Variational Autoencoders.” In: *Proceedings of the 2nd Mathematical and Scientific Machine Learning Conference*. Ed. by Joan Bruna, Jan Hesthaven, and Lenka Zdeborova. Vol. 145. Proceedings of Machine Learning Research. PMLR, 16–19 Aug 2022, pp. 386–425. URL: <https://proceedings.mlr.press/v145/goh22a.html>.
- [39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [40] Somdatta Goswami et al. “A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials.” In: *Computer Methods in Applied Mechanics and Engineering* 391 (2022), p. 114587.
- [41] Somdatta Goswami et al. “Deep transfer operator learning for partial differential equations under conditional shift.” In: *Nature Machine Intelligence* 4.12 (2022), pp. 1155–1164. DOI: 10.1038/s42256-022-00569-2. URL: <https://doi.org/10.1038/s42256-022-00569-2>.
- [42] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. “Strong stability-preserving high-order time discretization methods.” In: *SIAM review* 43.1 (2001), pp. 89–112.
- [43] Ishaan Gulrajani et al. “Improved training of wasserstein gans.” In: *Advances in neural information processing systems* 30 (2017).
- [44] Amiram Harten. “On the symmetric form of systems of conservation laws with entropy.” In: *Journal of Computational Physics* 49.1 (1983), pp. 151–164. ISSN: 0021-9991. DOI: [https://doi.org/10.1016/0021-9991\(83\)90118-3](https://doi.org/10.1016/0021-9991(83)90118-3). URL: <https://www.sciencedirect.com/science/article/pii/0021999183901183>.
- [45] Piet Hemker. “A numerical study of stiff two-point boundary problems.” PhD thesis. Universiteit van Amsterdam, Mar. 1977.
- [46] Qingguo Hong et al. *On the Activation Function Dependence of the Spectral Bias of Neural Networks*. 2022. arXiv: 2208.04924 [cs.LG]. URL: <https://arxiv.org/abs/2208.04924>.

- [47] Amanda A. Howard et al. “Multifidelity deep operator networks for data-driven and physics-informed problems.” In: *Journal of Computational Physics* 493 (2023), p. 112462. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2023.112462>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999123005570>.
- [48] Daniel Zhengyu Huang, Nicholas H. Nelsen, and Margaret Trautner. *An operator learning perspective on parameter-to-observable maps*. 2024. arXiv: 2402.06031 [cs.LG].
- [49] Thomas J. R. Hughes, Robert L. Taylor, and Worsak Kanoknukulchai. “A simple and efficient finite element for plate bending.” In: *International Journal for Numerical Methods in Engineering* 11.10 (1977), pp. 1529–1543. DOI: <https://doi.org/10.1002/nme.1620111005>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620111005>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620111005>.
- [50] Thomas J.R. Hughes. “Multiscale phenomena: Green’s functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods.” In: *Computer Methods in Applied Mechanics and Engineering* 127.1 (1995), pp. 387–401. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(95\)00844-9](https://doi.org/10.1016/0045-7825(95)00844-9). URL: <https://www.sciencedirect.com/science/article/pii/0045782595008449>.
- [51] Thomas J.R. Hughes, Leopoldo P. Franca, and Gregory M. Hulbert. “A new finite element formulation for computational fluid dynamics: VIII. The galerkin/least-squares method for advective-diffusive equations.” In: *Computer Methods in Applied Mechanics and Engineering* 73.2 (1989), pp. 173–189. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(89\)90111-4](https://doi.org/10.1016/0045-7825(89)90111-4). URL: <https://www.sciencedirect.com/science/article/pii/0045782589901114>.
- [52] Farzad Ismail, Philip L Roe, and Hiroaki Nishikawa. “A proposed cure to the carbuncle phenomenon.” In: *Computational Fluid Dynamics 2006: Proceedings of the Fourth International Conference on Computational Fluid Dynamics, ICCFD, Ghent, Belgium, 10-14 July 2006*. Springer. 2009, pp. 149–154.
- [53] Farzad Ismail and Philip L. Roe. “Affordable, entropy-consistent Euler flux functions II: Entropy production at shocks.” In: *Journal of Computational Physics* 228.15 (2009), pp. 5410–5436. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2009.04.021>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999109002113>.
