UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE MSII 269

Mathématiques, Sciences de l'Information et de l'Ingénieur

Laboratoire des sciences de l'Ingénieur, de l'Informatique et de l'Imagerie

ICube UMR 7357

THÈSE présentée par / DISSERTATION presented by:

Frédérique Lecourtier

soutenue le / defended on : **December 4, 2026**

pour obtenir le grade de / to obtain the grade of :
Docteur de l'université de Strasbourg / Strasbourg University Doctor

Discipline (Spécialité) / Discipline (Speciality) : Mathématiques

Hybrid finite element and neural network methods for parametric PDEs

THÈSE dirigée par / DISSERTATION supervisor :

Emmanuel FRANCK *(directeur)*	Directeur de recherche, HDR, Inria, Université de Strasbourg
Michel DUPREZ *(co-encadrant)*	Chargé de recherche, HDR, Inria, Université de Strasbourg
Vanessa LLERAS *(co-encadrante)*	Maître de conférences, Université de Montpellier

RAPPORTEURS :

Florian DE VUYST	Professeur des universités, HDR, Université de Technologie de Compiègne
Marco PICASSO	Professeur des universités, HDR, Institut de Mathématiques, EFPL

AUTRES MEMBRES DU JURY / OTHER MEMBERS OF THE JURY :

Hélène BARUCQ	Directrice de recherche, HDR, Inria, Université de Bordeaux
Alexandre ERN	Directeur de recherche, HDR, École des ponts
Christophe PRUD'HOMME	Professeur des universités, HDR, Université de Strasbourg

INVITÉS / INVITED MEMBERS :

Victor MICHEL-DANSAC	Chargé de recherche Inria, Inria, Université de Strasbourg

Contents

Acknowledgements i

List of Figures iii

List of Tables vii

List of Algorithms ix

1. Introduction 1
1.1. Context and motivation . . . 1
1.2. Outline and contributions . . . 4

2. Numerical methods for PDEs 7
2.1. Problem setting . . . 7
2.2. Some classical approximation methods . . . 8
2.2.1. Continuous finite element method (FEM) . . . 8
2.2.2. Discontinuous Galerkin (DG) approach . . . 11
2.2.2.1. Unified formulation of the DG method . . . 11
2.2.2.2. Numerical fluxes for the diffusive term . . . 14
2.2.2.3. Numerical flux for the convective term . . . 17
2.2.3. φ-FEM approach . . . 19
2.3. Physics-Informed Neural Networks (PINNs) . . . 23
2.3.1. Physics-Informed Neural Networks for parametric PDEs . . . 23
2.3.2. Improving PINN training and prediction . . . 25

3. Enriching continuous FEM 27
3.1. Enriching FEM with additive priors . . . 28
3.1.1. Construction of the modified problem . . . 29
3.1.2. Convergence analysis . . . 29
3.2. Enriching FEM with multiplicative priors . . . 31
3.2.1. Construction of the modified problem . . . 31
3.2.2. Convergence analysis . . . 32
3.3. Comparison of the two enriched methods . . . 35
3.4. Implementation details . . . 39
3.4.1. Using PINN prediction effectively . . . 39
3.4.2. Imposing boundary conditions . . . 40
3.5. Numerical results within the theoretical framework . . . 41
3.5.1. Setup of the numerical experiments . . . 43
3.5.1.1. Error estimates . . . 44
3.5.1.2. Gains achieved with the enriched bases . . . 44

3.5.2. 1D Poisson problem . . . 45
3.5.2.1. Construction of the two priors . . . 46
3.5.2.2. Error estimates – with the PINN prior . . . 47
3.5.2.3. Derivatives – with both priors . . . 49
3.5.2.4. Gains achieved with the additive and multiplicative approaches – with both priors . . . 51
3.5.2.5. Influence of the PINN initialization . . . 52
3.5.3. 1D general elliptic system and convection-dominated regime . . . 53
3.5.3.1. Error estimates . . . 54
3.5.3.2. Comparison of different approaches . . . 55
3.5.3.3. Gains achieved with the additive and the multiplicative approaches . . . 57
3.5.4. 1D non-smooth transmission problem . . . 58
3.5.4.1. Construction of the different priors . . . 59
3.5.4.2. Error estimates – with the three non-parametric priors 61
3.5.4.3. Derivatives – with the three non-parametric priors . . . 62
3.5.4.4. Gains achieved with the additive approach – with the parametric enriched PINN prior . . . 63
3.5.5. 2D Poisson problem in a square domain . . . 63
3.5.5.1. Low-frequency case . . . 64
3.5.5.2. Low-frequency case – Sobolev training . . . 70
3.5.5.3. Low-frequency case – Boundary loss training . . . 72
3.5.5.4. High-frequency case . . . 74
3.5.6. 2D anisotropic elliptic problem on a square . . . 76
3.5.6.1. Error estimates . . . 78
3.5.6.2. Comparison of different approaches . . . 78
3.5.6.3. Gains achieved with the additive approach . . . 79
3.5.7. 2D Poisson problem on an annulus, with mixed boundary conditions 80
3.5.7.1. Error estimates . . . 81
3.5.7.2. Comparison of different approaches . . . 81
3.5.7.3. Gains achieved with the additive approach . . . 83
3.5.8. 3D Poisson problem in a cube domain . . . 83
3.5.8.1. Error estimates . . . 84
3.5.8.2. Computation times of the different methods . . . 85
3.5.8.3. Gains achieved with the additive approach . . . 87
3.5.8.4. Influence of prior quality on the additive approach . . . 87
3.6. Numerical results beyond the theoretical framework: the Navier–Stokes equations . . . 88
3.6.1. Problem statement . . . 88
3.6.2. Construction of the parametric PINN prior . . . 89
3.6.3. Finite element discretization . . . 92
3.6.4. Extending the additive approach to the Navier–Stokes system . . . 94
3.6.5. Error estimates . . . 95

4. SciML library using differentiable tools **99**
4.1. Elements of Differentiable Programming . . . 101

4.2. Geometry and integration modules . . . 104
4.2.1. Quadrature module . . . 105
4.2.2. Mapping module . . . 106
4.2.3. Mesh module . . . 109
4.3. Basis functions module . . . 117
4.4. DG variable and post-processing modules . . . 119
4.5. DG scheme modules . . . 122
4.5.1. Linear weak forms . . . 122
4.5.2. Numerical fluxes . . . 124
4.5.3. Boundary conditions . . . 125
4.5.4. DG scheme assembly . . . 127
4.6. Solve framework modules . . . 129
4.6.1. Newton solve . . . 129
4.6.2. Matrix-free Newton solve . . . 130
4.6.3. The solver as a non-linear approximation space . . . 131
4.7. Extension to FEM . . . 131
4.7.1. Continuous FE variables . . . 132
4.7.2. FE scheme assembly . . . 133
4.8. Extension to φ-FEM . . . 135
4.8.1. Level-set-based cell and face classification . . . 136
4.8.2. φ-FEM scheme assembly . . . 136
4.9. Learnable formalization of the Galerkin schemes . . . 140
4.10. Numerical validations . . . 144
4.10.1. 1D Poisson problem . . . 144
4.10.2. 1D Poisson problem with mixed boundary conditions . . . 147
4.10.3. 1D general elliptic system and convection-dominated regime . . . 150
4.10.4. 1D non-linear Poisson problem . . . 153
4.10.5. 2D Poisson problem in a square domain . . . 155
4.10.6. 2D Poisson problem in a 3-lobed shape domain . . . 157
4.10.7. 1D reaction-diffusion inverse problem . . . 161

5. Learnable Galerkin methods 163
5.1. Dimensional study . . . 164
5.1.1. Classical approximation methods (FEM, DG) . . . 165
5.1.2. Learnable approximation methods (data-fitting, PINN) . . . 168
5.1.3. Limits of classical bases and motivation for neural enrichment . . . 170
5.2. Non-parametric enrichment (PINN loss) – 1D test case . . . 171
5.2.1. Finite element method – Patchwise basis . . . 174
5.2.1.1. A first study on the enrichment of bases . . . 175
5.2.1.2. Single-axis capacity study . . . 178
5.2.1.3. Choice of the multiplier parametrization . . . 181
5.2.1.4. Conclusion on the choice of the multiplier . . . 195
5.2.1.5. Revisiting the first study with the retained multiplier 196
5.2.2. Finite element method – Cellwise basis . . . 198
5.2.3. Discontinuous Galerkin – Patchwise basis . . . 202
5.2.4. Discontinuous Galerkin – Cellwise basis . . . 206

5.2.5. Comparison of the schemes, conclusion and limitations 210
5.3. Non-parametric enrichment (PINN loss) – 3D test case 212
5.4. Parametric enrichment (neural operator) – 1D test case 218
5.4.1. Linear transport problem 221
5.4.2. Turning point problem 227
5.4.3. Comparison, conclusion and limitations 232

6. Conclusion **235**
6.1. Summary and contributions 235
6.2. Limitations and perspectives 236

A. Notations and acronyms **239**
A.1. Notations and definitions 239
A.1.1. Numerical methods for PDEs 240
A.1.2. Enriched finite element methods 244
A.1.3. SciML library and learnable Galerkin methods 246
A.2. Acronyms 250

B. Code examples **251**
B.1. Minimal example: classical and learned FE solvers 251
B.2. Solving a batch of parametric problems 255

Bibliography **257**

Acknowledgements

I sincerely thank the reviewers for having accepted to evaluate this manuscript, and the members of the jury for the time and attention they devote to reading it. The acknowledgements are not written yet; they will be completed in the final version of the manuscript.

List of Figures

2.1. Fitted and unfitted meshes – 3-lobed domain 19
2.2. φ-FEM computational mesh – Cut elements and stabilized facets 21

3.1. Enriched FEM – offline/online pipeline 28
3.2. 1D Poisson problem – L^2 error convergence 47
3.3. 1D Poisson problem – Semi-H^1 error convergence 48
3.4. 1D Poisson problem – Additive vs multiplicative comparison theorem . . . 49
3.5. 1D Poisson problem – Prior prediction and derivatives, PINN prior 49
3.6. 1D Poisson problem – Prior prediction and derivatives, data prior 50
3.7. 1D elliptic problem – L^2 error convergence 55
3.8. 1D elliptic problem – Prior prediction and derivatives 56
3.9. 1D elliptic problem – Solutions and correction terms 56
3.10. 1D transmission problem – L^2 error convergence, three priors 61
3.11. 1D transmission problem – Prior prediction and derivatives, three priors . 62
3.12. 2D low-frequency Poisson problem – L^2 error convergence 65
3.13. 2D low-frequency Poisson problem – Semi-H^1 error convergence 65
3.14. 2D low-frequency Poisson problem – Solutions and correction terms 66
3.15. 2D low-frequency Poisson problem – Influence of the source term approximation degree 70
3.16. 2D low-frequency Poisson problem – L^2 error convergence, Sobolev training 71
3.17. 2D low-frequency Poisson problem – Training loss curves, strong BC vs BC loss training 72
3.18. 2D low-frequency Poisson problem – L^2 error convergence, BC loss training 73
3.19. 2D high-frequency Poisson problem – L^2 error convergence 74
3.20. 2D high-frequency Poisson problem – Solutions and correction terms . . . 75
3.21. 2D elliptic problem – L^2 error convergence 78
3.22. 2D elliptic problem – Solutions and correction terms 79
3.23. 2D Poisson problem on an annulus – L^2 error convergence 82
3.24. 2D Poisson problem on an annulus – Solutions and correction terms 82
3.25. 3D Poisson problem – L^2 error convergence 85
3.26. 3D Poisson problem – Error vs online computation time 86
3.27. 3D Poisson problem – Total cost vs number of parameters 87
3.28. 2D Navier–Stokes problem – Reference solutions 90
3.29. 2D Navier–Stokes problem – Parametric PINN architecture 90
3.30. 2D Navier–Stokes problem – Training loss curves 91
3.31. 2D Navier–Stokes problem – Parametric PINN error 92
3.32. 2D Navier–Stokes problem – Componentwise L^2 error convergence 97

4.1. `ScimBa` architecture – DG method 100
4.2. Mapping composition – 3-lobed domain 108
4.3. Physical normals – learnable mapping to the 3-lobed domain 112

4.4. Mapping chain – volumetric quadrature points . . . 113
4.5. Mapping chain – surface quadrature points . . . 114
4.6. `ScimBa` class hierarchy – `EllipticWeakModel` . . . 126
4.7. `ScimBa` architecture – FE extension . . . 132
4.8. `ScimBa` architecture – φ-FEM extension . . . 136
4.9. `ScimBa` class hierarchy – `PoissonDirichletPhiFEMWeakModel` . . . 139
4.10. 1D Poisson problem – DG convergence . . . 145
4.11. 1D Poisson problem – FEM convergence . . . 146
4.12. 1D Poisson problem – DG and FEM solutions . . . 147
4.13. 1D Poisson problem with mixed BC – DG convergence . . . 148
4.14. 1D Poisson problem with mixed BC – FEM convergence . . . 149
4.15. 1D Poisson problem with mixed BC – DG and FEM solutions . . . 149
4.16. 1D elliptic problem – DG convergence, Pe = 90 . . . 151
4.17. 1D elliptic problem – DG convergence, Pe = 500 . . . 151
4.18. 1D elliptic problem – FEM convergence, Pe = 90 . . . 152
4.19. 1D elliptic problem – DG and FEM solutions, Pe = 90 . . . 153
4.20. 1D non-linear Poisson problem – DG convergence . . . 154
4.21. 1D non-linear Poisson problem – DG solutions . . . 154
4.22. 2D Poisson problem – DG convergence . . . 155
4.23. 2D Poisson problem – FEM convergence . . . 156
4.24. 2D Poisson problem – DG and FEM solutions . . . 156
4.25. 2D Poisson problem on the 3-lobed domain – Physical meshes . . . 158
4.26. 2D Poisson problem on the 3-lobed domain – FEM and φ-FEM convergence 159
4.27. 2D Poisson problem on the 3-lobed domain – FEM and φ-FEM solutions . 160
4.28. 1D inverse problem – Training, DG solution and learned coefficient . . . 162

5.1. d-dimensional Poisson problem – DG convergence . . . 166
5.2. d-dimensional Poisson problem – FEM convergence . . . 167
5.3. 1D problem (PINN, FEM patchwise) – Classical and learned solutions . . 177
5.4. 1D problem (PINN, FEM patchwise) – Learned multiplier and enriched basis 178
5.5. 1D problem (PINN, FEM patchwise) – Budget study, affine multiplier . . 182
5.6. 1D problem (PINN, FEM patchwise) – Best loss vs gain, affine multiplier . 183
5.7. 1D problem (PINN, FEM patchwise) – Learned multiplier shape and sign, affine multiplier . . . 185
5.8. 1D problem (PINN, FEM patchwise) – Learned multiplier profiles, affine multiplier . . . 185
5.9. 1D problem (PINN, FEM patchwise) – Budget study, positive multiplier . 187
5.10. 1D problem (PINN, FEM patchwise) – Best loss vs gain, positive multiplier 187
5.11. 1D problem (PINN, FEM patchwise) – Learned multiplier shape, positive multiplier . . . 188
5.12. 1D problem (PINN, FEM patchwise) – Learned multiplier profiles, positive multiplier . . . 188
5.13. 1D problem (PINN, FEM patchwise) – Exploded runs shape, positive multiplier . . . 189
5.14. 1D problem (PINN, FEM patchwise) – Exploded run example, positive multiplier . . . 189
5.15. Bounded weights – Soft clipping of the weight norm . . . 191

5.16. 1D problem (PINN, FEM patchwise) – Budget study, positive multiplier with bounded weights . . . 191
5.17. 1D problem (PINN, FEM patchwise) – Best loss vs gain, positive multiplier with bounded weights . . . 192
5.18. 1D problem (PINN, FEM patchwise) – Learned multiplier shape, positive multiplier with bounded weights . . . 193
5.19. 1D problem (PINN, FEM patchwise) – Learned multiplier profiles, positive multiplier with bounded weights . . . 193
5.20. 1D problem (PINN, FEM patchwise) – Gain distribution, multiplier comparison . . . 194
5.21. 1D problem (PINN, FEM cellwise) – Classical and learned solutions . . . 201
5.22. 1D problem (PINN, FEM cellwise) – Learned multiplier and enriched basis 202
5.23. 1D problem (PINN, DG patchwise) – Classical and learned solutions . . . 205
5.24. 1D problem (PINN, DG patchwise) – Learned multiplier and enriched basis 205
5.25. 1D problem (PINN, DG cellwise) – Classical and learned solutions . . . 208
5.26. 1D problem (PINN, DG cellwise) – Learned multiplier and enriched basis . 209
5.27. 3D problem (PINN, FEM patchwise) – Exact solution . . . 213
5.28. 3D problem (PINN, FEM patchwise) – Training loss . . . 214
5.29. 3D problem (PINN, FEM patchwise) – L^2 error as a function of N_{dofs} . . . 215
5.30. 3D problem (PINN, FEM patchwise) – Solutions, errors and learned multiplier along the diagonal . . . 217
5.31. 1D linear transport problem (NO, DG patchwise) – Reference solutions and classical error . . . 222
5.32. 1D linear transport problem (NO, DG patchwise) – Gains over $\mathcal{D}_{\text{test}}$ . . . 224
5.33. 1D linear transport problem (NO, DG patchwise) – Learned multipliers . . 225
5.34. 1D linear transport problem (NO, DG patchwise) – Solutions and errors at the median-gain parameter . . . 226
5.35. 1D turning point problem (NO, DG patchwise) – Reference solutions and classical error . . . 228
5.36. 1D turning point problem (NO, DG patchwise) – Gains over $\mathcal{D}_{\text{test}}$ . . . 230
5.37. 1D turning point problem (NO, DG patchwise) – Learned multipliers . . . 231
5.38. 1D turning point problem (NO, DG patchwise) – Solutions and errors at the median-gain parameter . . . 232

List of Tables

3.1. 1D Poisson problem – Network and training parameters 46
3.2. 1D Poisson problem – L^2 errors and gains at fixed N 50
3.3. 1D Poisson problem – Semi-H^1 errors and gains at fixed N 50
3.4. 1D Poisson problem – Gains vs PINN prior and FEM 51
3.5. 1D Poisson problem – Gains vs data prior and FEM 52
3.6. 1D Poisson problem – Gains over 30 PINN priors 53
3.7. 1D elliptic problem – Network and training parameters 54
3.8. 1D elliptic problem – L^2 errors and gains at fixed N 56
3.9. 1D elliptic problem – Gains vs PINN prior and FEM 57
3.10. 1D transmission problem – Network and training parameters 60
3.11. 1D transmission problem – Gains vs PINN prior and FEM 63
3.12. 2D low-frequency Poisson problem – Network and training parameters . . 64
3.13. 2D low-frequency Poisson problem – L^2 errors and gains at fixed N 67
3.14. 2D low-frequency Poisson problem – Gains vs PINN prior and FEM . . . 67
3.15. 2D low-frequency Poisson problem – Cost to reach a target error 68
3.16. 2D low-frequency Poisson problem – Total cost vs number of parameters . 69
3.17. 2D low-frequency Poisson problem – Network and training parameters, Sobolev training 70
3.18. 2D low-frequency Poisson problem – Gains vs PINN prior and FEM, Sobolev training 71
3.19. 2D low-frequency Poisson problem – Network and training parameters, BC loss training 72
3.20. 2D low-frequency Poisson problem – Gains vs PINN prior and FEM, BC loss training 73
3.21. 2D high-frequency Poisson problem – Network and training parameters . 75
3.22. 2D high-frequency Poisson problem – L^2 errors and gains at fixed N . . . 75
3.23. 2D high-frequency Poisson problem – Gains vs PINN prior and FEM . . . 76
3.24. 2D elliptic problem – Network and training parameters 77
3.25. 2D elliptic problem – Gains vs PINN prior and FEM 79
3.26. 2D Poisson problem on an annulus – Network and training parameters . . 81
3.27. 2D Poisson problem on an annulus – Gains vs PINN prior and FEM . . . 83
3.28. 3D Poisson problem – Network and training parameters 84
3.29. 3D Poisson problem – Cost to reach a target error 86
3.30. 3D Poisson problem – Gains vs PINN prior and FEM 87
3.31. 3D Poisson problem – Gains vs prior quality 88
3.32. 2D Navier–Stokes problem – Network and training parameters 91

4.1. Adjoint method – gradient computation steps 143
4.2. 2D Poisson problem – Batch of solves 157

5.1. d-dimensional Poisson problem – Cost to reach a target error (FEM, DG) 167

5.2. d-dimensional Poisson problem – Data-fitting and PINN L^2 errors 169
5.3. 1D problem (PINN, FEM patchwise) – L^2 errors and gains, $k = 3$ 176
5.4. 1D problem (PINN, FEM patchwise) – L^2 errors and gains, $k = 4$ 176
5.5. 1D problem (PINN, FEM patchwise) – L^2 errors and gains vs polynomial degree 179
5.6. 1D problem (PINN, FEM patchwise) – L^2 errors and gains vs number of cells 180
5.7. 1D problem (PINN, FEM patchwise) – L^2 errors and gains vs network width 180
5.8. 1D problem (PINN, FEM patchwise) – Budget study summary, multiplier comparison 194
5.9. 1D problem (PINN, FEM patchwise) – L^2 errors and gains, $k = 3$, positive multiplier with bounded weights 197
5.10. 1D problem (PINN, FEM patchwise) – L^2 errors and gains, $k = 4$, positive multiplier with bounded weights 197
5.11. 1D problem (PINN, FEM cellwise) – L^2 errors and gains, $k = 3$ 200
5.12. 1D problem (PINN, FEM cellwise) – L^2 errors and gains, $k = 4$ 200
5.13. 1D problem (PINN, DG patchwise) – L^2 errors and gains, $k = 3$ 203
5.14. 1D problem (PINN, DG patchwise) – L^2 errors and gains, $k = 4$ 204
5.15. 1D problem (PINN, DG cellwise) – L^2 errors and gains, $k = 3$ 207
5.16. 1D problem (PINN, DG cellwise) – L^2 errors and gains, $k = 4$ 207
5.17. 3D problem (PINN, FEM patchwise) – Cost at equal time 216
5.18. 1D linear transport problem (NO, DG patchwise) – L^2 errors and gains, shared basis 223
5.19. 1D linear transport problem (NO, DG patchwise) – L^2 errors and gains, $\boldsymbol{\mu}$-dependent basis 223
5.20. 1D linear transport problem (NO, DG patchwise) – L^2 errors and gains, field-dependent basis 224
5.21. 1D turning point problem (NO, DG patchwise) – L^2 errors and gains, shared basis 229
5.22. 1D turning point problem (NO, DG patchwise) – L^2 errors and gains, μ-dependent basis 229

A.1. *Some common notations (function spaces, norms, etc.).* 239
A.2. *Section 2.1 – Parametric PDE.* 240
A.3. *Section 2.2.1 – Standard Finite element method.* 240
A.4. *Section 2.2.2 – Discontinuous Galerkin method.* 241
A.5. *Section 2.2.3 – φ-FEM method.* 242
A.6. *Section 2.3 – Physics-informed neural networks.* 243
A.7. *Chapter 3 – Enriched Finite element methods.* 244
A.8. *Section 3.5.1 – Setup of the numerical experiments.* 244
A.9. *Section 3.6 – 2D Navier–Stokes test case.* 245
A.10. *Section 4.2 – Geometry and integration modules of* `ScimBa`. 246
A.11. *Section 4.3 – Basis functions modules of* `ScimBa`. 247
A.12. *Section 4.4 – DG variable and post-processing modules of* `ScimBa`. 248
A.13. *Sections 4.5, 4.7 and 4.8 – Scheme assembly modules of* `ScimBa`. 248
A.14. *Section 4.9 – Learnable formalization of the Galerkin schemes.* 249
A.15. *Chapter 5 – Learnable Galerkin methods.* 249

List of Algorithms

5.1. Training of the enriched Galerkin space – PINN loss 173
5.2. Training of the enriched DG space – Data loss 220

1 Introduction

Contents

1.1. Context and motivation 1
1.2. Outline and contributions 4

1.1. Context and motivation

Partial differential equations (PDEs) are used to model a large number of problems encountered in science and engineering, such as fluid flow, the deformation of an elastic structure, or heat diffusion. In the vast majority of cases of practical interest, these equations do not admit an analytical solution, and numerical methods must be used to construct an approximation. Solving these problems both accurately and efficiently in terms of computational time is a central challenge. This is all the more true in many contexts where solutions must be computed quickly and repeatedly: real-time simulation and digital twins, interactive applications such as surgical simulation, design optimization, inverse problems, or uncertainty quantification, which all require a large number of resolutions.

Several major families of numerical methods, known as high-fidelity methods, have been developed to discretize these PDEs. Historically, the first were finite differences [87], which approximate differential operators directly through schemes built on a structured grid. However, their ease of implementation comes at a cost, in particular the geometric rigidity of a Cartesian grid, which is poorly suited to complex domains. The finite element method (FEM) [27, 55], in contrast, relies on a variational formulation of the problem and a discretization by an unstructured mesh. It offers great geometric flexibility and solid mathematical foundations, with a priori error analyses, which makes it today the dominant method in engineering. Finally, finite volume methods [58] directly discretize the conservative integral forms of the PDE over a set of control volumes, which guarantees by construction the local conservation of physical quantities. This property, which is particularly sought after in fluid mechanics and in the context of conservation laws, makes it the method of choice in these fields, even though it involves a mathematical analysis that is generally more complex to carry out than that of the finite element method. To these three families can be added spectral methods [34], which adopt a radically different approach by representing the solution no longer locally, but globally, using basis functions defined over the entire domain. This global representation makes them remarkably efficient for smooth solutions on simple geometries, but not very robust when faced with complex domains or solutions exhibiting singularities.

These classical high-fidelity methods nevertheless present limitations that motivate more recent developments. On the one hand, for numerical methods relying on a mesh,

constructing a mesh that exactly conforms to the boundary of a complex geometry can be costly, or even difficult. This difficulty has motivated the development of so-called unfitted methods, which embed the domain in a simple background mesh without requiring its boundary to fit with that of the physical domain. Two broad families are generally distinguished: immersed boundary methods (IBMs) [95, 105], and fictitious domain methods [66]. Among the latter, one can notably mention eXtended Finite Element Method (XFEM) [70] and cut finite element method (CutFEM) [30, 31], which impose the boundary condition by enriching the approximation space or through non-standard integration over cut elements, as well as the φ-FEM method, initially introduced in [53], which uses a level-set function to represent the domain, and thereby impose the boundary condition. A more radical response consists in dispensing with a mesh entirely, which is the principle of meshless methods [22], where the approximation is built from a set of points without any connectivity. Isogeometric analysis (IGA) [20, 62, 75] addresses the geometric difficulty differently, by using the same spline-based representation for the geometry and for the approximation, which avoids any geometry approximation but still relies on a mesh.

On the other hand, when the problem depends on one or several physical parameters and must be solved a large number of times for different values of these parameters, as in the contexts mentioned above, repeating a full resolution with one of these high-fidelity methods for each value quickly becomes prohibitive. This has motivated the development of reduced-order models (ROMs), whose common idea is to build offline a reduced-dimension approximation space from high-fidelity simulations, and then to solve online, at a lower cost, the partial differential equation projected onto this subspace for each new parameter. This approach is efficient when the set of solutions obtained as the parameter varies, the so-called solution manifold, is of small dimension, i.e. when it is well approximated by a low-dimensional linear subspace. Examples include proper orthogonal decomposition (POD) [23, 109] and the reduced basis method [104, 107], as well as proper generalized decomposition (PGD) [37]. Independently of the parametric nature of the problem, another strategy for reducing the cost of resolution consists in limiting the number of degrees of freedom, either by refining the mesh only where an a posteriori error estimator indicates it is needed [3], with basis functions that are local solutions of the PDE (Trefftz method [72, 77, 97]), or by hybridization, i.e. by solving a global system only on the mesh skeleton, as in hybridizable discontinuous Galerkin (HDG) methods [39, 76, 106] and hybrid high-order (HHO) methods [56].

Finally, these classical high-fidelity methods all rely on finite-dimensional approximation spaces built a priori using piecewise polynomials or global modes, whose expressivity may be insufficient to efficiently represent solutions exhibiting complex structures (e.g. boundary layers, singularities) without resorting to a very fine mesh. Two types of responses to this limitation can be found in the literature. The first consists in locally enriching the approximation space with special functions that capture the local features of the solution, which is the principle of the generalized finite element method (GFEM) [65, 120]. GFEM belongs to the family of partition-of-unity methods [11] (see also [12, 21]), of which XFEM [96, 121] is a particular case. The choice of these enrichment functions, however, remains largely problem-dependent, which makes them poorly suited to the parametric setting. This concept of enrichment can, however, be generalized by replacing these problem-dependent functions with a prior obtained by

other means. This is the case, for example, with the φ-FEM method, already mentioned above as a means of avoiding the meshing constraint, which can also be regarded as enriching the approximation space with a geometric prior, namely the level-set function φ. Since the solution is sought in the form $u = \varphi w$, with w the new unknown, φ-FEM is precisely a multiplicative enrichment, which is the structure we later reuse with a neural prior. Another example is the enrichment proposed in [33], where the authors deal with parametric problems by additively enriching finite element spaces with functions determined by PGD. The second, more radical response consists in giving up this finite-dimensional vector space altogether, in favor of a fundamentally different kind of approximating object, which is precisely the path taken by neural methods.

These last few years have thus seen the emergence of scientific machine learning (SciML), which proposes to replace, or complement, these classical approximation spaces with neural networks. Two main arguments motivate this approach. On the one hand, neural networks are universal approximators [43, 73], capable of representing complex functions without requiring discretization by a mesh. The sought solution is then no longer projected onto a finite-dimensional vector subspace, but onto a finite-dimensional submanifold of the function space, parameterized by the weights of the network. On the other hand, in the parametric context mentioned above, classical high-fidelity methods quickly become too costly. While linear ROMs lose their efficiency when the solution manifold is not of small dimension, as for transport-dominated problems, neural networks offer a natural alternative. Indeed, the parameters can simply be added as inputs of the network, and a single training phase, performed offline, then reduces the cost over the whole parametric family, in the same offline/online spirit as ROMs but with a non-linear approximation.

Two main paradigms stand out in the literature, distinguished by the way the network is trained. The first consists in minimizing a loss built from the PDE without requiring data, namely the strong-form residual in the case of Physics-Informed Neural Networks (PINNs) [108], an energy functional in the case of the Deep Ritz method [54], or a weak-form residual in the case of Variational PINNs (VPINNs) [82]. In this first paradigm, the solution is sought as the output of a neural network, which can also take the parameters as additional inputs, and it is the network itself that constitutes the approximation space. Similar methods already existed in the 1990s [84, 86, 93], but progress in deep learning and computing power has since considerably expanded their field of application. The second one, known as neural operators (NOs), relies on supervised learning from data and consists in learning an operator between function spaces (for instance from a coefficient field or a source term to the corresponding solution), as done by DeepONet [90] or Fourier Neural Operators (FNOs) [88]. Physics-informed variants, such as the physics-informed neural operator (PINO) [89], add a PDE-residual term to the loss. The second paradigm can moreover be seen as a non-linear generalization of the model-order-reduction methods mentioned above, with the learned non-linear manifold playing the role of the reduced space. We refer to [59] for a broader review of the different ways of combining neural networks with physical models, including this dichotomy.

These neural methods, however, present well-identified limitations, such as the lack of theoretical guarantees and reliable error estimators, training difficulties and a strong sensitivity to hyperparameters, and degraded performance for solutions that are not

smooth. Boundary conditions are also difficult to impose exactly, whether through penalization or through a level-set function. At present, these techniques are therefore not competitive with classical methods [69], in particular for a single resolution. They become advantageous, however, as soon as a multitude of resolutions is required, typically in the parametric setting.

To combine the advantages of these two large families of PDE-solving methods, high-fidelity methods (and their derived methods) and neural methods, several hybridization strategies have already been proposed in the literature. A first one consists in enriching the approximation space of a classical method using a neural prior, as in [129] for a GFEM basis or [63] for a discontinuous Galerkin basis. A second strategy consists in directly structuring the network's representation with finite elements, by interpolating its predictions onto a mesh [13, 130] or by embedding the shape functions directly in its architecture [118]. Other forms of hybridization also exist: a multiscale coupling, where a coarse classical resolution is locally corrected by a network [92], or a decomposition of the problem split according to its variables (space entrusted to the network, time discretized classically) [60]. A network prediction can also be used to accelerate a classical iterative solver, such as Newton's method [2, 101], an approach that we ourselves reuse in Section 3.6.3 to initialize the resolution of the Navier–Stokes system.

Differentiable programming has also been applied to numerical schemes themselves, which are written in an automatic differentiation framework so that gradients can be propagated through the resolution, notably for inverse design, as in JAX-FEM for finite elements [131]. Building on such solvers, a network can be trained through the solver itself to correct the scheme, with a neural source term in a differentiable discontinuous Galerkin (DG) solver [81] or corrections of the weak form in a finite element scheme [80].

It is this hybrid direction that we adopt in this manuscript: rather than replacing classical mesh-based methods, we build on their structure, either by enriching a fixed finite element approximation with a neural prior, or by making the numerical scheme itself differentiable so that it can be trained directly. These two complementary contributions, together with the outline of the manuscript, are presented in the following section.

1.2. Outline and contributions

This manuscript develops this hybridization around two main contributions, organized as follows.

Chapter 2 lays the foundations needed for the rest of the manuscript. We first introduce the general parametric elliptic problem considered throughout the thesis (Section 2.1). We then present the three classical numerical methods on which our contributions build, namely continuous finite elements (Section 2.2.1), DG methods (Section 2.2.2), and φ-FEM (Section 2.2.3). We conclude by introducing physics-informed neural networks (Section 2.3), which will serve as the neural prior in the first contribution. In the last chapter, they are used to build neural basis functions that enrich the approximation space of the learnable FEM and DG schemes, the network being trained through the solver by minimizing a PINN-type loss.

First contribution. The first contribution, presented in Chapter 3 and published as an article [16] in the "Mathematical Modelling and Numerical Analysis" (M2AN) journal, proposes two ways of enriching the classical continuous Lagrange finite element approximation space using a neural prior. The prior is incorporated into the approximation space either additively (Section 3.1) or multiplicatively (Section 3.2). For both approaches, we prove error estimates showing that the enrichment preserves the classical convergence orders of the finite element method, with a gain that depends explicitly on the quality of the prior, and in particular on the quality of its high-order derivatives. The two approaches are then compared (Section 3.3), before detailing their practical implementation (Section 3.4) and validating them numerically on parametric problems in dimensions 1, 2 and 3 (Sections 3.5 and 3.6). These experiments show that, for a given accuracy, a coarser mesh can be used with our enriched FEM than with the standard FEM, which reduces the computational cost, in particular in the parametric setting.

This work, to which I was the primary contributor, is the result of a collaboration between three Inria project teams (MIMESIS, MAKUTU and MACARON) and IMAG Montpellier. The code associated with this work is publicly available[1].

Second contribution. The second contribution, presented in Chapters 4 and 5, builds on a different coupling principle: rather than enriching a fixed finite element approximation space with a fixed neural prior, the solver itself (DG, FEM or φ-FEM) is made differentiable, and therefore learnable. Chapter 4 presents, for this purpose, part of the `ScimBa`[2] library, developed by the Inria project team MACARON. Initially written in `PyTorch`, it has been undergoing a rewrite in `JAX` since early 2026, the version on which this part of the manuscript is based. A first section presents the elements of differentiable programming on which the library relies (Section 4.1). The following sections detail the different modules built within the DG scheme (Sections 4.2 to 4.6). This same architecture is then extended, without modifying the underlying DG modules, to the continuous finite element method (Section 4.7) and then to φ-FEM (Section 4.8). The formalism making these three schemes learnable is then presented (Section 4.9), before concluding the chapter with numerical validations (Section 4.10).

Chapter 5 then builds on this differentiable framework to return to the lack of expressivity of classical approximation spaces, by enriching the Galerkin basis (Section 4.3) with a neural network. A dimensional study (Section 5.1) first motivates this enrichment, by showing the benefit of networks as the dimension increases, even though they do not offer the theoretical guarantees of Galerkin methods. The classical basis is therefore multiplied by a function depending on a network, the solution still being obtained by solving the scheme in the enriched basis. This enrichment is studied on advection-diffusion problems with a boundary layer, first for a fixed value of the parameter, in dimension 1 (Section 5.2) and then in dimension 3 (Section 5.3), where classical methods become costly, then for a whole family of parameters in dimension 1 (Section 5.4), a single network then being trained in a supervised manner and used without any new training. In all cases, for a fixed number of degrees of freedom, the results obtained show that the learned basis can reduce the L^2 error by several orders

[1] https://github.com/flecourtier/EnrichedFEMUsingPINNs
[2] https://www.scimba.org/

of magnitude compared with the classical basis. Moreover, in the parametric case, the study shows that making the basis depend on the parameter becomes essential when the boundary layer moves with it, a shared basis then no longer improving the classical solution.

This contribution differs from the previous one in two ways. First, the network is no longer a prior that is pretrained and then frozen, but is learned through the solver, thanks to the implicit differentiation of Chapter 4. Second, it is no longer a question of an additively or multiplicatively enriched space whose error is controlled a priori, but of a learned basis for which we have no error estimate. The results are therefore experimental, obtained on 1D and 3D problems and compared mainly at a fixed number of degrees of freedom, the training cost being only discussed qualitatively in 1D and compared with that of the classical method on the 3D problem, and constitute a feasibility study of the learned enrichment rather than a competitive solver. The code associated with this work is publicly available[3].

In this context, I have implemented, in close collaboration with Emmanuel Franck, Victor Michel-Dansac and Rémi Imbach, all of the modules specific to the DG scheme presented in the architecture of Figure 4.1, as well as its FEM and φ-FEM extensions presented in Figures 4.7 and 4.8. This architecture nevertheless relies on building blocks already present in `ScimBa`, which are not part of my contribution. These are the base class from which almost every module of the library inherits, and the optimization tools (optimizers such as adaptive moment estimation (Adam), limited-memory BFGS (LBFGS) or natural gradient methods, and network architectures such as PINNs or neural operators). Developed by other members of the team, they constitute the core of the training of the learnable Galerkin schemes presented. At the time of writing, a dedicated article presenting all of these developments is in preparation. It is also worth noting that the library's architecture was designed to remain reusable with minimal modifications, which has allowed extensions to be developed within the Inria project team MACARON, such as an extension to time-dependent problems and structured meshes, under development at the time of writing.

We begin, in the following chapter, by laying out the general framework of the parametric elliptic problem considered throughout this manuscript, together with the classical numerical methods and the physics-informed neural networks on which our two contributions are built.

[3] https://github.com/flecourtier/thesis-learnable-methods

2 Numerical methods for PDEs

Contents

2.1. Problem setting **7**

2.2. Some classical approximation methods **8**

2.2.1. Continuous finite element method (FEM) 8

2.2.2. Discontinuous Galerkin (DG) approach 11

2.2.3. φ-FEM approach 19

2.3. Physics-Informed Neural Networks (PINNs) **23**

2.3.1. Physics-Informed Neural Networks for parametric PDEs 23

2.3.2. Improving PINN training and prediction 25

The notations introduced in this chapter are recalled in Appendix A.1.1.

This chapter introduces the numerical methods, and the associated notations, that serve as the foundation for the rest of this manuscript. Section 2.1 first states the general parametric elliptic problem considered throughout this manuscript. Section 2.2 then presents the classical, mesh-based approximation methods used to discretize this problem for a fixed parameter $\boldsymbol{\mu}$: the continuous finite element method, the discontinuous Galerkin method, and an unfitted variant of FEM based on a level-set function (φ-FEM). Finally, Section 2.3 presents physics-informed neural networks, which, in contrast, seek the solution directly as the output of a neural network, trained by minimizing the (parametric) residual of the PDE.

2.1. Problem setting

Throughout this manuscript, we consider general parametric linear elliptic differential equations defined on a domain $\Omega \subset \mathbb{R}^d$ with d space dimensions, with a Lipschitz boundary $\Gamma := \partial\Omega$ (for instance a polytope). Consider a parameter space $\mathcal{M} := \{\boldsymbol{\mu} = (\mu_1, \dots, \mu_p) \in \mathbb{R}^p\}$. The typical problem of interest is, for one or several $\boldsymbol{\mu} \in \mathcal{M}$, to find $u(\cdot, \boldsymbol{\mu}) : \Omega \to \mathbb{R}$ such that:

$$\mathcal{L}\big(u\,; \boldsymbol{x}, \boldsymbol{\mu}\big) = f(\boldsymbol{x}, \boldsymbol{\mu}), \tag{2.1}$$

with $\boldsymbol{x} := (x_1, \dots, x_d) \in \Omega$ the space variable, and where $\mathcal{L}$ is the parametric differential operator defined by:

$$\mathcal{L}(\,\cdot\,; \boldsymbol{x}, \boldsymbol{\mu}) : u \mapsto R(\boldsymbol{x}, \boldsymbol{\mu})u + \boldsymbol{C}(\boldsymbol{x}, \boldsymbol{\mu}) \cdot \nabla u - \frac{1}{\mathrm{Pe}} \nabla \cdot (D(\boldsymbol{x}, \boldsymbol{\mu}) \nabla u), \tag{2.2}$$

with $f(\boldsymbol{x}, \boldsymbol{\mu}) \in L^2(\Omega)$ the source term, $R(\boldsymbol{x}, \boldsymbol{\mu}) \in L^\infty(\Omega)$ the reaction coefficient, $\boldsymbol{C}(\boldsymbol{x}, \boldsymbol{\mu}) \in [W^{1,\infty}(\Omega)]^d$ the convection coefficient, $D(\boldsymbol{x}, \boldsymbol{\mu}) \in [W^{1,\infty}(\Omega)]^{d\times d}$ the (symmetric and positive definite) diffusion matrix, and $\mathrm{Pe} \in \mathbb{R}_+^*$ the Péclet number representing the ratio between convection and diffusion. We suppose that the bilinear form associated with $\mathcal{L}$ is coercive. The differential operator is considered with Dirichlet, Neumann or Robin boundary conditions, which can also depend on $\boldsymbol{\mu}$.

Having introduced the general parametric elliptic problem (2.1) that serves as the model problem throughout this manuscript, we now turn to the classical, mesh-based methods used to discretize it for a fixed parameter $\boldsymbol{\mu}$.

2.2. Some classical approximation methods

In this section, we work with the parametric elliptic problem (2.1) and present three classical approximation methods, used throughout the rest of this manuscript, that discretize it directly for a fixed parameter $\boldsymbol{\mu}$. The first is the continuous finite element method (Section 2.2.1), and the second is the discontinuous Galerkin method (Section 2.2.2), in two variants, namely the Symmetric Interior Penalty Galerkin (SIPG) and the Non-symmetric Interior Penalty Galerkin (NIPG). The third is an unfitted finite element approach based on a level-set function (φ-FEM, Section 2.2.3).

All these approaches are based on a variational formulation of the problem, obtained by multiplying the PDE by a test function and integrating by parts. They are of course not the only classical paradigms for the direct discretization of PDEs, but the other major families (finite difference, finite volume and spectral methods), which are reviewed in Chapter 1, are not treated in this manuscript.

Notation 2.1. *It should be noted that, throughout this section, the notation u_h is used interchangeably across the various methods to represent the approximate solution to the PDE. This choice has been made for the sake of brevity, even though it actually denotes different objects depending on the method, as these belong to distinct spaces.*

2.2.1. Continuous finite element method (FEM)

We recall here the classical FEM and introduce the notation used throughout the manuscript. For conciseness, we work with a fixed $\boldsymbol{\mu}$ and omit the dependence on it. We first recall the variational formulation and its discretization with continuous Lagrange finite elements, then the Lagrange interpolation operator, and finally the expected convergence rates, which follow from the interpolation error and elliptic regularity estimates. To solve the problem (2.1) under consideration for a fixed parameter $\boldsymbol{\mu}$ with homogeneous Dirichlet boundary conditions using the continuous FEM, we rewrite it as the following variational problem:

$$\text{Find } u \in V^0 := H_0^1(\Omega) \text{ such that, } \forall v \in V^0,\ a(u, v) = l(v), \tag{2.3}$$

where the bilinear form a is given by:

$$a(u,v) := \int_\Omega R(\boldsymbol{x})\, u(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_\Omega v(\boldsymbol{x})\, \boldsymbol{C}(\boldsymbol{x}) \cdot \nabla u(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \frac{1}{\mathrm{Pe}} \int_\Omega D(\boldsymbol{x}) \nabla u(\boldsymbol{x}) \cdot \nabla v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},$$

while the linear form l reads:

$$l(v) := \int_\Omega f(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

Remark 2.2. *Since a is continuous on $V^0 \times V^0$ and coercive and l is continuous on V^0, the existence and uniqueness of the solution u are ensured by the Lax–Milgram theorem.*

Discrete problem. Let $\mathcal{T}_h$ be a mesh of the domain Ω composed of simplexes, where h denotes the characteristic size of the mesh, i.e. the largest diameter of the simplexes. We suppose that $\mathcal{T}_h$ satisfies the Ciarlet condition (see e.g. [55]) and that its boundary is exactly Γ, i.e. that the mesh is fitted to the domain, as illustrated in the left panel of Figure 2.1. Consider $V_h^0 \subset V_h \subset V := H^1(\Omega)$ the two continuous Lagrange finite element spaces of degree $k \geqslant 1$ defined by:

$$V_h := \left\{ v_h \in C^0(\Omega),\ \forall K \in \mathcal{T}_h,\ v_h|_K \in \mathbb{P}_k \right\}, \tag{2.4}$$

and

$$V_h^0 := \left\{ v_h \in C^0(\Omega),\ \forall K \in \mathcal{T}_h,\ v_h|_K \in \mathbb{P}_k,\ v_h|_\Gamma = 0 \right\}, \tag{2.5}$$

with $\mathbb{P}_k$ the space of polynomials with real coefficients of degree at most k. The solution to (2.3) is approximated by the solution u_h to:

$$\text{Find } u_h \in V_h^0 \text{ such that, } \forall v_h \in V_h^0,\ a(u_h, v_h) = l(v_h). \tag{2.6}$$

Lagrange interpolation. Let us now introduce some results used in the remainder of the manuscript, in particular in Chapter 3. We first define the Lagrange interpolation operator by:

$$\mathcal{I}_h : C^0(\Omega) \ni v \mapsto \sum_{i=1}^{N_{\mathrm{dofs}}} v\big(\boldsymbol{x}^{(i)}\big)\phi_i \in V_h, \tag{2.7}$$

with $(\boldsymbol{x}^{(i)})_{i\in\{1,\dots,N_{\mathrm{dofs}}\}}$ the N_{dofs} degrees of freedom (dofs) associated to the mesh, and $(\phi_i)_{i\in\{1,\dots,N_{\mathrm{dofs}}\}}$ the associated Lagrange shape functions of degree k. These Lagrange shape functions are the unique continuous piecewise polynomials of total degree at most k satisfying the interpolation property:

$$\phi_i(\boldsymbol{x}^{(j)}) = \delta_{ij}, \quad \forall i,j \in \{1,\dots,N_{\mathrm{dofs}}\},$$

where δ_{ij} is the Kronecker symbol.

Notation 2.3. *In the whole manuscript, for a Sobolev space H, the notation $|\cdot|_H$ and $\|\cdot\|_H$ denote respectively the semi-norm and the norm in H. Namely, for $v \in H^1(\Omega)$,*

we set:

$$|v|^2_{H^1(\Omega)} := \int_\Omega |\nabla v(\boldsymbol{x})|^2 \,\mathrm{d}\boldsymbol{x}, \quad \textit{and} \quad \|v\|^2_{H^1(\Omega)} := \int_\Omega |v(\boldsymbol{x})|^2 \,\mathrm{d}\boldsymbol{x} + \int_\Omega |\nabla v(\boldsymbol{x})|^2 \,\mathrm{d}\boldsymbol{x}.$$

More generally, for an integer $m \geqslant 1$, $|v|_{H^m(\Omega)}$ gathers the derivatives of order exactly m of v, and $\|v\|^2_{H^m(\Omega)}$ is the sum of the squared L^2 norms of all its derivatives up to order m.

Expected convergence rates. The following result gives a bound on the interpolation error:

Theorem 2.4 (see e.g. [55])**.** *There exists $C_s > 0$, independent of h, such that for all $v \in H^{s+1}(\Omega)$ and $1 \leqslant s \leqslant k$:*

$$\|v - \mathcal{I}_h v\|_{H^1} \leqslant C_s h^s |v|_{H^{s+1}}.$$

The next estimate is associated with the elliptic regularity:

Theorem 2.5 (see e.g. [57, 68])**.** *Suppose that Ω is a bounded open set, either convex or with a boundary Γ of class $C^{1,1}$, and that $R \in L^\infty(\Omega)$, $\boldsymbol{C} \in [W^{1,\infty}(\Omega)]^d$ and $D \in [W^{1,\infty}(\Omega)]^{d\times d}$ in the definition of $\mathcal{L}$ given in (2.2). There exists $C_e > 0$ such that for all $\xi \in L^2(\Omega)$ and every weak solution $w \in H^1_0(\Omega)$ to:*

$$\mathcal{L}^* w = \xi, \tag{2.8}$$

with homogeneous Dirichlet boundary condition, we have $w \in H^2(\Omega)$ and:

$$\|w\|_{H^2} \leqslant C_e \|\xi\|_{L^2}.$$

Here $\mathcal{L}^*$ represents the adjoint of the operator $\mathcal{L}$.

These estimates, combined with Céa's lemma, which uses the continuity and coercivity of a, give the following error estimate:

Theorem 2.6 (see e.g. [55])**.** *Let Ω be a convex polytope and $1 \leqslant s \leqslant k$. Considering $u \in H^{s+1}(\Omega)$ and $u_h \in V^0_h$ the solutions to (2.3) and (2.6), one has:*

$$|u - u_h|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} h^s |u|_{H^{s+1}}, \tag{2.9}$$

and

$$\|u - u_h\|_{L^2} \leqslant C_e C_1 C_s \frac{\gamma^2}{\alpha} h^{s+1} |u|_{H^{s+1}}, \tag{2.10}$$

where γ and α are respectively the constants of continuity and coercivity of a for the norm $|\cdot|_{H^1}$ on V^0, and C_1 is the constant C_s of Theorem 2.4 taken at $s = 1$.

Proof of Theorem 2.6. **H^1-error:** This is a direct application of Céa's lemma. By Galerkin orthogonality (difference of (2.3) and (2.6)), $a(u - u_h, v_h - u_h) = 0$ for all $v_h \in V^0_h$, so that, by coercivity and continuity of a:

$$\begin{aligned} \alpha |u - u_h|^2_{H^1} &\leqslant a(u - u_h, u - u_h) = a(u - u_h, u - v_h), \quad \forall v_h \in V^0_h, \\ &\leqslant \gamma |u - u_h|_{H^1} |u - v_h|_{H^1}, \quad \forall v_h \in V^0_h, \end{aligned}$$

which immediately leads to:

$$|u - u_h|_{H^1} \leqslant \frac{\gamma}{\alpha}|u - v_h|_{H^1}, \quad \forall v_h \in V_h^0.$$

We apply it to $v_h = \mathcal{I}_h u \in V_h^0$ with $\mathcal{I}_h$ the Lagrange interpolation operator (2.7); using the interpolation estimate given in Theorem 2.4, we obtain (2.9).

L^2-error: We follow the Aubin–Nitsche technique. Consider $w \in H_0^1(\Omega) \cap H^2(\Omega)$ solution to:

$$\mathcal{L}^* w = u - u_h,$$

with homogeneous Dirichlet boundary condition. Thanks to Theorem 2.5, one has:

$$\|w\|_{H^2} \leqslant C_e \|u - u_h\|_{L^2}. \tag{2.11}$$

Since $u - u_h \in H_0^1(\Omega)$, the definition of the adjoint gives:

$$\|u - u_h\|_{L^2}^2 = \int_\Omega (u - u_h)\, \mathcal{L}^* w \, \mathrm{d}\boldsymbol{x} = a(u - u_h, w).$$

Using then the Galerkin orthogonality and the continuity of the bilinear form a:

$$a(u - u_h, w) = a(u - u_h, w - \mathcal{I}_h w) \leqslant \gamma |u - u_h|_{H^1} |w - \mathcal{I}_h w|_{H^1}.$$

Thanks to Theorem 2.4 taken at $s = 1$ (which defines C_1) and (2.11):

$$|w - \mathcal{I}_h w|_{H^1} \leqslant C_e C_1 h \|u - u_h\|_{L^2},$$

which leads to the conclusion (2.10) by using (2.9). □

Remark 2.7. *For the sake of simplicity, we consider an elliptic boundary value problem with homogeneous Dirichlet conditions. More general boundary conditions, e.g. Robin-like conditions depending on the parameters, can be used as well.*

2.2.2. Discontinuous Galerkin (DG) approach

The goal of this section is to recall the DG method by working on a fixed $\boldsymbol{\mu}$, for which we omit the dependence on $\boldsymbol{\mu}$ for conciseness, as done in Section 2.2.1. We begin by presenting a unified formulation of the DG method (Section 2.2.2.1). Then, we present the numerical fluxes for the diffusive term used in the numerical experiments of this manuscript, that is, the SIPG and NIPG methods (Section 2.2.2.2). Finally, we present the upwind numerical flux for the convective term, which can be combined with either of them (Section 2.2.2.3).

2.2.2.1. Unified formulation of the DG method

The objective here is to present a unified formulation of the DG method as introduced in [9] (for the Poisson problem), which allows grouping the different variants of the DG method (SIPG, NIPG, etc.) under a single general form (see [10] for general advection-diffusion-reaction operators). Thus, we consider the problem (2.1)–(2.2) with homogeneous Dirichlet boundary conditions:

$$R\, u + \boldsymbol{C} \cdot \nabla u - \frac{1}{\mathrm{Pe}} \nabla \cdot (D \nabla u) = f \text{ in } \Omega, \qquad u = 0 \text{ on } \Gamma. \tag{2.12}$$

We begin by reformulating the diffusive part of the operator as a first-order system, which is easier to discretize with discontinuous elements. Indeed, the diffusive term is the only one for which both the trace of u and the trace of its normal derivative are simultaneously ambiguous at the interfaces, which motivates the introduction of an auxiliary flux variable; the reaction and (first-order) convection terms are instead discretized directly as volume contributions, without requiring such a reformulation (see Remark 2.8). We introduce the auxiliary variable $\boldsymbol{\varsigma} := \nabla u$, which gives the system:

$$\boldsymbol{\varsigma} = \nabla u, \qquad R\,u + \boldsymbol{C}\cdot\nabla u - \frac{1}{\mathrm{Pe}}\nabla\cdot(D\boldsymbol{\varsigma}) = f \text{ in } \Omega, \qquad u = 0 \text{ on } \Gamma.$$

As with its continuous equivalent defined in Section 2.2.1, let $\mathcal{T}_h$ be a mesh of the domain Ω satisfying the same assumptions, with h its characteristic size. We also define $\mathcal{F}_h^I$ as the set of interior interfaces of the mesh, i.e. faces shared by exactly two elements of $\mathcal{T}_h$, and $\mathcal{F}_h^B$ as the set of faces lying on Γ, so that $\mathcal{F}_h := \mathcal{F}_h^I \cup \mathcal{F}_h^B$ is the set of all faces associated with the mesh elements.

Consider V_h^{disc} and Σ_h^{disc} the two discontinuous Lagrange finite element spaces of degree $k \geqslant 1$ defined by:

$$V_h^{\mathrm{disc}} := \left\{ v_h \in L^2(\Omega),\ \forall K \in \mathcal{T}_h,\ v_h|_K \in \mathbb{P}_k \right\}, \tag{2.13}$$

and

$$\Sigma_h^{\mathrm{disc}} := \left\{ \boldsymbol{\tau}_h \in [L^2(\Omega)]^d,\ \forall K \in \mathcal{T}_h,\ \boldsymbol{\tau}_h|_K \in \mathbb{P}_k \right\},$$

with $\mathbb{P}_k$ the space of polynomials with real coefficients of degree at most k.

We seek $u_h \in V_h^{\mathrm{disc}}$ and $\boldsymbol{\varsigma}_h \in \Sigma_h^{\mathrm{disc}}$ satisfying, on each element $K \in \mathcal{T}_h$, the local weak formulation obtained by testing the system above against $v_K \in \mathbb{P}_k(K)$ and $\boldsymbol{\tau}_K \in [\mathbb{P}_k(K)]^d$, integrating over K, and integrating by parts the diffusive term, for all $K \in \mathcal{T}_h$:

$$\int_K \boldsymbol{\varsigma}_h\cdot\boldsymbol{\tau}_K = -\int_K u_h\,\nabla\cdot\boldsymbol{\tau}_K + \int_{\partial K} u_h\,\boldsymbol{n}_K\cdot\boldsymbol{\tau}_K,$$

$$\int_K R\,u_h\,v_K + \int_K \boldsymbol{C}\cdot\nabla u_h\,v_K + \frac{1}{\mathrm{Pe}}\int_K D\boldsymbol{\varsigma}_h\cdot\nabla v_K - \frac{1}{\mathrm{Pe}}\int_{\partial K} D\boldsymbol{\varsigma}_h\cdot\boldsymbol{n}_K\,v_K = \int_K f v_K,$$

where $\boldsymbol{n}_K$ denotes the outward unit normal to K.

In DG methods, the functions $u_h \in V_h^{\mathrm{disc}}$ and $\boldsymbol{\varsigma}_h \in \Sigma_h^{\mathrm{disc}}$ are discontinuous at the interfaces, so the traces $u_h|_{\partial K}$ and $D\boldsymbol{\varsigma}_h|_{\partial K}$ appearing in the equations above are ambiguous on $\mathcal{F}_h^I$, since there are two possible values on each side of each interface. To address this, we introduce the scalar numerical flux $\hat{u}_h$ and the vector numerical flux $\hat{\boldsymbol{\varsigma}}_h^D$, both constructed from the values on both sides of each interface. We replace the exact trace of u by $\hat{u}_h|_{\partial K}$, and the exact trace of the diffusive flux $D\boldsymbol{\varsigma}$ by $\hat{\boldsymbol{\varsigma}}_h^D|_{\partial K}$. The diffusive flux through ∂K, $D\boldsymbol{\varsigma}\cdot\boldsymbol{n}_K$, is thus approximated by $\hat{\boldsymbol{\varsigma}}_h^D\cdot\boldsymbol{n}_K$. This choice of fluxes fully defines the DG method and will be developed in the remainder of this section. We then seek $u_h \in V_h^{\mathrm{disc}}$ and $\boldsymbol{\varsigma}_h \in \Sigma_h^{\mathrm{disc}}$ such that for all $K \in \mathcal{T}_h$, $v_K \in \mathbb{P}_k(K)$ and $\boldsymbol{\tau}_K \in [\mathbb{P}_k(K)]^d$, we have:

$$\int_K \boldsymbol{\varsigma}_h\cdot\boldsymbol{\tau}_K = -\int_K u_h\,\nabla\cdot\boldsymbol{\tau}_K + \int_{\partial K} \hat{u}_h\,\boldsymbol{n}_K\cdot\boldsymbol{\tau}_K,$$

$$\int_K R\,u_h\,v_K + \int_K \boldsymbol{C}\cdot\nabla u_h\,v_K + \frac{1}{\mathrm{Pe}}\int_K D\boldsymbol{\varsigma}_h\cdot\nabla v_K - \frac{1}{\mathrm{Pe}}\int_{\partial K} \hat{\boldsymbol{\varsigma}}_h^D\cdot\boldsymbol{n}_K\,v_K = \int_K f v_K. \tag{2.14}$$

Remark 2.8. *Treating the reaction and convection terms as pure volume contributions, without a dedicated numerical flux, is standard and adequate in the diffusion-dominated regime, where the interior penalty term already present in most of the flux choices considered here (e.g. SIPG, NIPG) stabilizes the whole scheme. This treatment no longer makes sense in the convection-dominated regime (large Péclet number). Since the interior penalty term is scaled by $1/Pe$, it vanishes as $Pe \to \infty$, leaving no mechanism to couple neighboring elements through the convective term; a dedicated upwind numerical flux must then be introduced, as done in Section 2.2.2.3. A more general treatment, providing error estimates that are robust across the whole range of regimes, from diffusion- to reaction- to convection-dominated, is proposed in [10].*

Before summing the element-wise equations above, we introduce, following [9], the jump and average trace operators that let us rewrite the sums over ∂K, $K \in \mathcal{T}_h$, as integrals over $\mathcal{F}_h$. Let $F \in \mathcal{F}_h^I$ be an interior interface shared by two elements $K_L, K_R \in \mathcal{T}_h$; on F, the outward normals $\boldsymbol{n}_{K_L}$ and $\boldsymbol{n}_{K_R}$ satisfy $\boldsymbol{n}_{K_R}|_F = -\boldsymbol{n}_{K_L}|_F$. Since $v_h \in V_h^{\mathrm{disc}}$ and $\boldsymbol{\tau}_h \in \Sigma_h^{\mathrm{disc}}$ are discontinuous across F, we define the average $\{\cdot\}$ and the jump $[\![\cdot]\!]$ on F by:

$$\begin{aligned} \{v_h\} &:= \frac{1}{2}(v_h|_{K_L} + v_h|_{K_R}), \qquad [\![v_h]\!] := v_h|_{K_L}\boldsymbol{n}_{K_L}|_F + v_h|_{K_R}\boldsymbol{n}_{K_R}|_F, \\ \{\boldsymbol{\tau}_h\} &:= \frac{1}{2}(\boldsymbol{\tau}_h|_{K_L} + \boldsymbol{\tau}_h|_{K_R}), \qquad [\![\boldsymbol{\tau}_h]\!] := \boldsymbol{\tau}_h|_{K_L}\cdot\boldsymbol{n}_{K_L}|_F + \boldsymbol{\tau}_h|_{K_R}\cdot\boldsymbol{n}_{K_R}|_F. \end{aligned} \tag{2.15}$$

On a boundary face $F \in \mathcal{F}_h^B$, only one trace is available, so neither definition (2.15) applies as such; we extend only the jump of a scalar and the average of a vector to Γ, using $\boldsymbol{n}_K|_F$ for the (unique) element $K \in \mathcal{T}_h$ such that $F \subset \partial K$ (which coincides with the outward unit normal to Ω), by:

$$[\![v_h]\!] := v_h\,\boldsymbol{n}_K|_F, \qquad \{\boldsymbol{\tau}_h\} := \boldsymbol{\tau}_h, \qquad \text{on } F \in \mathcal{F}_h^B, \tag{2.16}$$

and leave $\{v_h\}$ and $[\![\boldsymbol{\tau}_h]\!]$ undefined on Γ. This choice is made precisely so that, in the two global equations below, the jump of a scalar (resp. the average of a vector) can be integrated over the whole of $\mathcal{F}_h$, while the average of a scalar (resp. the jump of a vector), left undefined on Γ, is only ever integrated over the interior set $\mathcal{F}_h^I$.

After summing over all elements and rewriting the boundary terms using the jump and average operators defined in (2.15) and (2.16), we obtain two global equations, valid for all $v_h \in V_h^{\mathrm{disc}}$ and $\boldsymbol{\tau}_h \in \Sigma_h^{\mathrm{disc}}$:

$$\int_\Omega \boldsymbol{\varsigma}_h \cdot \boldsymbol{\tau}_h = -\int_\Omega u_h \nabla_h \cdot \boldsymbol{\tau}_h + \int_{\mathcal{F}_h} [\![\hat{u}_h]\!] \cdot \{\boldsymbol{\tau}_h\} + \int_{\mathcal{F}_h^I} \{\hat{u}_h\}\,[\![\boldsymbol{\tau}_h]\!], \tag{2.17}$$

$$\begin{aligned} \int_\Omega R\,u_h\,v_h + \int_\Omega \boldsymbol{C}\cdot\nabla_h u_h\,v_h + \frac{1}{\mathrm{Pe}}\int_\Omega D\boldsymbol{\varsigma}_h\cdot\nabla_h v_h \\ - \frac{1}{\mathrm{Pe}}\int_{\mathcal{F}_h}\{\hat{\boldsymbol{\varsigma}}_h^D\}\cdot[\![v_h]\!] - \frac{1}{\mathrm{Pe}}\int_{\mathcal{F}_h^I}[\![\hat{\boldsymbol{\varsigma}}_h^D]\!]\{v_h\} = \int_\Omega f v_h, \end{aligned} \tag{2.18}$$

where, for any $v_h \in V_h^{\mathrm{disc}}$, $\nabla_h v_h$ is the broken gradient, i.e. the function whose restriction to each element $K \in \mathcal{T}_h$ is equal to $\nabla v_h|_K$. It is computed independently on each

element, as if v_h were an isolated polynomial, ignoring its jump across the interfaces (and similarly for $\nabla_h \cdot \boldsymbol{\tau}_h$, $\boldsymbol{\tau}_h \in \Sigma_h^{\text{disc}}$). This is necessary because v_h is discontinuous across $\mathcal{F}_h^I$, so it does not belong to $H^1(\Omega)$ and its usual gradient does not exist as an L^2 function on Ω.

Using the same tools as before, in particular by integrating by parts in (2.17) and substituting into (2.18) (see [9]), we eliminate $\boldsymbol{\varsigma}_h$ and obtain an equation involving only u_h. In particular, for the problem defined in (2.12), we obtain the following variational problem:

$$\text{Find } u_h \in V_h^{\text{disc}} \text{ such that, } \forall v_h \in V_h^{\text{disc}},\ a_h^{\text{disc}}(u_h, v_h) = l_h^{\text{disc}}(v_h), \tag{2.19}$$

with V_h^{disc} defined in (2.13), and where the bilinear form a_h^{disc} is defined by:

$$\begin{aligned} a_h^{\text{disc}}(u_h, v_h) := &\int_\Omega R\, u_h\, v_h + \int_\Omega \boldsymbol{C} \cdot \nabla_h u_h\, v_h + \frac{1}{\text{Pe}} \int_\Omega D \nabla_h u_h \cdot \nabla_h v_h \\ &+ \frac{1}{\text{Pe}} \int_{\mathcal{F}_h} \Big([\![\hat{u}_h - u_h]\!] \cdot \{ D \nabla_h v_h \} - \{ \hat{\boldsymbol{\varsigma}}_h^D \} \cdot [\![v_h]\!] \Big) \\ &+ \frac{1}{\text{Pe}} \int_{\mathcal{F}_h^I} \Big(\{ \hat{u}_h - u_h \} [\![D \nabla_h v_h]\!] - [\![\hat{\boldsymbol{\varsigma}}_h^D]\!] \{ v_h \} \Big), \end{aligned} \tag{2.20}$$

while the linear form l_h^{disc} is given by:

$$l_h^{\text{disc}}(v_h) := \int_\Omega f v_h. \tag{2.21}$$

This is the primal formulation of the DG method which is a classical variational equation: the right-hand side is the source term f tested against the test function, exactly as in the continuous weak formulation. The left-hand side a_h^{disc} replaces the bilinear form a (Section 2.2.1) of the continuous problem, adding all the interface terms that ensure communication between elements. As in the continuous case, the reaction and convection terms only contribute volume integrals, since we do not use a dedicated numerical flux for these terms (see Remark 2.8). Moreover, as for the continuous FEM, homogeneous Dirichlet conditions are considered here for simplicity; more general boundary conditions can be treated within the same unified framework, at the cost of adapting the boundary fluxes $\hat{u}_h$ and $\hat{\boldsymbol{\varsigma}}_h^D$ accordingly.

The richness of the different DG methods lies in the choice of the fluxes $\hat{u}_h$ and $\hat{\boldsymbol{\varsigma}}_h^D$, which determines the concrete form of a_h^{disc} and hence the stability, consistency, and accuracy properties of the method. The next section presents the diffusive fluxes that will be used in the numerical experiments of this manuscript.

2.2.2.2. Numerical fluxes for the diffusive term

We now specify the choices of numerical fluxes $\hat{u}_h$ and $\hat{\boldsymbol{\varsigma}}_h^D$; in particular, we focus on two numerical fluxes: the SIPG method and its non-symmetric version, the NIPG method.

SIPG flux. This method originates from the interior penalty (IP) method introduced by [49] for second-order elliptic and parabolic problems, analyzed in detail by [8], and recast, together with several other DG methods, in the unified flux formulation of [9].

The scalar flux $\hat{u}_h$ is set to the average $\{u_h\}$ on interior interfaces, and to the boundary value prescribed by the homogeneous Dirichlet condition on Γ, while the vector flux $\hat{\boldsymbol{\varsigma}}_h^D$ is set to the average of the diffusive flux $D\nabla_h u_h$, corrected by a term penalizing the jump of u_h:

$$\hat{u}_h = \begin{cases} \{u_h\} & \text{on } \mathcal{F}_h^I, \\ 0 & \text{on } \mathcal{F}_h^B, \end{cases} \qquad \hat{\boldsymbol{\varsigma}}_h^D = \{D\nabla_h u_h\} - \kappa \, [\![u_h]\!], \quad \text{on } \mathcal{F}_h, \tag{2.22}$$

where the penalty function $\kappa : \mathcal{F}_h \to \mathbb{R}$ is defined (face by face) by $\kappa|_F := \eta_F / h_F$, with h_F the diameter of the face F (in dimension $d = 1$, where a face is a point, the size of the adjacent cells, or the smaller of the two if they differ) and $\eta_F > 0$ a penalty parameter to be chosen large enough.

By a simple direct calculation, one has $\{\hat{u}_h - u_h\} = 0$ on $\mathcal{F}_h^I$ and $[\![\hat{u}_h - u_h]\!] = -[\![u_h]\!]$ on $\mathcal{F}_h$, and likewise $\{\hat{\boldsymbol{\varsigma}}_h^D\} = \hat{\boldsymbol{\varsigma}}_h^D$ and $[\![\hat{\boldsymbol{\varsigma}}_h^D]\!] = 0$ on $\mathcal{F}_h^I$. Substituting (2.22) into the general bilinear form (2.20) therefore makes the terms over $\mathcal{F}_h^I$ vanish entirely, and turns the remaining terms over $\mathcal{F}_h$ into the classical (symmetric) interior penalty form. We thus obtain:

$$\begin{aligned} a_h^{\mathrm{sipg}}(u_h, v_h) := \int_\Omega R\, u_h\, v_h + \int_\Omega \boldsymbol{C} \cdot \nabla_h u_h \, v_h + \frac{1}{\mathrm{Pe}} \int_\Omega D\nabla_h u_h \cdot \nabla_h v_h \\ - \frac{1}{\mathrm{Pe}} \int_{\mathcal{F}_h} \Big([\![u_h]\!] \cdot \{D\nabla_h v_h\} + \{D\nabla_h u_h\} \cdot [\![v_h]\!] \Big) + \frac{1}{\mathrm{Pe}} \mathcal{S}_h^{\mathrm{disc}}(u_h, v_h), \end{aligned} \tag{2.23}$$

where the stabilization (or interior penalty) term $\mathcal{S}_h^{\mathrm{disc}}$ is given by:

$$\mathcal{S}_h^{\mathrm{disc}}(u_h, v_h) := \int_{\mathcal{F}_h} \kappa \, [\![u_h]\!] \cdot [\![v_h]\!]. \tag{2.24}$$

The SIPG method is then defined by the variational problem (2.19) with a_h^{disc} replaced by a_h^{sipg}.

Remark 2.9. *In (2.23), we recognize, in the diffusive part, the same structure as Nitsche's method for weakly imposing Dirichlet boundary conditions on a conforming mesh [99]. The second line is symmetric in (u_h, v_h), so that a_h^{sipg} is symmetric whenever the advective term vanishes, which motivates the name of the method. The interior penalty term $\mathcal{S}_h^{disc}$ defined in (2.24) plays the same stabilizing role as the boundary penalty term of Nitsche's method, but is here applied at every interior interface of the mesh to control the jumps of the discrete solution.*

Since the reaction and convection terms only enter a_h^{sipg} through volume terms (Remark 2.8), the consistency argument of [9], stated there for the pure diffusion operator, carries over unchanged to the full operator (2.12). Thus, the exact solution u satisfies $a_h^{\mathrm{sipg}}(u, v_h) = l_h^{\mathrm{disc}}(v_h)$ for all $v_h \in V_h^{\mathrm{disc}}$. Stability and optimal convergence, however, are only established by [9] in the pure diffusion case; for the advection-diffusion-reaction operator considered here, robust stability and error estimates require a different treatment of the convective term, as discussed in Remark 2.8 and provided by [10].

NIPG flux. The NIPG method was introduced by [111] as a penalized (and thereby stabilized) variant of the Baumann–Oden method [17], and is likewise recast in the unified flux formulation of [9]. Unlike SIPG, where the penalty parameter η_F must be chosen large enough to ensure coercivity, NIPG remains stable for any $\eta_F > 0$, independently of the polynomial degree, a property first established by [111] and one of the original motivations for the method.

The vector flux $\hat{\boldsymbol{\varsigma}}_h^D$ is kept identical to the SIPG choice (2.22), but the scalar flux $\hat{u}_h$ is changed. Instead of the plain average $\{u_h\}$, each element K now uses its own value corrected by the jump, with the boundary value doubled instead of being set to zero:

$$\hat{u}_h = \begin{cases} \{u_h\} + \boldsymbol{n}_K \cdot [\![u_h]\!] & \text{on } \mathcal{F}_h^I, \\ 2\,u_h & \text{on } \mathcal{F}_h^B, \end{cases} \qquad \hat{\boldsymbol{\varsigma}}_h^D = \{D\nabla_h u_h\} - \kappa\,[\![u_h]\!] \quad \text{on } \mathcal{F}_h. \tag{2.25}$$

Unlike the SIPG flux, $\hat{u}_h$ is no longer conservative on $\mathcal{F}_h^I$. On the interior face F shared by $K_L, K_R \in \mathcal{T}_h$, the two elements see two different values, so that this flux is consistent but not conservative in the sense of [9]. A direct computation gives $(\hat{u}_h - u_h)|_{K_L} = -(\hat{u}_h - u_h)|_{K_R} = \frac{1}{2}[\![u_h]\!] \cdot \boldsymbol{n}_{K_L}$, so that:

$$\{\hat{u}_h - u_h\} = 0, \qquad [\![\hat{u}_h - u_h]\!] = [\![u_h]\!], \qquad \text{on } \mathcal{F}_h^I.$$

Thus, the jump relation $[\![\hat{u}_h - u_h]\!] = [\![u_h]\!]$ holds uniformly over $\mathcal{F}_h$, with the opposite sign compared to the SIPG flux. Since $\{\hat{u}_h - u_h\} = 0$ and $\hat{\boldsymbol{\varsigma}}_h^D$ is conservative on $\mathcal{F}_h^I$ exactly as for SIPG, the terms over $\mathcal{F}_h^I$ in the general bilinear form (2.20) vanish just as they did for SIPG, and substituting into (2.20) yields:

$$\begin{aligned} a_h^{\mathrm{nipg}}(u_h, v_h) := &\int_\Omega R\,u_h\,v_h + \int_\Omega \boldsymbol{C} \cdot \nabla_h u_h\,v_h + \frac{1}{\mathrm{Pe}}\int_\Omega D\nabla_h u_h \cdot \nabla_h v_h \\ &+ \frac{1}{\mathrm{Pe}}\int_{\mathcal{F}_h} \Big([\![u_h]\!] \cdot \{D\nabla_h v_h\} - \{D\nabla_h u_h\} \cdot [\![v_h]\!] \Big) + \frac{1}{\mathrm{Pe}}\,\mathcal{S}_h^{\mathrm{disc}}(u_h, v_h), \end{aligned} \tag{2.26}$$

with the same interior penalty term $\mathcal{S}_h^{\mathrm{disc}}$ as in (2.24). The NIPG method is then defined by the variational problem (2.19) with a_h^{disc} replaced by a_h^{nipg}.

Comparing a_h^{nipg} (2.26) with a_h^{sipg} (2.23), only the sign of the $[\![u_h]\!] \cdot \{D\nabla_h v_h\}$ cross term differs; the interior penalty term $\mathcal{S}_h^{\mathrm{disc}}$ and the other cross term $\{D\nabla_h u_h\} \cdot [\![v_h]\!]$ are unchanged. This single sign flip reflects the non-symmetry of a_h^{nipg} in (u_h, v_h) even when $\boldsymbol{C} = \boldsymbol{0}$ (unlike a_h^{sipg}), which motivates the name of the method.

Expected convergence rates for both fluxes. For the Poisson equation (taking $R = 0$, $\boldsymbol{C} = \boldsymbol{0}$, $D = \mathrm{Id}$ and $\mathrm{Pe} = 1$ in (2.12)), i.e.

$$-\Delta u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \Gamma, \tag{2.27}$$

the SIPG scheme converges at the optimal order in both the broken H^1 semi-norm and the L^2 norm (Theorem 2.10).