- [54] Antony Jameson. “Formulation of kinetic energy preserving conservative schemes for gas dynamics and direct numerical simulation of one-dimensional viscous compressible flow in a shock tube using entropy and kinetic energy preserving schemes.” In: *Journal of Scientific Computing* 34.2 (2008), pp. 188–208.

- [55] Guang-Shan Jiang and Chi-Wang Shu. “Efficient Implementation of Weighted ENO Schemes.” In: *Journal of Computational Physics* 126.1 (1996), pp. 202–228. ISSN: 0021-9991. DOI: <https://doi.org/10.1006/jcph.1996.0130>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999196901308>.
- [56] Pengzhan Jin, Shuai Meng, and Lu Lu. “MIONet: Learning Multiple-Input Operators via Tensor Product.” In: <https://doi.org/10.1137/22M1477751> 44 (6 Nov. 2022), A3490–A3514. ISSN: 1064-8275. DOI: 10.1137/22M1477751. URL: <https://epubs.siam.org/doi/10.1137/22M1477751>.
- [57] Patrick Kidger and Terry Lyons. “Universal Approximation with Deep Narrow Networks.” In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 2306–2327. URL: <https://proceedings.mlr.press/v125/kidger20a.html>.
- [58] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization.” In: *arXiv preprint arXiv:1412.6980* (2014).
- [59] Katiana Kontolati et al. “Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems.” In: *Nature Communications* 15.1 (2024), p. 5101. DOI: 10.1038/s41467-024-49411-w. URL: <https://doi.org/10.1038/s41467-024-49411-w>.
- [60] Seid Koric and Diab W Abueidda. “Data-driven and physics-informed deep learning operators for solution of heat conduction equation with parametric heat source.” In: *International Journal of Heat and Mass Transfer* 203 (2023), p. 123809.
- [61] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. “On universal approximation and error bounds for Fourier neural operators.” In: *J. Mach. Learn. Res.* 22.1 (Jan. 2021). ISSN: 1532-4435.
- [62] S. N. Kružkov. “First order quasilinear equations in several independent variables.” In: *Mathematics of the USSR-Sbornik* 10.2 (1970), p. 217. DOI: 10.1070/SM1970v010n02ABEH002156. URL: <https://dx.doi.org/10.1070/SM1970v010n02ABEH002156>.
- [63] Alexander Kurganov and Eitan Tadmor. “Solution of two-dimensional Riemann problems for gas dynamics without Riemann problem solvers.” In: *Numerical Methods for Partial Differential Equations: An International Journal* 18.5 (2002), pp. 584–608.
- [64] Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. “Error estimates for deepONets: A deep learning framework in infinite dimensions.” In: *Transactions of Mathematics and Its Applications* 6.1 (2022), tnac001.
- [65] Peter D Lax. “Weak solutions of nonlinear hyperbolic equations and their numerical computation.” In: *Communications on pure and applied mathematics* 7.1 (1954), pp. 159–193.

- [66] Philippe G Lefloch, Jean-Marc Mercier, and Christian Rohde. “Fully discrete, entropy conservative schemes of arbitrary order.” In: *SIAM Journal on Numerical Analysis* 40.5 (2002), pp. 1968–1992.
- [67] Camillo de Lellis and László Székelyhidi. “On Admissibility Criteria for Weak Solutions of the Euler Equations.” In: *Archive for Rational Mechanics and Analysis* 195.1 (2010), pp. 225–260. DOI: 10.1007/s00205-008-0201-x. URL: <https://doi.org/10.1007/s00205-008-0201-x>.
- [68] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations.” In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=c8P9NQVtmnO>.
- [69] Zongyi Li et al. “Fourier neural operator with learned deformations for pdes on general geometries.” In: *Journal of Machine Learning Research* 24.388 (2023), pp. 1–26.
- [70] Zongyi Li et al. “Geometry-informed neural operator for large-scale 3d pdes.” In: *Advances in Neural Information Processing Systems* 36 (2024).
- [71] Timur Linde et al. “Robust Euler codes.” In: *13th computational fluid dynamics conference*. 1997, p. 2098.
- [72] Ning Liu, Siavash Jafarzadeh, and Yue Yu. “Domain agnostic fourier neural operators.” In: *Advances in Neural Information Processing Systems* 36 (2024).
- [73] Ziming Liu et al. “Kan: Kolmogorov-arnold networks.” In: *arXiv preprint arXiv:2404.19756* (2024).