Theorem 2.10 (see [9])**.** *Let Ω be a convex polytope and $1 \leqslant s \leqslant k$, and assume the penalty parameter η_F in (2.22) is chosen large enough, independently of h, for a_h^{sipg} to be coercive on V_h^{disc}. Considering $u \in H^{s+1}(\Omega)$ and $u_h \in V_h^{disc}$ the solutions*

to the Poisson problem (2.27) and to (2.19) with $a_h^{disc} = a_h^{sipg}$*, there exists* $C > 0$*, independent of* h *and* u*, such that:*

$$\|\nabla_h(u - u_h)\|_{L^2(\Omega)} \leqslant Ch^s|u|_{H^{s+1}(\Omega)},$$

and

$$\|u - u_h\|_{L^2(\Omega)} \leqslant Ch^{s+1}|u|_{H^{s+1}(\Omega)}. \tag{2.28}$$

The L^2 estimate relies on SIPG being adjoint consistent, so that the classical Aubin–Nitsche duality argument applies. The optimal order in the broken H^1 semi-norm also holds for NIPG, as established originally in [111] and recast in [9]. In the L^2 norm, however, the two fluxes differ: NIPG is not adjoint consistent, because its scalar flux $\hat{u}_h$ is not conservative, and the duality argument breaks down. Unlike for SIPG, no lower bound on the penalty parameter η_F is required here. Under this sole assumption, [9] does not establish a convergence order for the NIPG L^2 error, but only the a priori upper bound:

$$\|u - u_h\|_{L^2(\Omega)} \leqslant Ch^s|u|_{H^{s+1}(\Omega)},$$

one order below the SIPG estimate (2.28). A sharper upper bound $Ch^{s+1}|u|_{H^{s+1}(\Omega)}$, matching the SIPG order, can be recovered with a "superpenalty", a result originally established for NIPG in [111] and recast in the unified framework of [9]. Its cost is a significantly worse conditioned linear system [35]. Since we do not use such a superpenalty, only the bound above is guaranteed a priori for the NIPG L^2 error observed in the numerical experiments of Chapter 4. In practice, the optimal order is nonetheless recovered in some of these experiments, depending in particular on the parity of the polynomial degree, as discussed in Section 4.10.1.

For the full advection-diffusion-reaction operator (2.12), the convergence rate additionally depends on the Péclet number Pe and on the reaction coefficient R, and can degrade outside the diffusion-dominated regime since the reaction and convection terms are treated here only through volume integrals, without a dedicated numerical flux (see Remark 2.8). A treatment providing robust error estimates across the whole range of regimes, from diffusion- to reaction- to convection-dominated, is given by [10]; we revisit this question numerically in Section 4.10.3. The next section addresses the convective term by introducing a dedicated upwind numerical flux, which can be combined with either of the two diffusive fluxes presented above.

2.2.2.3. Numerical flux for the convective term

We now return to the convective term left aside in Section 2.2.2.1, where it is discretized directly as the volume contribution $\int_K \boldsymbol{C} \cdot \nabla u_h \, v_K$ on each element $K \in \mathcal{T}_h$, without any interface term. As explained in Remark 2.8, this simple treatment is adequate in the diffusion-dominated regime, but degrades the convergence order as the Péclet number grows, since it provides no mechanism to control the jumps of the discrete solution. Following the local discontinuous Galerkin approach of [40], we now introduce a dedicated numerical flux for the convective term, chosen upwind with respect to $\boldsymbol{C}$; unlike the diffusive fluxes of Section 2.2.2.2, it can be combined independently with either the SIPG or the NIPG choice.

Upwind flux. For $F \in \mathcal{F}_h^I$, write $K_L, K_R \in \mathcal{T}_h$ for the two elements sharing F; for $F \in \mathcal{F}_h^B$, write $K \in \mathcal{T}_h$ for the unique element such that $F \subset \partial K$. We introduce the scalar numerical flux $\hat{u}_h^C$, analogous to $\hat{u}_h$ in Section 2.2.2.1 but now defined by taking, on each face, the value of u_h from the element that the convective field $\boldsymbol{C}$ leaves (the upwind element), with the homogeneous Dirichlet value used on the inflow part of Γ:

$$\hat{u}_h^C := \begin{cases} u_h|_{K_L} & \text{if } \boldsymbol{C}\cdot\boldsymbol{n}_{K_L} \geqslant 0, \\ u_h|_{K_R} & \text{if } \boldsymbol{C}\cdot\boldsymbol{n}_{K_L} < 0, \end{cases} \quad \text{on } \mathcal{F}_h^I, \qquad \hat{u}_h^C = \begin{cases} u_h & \text{if } \boldsymbol{C}\cdot\boldsymbol{n}_K \geqslant 0, \\ 0 & \text{if } \boldsymbol{C}\cdot\boldsymbol{n}_K < 0, \end{cases} \quad \text{on } \mathcal{F}_h^B. \tag{2.29}$$

We modify the local equation for u_h (2.14) established in Section 2.2.2.1 by adding, to the convective volume term, a face contribution built from the upwind flux $\hat{u}_h^C$ defined in (2.29), for all $K \in \mathcal{T}_h$ and $v_K \in \mathbb{P}_k(K)$:

$$\int_K R\,u_h\,v_K + \int_K \boldsymbol{C}\cdot\nabla u_h\,v_K + \int_{\partial K} (\boldsymbol{C}\cdot\boldsymbol{n}_K)\,(\hat{u}_h^C - u_h)\,v_K \\ + \frac{1}{\mathrm{Pe}}\int_K D\boldsymbol{\varsigma}_h\cdot\nabla v_K = \int_K f v_K + \frac{1}{\mathrm{Pe}}\int_{\partial K} \hat{\boldsymbol{\varsigma}}_h^D\cdot\boldsymbol{n}_K\,v_K,$$

the local equation for $\boldsymbol{\varsigma}_h$ and the choice of diffusive flux $\hat{\boldsymbol{\varsigma}}_h^D$ being unchanged. Since $\hat{u}_h^C = u_h|_K$ whenever K is the upwind element, i.e. whenever $\boldsymbol{C}\cdot\boldsymbol{n}_K \geqslant 0$, the added term vanishes identically on the outflow part of ∂K; it is only active on the inflow part, where it replaces the otherwise unused value $u_h|_K$ by the value coming from the upwind side.

Summing over $K \in \mathcal{T}_h$ and treating separately, on each interior face, the two cases $\boldsymbol{C}\cdot\boldsymbol{n}_{K_L} \geqslant 0$ and $\boldsymbol{C}\cdot\boldsymbol{n}_{K_L} < 0$, a direct computation shows that the added terms reduce, after simplification, to:

$$c_h^{\mathrm{up}}(u_h, v_h) := \int_{\mathcal{F}_h^I} \Big(\tfrac{1}{2}\,|\boldsymbol{C}\cdot\boldsymbol{n}_{K_L}|\,[\![u_h]\!]\cdot[\![v_h]\!] - \boldsymbol{C}\cdot[\![u_h]\!]\,\{v_h\} \Big) \\ + \int_{\mathcal{F}_h^B} \max(-\boldsymbol{C}\cdot\boldsymbol{n}_K, 0)\,u_h\,v_h, \tag{2.30}$$

where $\max(-\boldsymbol{C}\cdot\boldsymbol{n}_K, 0)$ denotes the (non-negative) inflow part of $\boldsymbol{C}\cdot\boldsymbol{n}_K$ on Γ.

Consistency and stabilizing effect. The scheme combining SIPG (resp. NIPG) with the upwind convective flux is then defined by the variational problem (2.19) with a_h^{disc} replaced by $a_h^{\mathrm{sipg}} + c_h^{\mathrm{up}}$ (resp. $a_h^{\mathrm{nipg}} + c_h^{\mathrm{up}}$), the term c_h^{up} being the same in both cases. Since the exact solution u is continuous, $[\![u]\!] = 0$ on $\mathcal{F}_h^I$, and $u = 0$ on Γ, both terms of c_h^{up} vanish identically when u_h is replaced by u: the addition of the upwind flux preserves consistency. Compared with the volume-only treatment of Section 2.2.2.1, c_h^{up} provides, through its first term, a direct control on the jumps $[\![u_h]\!]$ at a rate governed by $|\boldsymbol{C}\cdot\boldsymbol{n}_{K_L}|$ rather than by the diffusive penalty κ alone. This is precisely the mechanism missing in the convection-dominated regime, as anticipated in Remark 2.8. The corresponding stability and error analysis, robust across the diffusion-, reaction- and convection-dominated regimes, is carried out in [10].

2.2.3. φ-FEM approach

The methods presented in the previous sections (FEM, DG) rely on a mesh $\mathcal{T}_h$ whose boundary coincides exactly with Γ (see Figure 2.1). For complex (or evolving) geometries, building such a boundary-fitted mesh can be costly, or even difficult. An alternative is to embed Ω into a simple (typically Cartesian) mesh of an enclosing domain $\mathcal{O}$, so that $\Omega \subset \mathcal{O}$, without requiring the mesh boundary to coincide with Γ. This is the principle of so-called unfitted methods (see Chapter 1), among which XFEM [70], CutFEM [31], Shifted Boundary Method (SBM) [91], and the φ-FEM method presented in this section, introduced by Michel Duprez and Alexei Lozinski in [53].

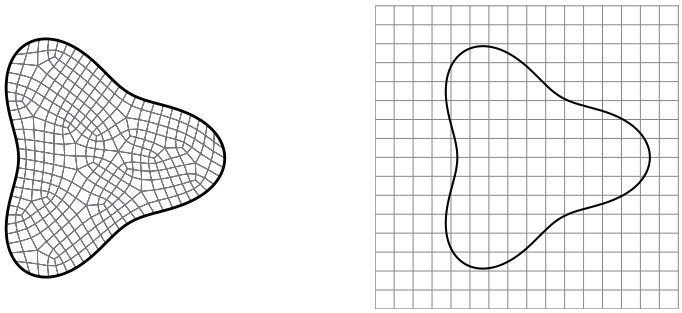


Figure 2.1.: Fitted and unfitted meshes for the same 3-lobed domain Ω. Left – Fitted mesh. Right – Unfitted mesh.

The idea of φ-FEM is to impose the Dirichlet condition not by constraining the finite element space, but by directly seeking the approximate solution as a product with a level-set function φ. This level-set function vanishes on the domain boundary Γ by construction, hence the name of the method. Unlike XFEM or CutFEM, this approach requires no non-standard integration, either over cut elements or on the immersed boundary Γ: all the integrals of the scheme are computed over whole elements of the background mesh, using the usual quadrature rules, which makes it particularly simple to implement. With a suitable stabilization (see below), it also retains a conditioning of the same order as that of a classical finite element method on a quasi-uniform mesh of equivalent size.

Since the founding paper [53], which addresses the Poisson problem with homogeneous Dirichlet boundary conditions (the setting on which this section focuses), the φ-FEM method has been extended to other operators and boundary conditions. These extensions cover Neumann and Robin conditions for a reaction-diffusion equation [51], the heat equation [52], linear elasticity and heat transfer [41], and the Stokes equations for particulate flows [50]. In these extensions, the Dirichlet condition is most often reimposed in a dual form, with an auxiliary variable only on a narrow band near the boundary, rather than through the direct product with the level-set φ presented below, this dual formulation combining more naturally with the (also dual) treatment of natural boundary conditions [41]. We do not detail this dual variant here and restrict ourselves to the homogeneous Dirichlet case and to the direct scheme, originally introduced in [53].

As in Section 2.2.1, we work at a fixed $\boldsymbol{\mu}$ and omit its dependence to lighten notation. We restrict ourselves here to the Poisson problem (formally, $R = 0$, $\boldsymbol{C} = \mathbf{0}$,

$D = \mathrm{Id}$, $\mathrm{Pe} = 1$) with homogeneous Dirichlet boundary conditions. The problem under consideration is therefore:

$$-\Delta u = f \text{ in } \Omega, \qquad u = 0 \text{ on } \Gamma. \tag{2.31}$$

We further assume that Ω is given implicitly by a level-set function φ, known, smooth, and behaving near Γ like the signed distance to Γ:

$$\Omega := \{\varphi < 0\}, \qquad \Gamma := \{\varphi = 0\}.$$

The central idea of φ-FEM is to seek the approximate solution as a product $u = \varphi w$, where w is a new unknown defined on a computational domain $\Omega_h \supset \Omega$ not fitted to Γ: the homogeneous Dirichlet condition is then automatically satisfied, whatever w, since φ vanishes on Γ by construction. Formally, assuming f and u extended to Ω_h so that $-\Delta u = f$ still holds, multiplying by a test function φv, $v \in H^1(\Omega_h)$, and integrating by parts over Ω_h gives:

$$\int_{\Omega_h} \nabla(\varphi w) \cdot \nabla(\varphi v) - \int_{\partial\Omega_h} \frac{\partial}{\partial \boldsymbol{n}}(\varphi w)\, \varphi v = \int_{\Omega_h} f \varphi v,$$

where $\boldsymbol{n}$ denotes the outward unit normal to Ω_h. Discretizing this formulation requires a few notations on the background mesh, which we introduce first. We then present the resulting φ-FEM scheme, which must be supplemented by a stabilization, and finally its a priori error estimates.

Mesh and facets. More precisely, we assume Ω is included in a simply-shaped enclosing domain $\mathcal{O}$ (typically a box), on which we introduce a background mesh $\mathcal{T}_h^{\mathcal{O}}$, quasi-uniform with characteristic size h. We approximate φ by its Lagrange interpolant $\varphi_h := \mathcal{I}_h^{\mathcal{O}}(\varphi)$, where $\mathcal{I}_h^{\mathcal{O}}$ denotes the same Lagrange interpolation operator as in (2.7), associated this time with the background mesh $\mathcal{T}_h^{\mathcal{O}}$ and with a degree at least k. The approximate boundary is then $\Gamma_h := \{\varphi_h = 0\}$. The computational mesh $\mathcal{T}_h$ is then defined as the subset of elements of $\mathcal{T}_h^{\mathcal{O}}$ intersecting $\{\varphi_h < 0\}$:

$$\mathcal{T}_h := \{K \in \mathcal{T}_h^{\mathcal{O}} : K \cap \{\varphi_h < 0\} \neq \emptyset\}, \tag{2.32}$$

and Ω_h denotes the domain it covers (generally slightly larger than Ω), with boundary $\partial\Omega_h$ (distinct from both Γ and $\Gamma_h = \{\varphi_h = 0\}$). As announced, $\mathcal{T}_h$ is not fitted to Γ: some of its elements are cut by Γ_h. We denote by $\mathcal{T}_h^\Gamma \subset \mathcal{T}_h$ the set of these cut elements:

$$\mathcal{T}_h^\Gamma := \{K \in \mathcal{T}_h : K \cap \Gamma_h \neq \emptyset\},$$

and we denote by Ω_h^Γ the narrow band they cover. Finally, we introduce $\mathcal{F}_h^\Gamma$, the set of interior facets of $\mathcal{T}_h$ belonging to a cut element:

$$\mathcal{F}_h^\Gamma := \{F \text{ internal face of } \mathcal{T}_h : \exists K \in \mathcal{T}_h^\Gamma,\ F \subset \partial K\}.$$

The various mesh and facet sets thus defined are illustrated in Figure 2.2 for a 3-lobed domain Ω embedded in a Cartesian background mesh $\mathcal{T}_h^{\mathcal{O}}$.

Having introduced these notations, we can now present the φ-FEM scheme for the Poisson problem with homogeneous Dirichlet boundary conditions (2.31).

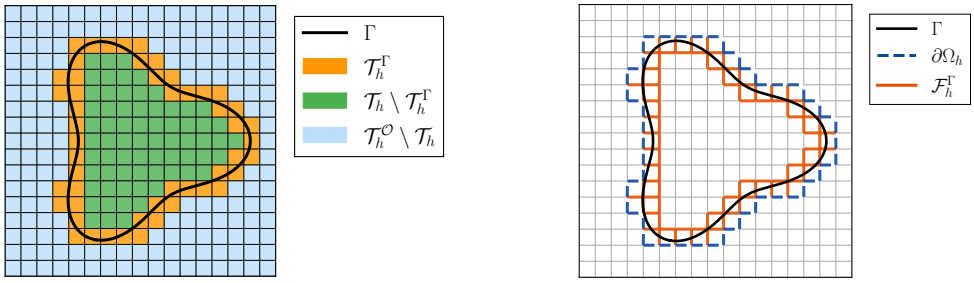


Figure 2.2.: Computational mesh $\mathcal{T}_h$ for a 3-lobed domain Ω (with boundary Γ), embedded in a Cartesian background mesh $\mathcal{T}_h^\mathcal{O}$. Left – Classification of the elements. Right – Classification of the facets.

φ-FEM scheme (direct). We consider the usual finite element space $V_h \subset H^1(\Omega_h)$ of degree $k \geqslant 1$, defined as in (2.4) but on $\mathcal{T}_h$ which covers Ω_h (unfitted to Γ), without any boundary condition imposed, that is:

$$V_h := \left\{ v_h \in C^0(\Omega_h),\ \forall K \in \mathcal{T}_h,\ v_h|_K \in \mathbb{P}_k \right\}.$$

Unlike V_h^0 in classical FEM, the Dirichlet condition is no longer carried here by the space, but by multiplication by φ_h (the interpolant of the level-set).

The direct φ-FEM scheme then consists of the following variational form:

$$\text{Find } w_h \in V_h \text{ such that, } \forall v_h \in V_h,\ a_h^{\text{phi}}(w_h, v_h) = l_h^{\text{phi}}(v_h), \tag{2.33}$$

where the forms a_h^{phi} and l_h^{phi} are given by:

$$a_h^{\text{phi}}(w,v) := \int_{\Omega_h} \nabla(\varphi_h w) \cdot \nabla(\varphi_h v) - \int_{\partial\Omega_h} \frac{\partial}{\partial \boldsymbol{n}}(\varphi_h w)\, \varphi_h v + \mathcal{S}_h(w,v),$$

and

$$l_h^{\text{phi}}(v) := \int_{\Omega_h} f \varphi_h v + \mathcal{S}_h^{\text{rhs}}(v),$$

with the stabilization terms $\mathcal{S}_h$ and $\mathcal{S}_h^{\text{rhs}}$ defined by:

$$\begin{aligned} \mathcal{S}_h(w,v) &:= \tau h \sum_{F \in \mathcal{F}_h^\Gamma} \int_F \left[\frac{\partial}{\partial \boldsymbol{n}}(\varphi_h w)\right] \left[\frac{\partial}{\partial \boldsymbol{n}}(\varphi_h v)\right] \\ &\quad + \tau h^2 \sum_{K \in \mathcal{T}_h^\Gamma} \int_K \Delta(\varphi_h w)\, \Delta(\varphi_h v), \\ \mathcal{S}_h^{\text{rhs}}(v) &:= -\tau h^2 \sum_{K \in \mathcal{T}_h^\Gamma} \int_K f\, \Delta(\varphi_h v), \end{aligned} \tag{2.34}$$

where $\tau > 0$ is a stabilization parameter independent of h, and $[\![\cdot]\!]$ denotes the (scalar) jump of a quantity across the facet $F \in \mathcal{F}_h^\Gamma$. The final approximate solution is reconstructed as $u_h := \varphi_h w_h$, living on Ω_h.

The first term of $\mathcal{S}_h$ corresponds to the "ghost penalty" stabilization introduced by [29] for $\mathbb{P}_1$ elements; the second term involves the Laplacian on cut elements, together with

$\mathcal{S}_h^{\text{rhs}}$, and is necessary to extend the coercivity of the scheme to $\mathbb{P}_k$ elements of arbitrary degree k. Without these stabilizations, the scheme loses not only its convergence, but also its good conditioning.

Remark 2.11. *Although φ-FEM is presented on a mesh of simplexes throughout this section, the meshes shown above in Figures 2.1 and 2.2 are deliberately drawn as (Cartesian) hypercubes. This anticipates the element type that will actually be used for φ-FEM in Chapter 4, where it is implemented and numerically validated within the `ScimBa` library.*

Expected convergence rates. The φ-FEM scheme enjoys a priori error estimates analogous to those of classical FEM (Theorem 2.6):

Theorem 2.12 (see [53])**.** *Assume $\Omega \subset \Omega_h$, suitable regularity on the level-set φ, and the stabilization parameter τ chosen large enough, independently of h, for a_h^{phi} to be coercive on V_h. Considering u the solution to the Poisson problem with homogeneous Dirichlet boundary conditions (2.31) and $u_h = \varphi_h w_h$ the φ-FEM solution to (2.33), there exists $C > 0$, independent of h, f and u, such that:*

$$|u - u_h|_{H^1(\Omega)} \leqslant Ch^k \|f\|_{H^k(\Omega_h)},$$

and

$$\|u - u_h\|_{L^2(\Omega)} \leqslant Ch^{k+1/2} \|f\|_{H^k(\Omega_h)}.$$

The φ-FEM scheme thus converges at the optimal order k in the H^1 semi-norm, and at the (slightly) suboptimal order $k + 1/2$ in the L^2 norm. The numerical results of [53] moreover suggest an optimal order $k + 1$ in the L^2 norm, better than the theoretical bound.

We thus recover, with φ-FEM, convergence guarantees close to those of classical FEM (optimal in the H^1 semi-norm, slightly suboptimal in the L^2 norm), while dispensing with the constraint of a mesh fitted to Γ. This geometric flexibility, however, comes at the cost, in the current state of the literature, of a more restricted theoretical framework (the Poisson problem with homogeneous Dirichlet boundary conditions, or the extensions recalled above) than that of the general operator $\mathcal{L}$ treated by continuous FEM and DG in the previous sections. This restriction should nonetheless be nuanced: as recalled above, φ-FEM has already been extended to a fairly broad range of operators and boundary conditions since its introduction, and remains an active area of ongoing development.

The methods presented in this section, i.e. FEM, DG, and φ-FEM, all seek the solution in a finite-dimensional space of piecewise polynomials attached to a mesh $\mathcal{T}_h$ of Ω, for which we have recalled the classical convergence theory. This paradigm is well suited to a single, fixed parameter $\boldsymbol{\mu}$, but becomes costly in the parametric setting of Section 2.1: a new discrete problem, and possibly a new mesh, must be assembled and solved for every $\boldsymbol{\mu} \in \mathcal{M}$. The next section replaces the polynomial approximation space with a neural network, whose weights are trained once to approximate the parametric solution map $\boldsymbol{\mu} \mapsto u(\cdot, \boldsymbol{\mu})$, but for which the a priori error estimates established above no longer apply.

2.3. Physics-Informed Neural Networks (PINNs)

Unlike the methods presented in the previous sections, where the approximate solution is sought in a polynomial space, the method presented here is based on a neural network. The approximation space is therefore no longer a finite-dimensional vector space, but a manifold parameterized by the network's weights. Finding an approximate solution amounts to solving an optimization problem over these weights, by minimizing a cost function based on the strong residual of the PDE. This approach, known as physics-informed neural networks, was popularized by [108] for solving a PDE with neural networks, building on an earlier idea introduced by [84]. This section is therefore dedicated to the presentation of PINNs, and more specifically to their parametric version, in Section 2.3.1. We also present several improvements to the classical PINN approach, in Section 2.3.2, which are used in the numerical experiments of Chapter 3. The basic formulation is used throughout this manuscript, in Chapter 3 and in Chapter 5, whereas the improvements are used only in Chapter 3.

2.3.1. Physics-Informed Neural Networks for parametric PDEs

The main idea is to recast a PDE as an optimization problem. We illustrate the method on our problem (2.1). Since this problem is parametric, we consider parametric PINNs. Unlike classical PINNs, trained for specific physical parameters, they seek to learn a generalized solution covering a range of parameters. They incorporate these parameters as additional inputs to the network, allowing greater flexibility in solving problems where physical conditions or properties vary. Besides, since they are based on neural networks, PINNs are well suited to high-dimensional problems. This advantage is compounded when one uses Monte Carlo integration to estimate the loss functions, as its convergence rate does not depend on the dimension, unlike that of grid-based quadrature rules.

Considering p parameters $\boldsymbol{\mu} = (\mu_1, \dots, \mu_p) \in \mathcal{M} \subset \mathbb{R}^p$, with some parameter space $\mathcal{M}$, the parametric PDE (2.1) reads, with non-homogeneous Dirichlet boundary conditions:

$$\begin{cases} \mathcal{L}\big(u(\boldsymbol{x}, \boldsymbol{\mu}); \boldsymbol{x}, \boldsymbol{\mu}\big) = f(\boldsymbol{x}, \boldsymbol{\mu}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}, \boldsymbol{\mu}) = g(\boldsymbol{x}, \boldsymbol{\mu}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{2.35}$$

with g the trace on Γ of an H^2 function, which we still denote g when it is evaluated in Ω. Note that the solution of the equation depends on the parameters $\boldsymbol{\mu}$, as do the operator $\mathcal{L}$ and the boundary conditions. We then denote $u_\theta(\cdot, \boldsymbol{\mu})$ the approximate PINN prediction for given parameters $\boldsymbol{\mu}$, where θ are the learnable weights of the network.

The first idea of PINNs comes from the observation that, by construction, neural networks with smooth activation functions are nothing but smooth functions of their weights and inputs. Therefore, neural networks form natural candidates for approximating solutions to PDEs, especially with the advent of automatic differentiation tools. In our case, a PINN is a neural network that takes $d + p$ inputs, where d is the dimension of the space variable $\boldsymbol{x} \in \Omega$ and p is the number of parameters $\boldsymbol{\mu} \in \mathcal{M}$. Classically, this neural network is a coordinate-based neural network, such as a multi-layer perceptron (MLP), which depends on several user-defined hyperparameters.

Notation 2.13. *A multi-layer perceptron is a neural network obtained by composing a sequence of affine maps, applying a pointwise non-linear activation function σ between consecutive layers (the last layer being purely affine, with no activation applied to the output). The hyperparameters used throughout this manuscript to specify an MLP are its activation function σ and* `layers`*, the sequence of integers giving the number of neurons of each hidden layer.*

Once this network is defined, solving the PDE can be rewritten as finding the optimal weights $\theta^\star$ defined by:

$$\theta^\star := \underset{\theta}{\operatorname{argmin}} \big(\omega_r \mathcal{J}_r(\theta) + \omega_b \mathcal{J}_b(\theta) + \omega_{\text{data}} \mathcal{J}_{\text{data}}(\theta)\big), \tag{2.36}$$

with ω_r, ω_b and ω_{data} some weights to balance the different terms of the loss function. In (2.36), the loss function has three terms. The first one is the residual loss function, defined by:

$$\mathcal{J}_r(\theta) := \int_{\mathcal{M}} \int_{\Omega} \big|\mathcal{L}\big(u_\theta(\boldsymbol{x}, \boldsymbol{\mu}); \boldsymbol{x}, \boldsymbol{\mu}\big) - f(\boldsymbol{x}, \boldsymbol{\mu})\big|^2 \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{\mu}. \tag{2.37}$$

The second one is the boundary loss function, defined by:

$$\mathcal{J}_b(\theta) := \int_{\mathcal{M}} \int_{\Gamma} \big|u_\theta(\boldsymbol{x}, \boldsymbol{\mu}) - g(\boldsymbol{x}, \boldsymbol{\mu})\big|^2 \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{\mu}. \tag{2.38}$$

The third one is the data loss function, defined by:

$$\mathcal{J}_{\text{data}}(\theta) := \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \big|u_\theta\big(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}_{\text{data}}^{(i)}\big) - u_{\text{data}}^{(i)}\big|^2, \tag{2.39}$$

where $\big(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}_{\text{data}}^{(i)}, u_{\text{data}}^{(i)}\big)_{i=1,\dots,N_{\text{data}}}$ are N_{data} known data points, with $u_{\text{data}}^{(i)}$ a reference solution at points $\boldsymbol{x}_{\text{data}}^{(i)}$ and for given parameters $\boldsymbol{\mu}_{\text{data}}^{(i)}$. These reference solutions can be the exact solutions of the parametric PDE, defined in this case by $u_{\text{data}}^{(i)} = u\big(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}_{\text{data}}^{(i)}\big)$. They can also be an approximation produced by a numerical method, such as finite elements on a fine mesh.

Solving the minimization problem (2.36) requires computing the gradient of the loss function with respect to θ, which involves calculating the integrals in (2.37) and (2.38). The most natural idea is to estimate them with a Monte Carlo method, see e.g. [32]. One could also use Gauss-type quadrature rules to evaluate integrals, as is done in VPINNs [82], but the limitation is the impossibility of selecting an adequate quadrature order due to the unknown properties of the neural network approximation. To apply the Monte Carlo method, we define so-called "collocation points" on $\Omega \times \mathcal{M}$ and its boundary $\Gamma \times \mathcal{M}$, denoted respectively by $\big(\boldsymbol{x}_{\text{col}}^{(i)}, \boldsymbol{\mu}_{\text{col}}^{(i)}\big)_{i=1,\dots,N_{\text{col}}}$ and $\big(\boldsymbol{x}_{\text{bc}}^{(i)}, \boldsymbol{\mu}_{\text{bc}}^{(i)}\big)_{i=1,\dots,N_{\text{bc}}}$. Then, we approximate the residuals and boundary losses (up to multiplicative constants absorbed in the weights ω_r and ω_b) by:

$$\mathcal{J}_r(\theta) \simeq \frac{1}{N_{\text{col}}} \sum_{i=1}^{N_{\text{col}}} \big|\mathcal{L}\big(u_\theta(\boldsymbol{x}_{\text{col}}^{(i)}, \boldsymbol{\mu}_{\text{col}}^{(i)}); \boldsymbol{x}_{\text{col}}^{(i)}, \boldsymbol{\mu}_{\text{col}}^{(i)}\big) - f(\boldsymbol{x}_{\text{col}}^{(i)}, \boldsymbol{\mu}_{\text{col}}^{(i)})\big|^2, \tag{2.40}$$

and

$$\mathcal{J}_b(\theta) \simeq \frac{1}{N_{\text{bc}}} \sum_{i=1}^{N_{\text{bc}}} \left| u_\theta\big(\boldsymbol{x}_{\text{bc}}^{(i)}, \boldsymbol{\mu}_{\text{bc}}^{(i)}\big) - g\big(\boldsymbol{x}_{\text{bc}}^{(i)}, \boldsymbol{\mu}_{\text{bc}}^{(i)}\big) \right|^2, \tag{2.41}$$

where N_{col} and N_{bc} are heuristically determined and should be large enough to ensure that the Monte Carlo integration is accurate enough. The precise values of these parameters will be given in the numerical experiments. In the case of complex geometries, one solution for obtaining a sample of these collocation points in the domain Ω is to use a level-set function, denoted φ (see Section 2.2.3). This function, which vanishes on the boundary of Ω, can be obtained in several ways. The authors of [122] propose different approaches to obtain a level-set function analytically in the case of polygonal or curved geometries. Learning-based approaches have also been proposed, notably in e.g. [102, 117].

Because of the minimization problem (2.36), the PINN u_θ does not exactly satisfy the boundary conditions. Moreover, the loss terms compete, which may require fine-tuning the coefficients between $\mathcal{J}_r$ and $\mathcal{J}_b$. In addition, classical PINNs do not include information on higher-order derivatives, and MLPs are subject to spectral bias. These limitations matter in particular for the enriched finite element methods of Chapter 3, where, as highlighted in Remarks 3.2 and 3.5, the neural network approximation u_θ serves as a prior whose derivatives must be well approximated. For these reasons, the next subsection recalls several improvements of classical PINNs in the literature.

2.3.2. Improving PINN training and prediction

Here, we focus on several ways of improving PINNs: strongly imposing the boundary conditions, adding a term involving the gradient of the residual to the loss function, and countering the spectral bias. Although these approaches are presented separately, they can easily be combined with one another.

Strong imposition of boundary conditions. To avoid the issues of classical PINNs discussed above, the authors of [63, 84] propose a method to enforce non-homogeneous Dirichlet boundary conditions exactly. To that end, they search the approximation u_θ of the solution to (2.35) of the form, for all $\boldsymbol{x} \in \Omega$ and $\boldsymbol{\mu} \in \mathcal{M}$:

$$u_\theta(\boldsymbol{x}, \boldsymbol{\mu}) = \varphi(\boldsymbol{x}) w_\theta(\boldsymbol{x}, \boldsymbol{\mu}) + g(\boldsymbol{x}, \boldsymbol{\mu}), \tag{2.42}$$

where φ and w_θ are, respectively, the level-set function and a neural network as defined above. Thus u_θ automatically satisfies the boundary conditions, since $u_\theta(\boldsymbol{x}, \boldsymbol{\mu}) = g(\boldsymbol{x}, \boldsymbol{\mu})$ for all $\boldsymbol{x} \in \Gamma$ and $\boldsymbol{\mu} \in \mathcal{M}$. Note that this level-set function can be used in a few different ways, firstly to generate a sample of points in Ω, as mentioned above, and secondly to impose boundary conditions. However, to use it directly in the formulation of the prior, it must be sufficiently regular. For example, the signed distance function is not a good candidate. Similar methods to impose boundary conditions exist for Robin and Neumann conditions (see [122]).

In this case, only the residual and data loss functions are minimized, and the minimization problem (2.36) becomes:

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \big(\omega_r \mathcal{J}_r(\theta) + \omega_{\text{data}} \mathcal{J}_{\text{data}}(\theta) \big).$$

Sobolev training for PINNs. As presented above, PINNs approximate the PDE solution by directly incorporating the equations into their training. Despite their effectiveness, these models can sometimes struggle to learn correctly, especially when the solution or its derivatives are complicated. The authors of [119] have proposed an approach called Sobolev training to try and overcome these difficulties, in the case of solutions with high regularity. Note that, in this case, the source term f and the coefficients of $\mathcal{L}$ need to be differentiable to ensure sufficient regularity and to compute the additional term. This method imposes constraints not only on the residual itself, but also on its spatial gradient, thereby indirectly constraining the derivatives of the approximate solution u_θ. In the context of solving the problem (2.35) under consideration, Sobolev training is applied by adding a cost term $\mathcal{J}_{\text{sob}}$ to the initial minimization problem (2.36):

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \big(\omega_r \mathcal{J}_r(\theta) + \omega_{\text{sob}} \mathcal{J}_{\text{sob}}(\theta) + \omega_b \mathcal{J}_b(\theta) + \omega_{\text{data}} \mathcal{J}_{\text{data}}(\theta)\big), \tag{2.43}$$

with $\mathcal{J}_r$, $\mathcal{J}_b$ and $\mathcal{J}_{\text{data}}$ defined as in (2.37) to (2.39) respectively, and ω_{sob} the new weight balancing the Sobolev term with the other terms of the loss function. The Sobolev loss function $\mathcal{J}_{\text{sob}}$ in (2.43) is defined by:

$$\mathcal{J}_{\text{sob}}(\theta) := \int_{\mathcal{M}} \int_{\Omega} \big|\nabla_{\boldsymbol{x}} \big(\mathcal{L}\big(u_\theta(\boldsymbol{x}, \boldsymbol{\mu}); \boldsymbol{x}, \boldsymbol{\mu}\big) - f(\boldsymbol{x}, \boldsymbol{\mu})\big)\big|^2 \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{\mu}, \tag{2.44}$$

where the integral is estimated by the Monte Carlo method, similarly to the other loss functions.

Overcoming the spectral bias. Multiple ways of overcoming the spectral bias of MLPs, i.e. their tendency to learn the low-frequency components of a function before its high-frequency ones, are available. For instance, in [123], the authors introduce Fourier features (FF) to improve the network, while the authors of [48] rely on a domain decomposition-based approach.

For high-frequency solutions (i.e. solutions with more than three wavelengths propagating), we use a variant of the FF of [123], which relies on modifying the input of the neural network. Indeed, the prior u_θ is then defined, for all $\boldsymbol{x} \in \Omega$ and $\boldsymbol{\mu} \in \mathcal{M}$, as:

$$u_\theta(\boldsymbol{x}, \boldsymbol{\mu}) = w_\theta\big(\boldsymbol{x}, \boldsymbol{\mu}; \sin(\pi \boldsymbol{a}_1 \cdot \boldsymbol{x}), \cos(\pi \boldsymbol{a}_1 \cdot \boldsymbol{x}), \dots, \sin(\pi \boldsymbol{a}_{N_{\text{FF}}} \cdot \boldsymbol{x}), \cos(\pi \boldsymbol{a}_{N_{\text{FF}}} \cdot \boldsymbol{x})\big),$$

where $\boldsymbol{a}_i \in \mathbb{R}^d$, for $i = 1, \dots, N_{\text{FF}}$, are additional trainable parameters, the same frequency vector $\boldsymbol{a}_i$ being shared between the sine and the cosine of a given feature i, so that their combination in the following layers can synthesize an arbitrary phase shift at that frequency. This makes it possible to learn higher-frequency solutions, by also learning the frequency itself, unlike in [123] where the frequencies are instead drawn at random and kept fixed throughout training. This MLP with FF needs the same parameters as the classical MLP (defined in Notation 2.13), but also the number N_{FF} of Fourier features.

Having introduced both the classical, mesh-based approximation methods and physics-informed neural networks, we now set φ-FEM and DG aside, and turn in Chapter 3 to our first contribution [16]: enriching continuous Lagrange finite element approximation spaces (Section 2.2.1) using neural networks, and in particular physics-informed neural networks (Section 2.3).

3 Enriching continuous FEM

Contents

3.1. Enriching FEM with additive priors 28
3.1.1. Construction of the modified problem 29
3.1.2. Convergence analysis 29
3.2. Enriching FEM with multiplicative priors 31
3.2.1. Construction of the modified problem 31
3.2.2. Convergence analysis 32
3.3. Comparison of the two enriched methods 35
3.4. Implementation details 39
3.4.1. Using PINN prediction effectively 39
3.4.2. Imposing boundary conditions 40
3.5. Numerical results within the theoretical framework 41
3.5.1. Setup of the numerical experiments 43
3.5.2. 1D Poisson problem 45
3.5.3. 1D general elliptic system and convection-dominated regime . . 53
3.5.4. 1D non-smooth transmission problem 58
3.5.5. 2D Poisson problem in a square domain 63
3.5.6. 2D anisotropic elliptic problem on a square 76
3.5.7. 2D Poisson problem on an annulus, with mixed boundary conditions 80
3.5.8. 3D Poisson problem in a cube domain 83
3.6. Numerical results beyond the theoretical framework: the Navier–Stokes equations 88
3.6.1. Problem statement 88
3.6.2. Construction of the parametric PINN prior 89
3.6.3. Finite element discretization 92
3.6.4. Extending the additive approach to the Navier–Stokes system . 94
3.6.5. Error estimates 95

The notations associated with the enriched FEM approaches, introduced in this chapter, are recalled in Appendix A.1.2. The notations relating to the classical finite element method (Table A.3) and the PINN architecture (Table A.6) are summarized in Appendix A.1.1.