- [74] Abimael F.D. Loula, Thomas J.R. Hughes, and Leopoldo P. Franca. “Petrov-Galerkin formulations of the Timoshenko beam problem.” In: *Computer Methods in Applied Mechanics and Engineering* 63.2 (1987), pp. 115–132. ISSN: 0045-7825. DOI: [https://doi.org/10.1016/0045-7825\(87\)90167-8](https://doi.org/10.1016/0045-7825(87)90167-8). URL: <https://www.sciencedirect.com/science/article/pii/0045782587901678>.
- [75] Lu Lu et al. “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators.” In: *Nature Machine Intelligence* 3.3 (2021), pp. 218–229.
- [76] Kjetil O Lye et al. “Iterative surrogate model optimization (ISMO): An active learning algorithm for PDE constrained optimization with deep neural networks.” In: *Computer Methods in Applied Mechanics and Engineering* 374 (2021), p. 113575.
- [77] Kjetil O. Lye, Siddhartha Mishra, and Deep Ray. “Deep learning observables in computational fluid dynamics.” In: *Journal of Computational Physics* 410 (2020), p. 109339. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2020.109339>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999120301133>.
- [78] Sabrina R Lynch et al. “Numerical considerations for advection-diffusion problems in cardiovascular hemodynamics.” In: *International journal for numerical methods in biomedical engineering* 36.9 (2020), e3378.

- [79] Romit Maulik, Omer San, and Jamey D. Jacob. “Spatiotemporally dynamic implicit large eddy simulation using machine learning classifiers.” In: *Physica D: Nonlinear Phenomena* 406 (2020), p. 132409. ISSN: 0167-2789. DOI: <https://doi.org/10.1016/j.physd.2020.132409>. URL: <https://www.sciencedirect.com/science/article/pii/S0167278919301630>.
- [80] Takeru Miyato et al. “Spectral normalization for generative adversarial networks.” In: *arXiv preprint arXiv:1802.05957* (2018).
- [81] Dhruv Patel et al. “Variationally mimetic operator networks.” In: *Computer Methods in Applied Mechanics and Engineering* 419 (2024), p. 116536. ISSN: 0045-7825. DOI: <https://doi.org/10.1016/j.cma.2023.116536>. URL: <https://www.sciencedirect.com/science/article/pii/S0045782523006606>.
- [82] Dhruv V Patel, Deep Ray, and Assad A Oberai. “Solution of physics-based Bayesian inverse problems with deep generative priors.” In: *Computer Methods in Applied Mechanics and Engineering* 400 (2022), p. 115428.
- [83] Allan Pinkus. “Approximation theory of the MLP model in neural networks.” In: *Acta Numerica* 8 (1999), pp. 143–195. DOI: 10.1017/S0962492900002919.
- [84] Joseph M Powers, Jeffrey D Bruns, and Aleksandar Jemcov. “Physical diffusion cures the carbuncle phenomenon.” In: *53rd AIAA Aerospace Sciences Meeting*. 2015, p. 0579.
- [85] Michael Prasthofer, Tim De Ryck, and Siddhartha Mishra. *Variable-Input Deep Operator Networks*. 2022. arXiv: 2205.11404 [cs.LG]. URL: <https://arxiv.org/abs/2205.11404>.
- [86] James J Quirk. *A contribution to the great Riemann solver debate*. Springer, 1997.
- [87] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.” In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2018.10.045>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999118307125>.
- [88] Hendrik Ranocha et al. “Efficient implementation of modern entropy stable and kinetic energy preserving discontinuous Galerkin methods for conservation laws.” In: *ACM Transactions on Mathematical Software* 49.4 (2023), pp. 1–30.
- [89] Deep Ray. “A Third-Order Entropy Stable Scheme for the Compressible Euler Equations.” In: *Theory, Numerics and Applications of Hyperbolic Problems II: Aachen, Germany, August 2016*. Springer. 2018, pp. 503–515.
- [90] Deep Ray and Jan S Hesthaven. “An artificial neural network as a troubled-cell indicator.” In: *Journal of computational physics* 367 (2018), pp. 166–191.
- [91] Deep Ray and Jan S Hesthaven. “Detecting troubled-cells on two-dimensional unstructured grids using a neural network.” In: *Journal of Computational Physics* 397 (2019), p. 108845.