In this chapter, we focus on continuous Lagrange approximation spaces (Section 2.2.1). We introduce our contribution [16] through two methods for enriching these spaces using neural networks, and in particular using PINNs (Section 2.3).

Thus, we assume that a prior knowledge of the solution to (2.1) is available. In what follows, we call this information a "prior". In the first few sections, we present the enriched methods considering that the parameter $\boldsymbol{\mu}$ is fixed, and that the prior is a general function. This prior is denoted by $\boldsymbol{x} \mapsto u_\theta(\boldsymbol{x})$ with parameters θ, and we assume that it can be constructed with the desired regularity $u_\theta \in H^{s+1}(\Omega) \cap H_0^1(\Omega)$ for $1 \leqslant s \leqslant k$, where k is the polynomial degree of the enriched FEM. However, from Section 3.4 onwards, the prior will be the prediction of a parametric PINN.

Figure 3.1 presents the pipeline of the enriched method, which consists of two phases: an offline phase, where the parametric PINN is trained to learn the prior, and an online phase, where the PINN is evaluated (for a fixed value of $\boldsymbol{\mu}$) to provide the prior for the enriched FEM resolution. The enriched FEM is then solved using this prior.

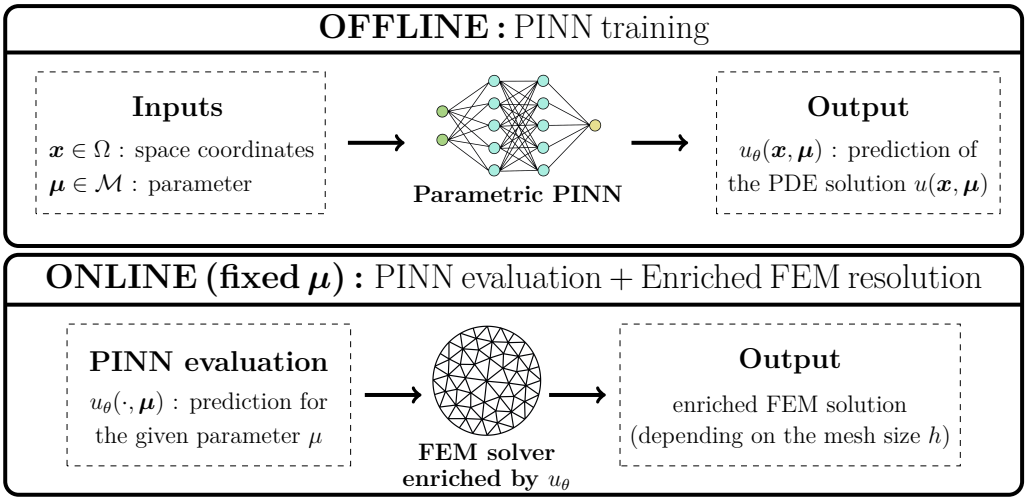


Figure 3.1.: Pipeline of the enriched method. Top – offline phase (PINN training). Bottom – online phase (PINN evaluation + enriched FEM resolution).

This chapter is organized as follows. We first present the enrichment with additive priors (Section 3.1) and with multiplicative priors (Section 3.2), each with its convergence estimate, and we then compare the two (Section 3.3). Section 3.4 details how the PINN prediction is plugged into the FEM solver and how the boundary conditions are imposed. Finally, we validate the methods numerically on test cases within the theoretical framework (Section 3.5) and beyond it, with the Navier–Stokes equations (Section 3.6).

3.1. Enriching FEM with additive priors

In Section 3.1.1, we first show how to use this prior to enrich classical finite element spaces. Then, in Section 3.1.2, we prove a convergence estimate for the resulting method.

3.1.1. Construction of the modified problem

In the general setting of FEM, we follow the Bubnov–Galerkin method [55], where the basis functions and the numerical solutions are in the same space (see (2.6), where both u_h and v_h are in V_h^0). As we intend to enrich the classical approximation space, we exploit the idea formalized as the Petrov–Galerkin method (e.g. [27, 45, 110]), where the test and trial functions belong to different spaces. This approach is often used for convection-dominated problems, see [5]. We propose to enrich the trial space using the prior u_θ by defining:

$$V_h^+ := \left\{ u_h^+ = u_\theta + p_h^+, \quad p_h^+ \in V_h^0 \right\}, \tag{3.1}$$

and we use the space V_h^0 for the test functions. Since we have assumed that $u_\theta \in H^{s+1}(\Omega) \cap H_0^1(\Omega)$, V_h^+ is also a subset of V^0, like V_h^0. Plugging this new trial space into the approximate problem (2.6), we obtain the formulation:

$$\text{Find } u_h^+ \in V_h^+ \text{ such that, } \forall v_h \in V_h^0,\ a(u_h^+, v_h) = l(v_h), \tag{3.2}$$

which leads to the following approximation problem:

$$\text{Find } p_h^+ \in V_h^0 \text{ such that, } \forall v_h \in V_h^0,\ a(p_h^+, v_h) = l(v_h) - a(u_\theta, v_h). \tag{3.3}$$

Therefore, we obtain a classical Galerkin approximation with a modified source term.

3.1.2. Convergence analysis

The objective is to prove that the FEM solution to problem (3.3) converges, with an error depending on the quality of the prior. Due to this dependence on the prior (neural network in the numerical section), we are not conducting an in-depth study of this gain. We are simply aiming to show that adding the prior does not degrade the convergence orders and that it is the high-order derivatives that control the gain in comparison with classical finite element method, which helps to guide choices regarding the prior.

Theorem 3.1. *Let $u \in H^{s+1}(\Omega)$ be the solution to problem (2.3) and $u_\theta \in H^{s+1}(\Omega) \cap H_0^1(\Omega)$ be a prior on u. We consider $u_h^+ \in V_h^+$ as the solution to the discrete problem (3.2) with V_h^+ the modified trial space defined in (3.1). The following estimates hold. For all $1 \leqslant s \leqslant k$:*

$$|u - u_h^+|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} C_{gain}^+ h^s |u|_{H^{s+1}}, \tag{3.4}$$

and, supposing that dual problem (2.8) admits strong solutions (i.e. for all $\xi \in L^2(\Omega)$ there exists an associated solution $w \in H_0^1(\Omega) \cap H^2(\Omega)$ to (2.8)):

$$\|u - u_h^+\|_{L^2} \leqslant C_e C_1 C_s \frac{\gamma^2}{\alpha} C_{gain}^+ h^{s+1} |u|_{H^{s+1}}, \tag{3.5}$$

with C_e, C_1, C_s, γ, α defined in Section 2.2.1 and:

$$C_{gain}^+ := \frac{|u - u_\theta|_{H^{s+1}}}{|u|_{H^{s+1}}}. \tag{3.6}$$

Remark 3.2. *The constant C^+_{gain} represents the potential gain compared to the error of the classical FEM presented in Theorem 2.6. Note that this constant is the same in L^2 norm and H^1 semi-norm.*

Proof of Theorem 3.1. **H^1-error:** To prove (3.4), we adapt the proof of Céa's lemma to the additive prior case. Considering the trial space defined in (3.1), the numerical solution u_h^+ is given by:

$$u_h^+ = u_\theta + p_h^+,$$

with $p_h^+ \in V_h^0 \subset V$ solution to (3.3). We have:

$$a(u-u_h^+, u-u_h^+) = a\big(u-u_h^+, (u-u_\theta)-v_h\big) + a\big(u-u_h^+, v_h-p_h^+\big), \quad \forall v_h \in V_h^0.$$

Let us first treat the second term on the right-hand side. By Galerkin orthogonality (difference of the continuous problem (2.3) and discrete problem (3.2)), we obtain:

$$a\big(u-u_h^+, v_h-p_h^+\big) = 0, \quad \forall v_h \in V_h^0.$$

Denoting by α and γ the coercivity and continuity constants of the bilinear form a, we have:

$$\begin{aligned} \alpha\big|u-u_h^+\big|_{H^1}^2 &\leqslant a\big(u-u_h^+, u-u_h^+\big) = a\big(u-u_h^+, (u-u_\theta)-v_h\big), && \forall v_h \in V_h^0, \\ &\leqslant \gamma\big|u-u_h^+\big|_{H^1}\big|(u-u_\theta)-v_h\big|_{H^1}, && \forall v_h \in V_h^0, \end{aligned}$$

which immediately leads to:

$$|u-u_h^+|_{H^1} \leqslant \frac{\gamma}{\alpha}\big|(u-u_\theta)-v_h\big|_{H^1}, \quad \forall v_h \in V_h^0.$$

We apply it to $v_h = \mathcal{I}_h(u-u_\theta) \in V_h^0$ with $\mathcal{I}_h$ the Lagrange interpolation operator (2.7) in V_h, it holds using interpolation estimate given in Theorem 2.4:

$$|u-u_h^+|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} h^s |u-u_\theta|_{H^{s+1}},$$

with C_s defined in Section 2.2.1.

The above expression can be rewritten as:

$$|u-u_h^+|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} C^+_{\text{gain}}\, h^s |u|_{H^{s+1}}\,, \tag{3.7}$$

with

$$C^+_{\text{gain}} = \frac{|u-u_\theta|_{H^{s+1}}}{|u|_{H^{s+1}}},$$

which completes the first part of the proof.

L^2-error: We follow the Aubin–Nitsche technique. Consider $w \in H_0^1(\Omega) \cap H^2(\Omega)$ solution to:

$$\mathcal{L}^* w = u - u_h^+,$$

with homogeneous Dirichlet boundary condition. Thanks to Theorem 2.5, one has:

$$\|w\|_{H^2} \leqslant C_e \|u-u_h^+\|_{L^2}. \tag{3.8}$$

Using the Galerkin orthogonality and the continuity of the bilinear form a:

$$\|u-u_h^+\|_{L^2}^2 = a(u-u_h^+, w-\mathcal{I}_h w) \leqslant \gamma |u-u_h^+|_{H^1} |w-\mathcal{I}_h w|_{H^1}.$$

Thanks to Theorem 2.4 and (3.8):

$$|w-\mathcal{I}_h w|_{H^1} \leqslant C_e C_1 h \|u-u_h^+\|_{L^2},$$

which leads to the conclusion by using (3.7). □

Remark 3.3. *The gain constant C^+_{gain} defined in (3.6) shows that the closer the prior is to the solution, the smaller is the error constant associated with the FEM while keeping the same order of accuracy. Therefore, as soon as $C^+_{gain} < 1$, the FEM with additive prior is more accurate than the classical one. While this gives us a particularly flexible constraint, our objective is to balance this gain by relaxing the contribution h^s, using a coarser grid and low-order polynomial, to reduce the computational cost of the FEM while maintaining accuracy. Nonetheless, the gain is related to the L^2 error associated with the derivatives of $(s+1)^{th}$ order (with $1 \leqslant s \leqslant k$) of the prior. This shows that the prior must accurately approximate the derivatives of the solution in addition to the solution itself. This highlights that we need to build our prior by ensuring a good approximation of the derivatives of the solution. It also shows that the higher the order of the finite elements k is, the better our prior should approximate the higher-order derivatives. Therefore, it is more appropriate to use only low-order FEM so that k remains small.*

Having shown how to enrich the trial space additively and derived the corresponding convergence estimate, controlled by how well the prior approximates the high-order derivatives of the solution, we now turn to an alternative, multiplicative way of injecting the same prior u_θ into the finite element approximation.

3.2. Enriching FEM with multiplicative priors

This section employs the same assumptions as in Section 3.1, namely that we have a sufficiently smooth prior u_θ on the solution u of the PDE (2.1). However, this prior is now multiplied by elements of V_h rather than added to them. We construct the underlying modified problem in Section 3.2.1. Then, similarly to the additive approach of Section 3.1, error estimates are obtained in Section 3.2.2. The objectives of this section on error estimates are the same as in the previous section.

3.2.1. Construction of the modified problem

To construct the modified problem in this case, we must ensure that the prior u_θ never vanishes. To that end, we lift the initial problem (2.1) by a constant $M \in \mathbb{R}_+$, chosen large enough to ensure that $u_M := u + M > 0$, to get:

$$\begin{cases} \mathcal{L}(u_M) = f, & \text{in } \Omega, \\ u_M = M, & \text{on } \Gamma. \end{cases} \tag{3.9}$$

We then introduce the associated variational problem, defined by:

$$\text{Find } u_M = u + M, \text{ with } u \in V^0 \text{ such that, } \forall v \in V^0,\ a(u_M, v) = l(v). \tag{3.10}$$

Therefore, solving (3.10), we recover the solution u of the initial problem (2.1) by setting:

$$u = u_M - M.$$

With problem (3.10), we associate the prior:

$$u_{\theta,M} := u_\theta + M > 0.$$

Let us now introduce a new modified finite element space, defined by:

$$V_h^\times = \left\{ u_{h,M}^\times = u_{\theta,M}\, p_h^\times, \quad p_h^\times \in 1 + V_h^0 \right\}, \tag{3.11}$$

with, for all $\boldsymbol{x} \in \Omega$, $u_{\theta,M}(\boldsymbol{x}) \neq 0$. From (3.10), this leads to the following approximate formulation:

$$\text{Find } p_h^\times \in 1 + V_h^0 \text{ such that, } \forall v_h \in V_h^0,\ a\big(u_{\theta,M}\, p_h^\times, u_{\theta,M} v_h\big) = l(u_{\theta,M} v_h). \tag{3.12}$$

Therefore, solving (3.12), we recover the solution $u_h^\times \in V_h^\times - M$ of the original problem (2.1) by setting:

$$u_h^\times := u_{h,M}^\times - M.$$

Based on the N_{dofs} dofs $\big(\boldsymbol{x}^{(i)}\big)_{i\in\{1,\dots,N_{\text{dofs}}\}}$ of the mesh, we consider the interpolation operator on $V_h^\times$ given by:

$$\mathcal{I}_h^\times : C^0(\Omega) \ni v \mapsto \sum_{i=1}^{N_{\text{dofs}}} \frac{v\big(\boldsymbol{x}^{(i)}\big)}{u_{\theta,M}\big(\boldsymbol{x}^{(i)}\big)} \phi_i^\times \in V_h^\times,$$

where the shape functions $\phi_i^\times$ associated to $V_h^\times$ are defined by:

$$\phi_i^\times := u_{\theta,M}\, \phi_i,$$

with ϕ_i the classical shape functions presented in Section 2.2.1. Note that the new interpolation operator $\mathcal{I}_h^\times$ is related to the classical Lagrange interpolation operator defined in (2.7) as follows:

$$\forall v \in C^0(\Omega), \qquad \mathcal{I}_h^\times(v) = u_{\theta,M}\, \mathcal{I}_h\left(\frac{v}{u_{\theta,M}}\right). \tag{3.13}$$

3.2.2. Convergence analysis

In this section, we finally prove that the modified FEM (3.12) converges to the solution to (3.9), and that it satisfies the same type of estimate as the classical one. Equipped with the lifting trick from Section 3.2.1, we can state the following convergence theorem.

Theorem 3.4. *Let $u_M \in H^{s+1}(\Omega)$ be the solution of the enhanced problem (3.10) and $u_{\theta,M} \in M + H^{s+1}(\Omega) \cap H_0^1(\Omega)$ be a prior on u_M. We consider $u_{h,M}^\times \in V_h^\times$ the solution to the finite element problem (3.12) with $V_h^\times$ the modified trial space defined in (3.11), considering $\mathbb{P}_k$ polynomials. We define $u = u_M - M$ and $u_h^\times = u_{h,M}^\times - M$. Then, for all $1 \leqslant s \leqslant k$:*

$$|u - u_h^\times|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} C_{gain,H^1}^{\times,M} h^s |u|_{H^{s+1}}, \tag{3.14}$$

and, supposing that dual problem (2.8) admits strong solutions (i.e. for all $\xi \in L^2(\Omega)$,

there exists an associated solution $w \in H_0^1(\Omega) \cap H^2(\Omega)$ to (2.8)):

$$\|u - u_h^\times\|_{L^2} \leqslant C_e C_1 C_s \frac{\gamma^2}{\alpha} C_{gain,L^2}^{\times,M} h^{s+1} |u|_{H^{s+1}}, \tag{3.15}$$

with C_e, C_1, C_s, γ, α defined in Section 2.2.1, and where:

$$C_{gain,H^1}^{\times,M} := \left| \frac{u_M}{u_{\theta,M}} \right|_{H^{s+1}} \frac{\|u_{\theta,M}\|_{W^{1,\infty}}}{|u|_{H^{s+1}}}, \tag{3.16}$$

and

$$C_{gain,L^2}^{\times,M} := C_{\theta,M} \left| \frac{u_M}{u_{\theta,M}} \right|_{H^{s+1}} \frac{\|u_{\theta,M}\|_{W^{1,\infty}}^2}{|u|_{H^{s+1}}}, \tag{3.17}$$

with

$$C_{\theta,M} := \|u_{\theta,M}^{-1}\|_{L^\infty} + 2|u_{\theta,M}^{-1}|_{W^{1,\infty}} + |u_{\theta,M}^{-1}|_{W^{2,\infty}}. \tag{3.18}$$

Remark 3.5. *The constants $C_{gain,H^1}^{\times,M}$ and $C_{gain,L^2}^{\times,M}$ represent the potential gains in both H^1 semi-norm and L^2 norm when using the multiplicative approach, compared to the error of the classical FEM presented in Theorem 2.6 with $\mathbb{P}_k$ polynomials.*

Proof of Theorem 3.4. **H^1-error:** Considering the trial space defined in (3.11), the numerical solution $u_{h,M}^\times$ is given by:

$$u_{h,M}^\times = u_{\theta,M}\, p_h^\times,$$

with $p_h^\times \in 1 + V_h^0 \subset 1 + V^0$ solution to (3.12). By coercivity of a:

$$\alpha |u_M - u_{h,M}^\times|_{H^1}^2 \leqslant a(u_M - u_{h,M}^\times, u_M - u_{h,M}^\times).$$

Thanks to (3.10) and (3.12), we have the following Galerkin orthogonality, for all $v_h \in V_h^0$:

$$a(u_M - u_{h,M}^\times, u_{\theta,M} v_h) = 0. \tag{3.19}$$

Now, choosing

$$v_h = p_h^\times - \mathcal{I}_h\left(\frac{u_M}{u_{\theta,M}}\right),$$

we note that v_h is well-defined since the lifting trick ensures that $u_{\theta,M} > 0$, and that it belongs to V_h^0. With this choice, we deduce by the definition (3.13) of $\mathcal{I}_h^\times$ that:

$$\begin{aligned} a(u_M - u_{h,M}^\times, u_M - u_{h,M}^\times) &= a\left(u_M - u_{h,M}^\times, u_M - u_{\theta,M}\mathcal{I}_h\left(\frac{u_M}{u_{\theta,M}}\right)\right) \\ &= a(u_M - u_{h,M}^\times, u_M - \mathcal{I}_h^\times(u_M)). \end{aligned}$$

By continuity of a:

$$|u_M - u_{h,M}^\times|_{H^1} \leqslant \frac{\gamma}{\alpha} |u_M - \mathcal{I}_h^\times(u_M)|_{H^1}. \tag{3.20}$$

Again, using the definition (3.13) of $\mathcal{I}_h^\times$:

$$|u_M - \mathcal{I}_h^\times(u_M)|_{H^1} \leqslant \|u_{\theta,M}\|_{W^{1,\infty}} \left\| \frac{u_M}{u_{\theta,M}} - \mathcal{I}_h\left(\frac{u_M}{u_{\theta,M}}\right) \right\|_{H^1}.$$

Finally, applying interpolation estimate given in Theorem 2.4, it holds:

$$|u_M - \mathcal{I}_h^{\times}(u_M)|_{H^1} \leqslant C_s \|u_{\theta,M}\|_{W^{1,\infty}} h^s \left| \frac{u_M}{u_{\theta,M}} \right|_{H^{s+1}}, \tag{3.21}$$

with C_s defined in Section 2.2.1. Combining the last inequality with (3.20), we obtain:

$$|u - u_h^{\times}|_{H^1} = |u_M - u_{h,M}^{\times}|_{H^1} \leqslant C_s \frac{\gamma}{\alpha} C_{\text{gain},H^1}^{\times,M} h^s |u|_{H^{s+1}}, \tag{3.22}$$

with $C_{\text{gain},H^1}^{\times,M}$ given in (3.16), which conclude the first part of the proof.

L^2-error: Again, we follow the Aubin–Nitsche strategy here. Consider $w - M \in H_0^1(\Omega) \cap H^2(\Omega)$ solution to:

$$\mathcal{L}^* w = u - u_h^{\times} = u_M - p_h^{\times} u_{\theta,M},$$

with $w = M$ on Γ. Thanks to Theorem 2.5, one has:

$$\|w\|_{H^2} \leqslant C_e \|u - u_h^{\times}\|_{L^2}.$$

Then, using the Galerkin orthogonality (3.19) for $v_h = \mathcal{I}_h \left(\frac{u_M}{u_{\theta,M}} \right)$:

$$\|u - u_h^{\times}\|_{L^2}^2 = \|u_M - p_h^{\times} u_{\theta,M}\|_{L^2}^2 = a(u_M - p_h^{\times} u_{\theta,M}, w) = a(u_M - p_h^{\times} u_{\theta,M}, w - \mathcal{I}_h^{\times}(w)).$$

Hence, by continuity of a:

$$\|u - u_h^{\times}\|_{L^2}^2 \leqslant \gamma |u_M - p_h^{\times} u_{\theta,M}|_{H^1} |w - \mathcal{I}_h^{\times}(w)|_{H^1}.$$

Using (3.22) and (3.21) for $s = 1$ to the term in the right-hand side:

$$\|u - u_h^{\times}\|_{L^2}^2 \leqslant C_1 C_s \frac{\gamma^2}{\alpha} \|u_{\theta,M}\|_{W^{1,\infty}} \left| \frac{w}{u_{\theta,M}} \right|_{H^2} C_{\text{gain},H^1}^{\times,M} h^{s+1} |u|_{H^{s+1}}.$$

Moreover,

$$\left| \frac{w}{u_{\theta,M}} \right|_{H^2} \leqslant C_{\theta,M} \|w\|_{H^2},$$

with $C_{\theta,M}$ given in (3.18). Thanks to the elliptic regularity, we obtain:

$$\|u - u_h^{\times}\|_{L^2} \leqslant C_e C_1 C_s \frac{\gamma^2}{\alpha} C_{\text{gain},L^2}^{\times,M} h^{s+1} |u|_{H^{s+1}},$$

with $C_{\text{gain},L^2}^{\times,M}$ defined in (3.17). □

Remark 3.6. *We note that the gain constants $C_{gain,H^1}^{\times,M}$ and $C_{gain,L^2}^{\times,M}$ are similar to the constant C_{gain}^{+} introduced in Section 3.1, in that, it depends on high-order derivatives of the prior. Hence, a high-quality prior necessarily involves a good approximation of the derivatives of the exact solution, and Remark 3.3 also applies in the present context. The major difference with the additive approach lies in the choice of the lifting constant M. To better understand this dependency in M, the following section*

provides a study of the behavior of our two gain constants when M goes to infinity. Moreover, the actual choice of M will be numerically investigated in Section 3.5.

Having derived the multiplicative counterpart of the additive convergence estimates, governed by a similar dependence on the high-order derivatives of the prior but now also involving the lifting constant M, we now turn to a direct comparison of the two gain constants, and in particular to their behavior as $M \to \infty$.

3.3. Comparison of the two enriched methods

This section aims to compare the two proposed methods, namely the additive approach presented in Section 3.1 and the multiplicative one proposed in Section 3.2. The constant M is chosen in the multiplicative approach so that $u_M > 0$. Let u be the solution of problem (2.3) and $u_\theta \in H^{s+1}(\Omega) \cap H_0^1(\Omega)$ be a prior on u, with $1 \leqslant s \leqslant k$ (k the polynomial degree of the finite element method). For clarity, we first recall the additive and multiplicative enrichments and their main error estimates, before showing that the multiplicative gain constants converge to the additive one when the lifting constant M goes to infinity (Theorem 3.7).

Additive approach. We consider $u_h^+ \in V_h^+$ as the solution to the finite element method associated to problem (3.2) with V_h^+ the modified trial space defined in (3.1), considering $\mathbb{P}_k$ polynomials. Using Theorem 3.1, we have for $1 \leqslant s \leqslant k$, the following additive theoretical gain constant:

$$C_{\text{gain}}^+ = \frac{|u - u_\theta|_{H^{s+1}}}{|u|_{H^{s+1}}}. \tag{3.23}$$

Multiplicative approach. Let $u_M = u + M$ be the solution of the enhanced problem (3.10) and $u_{\theta,M} = u_\theta + M$ be a prior on u_M. We consider $u_{h,M}^\times \in V_h^\times$ the solution to the finite element problem (3.12) with $V_h^\times$ the modified trial space defined in (3.11), considering $\mathbb{P}_k$ polynomials. Using Theorem 3.4, we have for $1 \leqslant s \leqslant k$, the following multiplicative theoretical gain constants:

$$C_{\text{gain},H^1}^{\times,M} = \left| \frac{u_M}{u_{\theta,M}} \right|_{H^{s+1}} \frac{\|u_{\theta,M}\|_{W^{1,\infty}}}{|u|_{H^{s+1}}}, \tag{3.24}$$

and

$$C_{\text{gain},L^2}^{\times,M} = C_{\theta,M} \left| \frac{u_M}{u_{\theta,M}} \right|_{H^{s+1}} \frac{\|u_{\theta,M}\|_{W^{1,\infty}}^2}{|u|_{H^{s+1}}}, \tag{3.25}$$

with $C_{\theta,M}$ given in (3.18).

Comparison of the two approaches. The following result proves that the upper bound in (3.14) and in (3.15) converges respectively to the one in (3.4) and in (3.5) when M goes to infinity. In other words, the multiplicative gain constants defined in (3.24) and (3.25) converge to the additive gain constant defined in (3.23) when M goes to infinity.

Theorem 3.7. *We have:*

$$C^{\times,M}_{gain,H^1} \underset{M\to\infty}{\longrightarrow} C^{+}_{gain}, \tag{3.26}$$

and

$$C^{\times,M}_{gain,L^2} \underset{M\to\infty}{\longrightarrow} C^{+}_{gain}. \tag{3.27}$$

Proof of Theorem 3.7. **Convergence in H^1:** According to the expressions (3.23) and (3.24) of the gain constants, the objective of the proof is to show that:

$$\|u_{\theta,M}\|_{W^{1,\infty}} \left|\frac{u_M}{u_{\theta,M}}\right|_{H^{s+1}} \underset{M\to\infty}{\longrightarrow} |u-u_\theta|_{H^{s+1}}.$$

Denoting by

$$E_\theta := u - u_\theta,$$

the error made by the prior u_θ when approximating the solution u, we have:

$$|u-u_\theta|_{H^{s+1}} = |E_\theta|_{H^{s+1}}.$$

On the one hand, we have:

$$\|u_{\theta,M}\|_{W^{1,\infty}} = \|u_\theta + M\|_{W^{1,\infty}} = M\left\|1+\frac{u_\theta}{M}\right\|_{W^{1,\infty}}.$$

On the other hand, we have:

$$\begin{aligned}\left|\frac{u_M}{u_{\theta,M}}\right|_{H^{s+1}} = \left|\frac{u+M}{u_\theta+M}\right|_{H^{s+1}} &= \left|\frac{u-u_\theta+u_\theta+M}{u_\theta+M}\right|_{H^{s+1}} \\ &= \left|1+\frac{u-u_\theta}{u_\theta+M}\right|_{H^{s+1}} = \frac{1}{M}\left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|_{H^{s+1}}.\end{aligned}$$

Multiplying these expressions, we obtain:

$$\|u_{\theta,M}\|_{W^{1,\infty}} \left|\frac{u_M}{u_{\theta,M}}\right|_{H^{s+1}} = \underbrace{\left\|1+\frac{u_\theta}{M}\right\|_{W^{1,\infty}}}_{\text{(I)}} \underbrace{\left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|_{H^{s+1}}}_{\text{(II)}}. \tag{3.28}$$

We now estimate term by term the right-hand side of the above equality (3.28), looking at their limits when M goes to infinity.

Term (I): By decomposing the first term, we obtain:

$$\text{(I)} = \left\|1+\frac{u_\theta}{M}\right\|_{W^{1,\infty}} = \left\|1+\frac{u_\theta}{M}\right\|_{L^\infty} + \frac{1}{M}\|\nabla u_\theta\|_{L^\infty} \underset{M\to\infty}{\longrightarrow} 1,$$

since

$$1 - \frac{\|u_\theta\|_{L^\infty}}{M} \leqslant \left\|1+\frac{u_\theta}{M}\right\|_{L^\infty} \leqslant 1 + \frac{\|u_\theta\|_{L^\infty}}{M}. \tag{3.29}$$

Term (II): Let us prove that:

$$\text{(II)} = \left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|_{H^{s+1}} \underset{M\to\infty}{\longrightarrow} |E_\theta|_{H^{s+1}}.$$

By the triangular inequality:

$$\left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|^2_{H^{s+1}} \leqslant \left|\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right|^2_{H^{s+1}} + |E_\theta|^2_{H^{s+1}},$$

and

$$|E_\theta|^2_{H^{s+1}} \leqslant \left|\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right|^2_{H^{s+1}} + \left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|^2_{H^{s+1}}.$$

Hence, by definition of the semi-norm $|\cdot|_{H^{s+1}}$:

$$\begin{aligned}\left|\left|\frac{E_\theta}{1+\frac{u_\theta}{M}}\right|^2_{H^{s+1}} - |E_\theta|^2_{H^{s+1}}\right| &\leqslant \left|\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right|^2_{H^{s+1}} = \left\|\nabla^{s+1}\left(\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right)\right\|^2_{L^2} \\ &\leqslant |\Omega|\left\|\nabla^{s+1}\left(\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right)\right\|^2_{L^\infty}.\end{aligned}$$

Then, using the general Leibniz rule, we have:

$$\begin{aligned}\left\|\nabla^{s+1}\left(\frac{E_\theta}{1+\frac{u_\theta}{M}} - E_\theta\right)\right\|_{L^\infty} &\leqslant \left\|\frac{\nabla^{s+1}E_\theta}{1+\frac{u_\theta}{M}} - \nabla^{s+1}E_\theta\right\|_{L^\infty} \\ &\quad + \sum_{r=1}^{s+1}\binom{s+1}{r}\left\|\nabla^{s+1-r}E_\theta\right\|_{L^\infty}\left\|\nabla^r\left(\frac{1}{1+\frac{u_\theta}{M}}\right)\right\|_{L^\infty} \\ &\leqslant \underbrace{\left\|\frac{\nabla^{s+1}E_\theta}{1+\frac{u_\theta}{M}} - \nabla^{s+1}E_\theta\right\|_{L^\infty}}_{(1)} \\ &\quad + \|E_\theta\|_{W^{s+1,\infty}}\sum_{r=1}^{s+1}\binom{s+1}{r}\underbrace{\left\|\nabla^r\left(\frac{1}{1+\frac{u_\theta}{M}}\right)\right\|_{L^\infty}}_{(2)}.\end{aligned}$$

We now estimate terms (1) and (2) in the right-hand side of the above inequality.

Term (1): Taking $M > \|u_\theta\|_{L^\infty}$, we obtain:

$$(1) \leqslant \left\|\frac{1}{1+\frac{u_\theta}{M}} - 1\right\|_{L^\infty}\|\nabla^{s+1}E_\theta\|_{L^\infty} \leqslant \frac{1}{M}\frac{\|u_\theta\|_{L^\infty}}{1-\frac{\|u_\theta\|_{L^\infty}}{M}}\|\nabla^{s+1}E_\theta\|_{L^\infty} \underset{M\to\infty}{\longrightarrow} 0.$$

Term (2): Let us define $g = 1 + \frac{u_\theta}{M}$. The Faà di Bruno formula (see [79]) gives the following expression for the r-th derivative of the inverse of g:

$$\nabla^r\left(\frac{1}{g}\right) = \frac{r!}{g^{r+1}}\sum\frac{(-1)^\alpha\alpha!}{\prod_{i=1}^r(i!)^{m_i}m_i!}g^{r-\alpha}\prod_{i=1}^r\left(\nabla^i g\right)^{m_i},$$

where the sum covers all r-tuples $(m_1,\dots,m_r)$ satisfying the constraint $r = \sum_{i=1}^r im_i$ and $\alpha = \sum_{i=1}^r m_i$.

Thus, using (3.29), there exists a constant $C > 0$ such that, for any $M > \|u_\theta\|_{L^\infty}$ and $1 \leqslant r \leqslant s+1$, the following estimate holds:

$$\begin{aligned}(2) &\leqslant C \frac{1}{\min_{x\in\Omega}\left|1+\frac{u_\theta(x)}{M}\right|^{r+1}} \sum \left\|1+\frac{u_\theta}{M}\right\|_{L^\infty}^{r-\alpha} \prod_{i=1}^{r} \left\|\nabla^i\left(1+\frac{u_\theta}{M}\right)\right\|_{L^\infty}^{m_i} \\ &\leqslant C \frac{1}{\left(1-\frac{\|u_\theta\|_{L^\infty}}{M}\right)^{r+1}} \sum \left(1+\frac{\|u_\theta\|_{L^\infty}}{M}\right)^{r-\alpha} \frac{1}{M^\alpha} \prod_{i=1}^{r} \|\nabla^i u_\theta\|_{L^\infty}^{m_i} \underset{M\to\infty}{\longrightarrow} 0,\end{aligned}$$

which leads to (3.26).

Convergence in L^2: Using the convergence of the gain for the H^1 semi-norm, we only need to prove that:

$$C_{\theta,M}\|u_{\theta,M}\|_{W^{1,\infty}} \underset{M\to\infty}{\longrightarrow} 1,$$

with $C_{\theta,M}$ given by:

$$C_{\theta,M} = \|u_{\theta,M}^{-1}\|_{L^\infty} + 2|u_{\theta,M}^{-1}|_{W^{1,\infty}} + |u_{\theta,M}^{-1}|_{W^{2,\infty}}.$$

Since

$$\frac{1}{M}\|u_{\theta,M}\|_{W^{1,\infty}} \underset{M\to\infty}{\longrightarrow} 1,$$

we are only required to prove that:

$$MC_{\theta,M} \underset{M\to\infty}{\longrightarrow} 1.$$

Considering $M > \|u_\theta\|_{L^\infty}$, we have:

$$M\|u_{\theta,M}^{-1}\|_{L^\infty} = \left\|\frac{1}{1+\frac{u_\theta}{M}}\right\|_{L^\infty} \leqslant \frac{1}{1-\frac{\|u_\theta\|_{L^\infty}}{M}} \underset{M\to\infty}{\longrightarrow} 1.$$

Moreover,

$$2M|u_{\theta,M}^{-1}|_{W^{1,\infty}} = 2M\left\|\frac{\nabla u_\theta}{(u_\theta+M)^2}\right\|_{L^\infty} \underset{M\to\infty}{\longrightarrow} 0.$$

Similarly,

$$M|u_{\theta,M}^{-1}|_{W^{2,\infty}} \underset{M\to\infty}{\longrightarrow} 0,$$

which leads to the conclusion (3.27). □

Having shown that the multiplicative gain constants converge to the additive one as the lifting constant M grows, so that the two enrichment strategies are asymptotically equivalent, we now turn to practical aspects of implementing these methods, in particular how the PINN prediction is effectively plugged into the FEM solver and how boundary conditions are imposed in each of the two approaches.

3.4. Implementation details

Before moving on to numerical experiments, we discuss some practical details regarding the implementation of the methods introduced in Sections 3.1 and 3.2. In Section 3.4.1, we first look at how to effectively plug the PINN prediction in the FEM solver. Then, in Section 3.4.2, we discuss the imposition of boundary conditions in the two proposed methods.

The tools used to implement the methods and obtain the numerical results in Section 3.5 are, on the one hand, the `PyTorch` [103] version of `ScimBa`[1] for the prior construction, in particular the implementation of the PINN, and, on the other hand, `FEniCS` [7] (version 2019.1.0) or `FEniCSx` [6, 14, 114, 115] (version 0.8) for the implementation of the finite element methods. For mesh generation, we use either `FEniCS`/`FEniCSx` or `mshr` mesh generators. Note that, we define the characteristic mesh size h as the length of the longest edge, for the space dimension $d \in \{1, 2, 3\}$.

Moreover, in Section 3.5, we do not specify the training times of the networks for each test case. To give the reader a rough idea, PINN training takes less than ten minutes on a NVIDIA RTX 2000 (8 GB VRAM). For cases using the LBFGS optimizer, training takes a little longer, but remains under an hour. Regarding the FEM part, the central processing unit (CPU) considered depends on the numerical cost required for each test case; we ran the solvers using either Intel I7-13800H or AMD EPYC 7713.

Notation 3.8. *To construct the PINN, we require some hyperparameters for the MLP (Notation 2.13) and the training phase. In particular, the MLP activation function is denoted by σ, and* `layers` *represents a sequence of integers describing the number of neurons associated with each hidden layer of the MLP. For training, we denote by* `lr` *the learning rate and N_{epochs} the number of epochs considered, as well as* `decay` *the multiplicative factor of the learning rate decay considered every* 20 *epochs thanks to* `PyTorch`*'s* `StepLR` *scheduler. Unless otherwise specified, the batch size corresponds to the number of collocation points chosen. The Adam optimizer [83] is used to train the network, but in some cases, we switch to the LBFGS optimizer [100] at the N_{switch}-th epoch.*

3.4.1. Using PINN prediction effectively

To be effective, our methods depend in practice on the quality of the approximation of the prior's derivatives, computed from automatic differentiation, and its precise integration on the domain.

Automatic differentiation. It is important to use the automatic differentiation offered by neural networks, enabling exact (in the sense of machine precision) derivative computation without having to manipulate complex symbolic expressions. In particular, in the context of PINNs, automatic differentiation plays a fundamental role in integrating the PDE under consideration. This automatic differentiation enables the two improved finite element methods to use the exact derivatives of the prior u_θ and thus avoid introducing an additional error in the computation of the derivative.

[1] https://www.scimba.org/

Numerical integration. Our approaches require the numerical integration, in the weak problem, of several functions with a closed-form expression (most notably the prior; also the right-hand side of the PDE for instance). This integration has to be done with sufficient precision for our methods to be effective. Thus, in e.g. the additive approach, a quadrature rule with a higher degree than the traditional FEM has to be applied to discretize the term $l(v_h) - a(u_\theta, v_h)$ in (3.3). This point, and the required degree of the quadrature rule, will be studied in more detail in the first 2D test case considered in Section 3.5.5.

Remark 3.9. *In practice, the source term in the additive approach is computed in the strong way. For instance, in the case of the Laplacian equation with $u_\theta = 0$ on Γ, the term:*

$$l(v_h) - a(u_\theta, v_h) = \int_\Omega f(\boldsymbol{x}) v_h(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_\Omega \nabla u_\theta(\boldsymbol{x}) \cdot \nabla v_h(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$$

is replaced by:

$$\int_\Omega \big(f(\boldsymbol{x}) + \Delta u_\theta(\boldsymbol{x})\big) v_h(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

If u_θ is not equal to zero on Γ, one needs to include a boundary term.

3.4.2. Imposing boundary conditions

We now focus on the crucial question of imposing boundary conditions. We first look at this problem in the context of the additive approach presented in Section 3.1 and then in the context of the multiplicative approach presented in Section 3.2.

For simplicity, we focus on (non-homogeneous) Dirichlet conditions. For our enriched FEM, just like in classical FEM, these boundary conditions are imposed by manually eliminating essential dofs, see [55], and more precisely by modifying the matrix and the right-hand side of the linear system. This approach is not needed for Neumann and Robin conditions.

Additive approach. In this first approach, if our Dirichlet problem satisfies $u = g$ on Γ, then p_h^+ has to satisfy:

$$p_h^+ = g - u_\theta \quad \text{on } \Gamma,$$

with u_θ the PINN prior. This non-homogeneous boundary condition becomes homogeneous as soon as u_θ is exact at the boundary, or, in other words, as soon as the boundary conditions (BCs) are imposed exactly in the PINN, as presented in Section 2.3.2.

Remark 3.10. *However, in the case of curved geometries (e.g. disks) where the meshes do not coincide with the boundary of the geometry, problems occur when $k > 2$, and especially on coarse meshes. In the numerical results, to avoid this problem and check the error estimates, we assume that $g = u$ on $\partial\Omega_h$ where Ω_h is the domain covered by the mesh (here fitted to the boundary, unlike the unfitted Ω_h of φ-FEM in Section 2.2.3). Furthermore, we need to be careful because even if, in PINN, the conditions are imposed exactly (as shown in Section 2.3.2), the prediction u_θ is not exact on the approximate boundary $\partial\Omega_h$. We made this choice here to simplify the problem, but in practice, solutions exist to improve the quality of the results.*

Multiplicative approach. In this second method, the boundary conditions are a bit more complex to handle. In Section 3.2, we have denoted by:

$$u_{h,M}^{\times} = u_{\theta,M}\, p_h^{\times}$$

the solution obtained by the multiplicative approach to the modified problem (3.9) with the prior $u_{\theta,M} = u_\theta + M$. Therefore, we can recover the solution $u_h^{\times}$ of the original problem (2.1) by setting $u_h^{\times} = u_{h,M}^{\times} - M$. We distinguish two cases, depending on whether the prior is the prediction of a standard PINN or of a PINN with exactly imposed boundary conditions.

Standard PINN. In the case where our prior u_θ is the prediction of a standard PINN, as presented in Section 2.3, the boundary conditions are not imposed exactly. Thus, if our problem satisfies $u = g$ on Γ, then $p_h^{\times}$ has to satisfy:

$$p_h^{\times} = \frac{g + M}{u_{\theta,M}} \quad \text{on } \Gamma.$$

PINN with exact BC. We now tackle the case where we exactly impose the boundary conditions in the PINN, as presented in Section 2.3.2. Then, supposing that $M > 0$, we have $u_{\theta,M} = g + M$ on Γ and therefore $p_h^{\times}$ has to satisfy:

$$p_h^{\times} = 1 \quad \text{on } \Gamma.$$

However, there is a specific case when this condition is not necessarily true. Indeed, if the boundary conditions are homogeneous, then $g = 0$ and $u_{\theta,M} = M$ on Γ. Considering that $u_\theta > 0$ in Ω, $M = 0$ is a possible choice. In this case, $u_{\theta,M} = u_{\theta,0} = 0$ on Γ, and $u_{h,M}^{\times} = u_{h,0}^{\times} = u_h^{\times}$ automatically satisfies the boundary conditions. Hence, imposing a boundary condition on $p_h^{\times}$ becomes unnecessary.

Remark 3.11. *In the numerical results of Section 3.5.3, one of these specific cases is considered in 1D. We will see that leaving $p_h^{\times}$ free gives better results here than imposing $p_h^{\times} = 1$ on Γ. Indeed, this approach leaves more freedom to capture the correct derivatives at the boundary.*

Having discussed how the PINN prediction is plugged into the FEM solver and how boundary conditions are imposed in both the additive and multiplicative approaches, we now turn to the numerical validation of these enriched methods on a range of test cases.

3.5. Numerical results within the theoretical framework

This section is dedicated to validating the proposed method on several numerical experiments, which are mostly instances of the problem (2.1) with spatial dimension $d \in \{1, 2, 3\}$, and with increasing complexity. The idea is to compare, in different ways, the additive and multiplicative approaches presented in Section 3.1 and Section 3.2 to the standard finite element method presented in Section 2.2.1. The multiplicative approach is only considered in dimension $d = 1$, showing that only in rare cases, and

with a good choice of the lifting constant M, does it provide better results than the additive one. We also show that the approach proposed in Section 2.3 for choosing the prior is more efficient than more classical ones, still in the 1D case.

In Section 3.5.1, we present the two tests that are performed for each test case. We are interested in three 1D test cases ($d = 1$): the Poisson problem with homogeneous Dirichlet conditions in Section 3.5.2, a general elliptic system in a convection-dominated regime in Section 3.5.3 and a non-smooth transmission problem in Section 3.5.4. We then consider three 2D cases ($d = 2$). We start with a Poisson problem with homogeneous Dirichlet conditions on a square domain in Section 3.5.5. We then continue with a more complicated elliptic problem, still with Dirichlet conditions and on a square domain, but with parameter-dependent anisotropy, in Section 3.5.6. In Section 3.5.7, we return to the 2D Poisson problem, but this time considering mixed boundary conditions on a ring-shaped domain. Finally, we consider a 3D Poisson problem with homogeneous Dirichlet conditions on a cube domain in Section 3.5.8. To reproduce these results, an open-source code is available on GitHub[2]. All these test cases are instances of the scalar, linear setting for which the additive and multiplicative approaches are backed by the theoretical results of Sections 3.1 and 3.2. Section 3.6 goes further and considers a non-linear extension, on a purely heuristic basis.

More specifically, in each test case, we consider the additive approach and use a prior constructed by a PINN (where we mostly impose boundary conditions using a level-set function). We perform an analysis of error estimates and an evaluation of the gains of the enriched approach over the standard approach on a set of parameters (in L^2 norm). In the first two 1D test cases (Sections 3.5.2 and 3.5.3), we also consider the multiplicative approach. A study in the H^1 semi-norm is performed in the first 1D and 2D test case (Sections 3.5.2 and 3.5.5.1). Additionally, this first 1D test case includes a comparative study between theoretical and numerical constants. In the third 1D test case, we focus on a problem where the gradient is discontinuous. Subsequently, we test the approach with other networks, notably: a data-driven network (Section 3.5.2) that justifies the use of PINNs, networks with weakly imposed conditions (Sections 3.5.4 and 3.5.5.3), and an improved PINN where we used Sobolev training (Section 3.5.5.2). Two studies on the numerical costs of the additive approach are carried out in Sections 3.5.5.1 and 3.5.8.2. We also perform a brief analysis of the influence of PINN quality, depending on the initialization of the network weights, the number of collocation points considered, and the number of epochs (Sections 3.5.2.5 and 3.5.8.4). In some test cases, we also look at the visualization of the solutions obtained by the different finite element approaches.

The first two 1D test cases show that the multiplicative approach only gives better results than the additive approach in certain very specific cases. The first test case also validates the theoretical results, in particular the importance of the derivatives of the prior by validating the choice of PINNs. The third 1D test case extends the theoretical framework to a non-smooth problem, showing that, once again, high-order derivatives have the greatest impact on the results. The first 2D test case (Poisson in a square) introduces training with Sobolev loss and discusses the numerical costs of the finite element approaches (in terms of number of degrees of freedom). It also discusses the imposition of weak or strong boundary conditions in the network and compares the gains between the "high frequency" and "low frequency" cases. The

[2] https://github.com/flecourtier/EnrichedFEMUsingPINNs

second 2D test case highlights the flexibility of the method for more complex problems, particularly with parameter-dependent anisotropy. The third 2D test case introduces the approach in a slightly more complex geometric setting (with an annulus) as well as Robin conditions. Finally, the 3D test case validates the approach in a larger dimension and analyzes the associated numerical costs. This time, the costs are expressed in terms of computation time, in the complete parametric framework. It also introduces the influence of PINN quality on the final results. More generally, all test cases show that, with the right choice of priors, the additive approach yields significant gains over the standard approach. In particular, the average results obtained over a set of parameters show that for elements $\mathbb{P}_1$, this enriched approach provides the same accuracy as the standard approach with meshes 5 (2D anisotropic problem) to 28 (low-frequency Poisson with Sobolev training) times coarser, which leads to a gain in computation time (from a purely online perspective). The 3D test case takes a closer look at execution times, particularly in the parametric context in which the enriched approach is developed. More specifically, gains in terms of mesh size make it possible to achieve a speed-up from 19 parameter sets to reach a relative accuracy of 10^{-3} (in the L^2 norm).

Remark 3.12. *In this chapter, and unlike the following chapters (Chapters 4 and 5), we focus solely on standard continuous finite element methods (no DG or φ-FEM). As stated in Section 3.4, the results presented here were obtained using the* `PyTorch` *version of* `ScimBa`*, chosen in particular for its implementation of PINNs. The aim of the following chapters is to develop and explain the implementation of its* `JAX` *version. The finite element approximations were performed using* `FEniCS/FEniCSx` *(see Section 3.4) and meshes composed of simplexes.*

Notation 3.13. *In the following set of numerical results, each mesh is characterized by its number of nodes per direction, denoted by N (see the note specific to each test case). This notation should not be confused with the number of cells N_{cells} (or its tuple per direction $\boldsymbol{N}_{cells}$) introduced in Chapter 4 for the logical mesh of* `ScimBa`*. As the mesh here is composed of simplexes (rather than a hypercube grid in the following chapter), N counts the vertices rather than the cells, and so the actual number of cells is not simply $(N-1)^d$. N is therefore nothing more than an analogy with $\boldsymbol{N}_{cells}$, and should be regarded as a notation specific to this chapter.*

3.5.1. Setup of the numerical experiments

For each of the proposed test cases, we consider a parametric problem, on which we train a PINN as presented in Section 2.3.1, to resolve it on a set of parameters, denoted by $\mathcal{M}$. Let $p = \dim(\mathcal{M})$ be the number of parameters, and consider a set $\mathcal{D}$ of N_μ parameter instances:

$$\mathcal{D} := \left\{ \boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N_\mu)} \right\},$$

with, for $j = 1, \dots, N_\mu$,

$$\boldsymbol{\mu}^{(j)} := \left(\mu_1^{(j)}, \dots, \mu_p^{(j)} \right) \in \mathcal{M}.$$

In the following, we denote by $u^{(j)}$ a reference solution to problem (2.35) for a given parameter $\boldsymbol{\mu}^{(j)}$ and by $u_h^{(j)}$ the solution obtained by the standard finite element

method (2.6), where V_h is the $\mathbb{P}_k$ Lagrange space defined in (2.4) and h is the characteristic mesh size. We also denote by $u_\theta^{(j)}$ the solution obtained by the parametric PINN, and by $u_{h,+}^{(j)}$ the solution obtained by the additive approach (3.3) with V_h^+ the $\mathbb{P}_k$ Lagrange space defined in (3.1). In some test cases, we also consider the solution $u_{h,M}^{(j)}$ of the multiplicative approach (3.12) with $V_h^\times$ the $\mathbb{P}_k$ Lagrange space defined in (3.11), depending on the lifting constant M.

Notation 3.14. *In the following, to estimate the error, we consider the reference solution to be either an analytical solution or a solution obtained with a very fine mesh and a high polynomial degree. More precisely, we need the characteristic size h_{ref} associated with the reference mesh to be much smaller than the size associated with the current mesh h, i.e. $h_{ref} \ll h$, and we consider $k_{ref} = 3$ the polynomial degree associated with the reference solution.*

In each test case, we investigate two aspects; the first in Section 3.5.1.1 involves verifying the error estimates and the second in Section 3.5.1.2 is the evaluation of the gains achieved by the proposed methods compared with the standard one.

3.5.1.1. Error estimates

Consider a small set $\mathcal{D}$ with $N_\mu = 2$ parameter instances. Given a fixed parameter $\boldsymbol{\mu}^{(j)}$, $j = 1, 2$, we start by testing the error estimates obtained in Theorem 3.1 for the additive approach. In the case $d = 1$, we are also interested in the error estimates in Theorem 3.4 for the multiplicative approach. By varying the mesh size h, we then estimate the errors obtained with the two methods. To evaluate these errors, we compare the approximations to the reference solution $u^{(j)}$ (see Notation 3.14). We then define by:

$$e_h^{(j)} := \frac{||u^{(j)} - u_h^{(j)}||_{L^2}}{||u^{(j)}||_{L^2}}, \text{ and } e_\theta^{(j)} := \frac{||u^{(j)} - u_\theta^{(j)}||_{L^2}}{||u^{(j)}||_{L^2}}, \tag{3.30}$$

the L^2 relative error obtained for the standard FEM and the PINN respectively. We further define:

$$e_{h,+}^{(j)} := \frac{||u^{(j)} - u_{h,+}^{(j)}||_{L^2}}{||u^{(j)}||_{L^2}}, \text{ and } e_{h,M}^{(j)} := \frac{||u^{(j)} - u_{h,M}^{(j)}||_{L^2}}{||u^{(j)}||_{L^2}}, \tag{3.31}$$

the L^2 relative errors obtained for the additive and multiplicative approach (depending on the lifting constant M), respectively. Relative errors in the semi-norm H^1 can be defined in the same way.

3.5.1.2. Gains achieved with the enriched bases

As we have trained the network to be parameter-dependent to predict a solution for a set of parameters, we are interested in the average gains we obtain with our enriched approaches compared to the PINN and the standard FEM. More precisely, for a fixed mesh size h and a fixed polynomial degree k, for a set $\mathcal{D}$ of N_μ parameter instances, the numerical gains obtained by the additive approach on PINN and standard FEM

are respectively defined for $j = 1, \dots, N_\mu$ by:

$$G_{+,\theta}^{(j)} := \frac{e_\theta^{(j)}}{e_{h,+}^{(j)}}, \quad \text{and} \quad G_+^{(j)} := \frac{e_h^{(j)}}{e_{h,+}^{(j)}}, \tag{3.32}$$

with $e_\theta^{(j)}$, $e_h^{(j)}$ and $e_{h,+}^{(j)}$ respectively the L^2 relative errors obtained with the PINN, the standard FEM and the additive approach, defined in Section 3.5.1.1. Similarly, the theoretical gains obtained by the multiplicative approach (depending on the lifting constant M) on PINN and standard FEM are respectively defined for $j = 1, \dots, N_\mu$ by:

$$G_{M,\theta}^{(j)} := \frac{e_\theta^{(j)}}{e_{h,M}^{(j)}}, \quad \text{and} \quad G_M^{(j)} := \frac{e_h^{(j)}}{e_{h,M}^{(j)}}, \tag{3.33}$$

with $e_{h,M}^{(j)}$ the L^2 relative error obtained with the multiplicative approach (depending on the lifting constant M), defined in Section 3.5.1.1. Therefore, we are interested in the minimum, maximum, mean and standard deviation obtained on the following samples:

$$G_{+,\theta} := \left\{ G_{+,\theta}^{(1)}, \dots, G_{+,\theta}^{(N_\mu)} \right\}, \quad \text{and} \quad G_+ := \left\{ G_+^{(1)}, \dots, G_+^{(N_\mu)} \right\}, \tag{3.34}$$

which respectively represent the gains obtained with our additive approach over PINN and over standard FEM on the set $\mathcal{D}$ of parameter instances. In the same way, we define $G_{M,\theta}$ and G_M, which respectively represent the gains obtained with our multiplicative approach over PINN and over standard FEM on $\mathcal{D}$ by:

$$G_{M,\theta} := \left\{ G_{M,\theta}^{(1)}, \dots, G_{M,\theta}^{(N_\mu)} \right\}, \quad \text{and} \quad G_M := \left\{ G_M^{(1)}, \dots, G_M^{(N_\mu)} \right\}. \tag{3.35}$$

3.5.2. 1D Poisson problem

In this section, we consider the problem (2.1) in its most simplified Poisson form, with homogeneous Dirichlet boundary conditions. In the 1D case ($d = 1$), we have:

$$\begin{cases} -\partial_{xx} u = f, & \text{in } \Omega \times \mathcal{M}, \\ u = 0, & \text{on } \Gamma \times \mathcal{M}, \end{cases} \tag{3.36}$$

with $\Omega = (0, 1)$, Γ its boundary and $\mathcal{M} \subset \mathbb{R}^p$ the parameter space (with p the number of parameters). We prescribe a family of exact solutions depending on the parameter vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \mathcal{M} = [0, 1]^3$ ($p = 3$ parameters) defined by:

$$u(x, \boldsymbol{\mu}) = \mu_1 \sin(2\pi x) + \mu_2 \sin(4\pi x) + \mu_3 \sin(6\pi x)\,.$$

Note that the associated right-hand side f in (3.36) also depends on $\boldsymbol{\mu}$. This problem is thus four-dimensional: one dimension in space and three dimensions for the parameters $\boldsymbol{\mu}$.

For this first test case, we construct two priors, as detailed in Section 3.5.2.1. The first one, denoted by u_θ, is built from a PINN as presented in Section 2.3. The second one, denoted by u_θ^{data} is constructed only from data (obtained from the analytical solution). The aim is to show that using physics-informed training to construct the prior leads to

better results than data-driven training. In this test case, we also compare the additive and multiplicative approaches (presented for several values of M), but only with $k = 1$ polynomial order to remain concise.

We first present the error estimates obtained with the PINN prior in Section 3.5.2.2. First, we check the orders of convergence of the two enriched approaches. Then, we verify the results expected in Section 3.3 by comparing the theoretical gain constants of the additive and multiplicative approaches. Afterwards, in Section 3.5.2.3, we compare, for a given parameter, the derivatives obtained with the two priors and analyze the associated gains. Then, we evaluate the gains obtained with the two priors in Section 3.5.2.4 on a sample of parameters. Finally, in Section 3.5.2.5, we analyze the influence of the weight initialization of the PINN prior on the gains obtained with our enriched approaches.

Notation 3.15. *In the following, the characteristic mesh size is* $h = \frac{1}{N-1}$*, where* N *represents the number of considered nodes.*

3.5.2.1. Construction of the two priors

The hyperparameters used to construct the two priors are presented in Table 3.1. We discuss below the specific differences in training both priors.

Network - MLP	
layers	$20, 80, 80, 80, 20, 10$
σ	sine

Training			
lr	9e-2	N_{epochs}	10 000
decay	0.99		
$N_{\text{col/data}}$	5 000		

Loss weights							
PINN prior u_θ				Data prior u_θ^{data}			
ω_r	1	ω_{data}	0	ω_r	0	ω_{data}	1
ω_b	0	ω_{sob}	0	ω_b	0	ω_{sob}	0

Table 3.1.: Network, training parameters (Notation 3.8) and loss weights for u_θ and u_θ^{data} in the *1D Poisson problem*. N_{col} denotes the number of collocation points for the PINN prior and N_{data} the number of data for the data prior.

Physics-informed training. For the first prior, we consider a parametric PINN, depending on the problem parameters $\boldsymbol{\mu}$, where we exactly impose the Dirichlet boundary conditions as presented in Section 2.3.2 and without using data in training. Thus, we construct u_θ as in (2.42) with the level-set function φ defined by:

$$\varphi(x) = x(x-1)\,,$$

which vanishes exactly on Γ and $g = 0$. Since we impose the boundary conditions by using the level-set function, we only consider the residual loss $\mathcal{J}_r$ approached by a Monte Carlo method as defined in (2.40) with $N_{\text{col}} = 5\,000$ collocation points uniformly chosen on $\Omega \times \mathcal{M}$. The hyperparameters are given in Table 3.1; we use the Adam optimizer [83].

Data-driven training. For the second prior considered, noted u_θ^{data}, a network is trained only on the data (constructed from the analytical solution). Therefore, we only consider the data loss $\mathcal{J}_{\text{data}}$ defined in (2.39), considering $N_{\text{data}} = 5\,000$ points. As for the physics-informed training, we still consider the hyperparameters defined in Table 3.1 and construct the prior u_θ^{data} in the same way using the level-set function.

3.5.2.2. Error estimates – with the PINN prior

In this section, we look at the theoretical results of the additive and multiplicative approaches, considering the PINN prior u_θ. First, we check the orders of convergence of Theorems 3.1 and 3.4 (in the L^2 norm), associated with both methods. Next, we numerically verify Theorem 3.7, showing that the multiplicative correction converges, for sufficiently large M, towards the additive correction (in both the L^2 norm and H^1 semi-norm).

Convergence rate. We test the error estimates of Theorems 3.1 and 3.4 for the following two sets of parameters:

$$\boldsymbol{\mu}^{(1)} = (0.3, 0.2, 0.1), \quad \text{and} \quad \boldsymbol{\mu}^{(2)} = (0.8, 0.5, 0.8)\,,$$

with the PINN prior u_θ. For $j \in \{1, 2\}$, the aim is to compare, by varying the mesh size h, the L^2 relative errors $e_h^{(j)}$ obtained with the standard FEM, defined in (3.30), $e_{h,+}^{(j)}$ obtained with the additive approach, defined in (3.31) and $e_{h,M}^{(j)}$ obtained with the multiplicative approach (taking $M = 3$ and $M = 100$), defined in (3.31). The results are presented in Figure 3.2 for polynomial orders $k = 1$ and $k = 2$, with h depending on the number of nodes $N \in \{16, 32, 64, 128, 256\}$ as presented in Notation 3.15. We also present in Figure 3.3 the results obtained for the semi-norm H^1.

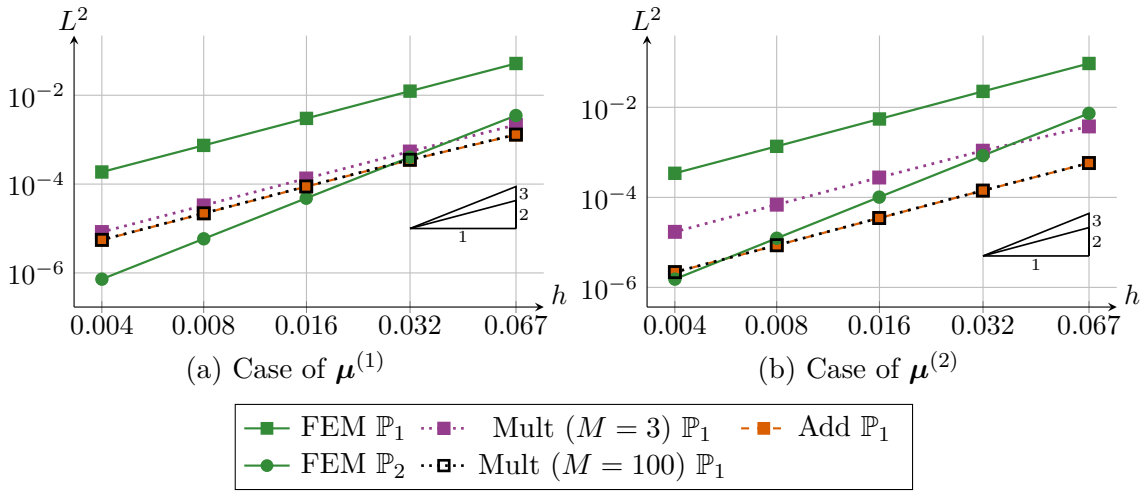


Figure 3.2.: Considering the *1D Poisson problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) with $k = 1$ and $k = 2$, the additive approach $e_{h,+}^{(1)}$ (dashed lines) with $k = 1$ and the multiplicative approach $e_{h,M}^{(1)}$ (dotted lines) with $k = 1$ ($M = 3$ and $M = 100$), for $\boldsymbol{\mu}^{(1)}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

The results of Figures 3.2 and 3.3 show that all the enriched finite elements increase the accuracy of the method and that they also converge at the same rate as the classical approach (i.e. for polynomial approximation of order $k = 1$, the convergence order in the L^2 norm is 2 and in the semi-norm H^1 is 1). Furthermore, the theoretical analysis (which showed that the multiplicative correction has the same error as the additive

one when $M \to \infty$) is confirmed for both sets of parameters. In addition, Figures 3.2 and 3.3 also shows that this multiplicative enrichment can be less efficient for small M when the $(k+1)^{\text{th}}$ derivative of the solution is large. Indeed, for the first parameter considered in Figures 3.2a and 3.3a, for which the second derivative takes lower values, we observe that the multiplicative approach with small M is closer to the additive one than for the second set of parameters considered in Figures 3.2b and 3.3b, for which the derivatives are larger. Moreover, it seems that we gain almost one order of interpolation with the additive approach: the additive method with polynomial order $k = 1$ gives relative errors close to the original FEM with $k = 2$, although the rate of convergence is different.

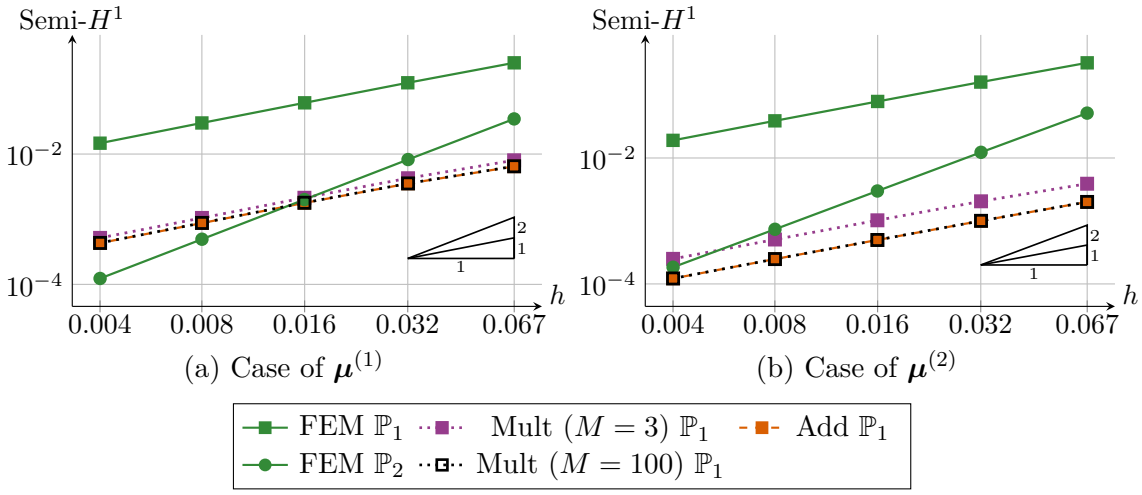


Figure 3.3.: Considering the *1D Poisson problem* and the PINN prior u_θ. Left – Semi-H^1 relative error as a function of h, obtained with the standard FEM (solid lines) with $k = 1$ and $k = 2$, the additive approach (dashed lines) with $k = 1$ and the multiplicative approach (dotted lines) with $k = 1$ ($M = 3$ and $M = 100$), for $\boldsymbol{\mu}^{(1)}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

Gain constants. We consider the first parameter $\boldsymbol{\mu}^{(1)}$ and the PINN prior u_θ. We now evaluate the gain constants C^{+}_{gain}, $C^{\times,M}_{\text{gain},H^1}$ and $C^{\times,M}_{\text{gain},L^2}$ (for different values of M), which are respectively defined in Theorems 3.1 and 3.4 for the additive and multiplicative approaches. The idea is to check the convergence of the two multiplicative gain constants towards the additive one, as proven in Theorem 3.7. The results are presented in Figure 3.4 in L^2 norm and H^1 semi-norm, considering $M > \min |u_\theta|$.

Figure 3.4 shows that the multiplicative gain constant converges to the additive gain constant when M increases, as expected in the theoretical results of Theorem 3.7. Furthermore, we can compare the expected theoretical gains with the numerical gains obtained. For the additive approach, it would appear that the expected theoretical gain (approximately 34.01) corresponds well with the numerical results of Tables 3.2 and 3.3 (left subtable). As for the multiplicative approach, it appears that these constants align well when M becomes large; however, for smaller M, the theoretical error estimates seem suboptimal. For example, for $M = 3$, we find a theoretical gain of approximately 8.22 in semi-norm H^1 compared to the 30 obtained in practice in Table 3.3 (left subtable). Moreover, in Figure 3.4, we see that the theoretical gain (in L^2 norm and H^1 semi-norm)

converges to 0 when M tends to $\min|u_\theta|$, whereas in practice the numerical gains obtained have never been less than 1. More precisely, for this test case in the H^1 semi-norm, we converge to 2.45 numerically when M is small.

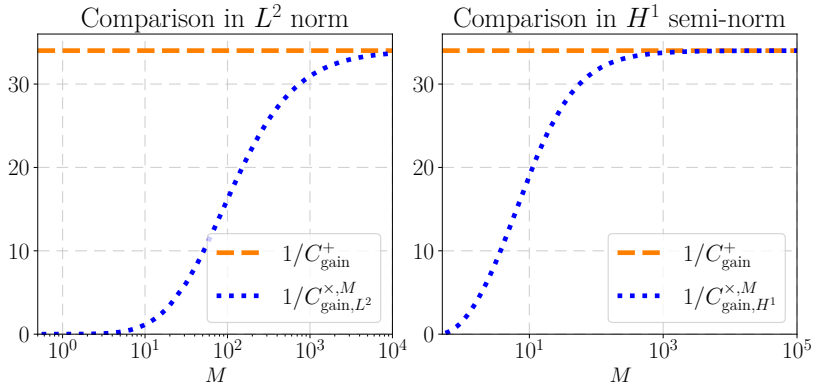


Figure 3.4.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$ and the PINN prior u_θ. Left – Numerical illustration of Theorem 3.7 for the L^2 relative error. Right – Same for the semi-H^1 relative error.

3.5.2.3. Derivatives – with both priors

To better explain the results of Section 3.5.2.2, we compare the solution, the first- and second-order derivatives between the exact solution and the prediction of both priors, for selected parameter $\boldsymbol{\mu}^{(1)}$. Figures 3.5 and 3.6 respectively present this comparison for the PINN prior u_θ and the data prior u_θ^{data}. We also compare the relative errors and gains obtained with these two priors for $N \in \{16, 32\}$ in Tables 3.2 and 3.3, respectively in L^2 norm and H^1 semi-norm. More precisely for the L^2 norm, we evaluate the additive error $e^{(1)}_{h,+}$ and the additive gain on FEM $G^{(1)}_{+}$, respectively defined in (3.31) and (3.32), for both PINN and data priors. We also evaluate the multiplicative error $e^{(1)}_{h,M}$ and the multiplicative gain on FEM $G^{(1)}_{M}$ defined in (3.31) and (3.33), for both priors, with $M = 3$ and $M = 100$.

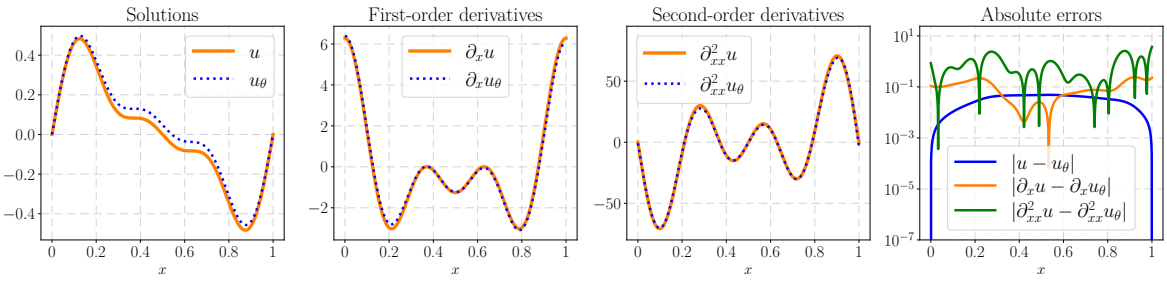


Figure 3.5.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$ and the PINN prior u_θ. Comparison between the analytical solution and the network prediction. From left to right – solution; first derivative; second derivative; errors.

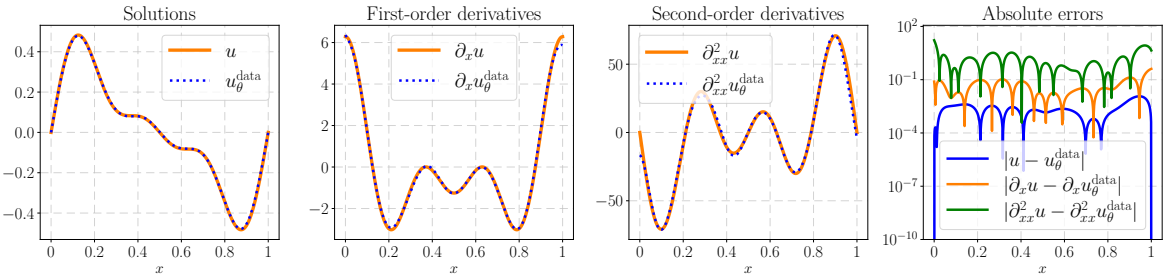


Figure 3.6.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$ and the data prior u_θ^{data}. Comparison between the analytical solution and the network prediction. From left to right – solution; first derivative; second derivative; errors.

	FEM
N	**error**
16	$5.18 \cdot 10^{-2}$
32	$1.24 \cdot 10^{-2}$

		PINN prior u_θ		**Data prior u_θ^{data}**	
method	**N**	**error**	**gain**	**error**	**gain**
Add	16	$1.29 \cdot 10^{-3}$	40.34	$3.51 \cdot 10^{-3}$	14.78
	32	$3.49 \cdot 10^{-4}$	35.41	$8.8 \cdot 10^{-4}$	14.06
Mult (M=3)	16	$2.15 \cdot 10^{-3}$	24.13	$3.38 \cdot 10^{-3}$	15.36
	32	$5.41 \cdot 10^{-4}$	22.86	$8.53 \cdot 10^{-4}$	14.51
Mult (M=100)	16	$1.3 \cdot 10^{-3}$	39.84	$3.5 \cdot 10^{-3}$	14.83
	32	$3.53 \cdot 10^{-4}$	35.08	$8.78 \cdot 10^{-4}$	14.09

Table 3.2.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N \in \{16, 32\}$, the PINN prior u_θ and the data prior u_θ^{data}. Left – L^2 relative error obtained with the standard FEM. Right – L^2 relative errors and gains with respect to the standard FEM obtained with the additive approach and the multiplicative approach ($M = 3$ and $M = 100$), for both priors.

	FEM
N	**error**
16	$2.51 \cdot 10^{-1}$
32	$1.24 \cdot 10^{-1}$

		PINN prior u_θ		**Data prior u_θ^{data}**	
method	**N**	**error**	**gain**	**error**	**gain**
Add	16	$6.45 \cdot 10^{-3}$	38.94	$1.74 \cdot 10^{-2}$	14.45
	32	$3.53 \cdot 10^{-3}$	35.21	$8.91 \cdot 10^{-3}$	13.93
Mult (M=3)	16	$8.02 \cdot 10^{-3}$	31.31	$1.64 \cdot 10^{-2}$	15.32
	32	$4.26 \cdot 10^{-3}$	29.16	$8.51 \cdot 10^{-3}$	14.59
Mult (M=100)	16	$6.49 \cdot 10^{-3}$	38.7	$1.74 \cdot 10^{-2}$	14.48
	32	$3.54 \cdot 10^{-3}$	35.04	$8.9 \cdot 10^{-3}$	13.95

Table 3.3.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N \in \{16, 32\}$, the PINN prior u_θ and the data prior u_θ^{data}. Left – Semi-H^1 relative error obtained with the standard FEM. Right – Semi-H^1 relative errors and gains with respect to the standard FEM obtained with the additive approach and the multiplicative approach ($M = 3$ and $M = 100$), for both priors.

The results reported in Figures 3.5 and 3.6 and Tables 3.2 and 3.3 show that, even if the approach chosen to build the prior (physics-informed or data-driven training) gives a good approximation of the solution, the important point lies in the derivatives and mainly in the second-order derivatives, which are clearly better learned by PINNs. Indeed, while the enriched FEM solution is more accurate using the PINN prior (Tables 3.2 and 3.3), we see from Figures 3.5 and 3.6 that the raw PINN approximates the solution u less accurately than the raw data-driven, but the PINN better approximates the derivatives. As the error of the enriched FEM is mainly due to the $(k+1)^{\text{th}}$ derivatives of the network being close to the $(k+1)^{\text{th}}$ derivatives of the solution, this explains why the enriched FEM with data prior does not perform as well as with a PINN prior. Therefore, PINNs have two advantages: they do not require training data and give better results. Their main shortcoming is that training takes longer; data could be used in addition to physics-informed training to speed up the process, but we do not expect it to improve the accuracy of the prior derivatives. However, we mention that if data are available for first- and second-order derivatives, it could also be used to improve a purely data-driven prior.

3.5.2.4. Gains achieved with the additive and multiplicative approaches – with both priors

Considering a set $\mathcal{D}$ of $N_\mu = 100$ parameter instances, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34) with the PINN prior u_θ and with the data prior u_θ^{data}. We also compute $G_{M,\theta}$ and G_M, defined in (3.35), similarly for both priors. For fixed polynomial order $k = 1$ and $N \in \{20, 40\}$, the results with the physics-informed prior u_θ and the data-driven prior u_θ^{data} are respectively presented in Tables 3.4 and 3.5.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
method	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
Add	20	10.47	251.6	92.19	50.96	26.49	271.92	140.74	55.16
	40	42.12	969.51	362.88	195.53	23.4	258.37	134.11	51.86
Mult (M=3)	20	5.19	106.29	25.34	20.57	3.93	163.81	42.72	34.93
	40	17.25	443.21	99.66	85.19	3.16	149.83	40.22	33.75
Mult (M=100)	20	10.48	253.57	92.59	51.9	26.29	270.87	140.82	55
	40	42.17	979.91	364.38	199.36	23.28	257.71	134.15	51.62

Table 3.4.: Considering the *1D Poisson problem*, $k = 1$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach and the multiplicative approach ($M = 3$ and $M = 100$) with respect to the PINN prior. Right – Same with respect to the standard FEM.