- [92] Deep Ray et al. “Solution of physics-based inverse problems using conditional generative adversarial networks with full gradient penalty.” In: *Computer Methods in Applied Mechanics and Engineering* 417 (2023). A Special Issue in Honor of the Lifetime Achievements of T. J. R. Hughes, p. 116338. ISSN: 0045-7825. DOI: <https://doi.org/10.1016/j.cma.2023.116338>. URL: <https://www.sciencedirect.com/science/article/pii/S0045782523004620>.
- [93] AM Rogerson and E Meiburg. “A numerical study of the convergence properties of ENO schemes.” In: *Journal of Scientific Computing* 5 (1990), pp. 151–167.
- [94] Omer San and Kursat Kara. “Evaluation of Riemann flux solvers for WENO reconstruction schemes: Kelvin–Helmholtz instability.” In: *Computers & Fluids* 117 (2015), pp. 24–41.
- [95] Chi-Wang Shu and Stanley Osher. “Efficient implementation of essentially non-oscillatory shock-capturing schemes, II.” In: *Journal of Computational Physics* 83.1 (1989), pp. 32–78.
- [96] Anand Pratap Singh and Karthik Duraisamy. “Using field inversion to quantify functional errors in turbulence closures.” In: *Physics of Fluids* 28.4 (Apr. 2016), p. 045110. ISSN: 1070-6631. DOI: 10.1063/1.4947045. eprint: https://pubs.aip.org/aip/pof/article-pdf/doi/10.1063/1.4947045/14093553/045110_1_online.pdf. URL: <https://doi.org/10.1063/1.4947045>.
- [97] Eitan Tadmor. “Entropy stability theory for difference approximations of nonlinear conservation laws and related time-dependent problems.” In: *Acta Numerica* 12 (2003), pp. 451–512.
- [98] Matthew Tancik et al. “Fourier features let networks learn high frequency functions in low dimensional domains.” In: *Advances in neural information processing systems* 33 (2020), pp. 7537–7547.
- [99] Eleuterio F Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media, 2013.
- [100] Sifan Wang, Hanwen Wang, and Paris Perdikaris. “Learning the solution operator of parametric partial differential equations with physics-informed DeepONets.” In: *Science advances* 7.40 (2021), eabi8605.
- [101] Wuzhe Xu, Yulong Lu, and Li Wang. “Transfer learning enhanced deepnet for long-time prediction of evolution equations.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 9. 2023, pp. 10629–10636.
- [102] Liu Yang, Dongkun Zhang, and George Em Karniadakis. “Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations.” In: *SIAM Journal on Scientific Computing* 42.1 (2020), A292–A317. DOI: 10.1137/18M1225409. eprint: <https://doi.org/10.1137/18M1225409>. URL: <https://doi.org/10.1137/18M1225409>.

- [103] Yibo Yang, Georgios Kissas, and Paris Perdikaris. “Scalable uncertainty quantification for deep operator networks using randomized priors.” In: *Computer Methods in Applied Mechanics and Engineering* 399 (2022), p. 115399. ISSN: 0045-7825. DOI: <https://doi.org/10.1016/j.cma.2022.115399>. URL: <https://www.sciencedirect.com/science/article/pii/S0045782522004595>.
- [104] Dmitry Yarotsky and Anton Zhevnerchuk. *The phase diagram of approximation rates for deep neural networks*. <https://arxiv.org/abs/1906.09477>. 2019. DOI: 10.48550/ARXIV.1906.09477.
- [105] Huaqian You et al. “Learning deep implicit Fourier neural operators (IFNOs) with applications to heterogeneous material modeling.” In: *Computer Methods in Applied Mechanics and Engineering* 398 (2022), p. 115296.
- [106] Runpeng Yu, Weihao Yu, and Xinchao Wang. “Kan or mlp: A fairer comparison.” In: *arXiv preprint arXiv:2407.16674* (2024).
- [107] Lei Zhang et al. “Hierarchical deep-learning neural networks: finite elements and beyond.” English (US). In: *Computational Mechanics* 67.1 (Jan. 2021). Publisher Copyright: © 2020, Springer-Verlag GmbH Germany, part of Springer Nature., pp. 207–230. ISSN: 0178-7675. DOI: 10.1007/s00466-020-01928-9.
- [108] J.S.B. van Zwieten, G.J. van Zwieten, and W. Hoitinga. *Nutils 7.0*. 2022. DOI: 10.5281/zenodo.6006701.