The previous results indicate that the average gain provided by the enriched FEM with the PINN prior is significant, particularly when using the additive approach. These findings also confirm the behavior of the multiplicative prior method for varying values of M. In contrast, when applied with data-driven instead of physics-informed training,

the same method does not yield similarly favourable results. Consequently, in the experiments we perform below, we only employ the PINN prior.

		Gains in L^2 rel error of our method w.r.t. Data Network				Gains in L^2 rel error of our method w.r.t. FEM			
method	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
Add	20	1.34	10.95	3.33	2.09	6.91	60.85	26.12	12.45
	40	4.35	34.37	11.19	6.97	7.13	39.34	20.55	7.94
Mult (M=3)	20	0.85	10.4	2.56	1.98	3.54	53.99	19.66	11.11
	40	2.64	35.3	8.68	6.98	2.18	39.06	15.66	8.25
Mult (M=100)	20	1.35	10.96	3.33	2.09	6.9	60.73	26.11	12.43
	40	4.35	34.41	11.19	6.97	7.13	39.38	20.55	7.93

Table 3.5.: Considering the *1D Poisson problem*, $k = 1$ and the data prior u_θ^{data}. Left – Gains in L^2 relative error of the additive approach and the multiplicative approach ($M = 3$ and $M = 100$) with respect to the data prior. Right – Same with respect to the standard FEM.

Furthermore, it is important to note that we cannot expect enriched methods to have generalization or extrapolation properties for parameters $\boldsymbol{\mu} \notin \mathcal{M}$. Indeed, these potential extrapolation properties would depend on those of the PINN. Since the PINN is a very smooth function, we might expect it to give reasonable results near the boundaries of $\mathcal{M}$. Intuitively, we might also expect that the further the parameters are from the training domain, the worse the results are. However, there is no way to control this potential extrapolation property. For this test case, considering the additive approach with the PINN prior, $N = 20$ and a sample of 100 parameters in $[1, 1.1]^3$ (close to $\mathcal{M}$), we find a mean gain of 9.18, compared to the 140.74 in Table 3.4 for parameters in $\mathcal{M}$: the gain has been lowered by a factor of 15. If we move away from the training area with parameters in $[1.1, 1.5]^3$, we once again halve this factor with a mean gain of 4.48, confirming the previous intuition.

3.5.2.5. Influence of the PINN initialization

In this section, we focus on the influence of network weight initialization on the quality of our results. To do this, we consider a set of 30 PINN initializations with the hyperparameters defined in Table 3.1. Thus, for each network, we are interested in the same way as in Section 3.5.2.4, in the gain of our enriched approaches on a set of $N_\mu = 50$ parameter instances. For each prior, we evaluate the minimum, maximum, mean, and standard deviation on this sample. We then regroup these 30 values in Table 3.6, in the form *mean*±*std*, where *mean* and *std* represent the mean and standard deviation, respectively, on the 30 networks considered.

In Table 3.6, we can see that the initialization of the PINN weights has an influence on the gains obtained with the different enriched approaches. More specifically, for the additive approach ($N = 20$), for example, we can see that the minimum gain obtained on the sample averages 23 across the 30 networks considered, with a standard deviation of 6. It would therefore seem that there is always a gain in using the enriched approach,

even if initialization influences its magnitude. On the other hand, even though the maximum is on average 307, the standard deviation is close to 50, which is not negligible. However, the results on the mean column seem close to the results obtained in Table 3.4. The results obtained with the multiplicative approach also appear to be consistent with those obtained previously.

		Gains in L^2 rel error of our method w.r.t. FEM			
method	**N**	**min**	**max**	**mean**	**std**
Add	20	23.04 ± 5.66	306.85 ± 50.27	148.56 ± 24.05	61.57 ± 10.46
Add	40	21.65 ± 5.19	278.92 ± 44.55	141.72 ± 21.92	56.69 ± 8.62
Mult (M=3)	20	6.10 ± 2.12	166.78 ± 52.32	44.54 ± 10.76	31.25 ± 8.81
Mult (M=3)	40	4.76 ± 1.69	154.64 ± 49.58	41.51 ± 10.18	29.74 ± 8.41
Mult (M=100)	20	23.03 ± 5.65	305.61 ± 50.89	148.07 ± 23.85	61.28 ± 10.31
Mult (M=100)	40	21.64 ± 5.17	277.30 ± 44.34	141.29 ± 21.77	56.45 ± 8.51

Table 3.6.: Considering the *1D Poisson problem*, $k = 1$ and a set of 30 PINN priors. Gains in L^2 relative error of the additive approach and the multiplicative approach ($M = 3$ and $M = 100$) with respect to the standard FEM (*mean*$\pm$*std* over the 30 networks).

3.5.3. 1D general elliptic system and convection-dominated regime

In this experiment, we consider the problem (2.1) in a more complex form, still in a 1D ($d = 1$) configuration:

$$\begin{cases} \partial_x u - \dfrac{1}{\mathrm{Pe}} \partial_{xx} u = r, & \text{in } \Omega \times \mathcal{M}, \\ u = 0, & \text{on } \Gamma \times \mathcal{M}, \end{cases}$$

with $\Omega = (0, 1)$ and Γ its boundary, r the reaction constant term, and Pe the Péclet number, describing the ratio between the convection and the diffusion terms. For all $x \in \Omega$, the analytical solution reads:

$$u(x, \boldsymbol{\mu}) = r \left(x - \frac{e^{\mathrm{Pe}\, x} - 1}{e^{\mathrm{Pe}} - 1} \right) , \tag{3.37}$$

with $p = 2$ parameters $\boldsymbol{\mu} = (r, \mathrm{Pe}) \in \mathcal{M} = [1, 2] \times [10, 100]$.

Remark 3.16. *In the large Péclet regime, i.e. for convection-dominated flows, the classical finite element method may generate oscillations when no specific treatment is applied, see e.g. [78].*

In this test case, we construct only one prior, denoted u_θ, built from a PINN as presented in Section 2.3. We also compare the additive and multiplicative approaches by considering polynomial order $k = 1$, and since the solution is positive in Ω, we consider $M = 0$ for the multiplicative approach. We start by evaluating the error

in Section 3.5.3.1, then we compare the derivatives of the PINN prior and compare the different approaches in Section 3.5.3.2. Finally, we evaluate the gains obtained in Section 3.5.3.3 on a sample of parameters. As we are dealing with a specific case, we compare two methods for imposing boundary conditions, as presented in Section 3.4.2 for the multiplicative approach: the strong and the weak approaches.

Notation 3.17. *As in Section 3.5.2, the characteristic mesh size is* $h = \frac{1}{N-1}$*, where* N *is the number of nodes considered.*

We consider a parametric PINN, depending on the problem parameters $\boldsymbol{\mu}$, where we exactly impose the Dirichlet boundary conditions as presented in Section 3.4.2. We define the prior u_θ and the level-set φ as in Section 3.5.2.

Network - MLP	
layers	$40, 40, 40, 40, 40$
σ	tanh

Training			
lr	1e-3	N_{epochs}	20 000
decay	0.99		
N_{col}	5 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0

Table 3.7.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *1D elliptic problem.*

Therefore, we only consider the residual loss $\mathcal{J}_r$ approached by a Monte Carlo method as defined in (2.40) with $N_{\text{col}} = 5\,000$ collocation points (uniformly chosen on $\Omega \times \mathcal{M}$) and we seek to solve the minimization problem (2.36). The hyperparameters are given in Table 3.7; we use the Adam optimizer [83].

3.5.3.1. Error estimates

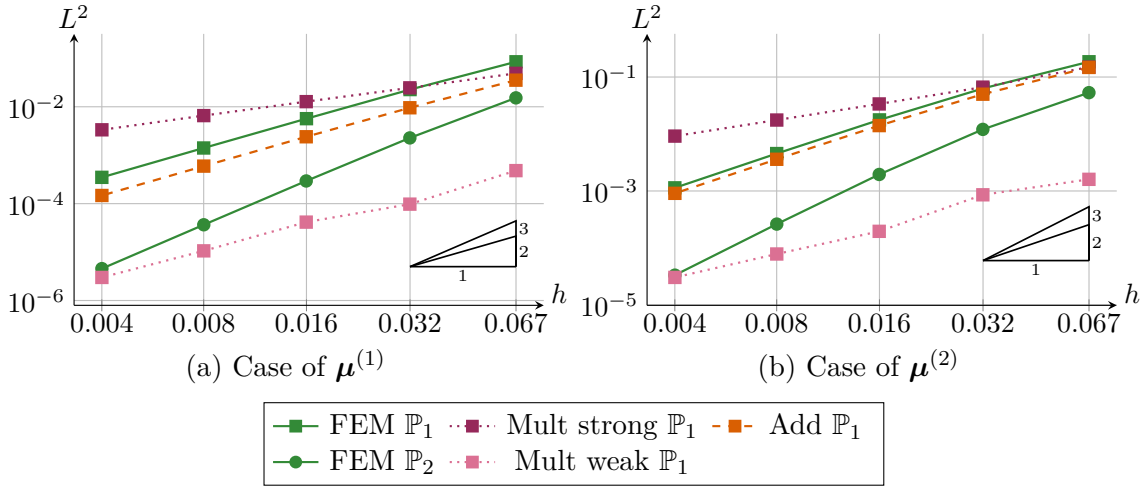
We start by testing the error estimates (Theorems 3.1 and 3.4) for the following two sets of parameters:

$$\boldsymbol{\mu}^{(1)} = (1.2, 40), \quad \text{and} \quad \boldsymbol{\mu}^{(2)} = (1.5, 90)\,,$$

by considering the PINN prior u_θ. For $j \in \{1, 2\}$, the aim is to compare for different mesh sizes h, the L^2 relative errors $e_h^{(j)}$ obtained with the standard FEM, defined in (3.30), $e_{h,+}^{(j)}$ obtained with the additive approach and $e_{h,M}^{(j)}$ obtained with the multiplicative approach (taking $M = 0$), defined in (3.31). We consider the two implementations of the boundary conditions for the multiplicative approach: the strong and the weak BC, as presented in Section 3.4.2. The results are presented in Figure 3.7 by varying the mesh size h, considering $N \in \{16, 32, 64, 128, 256\}$ as presented in Notation 3.17.

In Figure 3.7, we see that the enriched approaches seem to give better results than standard FEM except for the multiplicative approach with strong imposition of boundary conditions. Moreover, this approach, which imposes $p_h^\times = 1$ on Γ, does not follow the expected convergence order (i.e. for $k = 1$, the expected convergence order in the L^2 norm is 2, compared to 1 obtained numerically). The additive approach seems much less effective here than in the previous experiment of Section 3.5.2, whereas the multiplicative approach with weak BC seems to significantly improve the results

obtained with standard FEM. Nevertheless, these two approaches appear to respect the expected convergence order in the L^2 norm, namely 2 for polynomials of order $k = 1$. A comparative study of the different methods is given in Section 3.5.3.2. In addition, we see that the standard FEM with polynomial order $k = 2$ is clearly less accurate than the multiplicative approach using weak BC applied with polynomial order $k = 1$.



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.7.: Considering the *1D elliptic problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) with $k = 1$ and $k = 2$, the additive approach $e_{h,+}^{(1)}$ (dashed lines) with $k = 1$ and the multiplicative approach $e_{h,M}^{(1)}$ (dotted lines) with $k = 1$, considering strong and weak BC, for $\boldsymbol{\mu}^{(1)}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

3.5.3.2. Comparison of different approaches

We now focus on the second parameter $\boldsymbol{\mu}^{(2)}$. We first look at the PINN prediction for this parameter and its derivatives in Figure 3.8. As in the previous section, we consider the following approaches: standard FEM, the additive approach and the multiplicative approach (with $M = 0$) with strong or weak imposition of boundary conditions. We compare the different methods in Table 3.8, where we can see the different errors obtained with the considered methods for $k = 1$ and $N \in \{16, 32\}$ as well as the gains obtained in comparison with standard FEM. Next, we take a closer look at the solutions obtained with the different approaches in Figure 3.9; for each method, we compare the solution obtained (u_h for standard FEM, u_h^+ for the additive approach and $u_h^\times$ for the multiplicative approach, with strong or weak BC imposition) with the analytical solution u. For the enriched methods, using the PINN prior, we also compare the proposed correction; namely, for the additive approach, we compare p_h^+ with $u - u_\theta$ and for the multiplicative one $p_h^\times$ with u/u_θ (with $u_\theta > 0$ in Ω).

In Figure 3.8, we can see that PINN has difficulties to capture the solution and that the prediction it provides is far from the analytical solution. As for its derivatives, they seem to be relatively inaccurate compared to the analytical. Indeed, since the PINN is a smooth function, it has trouble approximating functions with very sharp gradients such as the one of (3.37).

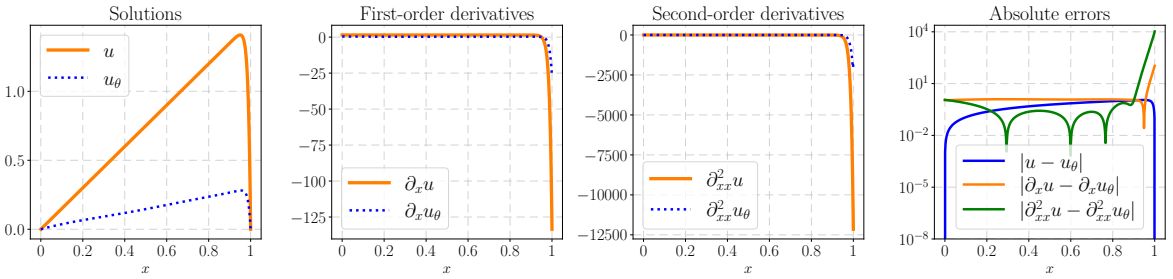
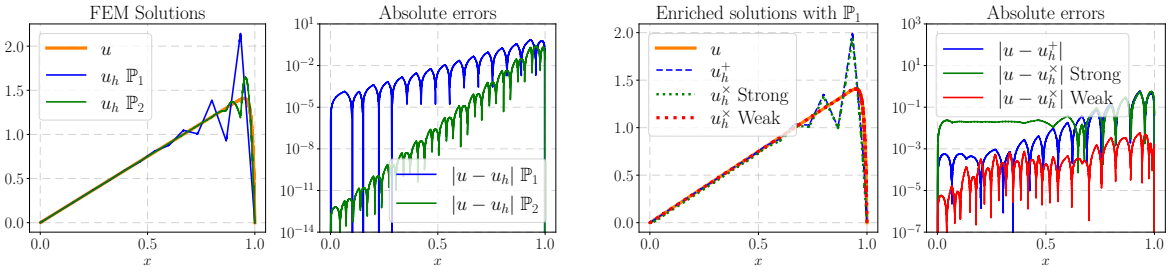


Figure 3.8.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$ and the PINN prior u_θ. Comparison between the analytical solution and the network prediction. From left to right – solution; first derivative; second derivative; errors.

	FEM
N	**error**
16	$1.85 \cdot 10^{-1}$
32	$6.31 \cdot 10^{-2}$

		PINN prior u_θ	
method	**N**	**error**	**gain**
Add	16	$1.47 \cdot 10^{-1}$	1.26
	32	$5 \cdot 10^{-2}$	1.26
Mult (strong)	16	$1.49 \cdot 10^{-1}$	1.24
	32	$6.65 \cdot 10^{-2}$	0.95
Mult (weak)	16	$1.6 \cdot 10^{-3}$	115.82
	32	$8.57 \cdot 10^{-4}$	73.7

Table 3.8.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$, $k = 1$, $N \in \{16, 32\}$ and the PINN prior u_θ. Left – L^2 relative error obtained with the standard FEM. Right – L^2 relative errors and gains with respect to the standard FEM obtained with the additive approach and the multiplicative approach with $M = 0$ (strong and weak BC).



(a) Standard FEM solutions ($k \in \{1, 2\}$) and absolute errors

(b) Enriched solutions ($k = 1$) and absolute errors

Figure 3.9.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$, $N = 16$ and the PINN prior u_θ. Comparison of the solutions obtained with the standard FEM, the additive approach and the multiplicative approach with $M = 0$ (strong and weak BC) with the analytical solution. For each enriched approach, comparison of the correction term with the analytical one.

In Figure 3.9 and Table 3.8, we present a comparison of the different approaches proposed. In Figure 3.9a, we first notice the oscillations anticipated in Remark 3.16 for standard FEM at both polynomial orders $k = 1$ and $k = 2$. This behavior is also seen in the additive enrichment (blue dashed line in Figure 3.9b), which does not seem to give better results than standard FEM due to the derivatives presented in Figure 3.8, and for the multiplicative approach with strong boundary conditions (green dotted line in Figure 3.9b). However, weakly imposing the boundary coniditions gives the appropriate results (red dotted line in Figure 3.9b). It seems that for this specific test case, the prediction obtained (close to a multiplicative constant of the solution) favors the multiplicative approach over the additive one.

More specifically, since the accuracy of enriched approaches is completely dependent on the prior, it is not clear in advance which approach (additive or multiplicative) yields the best results. However, this strong dependence means that we can intuitively choose the enriched approach based on our knowledge of PINNs. For example, as explained above, since PINNs are regular functions, they have difficulty approximating functions with very sharp gradients, as in this test case. Thus, knowing the nature of the test case in question, we can get an idea of the quality of the prior and therefore estimate which approach should be more compatible with it. To go further, if we want a more precise idea of the approach to focus on, we could, after training, calculate the theoretical gain constants using a coarse FEM as a reference solution, for example, which in a parametric approach could be seen as an offline cost. However, there is no theoretical guarantee that the constants exhibit the same behavior for different parameters, particularly depending on the type of parameter considered.

3.5.3.3. Gains achieved with the additive and the multiplicative approaches

Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances, we evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34) with the PINN prior u_θ. We evaluate $G_{M,\theta}$ and G_M, defined in (3.35), in the same way with $M = 0$ for the multiplicative approach, by considering the two implementations of the boundary conditions; strong and weak BC. The results are presented in Table 3.9 for fixed $k = 1$ and $N \in \{20, 40\}$ fixed.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
method	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
Add	20	7.01	93.15	17.97	16.59	1.2	52.18	3.35	6.57
	40	21.19	363.77	68.7	69.83	1.2	46.32	3.29	6.17
Mult (strong)	20	5.29	19.77	9.35	2.51	0.33	13.95	1.58	1.76
	40	10.48	39.89	18.9	4.48	0.16	6.74	0.87	0.84
Mult (weak)	20	69.49	6 247.43	1 479.77	1 094.37	65.32	359.24	186.72	77.12
	40	262.57	24 907.73	4 423.73	3 674.41	54.45	252.57	148.83	48.35

Table 3.9.: Considering the *1D elliptic problem*, $k = 1$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach and the multiplicative approach with $M = 0$ (strong and weak BC) with respect to the PINN prior. Right – Same with respect to the standard FEM.

Table 3.9 confirms the above results. The multiplicative approach with weak BC seems to give the best results on our set $\mathcal{D}$ of parameter instances. The additive and multiplicative approaches with strong BC imposition do not appear to be very effective on this test case. In particular, even though the additive approach improves the standard FEM error by a factor of 3, we have seen in Section 3.5.3.2 that the solutions obtained do not correspond to the expected solution, whereas the multiplicative approach with low BCs does. In the following, we will only consider the additive approach, as it seems to be the most efficient one, except in special cases such as the one under consideration in this Section 3.5.3. Indeed, the following test cases will not contain boundary layers or strong gradients.

3.5.4. 1D non-smooth transmission problem

Let $\Omega = (0, \pi)$ and Γ its boundary. In this test case, the physical properties of the medium are discontinuous with respect to space, and we define the interface point $I := \frac{\pi}{2}$ at the middle of the domain. We consider the following one-dimensional non-smooth transmission problem:

$$\begin{cases} -D\partial_{xx}u = f, & \text{in } \Omega \setminus \{I\} \times \mathcal{M}, \\ u = 0, & \text{on } \Gamma \times \mathcal{M}, \\ \lim\limits_{x \to I^+} D(x, \boldsymbol{\mu})\partial_x u(x, \boldsymbol{\mu}) = \lim\limits_{x \to I^-} D(x, \boldsymbol{\mu})\partial_x u(x, \boldsymbol{\mu}), & \boldsymbol{\mu} \in \mathcal{M}, \\ \lim\limits_{x \to I^+} u(x, \boldsymbol{\mu}) = \lim\limits_{x \to I^-} u(x, \boldsymbol{\mu}), & \boldsymbol{\mu} \in \mathcal{M}, \end{cases}$$

where the diffusion coefficient D is defined by:

$$D(x, \boldsymbol{\mu}) := \begin{cases} D_1, \text{ if } x \in [0, I], \\ D_2, \text{ if } x \in (I, \pi], \end{cases}$$

with $\boldsymbol{\mu} = (D_1, D_2) \in \mathcal{M} = [2.5, 3.5] \times [0.5, 1.5]$ (we have $p = 2$ parameters). We prescribe a family of exact solutions (depending on the parameter vector) defined by:

$$u(x, \boldsymbol{\mu}) = \begin{cases} 1/D_1 \, \sin(2x), \text{ if } x \in (0, I), \\ 1/D_2 \, \sin(2x), \text{ if } x \in (I, \pi), \end{cases}$$

and we deduce the associated right-hand side $f(x, \boldsymbol{\mu}) = 4 \sin(2x)$.

At first, we work in a non-parametric framework, focusing on the parameter $\boldsymbol{\mu}^{(1)} = (3, 1)$. Therefore, in Section 3.5.4.2, we first present the error estimates obtained on this parameter, considering three different priors (whose construction is detailed in Section 3.5.4.1). The first one (denoted by u_θ) is a PINN with a generic MLP architecture. The second one (denoted by u_θ^{data}) is constructed from data on the solution and its derivatives (obtained from the analytical solution). For the third one (denoted by u_θ^{sing}), we use a PINN with the architecture proposed in [125], specifically designed to better capture the behavior of the solution around the interface by enriching the MLP with a singular function. The aim is to show that for this type of problem, using an architecture adapted to the problem leads to better results than a generic

one. More specifically, we show that a naive network is not suited to the enriched approach we propose. However, by enriching the network in a similar way that classical finite element methods are enriched to capture known discontinuities, we obtain results similar to the other test cases considered, for low values of the polynomial degree k. To do this, we focus in Section 3.5.4.3 on the predictions and derivatives of the different non-parametric priors and analyze the results of the enriched approach with the three priors. Finally, in Section 3.5.4.4, we return to our parametric framework by constructing the enriched prior u_θ^{sing} in a parametric way (considering $\boldsymbol{\mu} \in \mathcal{M}$) and evaluate the gains obtained with it on a sample of parameters.

Notation 3.18. *As in the previous sections, the characteristic mesh size is $h = \frac{1}{N-1}$, where N is the number of nodes considered. For the finite element methods, we choose to consider that the interface I lies on a mesh node (by taking odd values of N) to avoid integration issues.*

3.5.4.1. Construction of the different priors

Here, we discuss the differences in training the three non-parametric networks (used in Sections 3.5.4.2 and 3.5.4.3) for the specific parameter $\boldsymbol{\mu}^{(1)}$. We also add the construction of the parametric network u_θ^{sing} (considered in Section 3.5.4.4). For PINNs, we do not strongly impose the boundary conditions (i.e. with a level-set function), but rather include them in the loss function.

Non-parametric physics-informed training with classical architecture. The first non-parametric prior u_θ is a PINN with a generic MLP architecture. Since the problem is not defined on I, we can simply consider that the integral of the residual loss is divided between the two subdomains (to the left and right of I). By applying an independent Monte Carlo method, we obtain the same approximate residual loss $\mathcal{J}_r$ defined in (2.40) (considering $N_{\text{col}} = 3\,000$ collocation points uniformly chosen on $\Omega \setminus \{I\}$). Since the boundary conditions are not strongly imposed in this network, we also include the boundary loss $\mathcal{J}_b$ defined in (2.41). It should be noted that, by definition of the network in question, the continuity condition is automatically validated, but the flow condition cannot be satisfied. In practice, the considered prior aims to be accurate on the residual of transmission problem, while the solution itself is very poorly represented. The hyperparameters used are given in Table 3.10.

Non-parametric data-driven training with classical architecture. For the second non-parametric prior u_θ^{data}, a network is trained only on the data (constructed from the analytical solution), considering the same architecture as above. In Section 3.5.2, we have already seen that a network based on data is not sufficient to correctly capture the derivatives of the solution. Hence, based on [125], we define the following modified data loss function, including information on the derivatives:

$$\mathcal{J}_{\text{data}}(\theta) := \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| u_\theta^{\text{data}}\left(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}^{(1)}\right) - u\left(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}^{(1)}\right) \right|^2 + \left| \partial_x u_\theta^{\text{data}}\left(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}^{(1)}\right) - \partial_x u\left(\boldsymbol{x}_{\text{data}}^{(i)}, \boldsymbol{\mu}^{(1)}\right) \right|^2,$$

with $N_{\text{data}} = 3\,000$ points uniformly chosen on $\Omega \setminus \{I\}$. The hyperparameters used are also given in Table 3.10.

Network - MLP	
layers	$20, 20, 20$
σ	tanh

Training			
lr	1e-3	N_{epochs}	2 000
decay	0.99		
$N_{\text{col/data}}$	3 000		

Loss weights					
u_θ		u_θ^{data}		u_θ^{sing}	
ω_r	1	ω_{data}	1	ω_r	1
ω_b	20			ω_b	20
				ω_{int}	1

Table 3.10.: Network, training parameters (Notation 3.8) and loss weights for non-parametric priors u_θ, u_θ^{data} and u_θ^{sing} in the *1D transmission problem*. N_{col} denotes the number of collocation points for the PINN priors and N_{data} the number of data for the data prior. Weights not specified in the table are defined to be zero.

Non-parametric physics-informed training with enriched architecture. The third non-parametric prior u_θ^{sing} is a PINN with an enriched architecture specifically designed to capture the behavior of the solution around the interface. This architecture was proposed in [125] under the name ReCoNNs (Regularity-Conforming Neural Networks). More precisely, we define the non-parametric network as follows:

$$u_\theta^{\text{sing}}(x, \boldsymbol{\mu}) := w_\theta^0(x, \boldsymbol{\mu}) + w_\theta^1(x, \boldsymbol{\mu}) \frac{|x - I|}{2}, \tag{3.38}$$

with $w_\theta = (w_\theta^0, w_\theta^1)$ a classic MLP and $\boldsymbol{\mu} = \boldsymbol{\mu}^{(1)}$. This architecture allows us to capture the discontinuity of the first derivative of the solution at the interface I. The network is trained in a physics-informed way, minimizing the same residual loss function $\mathcal{J}_r$ as for the first prior and the same boundary loss function $\mathcal{J}_b$. The continuity condition is automatically satisfied by the construction of the network. For the flow condition, we then add the interface loss $\mathcal{J}_{\text{int}}$ defined by:

$$\mathcal{J}_{\text{int}}(\theta) := \left| \lim_{x \to I^+} D(x, \boldsymbol{\mu}^{(1)}) \partial_x u_\theta^{\text{sing}}(x, \boldsymbol{\mu}^{(1)}) - \lim_{x \to I^-} D(x, \boldsymbol{\mu}^{(1)}) \partial_x u_\theta^{\text{sing}}(x, \boldsymbol{\mu}^{(1)}) \right|^2,$$

where the limits of network derivatives at the interface are calculated analytically, as detailled in [125]. The minimization problem then becomes:

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \left(\omega_r \mathcal{J}_r(\theta) + \omega_b \mathcal{J}_b(\theta) + \omega_{\text{int}} \mathcal{J}_{\text{int}}(\theta) \right),$$

with ω_r, ω_b and ω_{int} some weights to balance the different terms of the loss function. The hyperparameters used are still given in Table 3.10.

Parametric physics-informed training with enriched architecture. We consider the same enriched architecture as above, but in a parametric way, for Section 3.5.4.4. Thus,

we define the parametric network as defined in (3.38), considering $\boldsymbol{\mu} \in \mathcal{M}$ and the modified interface loss function:

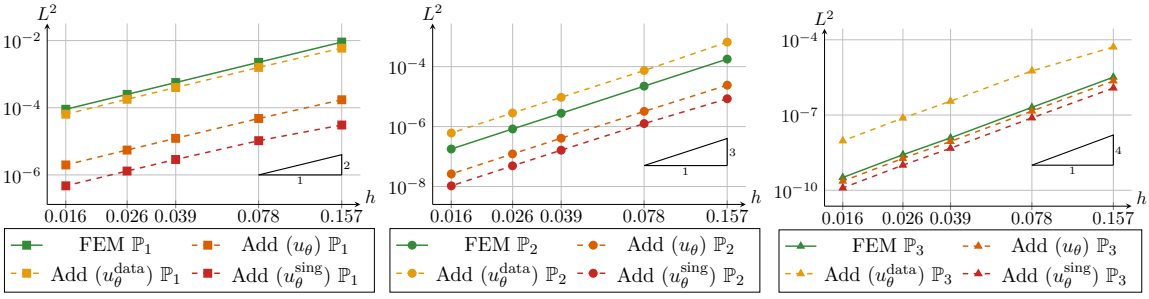
$$\mathcal{J}_{\text{int}}(\theta) := \frac{1}{N_{\text{col}}} \sum_{i=1}^{N_{\text{col}}} \left| \lim_{x \to I^+} D(x, \boldsymbol{\mu}_{\text{col}}^{(i)}) \partial_x u_\theta^{\text{sing}}(x, \boldsymbol{\mu}_{\text{col}}^{(i)}) - \lim_{x \to I^-} D(x, \boldsymbol{\mu}_{\text{col}}^{(i)}) \partial_x u_\theta^{\text{sing}}(x, \boldsymbol{\mu}_{\text{col}}^{(i)}) \right|^2 .$$

The hyperparameters used for the network and training are the same as for the non-parametric PINN prior with enriched architecture (defined in Table 3.10), except for the number of epochs set to $n_{epochs} = 10\,000$ and the initial learning rate set to $lr = 1.8 \cdot 10^{-2}$.

3.5.4.2. Error estimates – with the three non-parametric priors

We perform the same test as in the previous sections, considering only the additive approach with the three non-parametric priors presented in Section 3.5.4.1. The results in L^2 norm are presented for $\boldsymbol{\mu}^{(1)}$ in Figure 3.10 for fixed $k \in \{1, 2, 3\}$ and by varying the number of nodes $N \in \{21, 41, 61, 81, 101\}$ (Notation 3.18).

The results of the enriched approach are obtained by calculating the source term weakly, meaning only using the first derivatives of the prior. Also, like in the other test cases, the derivatives are calculated analytically using `PyTorch`. However, in the case of the particular architecture used in u_θ^{sing}, the derivatives produced are discontinuous, which must be taken into account during integration. More precisely, using an odd number of nodes N (as stated in Notation 3.18), two cells of the mesh intersect at the interface I. Using a Galerkin discontinuous space to represent the first derivative of the prior, we define the value at I of the left (resp. right) cell by its value at $I - \epsilon$ (resp. $I + \epsilon$) with $\epsilon := 10^{-6}$. This way, we ensure that the correct value of the derivative is used on each side of the interface during integration.



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$

Figure 3.10.: Considering the *1D transmission problem* with $\boldsymbol{\mu}^{(1)}$. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid line) and the additive approach $e_{h,+}^{(1)}$ (dashed lines), with $k = 1$, by considering the PINN prior with classic architecture u_θ, the data-driven prior u_θ^{data} and the PINN with enriched architecture u_θ^{sing} (all in a non-parametric way). Middle – Same with $k = 2$. Right – Same with $k = 3$.

The results presented in Figure 3.10 show that the additive approach with the PINN prior and the enriched architecture u_θ^{sing} provides the best results. In particular, we observe that for polynomial order $k = 1$, the errors obtained with PINNs are much lower

than those obtained with the data network, and that the gain compared to standard FEM is significant. Furthermore, we also observe that the convergence rates are similar for all three priors and match those of the classical FEM. This confirms that using a network architecture adapted to the problem yields better results in our enriched finite element method. On the other hand, we can see that the data-driven network seems to deteriorate the results of the standard approach, particularly for $k \neq 1$, which is a behavior we did not observe elsewhere. As this test case does not enter into the theoretical framework of Section 3.1, we cannot assert the causes of this; further study is warranted. However, from what we have seen so far, we have observed that the more we increase the polynomial order, the more the order of the derivatives that control the error increases. If the reasoning is similar here, a network trained solely by data could produce such a result. To better understand these results, we now analyze the predictions and derivatives of the three priors in Section 3.5.4.3.

3.5.4.3. Derivatives – with the three non-parametric priors

To better explain the results of Section 3.5.4.2 for selected parameter $\boldsymbol{\mu}^{(1)}$, we compare the solution, the first- and second-order derivatives between the exact solution and the prediction of the three priors constructed in Section 3.5.4.1. For analytical second derivatives, they are calculated independently on the left and right intervals. The results are presented in Figure 3.11.

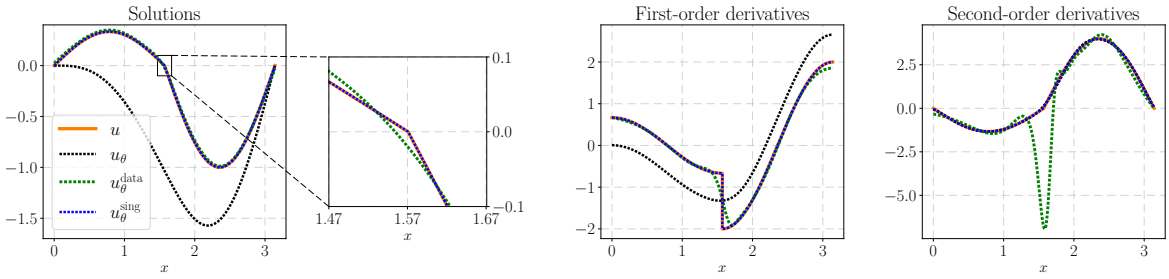


Figure 3.11.: Considering the *1D transmission problem* with $\boldsymbol{\mu}^{(1)}$ and the three non-parametric priors: the PINN prior with classic architecture u_θ (black), the data-driven prior u_θ^{data} (green) and the PINN with enriched architecture u_θ^{sing} (blue). Comparison between the analytical solution (orange) and the network prediction. From left to right – solution; zoom on solution at interface; first derivative; second derivative.

In Figure 3.11, we see that the prior that gives the best approximation of the solution is the PINN with the singular architecture, followed by the data-driven network, which seems to smooth the solution at the interface somewhat. The standard PINN, on the other hand, has great difficulty capturing the solution. In terms of first derivatives, we see that the enriched PINN is also very successful, as is the case for the solution. The data-driven network seems to give a smooth approximation of the first derivative, while the standard PINN still seems quite far off. However, the data-driven network has the most difficulty approximating second derivatives, followed by the standard PINN, while the singular network approximates them very well. In view of the results of Section 3.5.4.2, it would therefore seem that it is the higher-order derivatives that have

the greatest impact on the results, even if the prior itself is not optimal. This conclusion also holds true in other test cases where our theoretical framework is applicable.

3.5.4.4. Gains achieved with the additive approach – with the parametric enriched PINN prior

Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances and the parametric enriched PINN prior u_θ^{sing} defined in Section 3.5.4.1, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34). The results are presented in Table 3.11 for $k \in \{1, 2, 3\}$ and $N \in \{21, 41\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	21	37.43	185.85	101.87	40.6	32.41	270.99	139.17	41.77
	41	141.02	629.45	383.64	133.39	33.64	231.89	132.2	35.7
2	21	211.26	3 555.64	1 033.67	643.81	16.72	34.59	25.08	4.7
	41	1 057.29	25 483.75	5 719.05	4 470.3	13.52	19.16	16.56	1.54
3	21	770.08	28 142.92	4 569.45	4 825.81	1.48	2.68	1.79	0.23
	41	11 038.01	354 298.53	58 440.14	60 029.7	1.2	2.02	1.45	0.22

Table 3.11.: Considering the *1D transmission problem*, $k \in \{1, 2, 3\}$ and the parametric PINN prior with enriched architecture u_θ^{sing}. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

The results in Table 3.11 confirm the results obtained for $\boldsymbol{\mu}^{(1)}$ in Section 3.5.4.2. It would appear that the additive approach with the enriched PINN prior u_θ^{sing} provides significant gains, particularly for $k = 1$. For $k = 3$, the minimum obtained is greater than 1, so we see that, unlike when using the data-driven prior, the results of the standard method do not appear to be degraded. On the other hand, as the degree of the polynomial increases, the average gains appear to decrease, and quite considerably so.

3.5.5. 2D Poisson problem in a square domain

We now consider the problem of Section 3.5.2 but in two dimensions ($d = 2$), with:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega \times \mathcal{M}, \\ u = 0, & \text{on } \Gamma \times \mathcal{M}, \end{cases} \tag{3.39}$$

with Δ the Laplace operator on the domain $\Omega = (-0.5\pi, 0.5\pi)^2$ with boundary Γ, and $\mathcal{M} \subset \mathbb{R}^p$ the parameter space (with p the number of parameters). We define the right-hand side f such that the solution is given by:

$$u(\boldsymbol{x}, \boldsymbol{\mu}) = \exp\left(-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2}{2}\right) \sin(\kappa x_1) \sin(\kappa x_2), \tag{3.40}$$

with $\boldsymbol{x} = (x_1, x_2) \in \Omega$ and some parameters $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathcal{M} = [-0.5, 0.5]^p$, hence with $p = 2$ parameters. With an abuse of language as well, we refer to the quantity κ in (3.40) as the frequency of the solution, in the sense that it characterizes the number of oscillations in the solution.

We start with a "low frequency" case in Section 3.5.5.1, taking $\kappa = 2$ and considering a PINN where we impose the Dirichlet boundary conditions as presented in Section 2.3.2, i.e. using a level-set function. To further improve the prior quality, we introduce an augmented loss function in Section 3.5.5.2 by using the Sobolev training presented in Section 2.3.2. Afterwards, we test another loss in Section 3.5.5.3 that includes the Dirichlet condition, or in other words, that does not use a level-set function. Finally, we consider a "higher frequency" case in Section 3.5.5.4, with $\kappa = 8$.

Notation 3.19. *In the following, the characteristic mesh size $h = \frac{\pi\sqrt{2}}{N-1}$ is defined as a function of N, considering a cartesian mesh of N^2 nodes for our squared 2D domain of length π.*

3.5.5.1. Low-frequency case

We consider a "low-frequency" problem, taking $\kappa = 2$. In this section, we consider the additive approach, as presented in Section 3.1, by considering the PINN prior u_θ. We start by testing the error estimates in (in L^2 norm and H^1 semi-norm) with polynomial order $k \in \{1, 2, 3\}$, then we compare the different approaches. We evaluate the gains obtained on a sample of parameters. Then, we compare the numerical costs of the different methods. Finally, we discuss the importance of integrating analytical functions, as presented in Section 3.4.1.

Since the problem under consideration is parametric we deploy a parametric PINN, which depends on both the space variable $\boldsymbol{x} = (x_1, x_2) \in \Omega$ and the parameters $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathcal{M}$. Moreover, we strongly impose the Dirichlet boundary conditions, as presented in Section 2.3.2. To do this, we use the prior u_θ defined in (2.42), where we choose the level-set function φ defined by:

$$\varphi(\boldsymbol{x}) = (x_1 + 0.5\pi)(x_1 - 0.5\pi)(x_2 + 0.5\pi)(x_2 - 0.5\pi).$$

Thus, we only consider the residual loss $\mathcal{J}_r$ approached by a Monte Carlo method as defined in (2.40) with $N_{\text{col}} = 6\,000$ collocation points uniformly chosen on $\Omega \times \mathcal{M}$. The parametric network is defined as an MLP with the hyperparameters defined in Table 3.12; we use the Adam optimizer and then switch to the LBFGS optimizer after the n_{switch}-th epoch.

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	sine

Training - with LBFGS			
lr	1.7e-2	N_{epochs}	5 000
decay	0.99	n_{switch}	1 000
N_{col}	6 000		

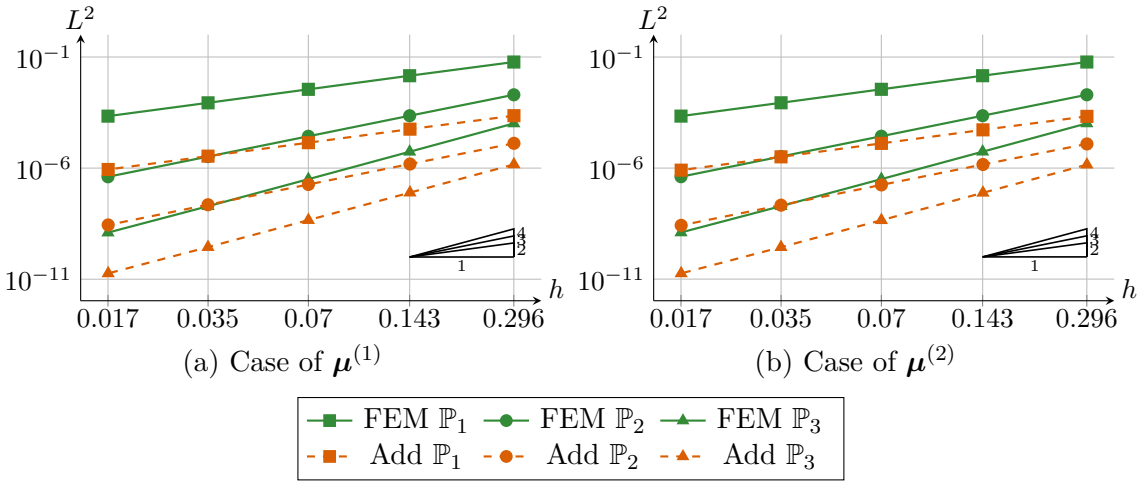
Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0

Table 3.12.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *2D low-frequency Poisson problem.*

Error estimates. We start by testing the error estimates of Theorem 3.1 for the following two sets of parameters, randomly selected in $\mathcal{M}$:

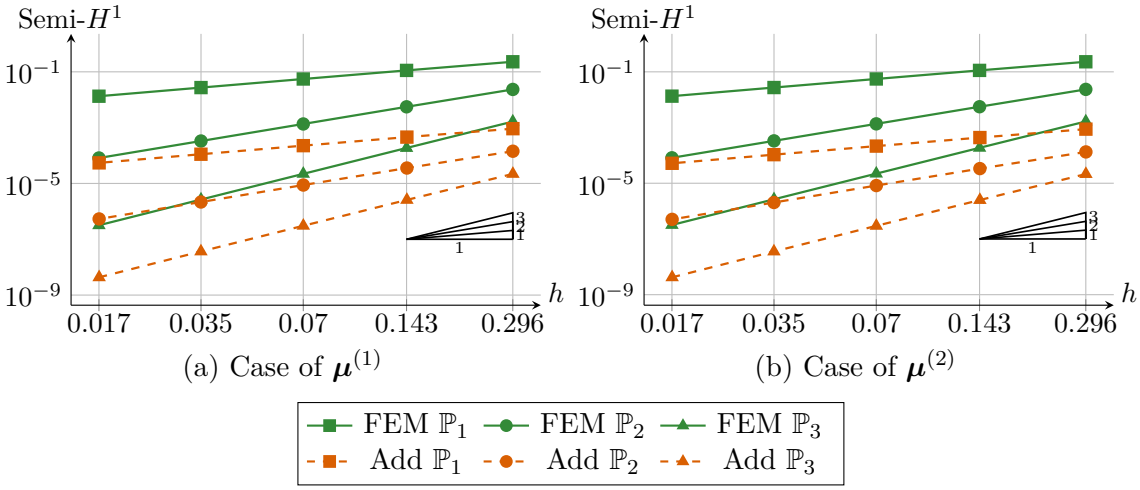
$$\boldsymbol{\mu}^{(1)} = (0.05, 0.22), \quad \text{and} \quad \boldsymbol{\mu}^{(2)} = (0.1, 0.04)\ ,$$

by considering the PINN prior u_θ. We perform the same tests as in the previous sections, but this time considering only the additive approach. The results are presented in Figures 3.12 and 3.13 (respectively in norm L^2 and semi-norm H^1) for fixed $k \in \{1, 2, 3\}$ with h depending on $N \in \{16, 32, 64, 128, 256\}$, as presented in Notation 3.19.



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.12.: Considering the *2D low-frequency Poisson problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k \in \{1, 2, 3\}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.13.: Considering the *2D low-frequency Poisson problem* and the PINN prior u_θ. Left – Semi-H^1 relative error as a function of h, obtained with the standard FEM (solid lines) and the additive approach (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k \in \{1, 2, 3\}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

In Figures 3.12 and 3.13, we observe the expected behavior. Indeed, the error decreases with the correct order of accuracy as the mesh size h decreases (i.e. with a

slope of $k+1$ in the L^2 norm and k in the H^1 semi-norm). This observation is valid for both the classical and enriched FEM. Moreover, we observe that the error constant of the additive approach is significantly lower than that of the classical FEM. In the following paragraph, we will compare these different approaches in more detail. As noted in the 1D case, we can see that the additive enriched approach for $k = 1$ (resp. $k = 2$) seems to give relative errors close to the original FEM for $k = 2$ (resp. $k = 3$), although the rate of convergence is different.

Comparison of different approaches. We now focus on the first parameter $\boldsymbol{\mu}^{(1)}$. We compare the standard FEM with the additive approach, first in the Table 3.13 where we can see the different errors obtained with the different methods for $k = 1$ fixed and $N \in \{16, 32\}$ as well as the gains obtained in comparison with standard FEM. Next, we take a closer look at the solution obtained with the different approaches in Figure 3.14; for each method, we compare the solution obtained (u_h for standard FEM and u_h^+ for the additive approach) with the analytical solution u. For the enriched method, using the PINN prior u_θ, we also compare the proposed correction; namely, for the additive approach, we compare p_h^+ with $u - u_\theta$.

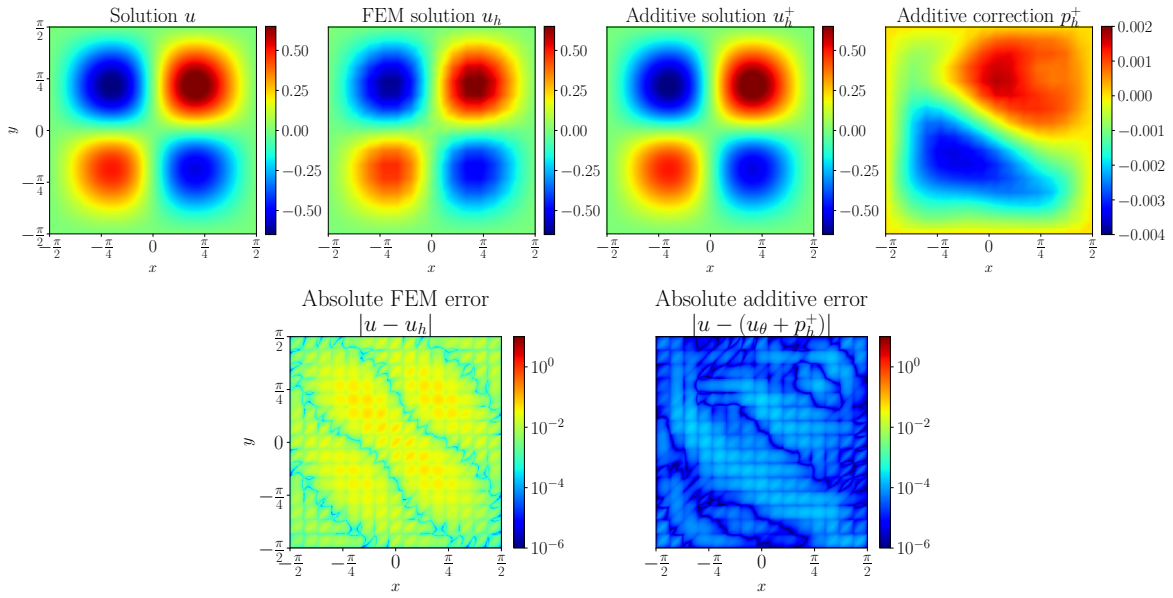


Figure 3.14.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N = 16$ and the PINN prior u_θ. Comparison of the solutions obtained with the standard FEM and the additive approach with the analytical solution. For the additive approach, comparison of the correction term with the analytical one.

In Table 3.13, we observe that the additive approach significantly improves the error of the standard FEM, with gains of around 260 for $N = 16$ and $k = 1$, which is equivalent to refining the mesh by a factor of 16 for $\mathbb{P}_1$ elements. Indeed, in this case, our enriched approach gives much better results than standard FEM. In Figure 3.14, we observe that the solution obtained with the additive approach is very close to the

analytical solution, with a correction term that is also very close to the analytical one. This shows the effectiveness of the additive approach in this case.

FEM		PINN prior u_θ			
N	**error**	**method**	**N**	**error**	**gain**
16	$5.95 \cdot 10^{-2}$	Add	16	$2.29 \cdot 10^{-4}$	260.38
32	$1.44 \cdot 10^{-2}$		32	$5.67 \cdot 10^{-5}$	253.66

Table 3.13.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N \in \{16, 32\}$ and the PINN prior u_θ. Left – L^2 relative error obtained with the standard FEM. Right – L^2 relative errors and gains with respect to the standard FEM obtained with the additive approach.

Gains achieved with the additive approach. Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34). The results are presented in Table 3.14 for $k \in \{1, 2, 3\}$ and $N \in \{20, 40\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				Gains in L^2 rel error of our method w.r.t. FEM			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	20	15.69	48.33	33.62	5.56	134.32	377.36	269.39	43.66
	40	61.4	195.53	135.25	23.18	131.2	362.12	262.13	41.68
2	20	244.05	993.21	653.07	153.16	67.02	164.65	134.85	21.29
	40	2 055.16	8 340.94	5 501.88	1 286.49	66.45	159.48	131.86	20.34
3	20	2 780.52	11 705.79	7 542.95	1 766.24	39.52	72.65	61.55	7.02
	40	50 649.24	211 156.91	136 714.04	31 882.69	39.83	72.62	61.67	6.87

Table 3.14.: Considering the *2D low-frequency Poisson problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

In Table 3.14, we observe (left subtable) that our method significantly improves the error of the PINN, especially for large values of k, where the enrichment is performed in a richer approximation space. Moreover, we also observe (right subtable) significant gains with respect to classical FEM. For instance, as expected from the results of the previous paragraph, the mean gains for $k = 1$ are around 270, which corresponds to refining the mesh approximately 16 times for $\mathbb{P}_1$ elements. This means that our $\mathbb{P}_1$ enhanced bases capture the solution as accurately as classical $\mathbb{P}_1$ bases with a mesh sixteen times finer. For $k = 2$ and $k = 3$, the mean gains are around 134 and 61, respectively, which corresponds to refining the mesh approximately 5 times for $\mathbb{P}_2$ elements and 2.8 times for $\mathbb{P}_3$ elements. A natural follow-up question consists in assessing the impact of the PINN quality on our results. This will be the subject of the Section 3.5.5.2.

Costs of the different methods. To more accurately assess the benefits of using the enriched methods, we look in this section at the costs of the different methods proposed, considering the parameter $\boldsymbol{\mu}^{(1)}$. Thus, we consider that the cost of using the PINN prior u_θ, corresponds to the total number of weights of the network considered. In this case, it is given as an MLP with the hyperparameters defined in Table 3.12, for a total of $N_\theta = 12\,461$ weights.

For the different finite element methods, we endeavor to determine, for a fixed polynomial degree k, the characteristic mesh size h (depending on N as described in Notation 3.19) required to reach a given fixed error e. In Table 3.15, we study, for $k \in \{1, 2, 3\}$, considering standard FEM and the additive approach, the N required to achieve the same error e. More precisely, the characteristic mesh size required by standard FEM so that $e_h^{(1)} \approx e$ and the one required by the additive approach so that $e_{h,+}^{(1)} \approx e$. Depending on the polynomial degree k, we can also determine the number of degrees of freedom N_{dofs} associated with each case. These results are obtained by interpolating the convergence curves of Figure 3.12 for the different methods for a given e.

		N		$N_{\mathbf{dofs}}$	
k	**e**	**FEM**	**Add**	**FEM**	**Add**
1	$1 \cdot 10^{-3}$	119	8	14 161	64
	$1 \cdot 10^{-4}$	379	24	143 641	576
2	$1 \cdot 10^{-4}$	42	8	6 889	225
	$1 \cdot 10^{-5}$	89	17	31 329	1 089
3	$1 \cdot 10^{-5}$	28	10	6 724	784
	$1 \cdot 10^{-6}$	48	18	20 164	2 704

Table 3.15.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Characteristic N (associated with the characteristic mesh size h) required to reach a fixed error e for the standard FEM and the additive approach. Right – Number of degrees of freedom N_{dofs} associated with each case.

In Table 3.15, we see that the additive approach proposed in Section 3.1 requires a much coarser mesh than standard FEM to achieve the same e error. This is due to the error estimations of Theorem 3.1 which show that the error of the enhanced FEM is significantly lower than that of the classical FEM (depending on the quality of the prior). This is also reflected in the number of degrees of freedom required to achieve the same error e.

However, the enriched approaches proposed require using the prior PINN u_θ, which also includes its inference cost. For this reason, it is also interesting to study these same costs on a set of parameters, say of size $N_\mu = 100$. Since we are in the context of parametric PINN, we can estimate that the computational cost of solving (3.39) on this sample of N_μ parameters corresponds, for the additive approach, to N_μ times its number of DOFs plus the cost of using PINN (i.e. its total number of weights), thus $N_\mu \times N_{\text{dofs}} + N_\theta$ (with N_{dofs} the number of DOFs associated to the additive approach).

For standard FEM, this cost is equivalent to N_μ times its estimated number of DOFs $N_\mu \times N_{\text{dofs}}$ (with N_{dofs} the number of DOFs associated with standard FEM). We then compare these costs for a set of $N_\mu = 100$ parameters, considering the same error e to be achieved for both methods. The results are presented in Table 3.16. Note that these results (right subtable) are, in fact, only an estimate of the real cost of solving N_μ problems. In practice, the number of degrees of freedom N_{dofs} associated with each method depends on the parameter itself. The error to be achieved e requires more or less fine meshes for each parameter.

		$N_\mu = 1$		$N_\mu = 100$	
k	**e**	**FEM**	**Add**	**FEM**	**Add**
1	$1 \cdot 10^{-3}$	14 161	12 525	1 416 100	18 861
	$1 \cdot 10^{-4}$	143 641	13 037	14 364 100	70 061
2	$1 \cdot 10^{-4}$	6 889	12 686	688 900	34 961
	$1 \cdot 10^{-5}$	31 329	13 550	3 132 900	121 361
3	$1 \cdot 10^{-5}$	6 724	13 245	672 400	90 861
	$1 \cdot 10^{-6}$	20 164	15 165	2 016 400	282 861

Table 3.16.: Considering the *2D low-frequency Poisson problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Total costs of the standard FEM and the additive approach to reach an error e for a set of $N_\mu = 1$ parameter. Right – Same for a set of $N_\mu = 100$ parameters.

In Table 3.16, for $N_\mu = 1$ (left subtable), we can see that the cost of the additive method is generally lower than that of standard FEM, even though they are of the same range. However, it is important to note that these are not entirely comparable: in fact, a large part of the cost of the additive method lies in PINN prediction, which is more or less well estimated depending on a number of hyper-parameters (number of epochs, learning rate, etc.). If we then take $N_\mu = 100$ (right subtable), we can see that the cost of standard FEM becomes radically higher than with the additive approach. This is why the improved approach is particularly interesting for solving the (3.39) problem on a set of parameters.

Integration of analytical functions. This section aims to discuss one of the important points, presented in Section 3.4.1, that enables PINN to be used effectively and can make our enriched methods more or less effective. Indeed, according to (3.3), we have to integrate $f + \Delta u_\theta$ multiplied by the test function. To perform this integration, we first interpolate this term on a polynomial space and then integrate it exactly. The degree of this polynomial approximation is an important parameter to make our technique effective.

The goal here is simply to show that for enriched approaches to be effective, particularly the additive method, this polynomial approximation must be of a sufficiently high degree. Consider the parameter $\boldsymbol{\mu}^{(1)}$, a polynomial degree $k = 3$ and a number of nodes N^2 with $N = 128$. In Figure 3.15, we display the L^2 relative error of the additive approach with respect to the degree of polynomial approximation of $f + \Delta u_\theta$.

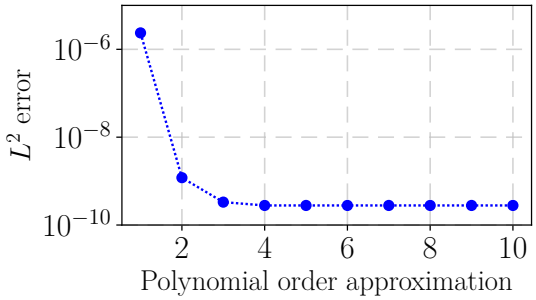


Figure 3.15.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N = 128$ and the PINN prior u_θ. L^2 relative error $e^{(1)}_{h,+}$ of the additive approach as a function of the polynomial degree used to approximate $f + \Delta u_\theta$.

In Figure 3.15, we observe that the error decreases as the degree of polynomial approximation increases. This shows the importance of properly interpolating analytical functions in the context of enriched methods.

3.5.5.2. Low-frequency case – Sobolev training

This section focuses on the same problem as in Section 3.5.5.1. The aim here is to show that the network quality has a non-negligible impact on the results obtained with our method and that if the network is better, our results are, too. To this end, we defined a new prior u_θ^{sob} by using the Sobolev training presented in Section 2.3.2, where the derivatives of the solution should be better approximated than by a standard training and compared it with the PINN prior u_θ defined in Section 3.5.5.1. We start by testing the error estimation with $k \in \{1, 2, 3\}$ polynomial order and evaluate the gains obtained on the same sample of parameters as in Section 3.5.5.1.

We deploy here a parametric PINN, denoted by u_θ^{sob}, where we consider the residual loss $\mathcal{J}_r$ and the Sobolev loss $\mathcal{J}_{\text{sob}}$, respectively defined in (2.37) and (2.44), whose integrals are both approximated by a Monte Carlo method with $N_{\text{col}} = 6\,000$ collocation points. We then solve the minimization problem defined in (2.43) considering that the Dirichlet boundary conditions are strongly imposed as in Section 3.5.5.1. The hyperparameters are defined in Table 3.17; we use the Adam optimizer.

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	sine

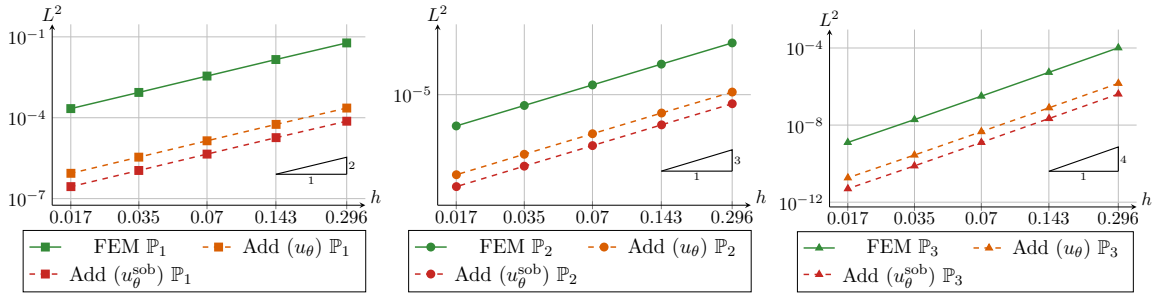
Training - with LBFGS			
lr	1.7e-2	N_{epochs}	3 000
decay	0.99	batch size	2 000
N_{col}	6 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0.1

Table 3.17.: Network, training parameters (Notation 3.8) and loss weights for u_θ^{sob} (Sobolev training) in the *2D low-frequency Poisson problem*.

Remark 3.20. *Adding the Sobolev loss can make training more difficult, so we only consider* 3 000 *epochs but a batch size of* 2 000*. This means that for each epoch, the weights are updated 3 times (because* $N_{col} = 6\,000$*).*

Error estimates. For simplicity, we consider the first parameter $\boldsymbol{\mu}^{(1)} = (0.05, 0.22)$ presented in Section 3.5.5.1. We perform the same test as in the previous section, considering the two priors, u_θ and u_θ^{sob}, in the enriched approach. The results are presented in Figure 3.16 for fixed $k \in \{1, 2, 3\}$ (in L^2 norm). We observe that Sobolev training improves the results obtained with the L^2 training, for $k \in \{1, 2, 3\}$. This shows the impact of the quality of the network prediction on our method. To further investigate this, we evaluate the gains obtained with the Sobolev training on the same sample of parameters as in Section 3.5.5.1.



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$

Figure 3.16.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid line) and the additive approach $e_{h,+}^{(1)}$ (dashed lines), with $k = 1$, by considering the PINN prior with standard training u_θ and Sobolev training u_θ^{sob}. Middle – Same with $k = 2$. Right – Same with $k = 3$.

Gains achieved with the additive approach. Considering the same set $\mathcal{D}$ of $N_\mu = 50$ parameter instances as in Section 3.5.5.1, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34) considering the PINN prior u_θ^{sob} using Sobolev training. The results are presented in Table 3.18 for $k \in \{1, 2, 3\}$ and $N \in \{20, 40\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	20	16.14	80.38	41.1	15.04	188.69	1 085.32	768.22	200.13
	40	63.93	324.11	166.45	61.46	187.82	1 051.55	751.01	190.72
2	20	263.88	1 751.13	896.03	394.57	147.08	536.32	414.91	78.1
	40	2 235.97	14 849.11	7 600.79	3 349.18	146.35	528.57	408.6	76.94
3	20	3 040.5	22 972.38	10 662.78	5 092.35	91.84	244.09	192.96	32.23
	40	55 299.93	416 438.72	194 157.02	92 537.38	93.16	246.78	194.31	32.51

Table 3.18.: Considering the *2D low-frequency Poisson problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ^{sob} (Sobolev training). Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

The gains reported in Table 3.18 show that, compared to L^2 training, Sobolev training increases the mean gains by a factor of about 3. This corresponds to almost half an additional mesh refinement for the $\mathbb{P}_1$ elements. We also note that this Sobolev training is particularly interesting for higher polynomial degrees, with standard L^2 training having lower gains than for $k = 1$.

3.5.5.3. Low-frequency case – Boundary loss training

In this section, we focus on the same problem as in Section 3.5.5.1 and Section 3.5.5.2. We now turn to a standard PINN, denoted by u_θ^{bc}, where we impose the boundary conditions in the loss function (no longer with the level-set function). The aim here is to show that our enriched methods also work with priors that do not have exact boundary conditions. To this end, we start by testing the error estimation and evaluate the gains obtained on the same sample of parameters as in Section 3.5.5.1.

We deploy a new parametric PINN, denoted u_θ^{bc}, where we consider the residual loss function $\mathcal{J}_r$ and the boundary loss function $\mathcal{J}_b$, respectively defined in (2.40) and (2.41). These integrals are both approximated by a Monte Carlo method considering $N_{\text{col}} = 6\,000$ collocation points and $N_{\text{bc}} = 2\,000$ boundary collocation points. We insist that the collocation points are resampled at each epoch during the training process. Here, we no longer strongly impose the Dirichlet boundary conditions. The hyperparameters are defined in Table 3.19; we use the Adam optimizer and then switch to the LBFGS optimizer after the n_{switch}-th epoch. The training loss curves are presented in Figure 3.17 for both priors u_θ and u_θ^{bc}.

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	sine

Training - with LBFGS			
lr	1.7e-2	N_{epochs}	5 000
decay	0.99	n_{switch}	1 000
N_{col}	6 000	N_{bc}	2 000

Loss weights			
ω_r	1	ω_{data}	0
ω_b	30	ω_{sob}	0

Table 3.19.: Network, training parameters (Notation 3.8) and loss weights for u_θ^{bc} (BC loss training) in the *2D low-frequency Poisson problem*.

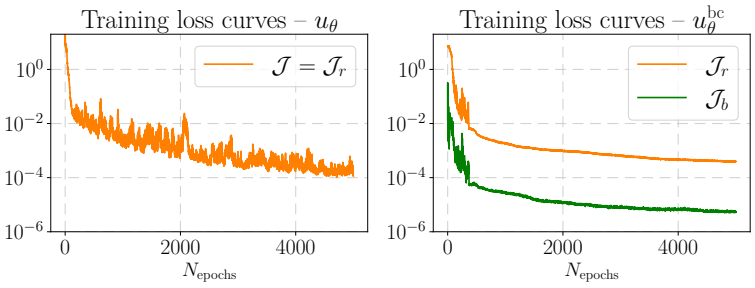
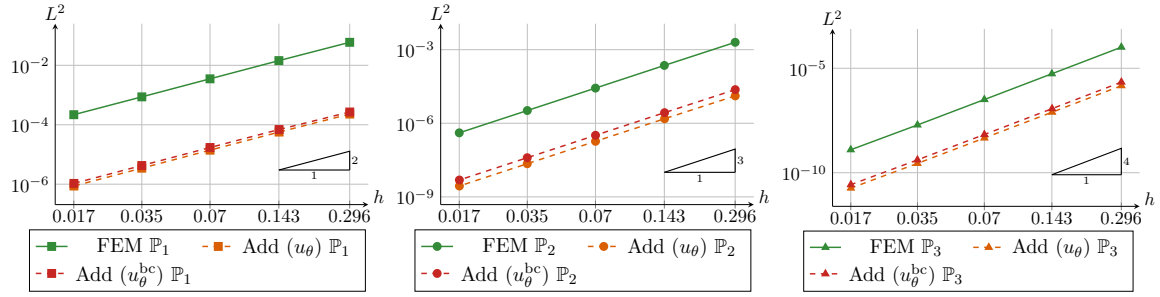


Figure 3.17.: Considering the *2D low-frequency Poisson problem*. Training loss curves. Left – PINN prior u_θ with strong boundary conditions (Section 3.5.5.1). Right – PINN prior u_θ^{bc} with BC loss training (Section 3.5.5.3).

Error estimates. We consider the first parameter $\boldsymbol{\mu}^{(1)} = (0.05, 0.22)$ presented in Section 3.5.5.1 and perform the same test as before, considering the two priors u_θ and u_θ^{bc}. The results are presented in Figure 3.18 for fixed $k \in \{1, 2, 3\}$ (in L^2 norm). We can see in Figure 3.18 that the additive approach also works when the prior is not exact on the boundary, as here with u_θ^{bc}. In particular, for $k \in \{1, 2, 3\}$ and the parameter $\boldsymbol{\mu}^{(1)}$, our enriched approach using the prior u_θ^{bc} seems to give very similar results to those obtained with u_θ even if the approach with level-set is, for this case, slightly better.



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$

Figure 3.18.: Considering the *2D low-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid line) and the additive approach $e_{h,+}^{(1)}$ (dashed lines), with $k = 1$, by considering the PINN prior with standard training u_θ and the BC loss training u_θ^{bc}. Middle – Same with $k = 2$. Right – Same with $k = 3$.

Gains achieved with the additive approach. Considering the same set $\mathcal{D}$ of $N_\mu = 50$ parameter instances as in Section 3.5.5.1, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34) considering the PINN prior u_θ^{bc} using BC loss training. The results are presented in Table 3.20 for $k \in \{1, 2, 3\}$ and $N \in \{20, 40\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	20	21.37	54.17	34.72	6.45	81.13	246.75	168.01	45.92
	40	85.88	220.61	139.04	26.75	80.46	236.88	162.51	43.23
2	20	356.19	1 248.67	579.58	179.56	48.74	91.6	70.04	12.06
	40	3 046.74	10 658.21	4 959.71	1 534.67	48.42	90.91	69.6	11.98
3	20	5 100.32	16 798.35	8 398.62	2 524.2	27.54	52.09	40.41	6.78
	40	92 448.59	308 340.35	152 537.79	46 156.7	27.93	52.21	40.55	6.75

Table 3.20.: Considering the *2D low-frequency Poisson problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ^{bc} (BC loss training). Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

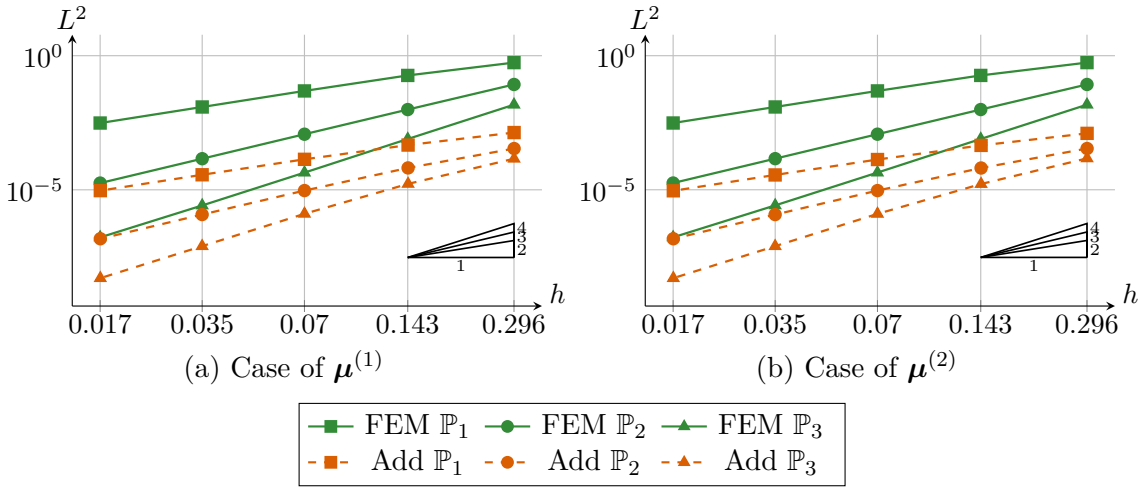
The gains reported in Table 3.20 show that for this test case, the use of the prior u_θ

(using the level-set) in our enriched approach, seems to give better gains than those of Table 3.14, considering the current prior u_θ^{bc}. This may be due to the addition of the ω_{bc} hyperparameter for balancing losses in training, which may make training less efficient. However, the results obtained with the current prior u_θ^{bc} are still very good, and the gains are still significant compared to the standard FEM.

3.5.5.4. High-frequency case

To increase in complexity, we investigate a higher-frequency problem by taking $\kappa = 8$ in (3.40). In this section, we start by testing the error estimates. Then, we compare the different methods and evaluate the gains obtained on a sample of parameters. This time, we use the Fourier features from [123] as presented in Section 2.3.2 to construct the PINN prior u_θ. The hyperparameters are defined in Table 3.21; we use the Adam optimizer and then switch to the LBFGS optimizer after the n_{switch}-th epoch. We consider $N_{\text{col}} = 6\,000$ collocation points, uniformly chosen on Ω. We impose the Dirichlet boundary conditions as in Section 3.5.5.1 using the level-set function and the same residual loss.

Error estimates. We perform the same test as in Section 3.5.5.1, with a standard training (Sobolev training is not considered). The results are displayed in Figure 3.19, where we observe the expected behavior. Indeed, all schemes have the correct order of accuracy, and the enhanced FEM has a significantly lower error constant than the classical FEM.



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.19.: Considering the *2D high-frequency Poisson problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k \in \{1, 2, 3\}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

Comparison of different approaches. Considering the first parameter $\boldsymbol{\mu}^{(1)}$, we perform the same comparison as in Section 3.5.5.1 for the high-frequency Poisson problem. The results are presented in Table 3.22 and Figure 3.20.

Network - MLP w/ FF	
layers	$40, 60, 60, 60, 40$
σ	sine
n_f	40

Training - with LBFGS			
lr	1.7e-2	N_{epochs}	20 000
decay	0.99	n_{switch}	1 000
N_{col}	6 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0

Table 3.21.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *2D high-frequency Poisson problem*.

FEM	
N	**error**
16	$5.53 \cdot 10^{-1}$
32	$1.84 \cdot 10^{-1}$

PINN prior u_θ			
method	**N**	**error**	**gain**
Add	16	$1.36 \cdot 10^{-3}$	405.23
	32	$4.66 \cdot 10^{-4}$	394.18

Table 3.22.: Considering the *2D high-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N \in \{16, 32\}$ and the PINN prior u_θ. Left – L^2 relative error obtained with the standard FEM. Right – L^2 relative errors and gains with respect to the standard FEM obtained with the additive approach.

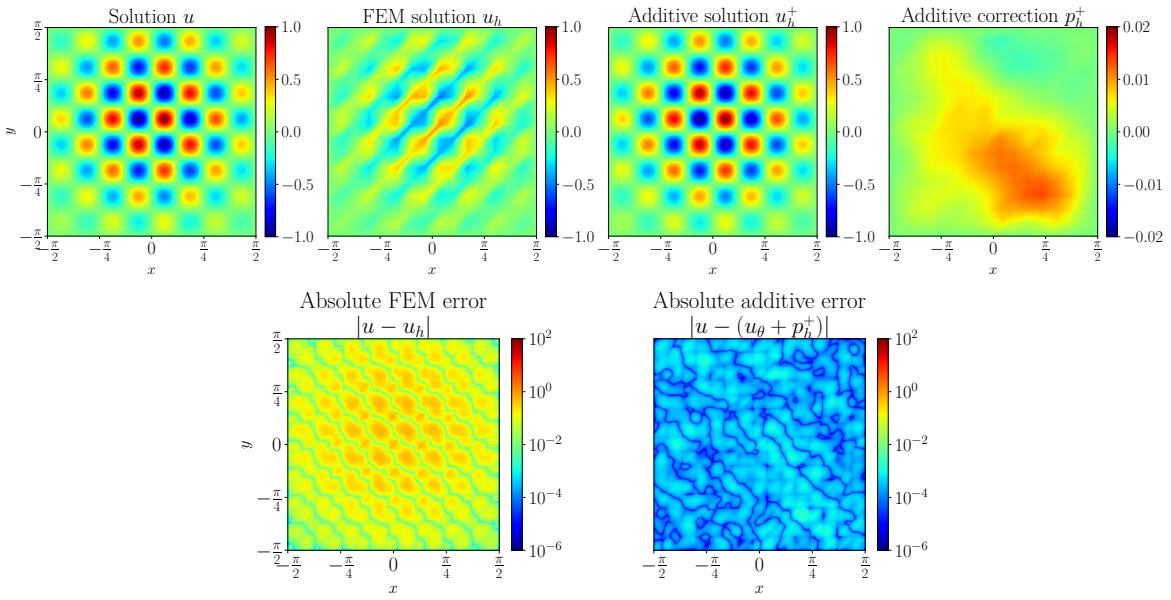


Figure 3.20.: Considering the *2D high-frequency Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $N = 16$ and the PINN prior u_θ. Comparison of the solutions obtained with the standard FEM and the additive approach with the analytical solution. For the additive approach, comparison of the correction term with the analytical one.

We can see here that the gains obtained in Table 3.22 are much better than for the "low frequency" case presented in Section 3.5.5.1. This is, in fact, due to FEM's difficulty in approximating the solution for high frequencies, especially on coarse meshes.

In fact, for the same choice of parameters, the FEM error on this high-frequency problem is 10 times worse than on the low-frequency one, which explains why our gains are so much greater. This also makes the use of the proposed enriched methods particularly interesting. Moreover, we note that while FEM provides a reasonable approximation of the mean of the solution (as evidenced by the second figure on the top row of Figure 3.20), it is unable to correctly resolve the small-scale oscillating behavior of the solution. The additive correction restores this ability, and the new solution (third figure on the top row of Figure 3.20) is much better able to capture the oscillations.

Gains achieved with the additive approach. We now evaluate the gains $G_{+,\theta}$ and G_+, defined in (3.34), using the same set $\mathcal{D}$ of $N_\mu = 50$ parameter instances. The results are reported in Table 3.23 for $k \in \{1, 2, 3\}$ and $N \in \{20, 40\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	20	9.16	36.18	19.8	6.63	112.19	450.8	349.84	82.9
	40	26.1	111.31	58.78	19.77	106.01	388.91	308.44	71.79
2	20	35.89	164.85	87.44	29.21	65.44	210.59	159.41	38.96
	40	204.25	1 088.02	516.55	179.21	52.5	139.6	109.72	22.04
3	20	100.37	533.16	252.86	88.38	32.91	81.05	63	12.73
	40	971.7	6 165.56	2 592.38	978.71	19.8	41.1	33.53	4.98

Table 3.23.: Considering the *2D high-frequency Poisson problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

The same results can be observed in Table 3.23 as for the $\boldsymbol{\mu}^{(1)}$ parameter. These could be improved by considering a Sobolev training, as in the "low-frequency" case presented in Section 3.5.5.2.

3.5.6. 2D anisotropic elliptic problem on a square

In this section, we consider the (2.1) problem in a more complex form than in Section 3.5.5, by considering the following elliptic problem with homogeneous Dirichlet boundary conditions, in the 2D case ($d = 2$):

$$\begin{cases} -\text{div}(D\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases}$$

with $\Omega = (0, 1)^2$, Γ its boundary and $\mathcal{M} \subset \mathbb{R}^p$ the parameter space (with p the number of parameters). Considering $\boldsymbol{x} = (x_1, x_2) \in \Omega$, we define $p = 4$ parameters $\boldsymbol{\mu} = (\mu_1, \mu_2, \epsilon, \sigma) \in \mathcal{M} = [0.4, 0.6] \times [0.4, 0.6] \times [0.01, 1] \times [0.1, 0.8]$. We define the

(symmetric and positive definite) diffusion matrix D by:

$$D(\boldsymbol{x}, \boldsymbol{\mu}) := \begin{pmatrix} \epsilon x_1^2 + x_2^2 & (\epsilon - 1)x_1 x_2 \\ (\epsilon - 1)x_1 x_2 & x_1^2 + \epsilon x_2^2 \end{pmatrix},$$

and the right-hand side f by:

$$f(\boldsymbol{x}, \boldsymbol{\mu}) := \exp\left(-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2}{0.025\sigma^2} \right).$$

Note that the matrix D has eigenvalues $x_1^2 + x_2^2$ and $\epsilon(x_1^2 + x_2^2)$, leading to a diffusion process whose anisotropy increases as ϵ decreases.

In this section, we consider the following three sets of parameters:

$$\boldsymbol{\mu}^{(1)} = (0.51, 0.54, 0.52, 0.55), \quad \boldsymbol{\mu}^{(2)} = (0.48, 0.53, 0.41, 0.89),$$
$$\text{and} \quad \boldsymbol{\mu}^{(3)} = (0.46, 0.52, 0.05, 0.12).$$

In Section 3.5.6.1, we start by testing the error estimates of Theorem 3.1. Then, we compare the two different approaches in Section 3.5.6.2. Finally, we evaluate the gains achieved with the additive approach in Section 3.5.6.3.

Notation 3.21. *In the following, the characteristic mesh size $h = \frac{\sqrt{2}}{N-1}$ is defined as a function of N, considering a cartesian mesh of N^2 nodes.*

We consider a parametric PINN where we exactly impose the Dirichlet boundary conditions as presented in Section 2.3.2. Thus, we construct u_θ as in (2.42) with the level-set function φ defined by:

$$\varphi(\boldsymbol{x}) = x_1(x_1 - 1)x_2(x_2 - 1).$$

Since we impose the boundary conditions by using the level-set function, we only consider the residual loss $\mathcal{J}_r$ approached by a Monte Carlo method as defined in (2.40) with $N_{\text{col}} = 8\,000$ collocation points uniformly chosen on $\Omega \times \mathcal{M}$. The hyperparameters are given in Table 3.24; we use the Adam optimizer [83].

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	tanh

Training			
lr	1.6e-2	N_{epochs}	15 000
decay	0.99		
N_{col}	8 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0

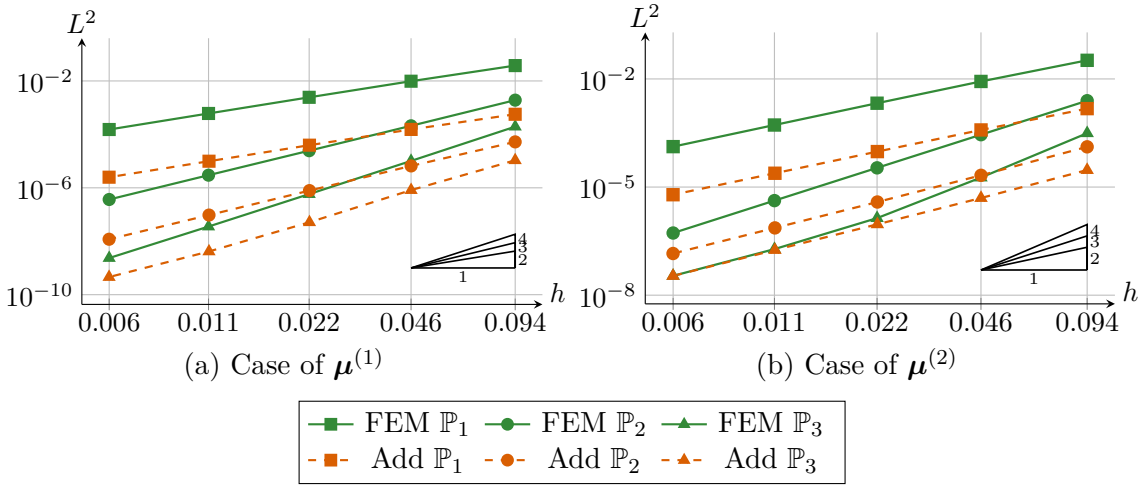
Table 3.24.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *2D elliptic problem.*

Remark 3.22. *Here, we do not know the analytical solution associated with the problem under consideration. So, in order to analyze the results obtained, we define u as a reference solution u_{ref} obtained from a FEM solver on an over-refined mesh of characteristic mesh size h_{ref} and with k_{ref} polynomial order. In this section,*

we set $N_{ref} = 1\,000$ (and the associated characteristic mesh size h_{ref}, as defined in Notation 3.21) and $k_{ref} = 3$. The linear systems resulting from the finite element discretization is solved using the Conjugate Gradient iterative method (tolerance: 10^{-13}, maximum number of iterations: $10\,000$) preconditioned by the algebraic multigrid (AMG).

3.5.6.1. Error estimates

We first test the error estimates (Theorem 3.1) for the following two sets of parameters, randomly chosen from $\mathcal{M}$: $\boldsymbol{\mu}^{(1)}$ and $\boldsymbol{\mu}^{(2)}$. We perform the same tests as in the Section 3.5.5, considering only the additive approach. The results are presented in Figure 3.21 for a fixed $k \in \{1, 2, 3\}$ with $N \in \{16, 32, 64, 128, 256\}$, as presented in Notation 3.21.



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.21.: Considering the *2D elliptic problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k \in \{1, 2, 3\}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

As in the other test cases, the two approaches tested appear to respect the correct slopes of Theorem 3.1. The additive approach seems to be more efficient than the standard FEM for polynomial orders $k \in \{1, 2, 3\}$ and for the two sets of parameters considered.

3.5.6.2. Comparison of different approaches

We perform the same comparison as in Section 3.5.5.1 for this 2D elliptic problem. We focus on the third parameter $\boldsymbol{\mu}^{(3)}$ by taking a closer look at the solution obtained with the different approaches in Figure 3.22 considering $N = 16$ and $k = 2$.

We observe that the enriched FEM provides a more accurate solution compared to the standard FEM one. The results indicate that the additive approach is particularly effective in capturing the solution's finer details. This demonstrates its potential in solving anisotropic problems with higher accuracy than standard methods.

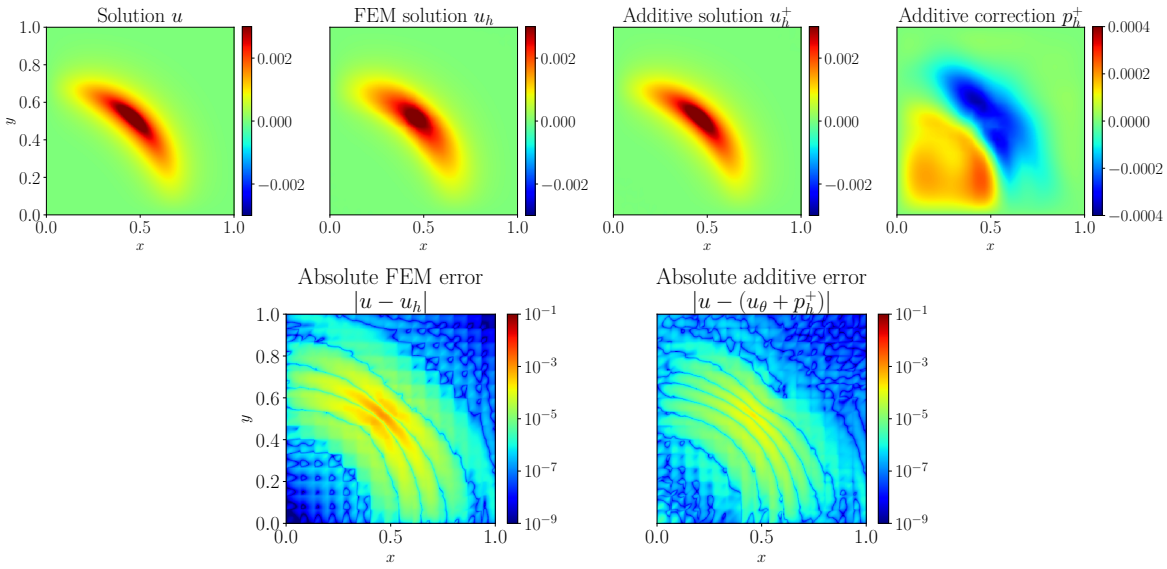


Figure 3.22.: Considering the *2D elliptic problem* with $\boldsymbol{\mu}^{(3)}$, $k = 2$, $N = 16$ and the PINN prior u_θ. Comparison of the solutions obtained with the standard FEM and the additive approach with the reference solution. For the additive approach, comparison of the correction term with the reference one.

3.5.6.3. Gains achieved with the additive approach

Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances, we evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34). The results are presented in Table 3.25 for $k \in \{1, 2, 3\}$ fixed and $N \in \{20, 40\}$ fixed.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	20	2.26	50.1	17.35	11.21	7.12	82.57	35.67	17.55
	40	4.84	185.73	62.52	43.15	6.8	77.55	33.91	16.36
2	20	15.03	623.17	213.18	143.68	3.54	35.88	18.32	7.95
	40	118.35	4 942.28	1 560	1 096.15	2.37	31.82	14.83	7.49
3	20	151.67	3 295.53	1 111.11	750.19	1.33	26.51	8.32	5.02
	40	1 939.42	62 146.51	14 375.23	12 088.42	1.03	19.8	5.77	3.86

Table 3.25.: Considering the *2D elliptic problem*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

As in the previous test cases, the additive approach seems to be more efficient than the standard FEM for the three polynomial orders $k \in \{1, 2, 3\}$ and for the two mesh sizes $N \in \{20, 40\}$. However, the gains obtained are less significant than in the previous test cases. This is due to the fact that the problem under consideration is more complex,

and thus, the prior u_θ is less accurate. Namely, PINNs have trouble converging on such highly anisotropic problems, see e.g. [133].

3.5.7. 2D Poisson problem on an annulus, with mixed boundary conditions

This section concerns the problem (2.1), considering the Poisson problem with mixed (Dirichlet and Robin) boundary conditions defined in two space dimensions ($d = 2$) by:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega \times \mathcal{M}, \\ u = g, & \text{on } \Gamma_E \times \mathcal{M}, \\ \dfrac{\partial u}{\partial n} + u = g_R, & \text{on } \Gamma_I \times \mathcal{M}, \end{cases}$$

with $\mathcal{M} \subset \mathbb{R}^p$ the parameter space (with p the number of parameters). We consider Ω to be an annulus centered at the origin, defined by the unit circle with a circular hole of radius 0.25. Or, in other words, Ω is defined by:

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^2 \;\middle|\; 0.25 < \sqrt{x_1^2 + x_2^2} < 1 \right\}.$$

We then define $\Gamma := \Gamma_I \cup \Gamma_E$ the boundary of Ω, with Γ_I the inner boundary (the hole) and Γ_E the outer boundary (the unit circle). We consider the analytical solution defined for all $\boldsymbol{x} = (x_1, x_2) \in \Omega$ by:

$$u(\boldsymbol{x}, \boldsymbol{\mu}) = 1 - \frac{\ln\left(\mu_1 \sqrt{x_1^2 + x_2^2}\right)}{\ln(4)},$$

with some parameters $\boldsymbol{\mu} = \mu_1 \in [2.4, 2.6]$ ($p = 1$ parameter), and the associated right-hand side $f = 0$. The Dirichlet condition g on Γ_E and the Robin condition g_R on Γ_I are defined by:

$$g(\boldsymbol{x}, \boldsymbol{\mu}) := 1 - \frac{\ln(\mu_1)}{\ln(4)}, \quad \text{and} \quad g_R(\boldsymbol{x}, \boldsymbol{\mu}) := 2 + \frac{4 - \ln(\mu_1)}{\ln(4)}.$$

These boundary conditions are thus parameter-dependent, contrary to the previous cases, which makes the problem more complex. To avoid geometric errors, we apply Remark 3.10, by considering that $g = u$ on $\Gamma_{E,h}$ and $g_R = \frac{\partial u}{\partial n}$ on $\Gamma_{I,h}$, with $\Gamma_{E,h}$ and $\Gamma_{I,h}$ the respective outer and inner boundaries of Ω_h, the domain covered by the (fitted) mesh. Note also that u_θ is not exact on these approximate boundaries.

In this section, we consider the additive approach, as presented in Section 3.1, by considering the PINN prior u_θ. We start by testing the error estimates in Section 3.5.7.1. Then, we compare the different approaches in Section 3.5.7.2 and evaluate the gains obtained in Section 3.5.7.3 on a sample of parameters.

Since the problem under consideration is parametric, we deploy a parametric PINN, which depends on both the space variable $\boldsymbol{x} = (x_1, x_2) \in \Omega$ and the parameters $\boldsymbol{\mu} = \mu_1 \in \mathcal{M}$. To improve the derivatives' quality, we consider the Sobolev training presented in Section 2.3.2. Moreover, we strongly impose the Dirichlet boundary

conditions, as explained in Section 2.3.2, by using the formulation proposed in [122]. To do this, we define the prior:

$$u_\theta := \frac{\varphi_E}{\varphi_E + \varphi_I^2}\left[w_\theta + \varphi_I\big(w_\theta - \nabla\varphi_I \cdot \nabla w_\theta - g_R\big)\right] + \frac{\varphi_I^2}{\varphi_E + \varphi_I^2} g + \varphi_E \varphi_I^2 w_\theta, \tag{3.41}$$

where w_θ is the neural network under consideration and φ_I and φ_E are respectively the signed distance functions to Γ_I and Γ_E defined by:

$$\varphi_I(\boldsymbol{x}) := \sqrt{x_1^2 + x_2^2} - 0.25, \quad \varphi_E(\boldsymbol{x}) := 1 - \sqrt{x_1^2 + x_2^2},$$

which cancels out exactly on Γ_I and Γ_E. Note that the level-sets considered are signed distance functions in this specific test case, which is not the case in the other test cases. In this test case, this is necessary because of the formulation proposed in (3.41) by [122].

In this case, we consider an MLP with 5 layers and a tanh activation function with the hyperparameters defined in Table 3.26; we use the Adam optimizer [83]. We consider the residual and Sobolev losses in the same way as in Section 3.5.5.2 with $N_{\text{col}} = 6\,000$ collocation points.

Network - MLP	
layers	$40, 40, 40, 40, 40$
σ	tanh

Training - with LBFGS			
lr	1e-2	N_{epochs}	4 000
decay	0.99	n_{switch}	3 000
N_{col}	6 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0.1

Table 3.26.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *2D Poisson problem on an annulus.*

3.5.7.1. Error estimates

We start by testing the error estimation of Theorem 3.1 for the following two sets of parameters, uniformly selected from $\mathcal{M}$:

$$\boldsymbol{\mu}^{(1)} = (2.51), \quad \text{and} \quad \boldsymbol{\mu}^{(2)} = (2.54)\ ,$$

by considering the PINN prior u_θ. We perform the same tests as in the previous sections, considering only the additive approach. The results are presented in Figure 3.23 for fixed $k \in \{1, 2, 3\}$.

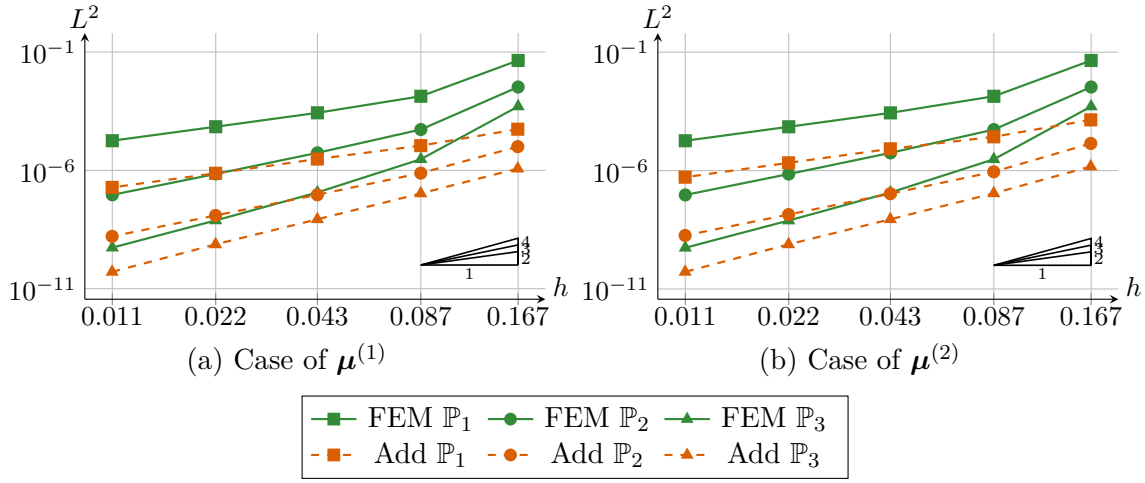
As expected, we see in Figure 3.23 that the error estimates are confirmed by the numerical results obtained with the standard FEM and the additive approach. The error decreases with the correct order of convergence for these two methods. Furthermore, the enriched approach provides a better accuracy than the standard FEM, as expected.

3.5.7.2. Comparison of different approaches

We perform the same comparison as in Section 3.5.5.1 for this Poisson problem on an annulus. We focus on the first parameter $\boldsymbol{\mu}^{(1)}$ by taking a closer look at the solution

obtained with the different approaches in Figure 3.24 considering $h \simeq 1.67 \cdot 10^{-1}$ and $k = 1$.

Once again, we observe that the enriched approach provides a significant improvement in accuracy compared to the standard FEM. This demonstrates the effectiveness of incorporating neural network priors in the case of mixed boundary conditions on more complex geometries than squares (here, on an annulus).



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.23.: Considering the *2D Poisson problem on an annulus* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k \in \{1, 2, 3\}$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

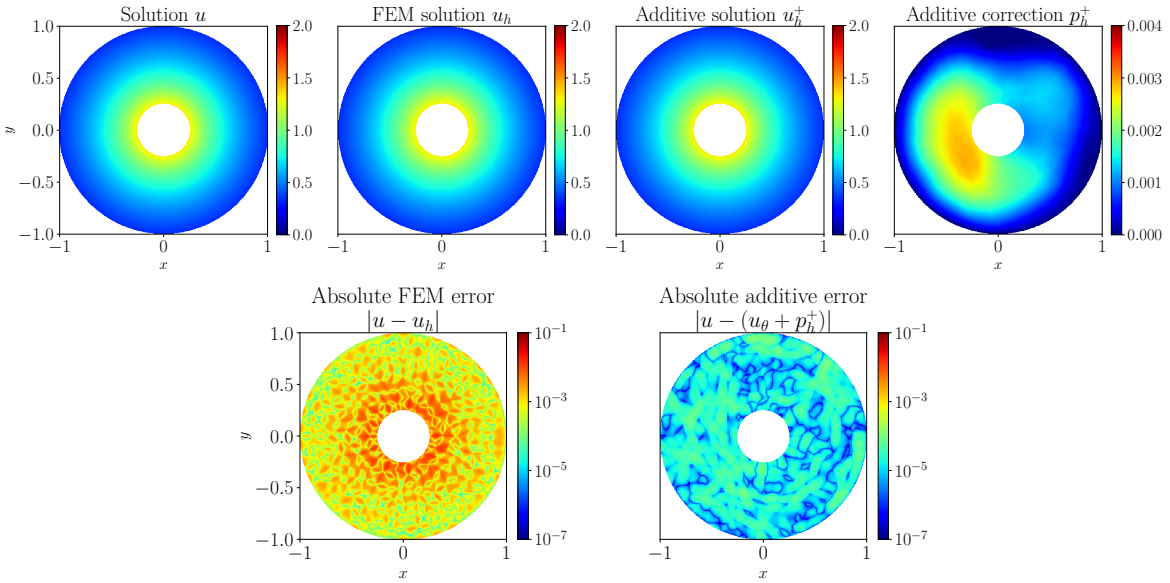


Figure 3.24.: Considering the *2D Poisson problem on an annulus* with $\boldsymbol{\mu}^{(1)}$, $k = 1$, $h \simeq 1.67 \cdot 10^{-1}$ and the PINN prior u_θ. Comparison of the solutions obtained with the standard FEM and the additive approach with the analytical solution. For the additive approach, comparison of the correction term with the analytical one.

3.5.7.3. Gains achieved with the additive approach

Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34). The results are presented in Table 3.27 for $k \in \{1, 2, 3\}$ and $h \in \{1.33 \cdot 10^{-1}, 6.90 \cdot 10^{-2}\}$.

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
k	**h**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
1	$1.33 \cdot 10^{-1}$	51.44	154.48	124.76	31.45	15.12	137.72	55.5	38.24
	$6.9 \cdot 10^{-2}$	186.44	569.39	460.53	118.34	14.35	124.91	51.72	34.68
2	$1.33 \cdot 10^{-1}$	551.64	5 688.06	3 401.69	1 674.34	31	77.46	58.41	15.46
	$6.9 \cdot 10^{-2}$	3 148.05	39 267.58	21 619.73	11 644.3	28.45	58.98	47.17	10.1
3	$1.33 \cdot 10^{-1}$	2 005.65	51 857	19 839.47	14 735.78	18.72	21.49	20.6	0.82
	$6.9 \cdot 10^{-2}$	29 270.47	681 417.4	281 187.12	197 765.09	17.92	19.85	19.33	0.57

Table 3.27.: Considering the *2D Poisson problem on an annulus*, $k \in \{1, 2, 3\}$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

As in previous sections, the PINN-enriched approach seems to give better results than standard FEM. For $k = 1$ we see a gain of 50 on average for this test case, which is equivalent to refining the mesh by a factor of 7 for $\mathbb{P}_1$ elements.

3.5.8. 3D Poisson problem in a cube domain

We now consider the problem (3.39) of Section 3.5.5.1 but in three dimensions ($d = 3$), in the space domain $\Omega = (-0.5\pi, 0.5\pi)^3$ and the parameter domain $\mathcal{M} = [-0.5, 0.5]^3$. We define the right-hand side f such that the solution is given for all $\boldsymbol{x} = (x_1, x_2, x_3) \in \Omega$ and $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \mathcal{M}$ by:

$$u(\boldsymbol{x}, \boldsymbol{\mu}) = \exp\left(-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2 + (x_3 - \mu_3)^2}{2}\right) \sin(2x_1) \sin(2x_2) \sin(2x_3).$$

In this section, we study the additive approach with $k = 1$. In Section 3.5.8.1, we start by testing the error estimates. Then, in Section 3.5.8.2, we conduct a study of the computation times of the different methods. In Section 3.5.8.3, we evaluate the gains achieved with the enriched approach. Finally, we discuss the importance of the prior quality in Section 3.5.8.4.

Notation 3.23. *In this section, the cube Ω of side length π is discretized using a Cartesian mesh with N^3 nodes. Consequently, the characteristic mesh size is defined as a function of N by $h = \frac{\pi\sqrt{3}}{N-1}$.*

We deploy the same type of parametric PINN as in the previous sections, where we strongly impose the Dirichlet boundary conditions. This PINN depends on both the spatial variable $\boldsymbol{x}$ and the parameters $\boldsymbol{\mu}$. We use the prior u_θ defined in (2.42), where we choose the level-set function φ defined by:

$$\varphi(\boldsymbol{x}) = (x_1 + 0.5\pi)(x_1 - 0.5\pi)(x_2 + 0.5\pi)(x_2 - 0.5\pi)(x_3 + 0.5\pi)(x_3 - 0.5\pi).$$

The residual loss function (2.40) is approximated with $N_{\text{col}} = 40\,000$ collocation points uniformly chosen in $\Omega \times \mathcal{M}$. The hyperparameters are defined in Table 3.28; we use the Adam optimizer and then switch to the LBFGS optimizer after the n_{switch}-th epoch.

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	tanh

Training - with LBFGS			
lr	1.7e-2	N_{epochs}	5 000
decay		n_{switch}	2 000
N_{col}	40 000		

Loss weights			
ω_r	1	ω_{data}	0
ω_b	0	ω_{sob}	0

Table 3.28.: Network, training parameters (Notation 3.8) and loss weights for u_θ in the *3D Poisson problem.*

Remark 3.24. *The results are obtained using the iterative conjugate gradient method (tolerance:* 10^{-8}*, maximum number of iterations:* 1 000*) preconditioned by HYPRE's BoomerAMG (a classical algebraic multigrid method) with a strong connection threshold of* 0.7 *and* 4 *levels of non-local aggregation using* 2 *paths (recommended options for the Poisson problem in* **FEniCSx**[3]*). For both FEM approaches, the right-hand side is interpolated on a* $\mathbb{P}_2$ *space. This means that for the additive approach, the derivatives of the network are evaluated analytically (using* `torch.autograd`*) on the degrees of freedom associated to the space* $\mathbb{P}_2$*. To give an order of magnitude, the evaluation of the second derivatives of the prior, on the mesh with* $N = 100$ *(i.e.* 7 880 599 *evaluation points for a* $\mathbb{P}_2$ *space) takes* 1.01 *seconds on an NVIDIA H100 graphics processing unit (GPU).*

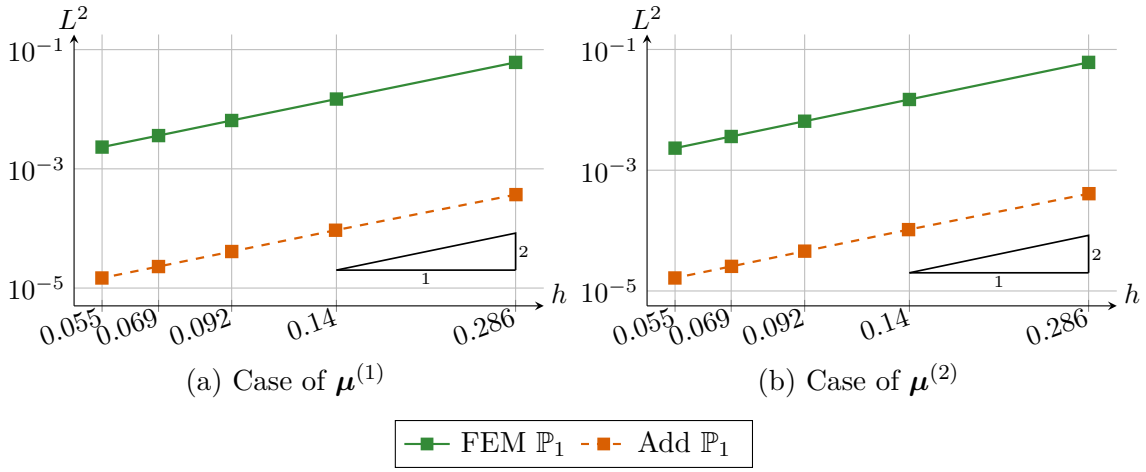
3.5.8.1. Error estimates

We start by testing the error estimates of Theorem 3.1 for the following two sets of parameters, randomly selected in $\mathcal{M}$:

$$\boldsymbol{\mu}^{(1)} = (0.05, 0.22, 0.1), \quad \text{and} \quad \boldsymbol{\mu}^{(2)} = (0.04, -0.08, 0.15),$$

by considering the PINN prior u_θ and by performing the same tests as in the previous sections. The results are presented in Figure 3.25 with h depending on $N \in \{20, 40, 60, 80, 100\}$, as presented in Notation 3.23. We make the same observations as in the previous sections, both in terms of convergence rate and gains.

[3] https://github.com/FEniCS/performance-test



(a) Case of $\boldsymbol{\mu}^{(1)}$ (b) Case of $\boldsymbol{\mu}^{(2)}$

Figure 3.25.: Considering the *3D Poisson problem* and the PINN prior u_θ. Left – L^2 relative error as a function of h, obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) for $\boldsymbol{\mu}^{(1)}$, with $k = 1$. Right – Same for $\boldsymbol{\mu}^{(2)}$.

3.5.8.2. Computation times of the different methods

Just like in Section 3.5.5.1, we are now interested in the numerical costs of the different approaches, but this time in terms of computation time. First, we perform a study in a non-parametric framework, considering only the parameter $\boldsymbol{\mu}^{(1)}$. Thus, in Figure 3.26, we examine the execution time of the two approaches (training time not included for the additive approach), comparing it to the relative L^2 error. In Table 3.29, we perform a study similar to the one done previously. More precisely, for a given error e, we look at the numerical cost of the two approaches that allow us to achieve it. Therefore, for a given e, we report the characteristic mesh size h, the number of degrees of freedom N_{dofs}, and the execution time. In this purely online context on the parameter $\boldsymbol{\mu}^{(1)}$, the times indicated include all the steps necessary to solve the FEM problem, i.e. mesh construction, system assembly, solving the linear system, and, for the additive approach, the cost of evaluating the derivatives of the prior.

However, the offline cost of the additive approach cannot be neglected, as it includes the training time of the prior (in this case: 707.84 seconds). This is why, in Figure 3.27, we are interested in the numerical costs of the two approaches in a parametric framework (since PINN is trained parametrically). To do this, we seek to determine, for a fixed error e, the number of parameter instances at which the additive approach is better than the standard approach, including the PINN training time. More precisely, the offline cost of the standard approach consists simply of constructing the mesh (of the size required to achieve e), while the additive approach includes the construction of the (coarser) mesh as well as the training time. For the online phase, both approaches include all the FEM steps (assembly and resolution of the system as well as the evaluation of derivatives for the enriched method). It should be noted that the online costs are much higher for the standard approach, due to a finer mesh size.

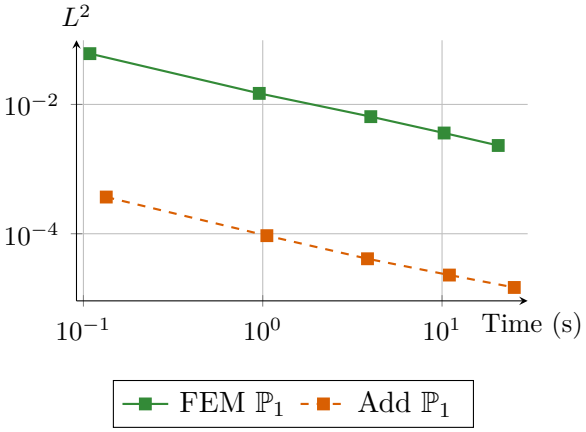


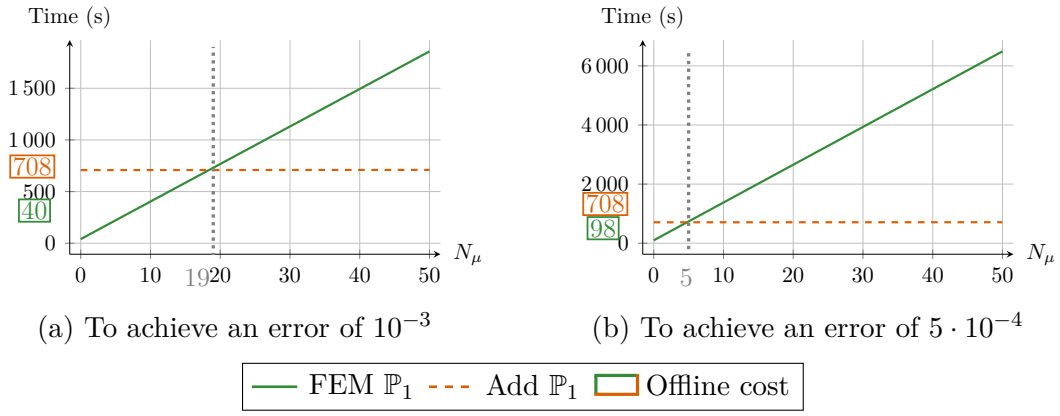
Figure 3.26.: Considering the *3D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$ and the PINN prior u_θ. L^2 relative error obtained with the standard FEM $e_h^{(1)}$ (solid lines) and the additive approach $e_{h,+}^{(1)}$ (dashed lines) as a function of the online computation time (in seconds), including mesh construction.

	N		N_{dofs}		**Computation time**	
e	**FEM**	**Add**	**FEM**	**Add**	**FEM**	**Add**
$1 \cdot 10^{-3}$	152	12	$3.51 \cdot 10^6$	$1.73 \cdot 10^3$	$(7.65 \cdot 10^1)$	$5.2 \cdot 10^{-2}$
$1 \cdot 10^{-4}$	484	39	$1.13 \cdot 10^8$	$5.93 \cdot 10^4$	$(2.8 \cdot 10^3)$	$9.26 \cdot 10^{-1}$
$1 \cdot 10^{-5}$	1 539	122	$3.65 \cdot 10^9$	$1.82 \cdot 10^6$	$(1.02 \cdot 10^5)$	$(5.26 \cdot 10^1)$

Table 3.29.: Considering the *3D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, $k = 1$ and the PINN prior u_θ. Left – Characteristic N (associated with the characteristic mesh size h) required to reach a fixed error e for the standard FEM and the additive approach. Middle – Number of degrees of freedom N_{dofs} associated with each case. Right – Execution time (in seconds) of the online phase associated with each case. The results in brackets are extrapolations provided to give an order of magnitude.

In Figure 3.26, we observe that for a given error, the additive approach is significantly faster than the standard FEM. This is confirmed in Table 3.29, where we can see that a coarser mesh can be used to achieve a given accuracy with our enriched FEM compared to the standard FEM, leading to reduced computational time. For example, to reach an error of 10^{-3}, we need $N = 152$ (execution time: 76.5 seconds) with the standard FEM, whereas with the additive approach, we only need $N = 12$ (execution time: 0.052 seconds). This represents a speed-up of approximately 1 471 times.

However, the cost of training the network cannot be ignored. In Figure 3.27, we see that when we include a parametric context (including offline cost), the additive approach becomes more advantageous than the standard approach when using $N_\mu = 19$ parameter sets for an error of 10^{-3} (left figure) and $N_\mu = 5$ parameter sets for an error of $5 \cdot 10^{-4}$ (right figure). And it is with this framework that approaches enriched by PINNs become interesting.



(a) To achieve an error of 10^{-3}

(b) To achieve an error of $5 \cdot 10^{-4}$

Figure 3.27.: Considering the *3D Poisson problem*, $k = 1$ and the PINN prior u_θ. Computation time (in seconds) of the total process (including the offline phase) as a function of the number of parameter instances N_μ to be solved. The vertical dashed line represents the number of parameter instances from which the additive approach becomes more efficient than the standard FEM.

3.5.8.3. Gains achieved with the additive approach

Considering a set $\mathcal{D}$ of $N_\mu = 50$ parameter instances, we now evaluate the gains $G_{+,\theta}$ and G_+ defined in (3.34). The results are presented in Table 3.30 for $k = 1$ and $N \in \{20, 40\}$, where we observe gains relatively close to the 2D test case (Section 3.5.5.1).

		Gains in L^2 rel error of our method w.r.t. PINN				**Gains in L^2 rel error of our method w.r.t. FEM**			
method	**N**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
Add	20	8.57	20.19	14.11	2.69	75.34	166.98	137.34	24.07
	40	33.67	82.21	56.47	11.15	75.57	161.7	133.31	22.14

Table 3.30.: Considering the *3D Poisson problem*, $k = 1$ and the PINN prior u_θ. Left – Gains in L^2 relative error of the additive approach with respect to the PINN prior. Right – Same with respect to the standard FEM.

3.5.8.4. Influence of prior quality on the additive approach

Lastly, we focus on the impact of PINN quality on the results obtained with the additive approach. In Table 3.31, we look at the average gain obtained with different priors on the same sample of $N_\mu = 50$ parameters as in Section 3.5.8.3. More specifically, we train 9 PINNs (including the PINN from the previous sections) by varying only the number of epochs N_{epochs} and the number of collocation points N_{col}.

In Table 3.31, we can see that the number of collocation points considered seems to have a significant impact on the gains obtained. On the other hand, the number of epochs appears to be less significant. The previous results, therefore, seem to have

been obtained by choosing the training among the nine that gives the best gains, while having a relatively correct number of epochs.

	$N_{\mathbf{epochs}}$		
$N_{\mathbf{col}}$	2 500	5 000	10 000
10 000	57.27	76.38	77.74
20 000	78.33	87.79	88.26
40 000	110.52	137.34	138.59

Table 3.31.: Considering the *3D Poisson problem* and $k = 1$. Mean gain in L^2 relative error of the additive approach with respect to the standard FEM, by varying the quality of the PINN prior u_θ (by changing N_{epochs} and N_{col}).

Having validated the additive and multiplicative approaches on scalar, linear problems for which Sections 3.1 and 3.2 provide error estimates, we now turn to a non-linear test case involving the incompressible Navier–Stokes equations, which fall outside this theoretical framework, and to which we apply the additive approach in a purely heuristic manner.

3.6. Numerical results beyond the theoretical framework: the Navier–Stokes equations

This section considers a further test case, the stationary incompressible Navier–Stokes equations coupled with a temperature equation via the Boussinesq approximation, on the classical heated square cavity [44]. It leaves the scalar, linear framework of problem (2.1) considered up to Section 3.5. Neither the enrichment theory of Sections 3.1 and 3.2 nor the associated error estimates (Theorems 3.1 and 3.4) apply, since these were only established for the scalar, linear operator (2.2), whereas this test case corresponds to a mixed, non-linear system of PDEs. Unlike the test cases of Section 3.5, the additive approach of Section 3.1 is therefore applied to this problem on a purely heuristic basis, with no theoretical guarantee attached to it, and it is not part of our contribution [16]. This more demanding test case is also an opportunity to discuss, beyond the theoretical framework, some practical limitations of the additive approach.

We first present the test case and the problem under consideration in Section 3.6.1. The parametric PINN $\boldsymbol{U}_\theta$ used as a prior is then built in Section 3.6.2. Section 3.6.3 introduces the standard finite element discretization, along with the two Newton-solver initialization strategies considered. The additive approach is then extended to this non-linear, vector-valued setting in Section 3.6.4. Finally, Section 3.6.5 evaluates the error estimates obtained with the additive approach.

3.6.1. Problem statement

Let $\Omega = (-1, 1)^2$ be the square cavity, $\boldsymbol{x} = (x_1, x_2) \in \Omega$, and $\boldsymbol{e}_2 = (0, 1)$. We seek the velocity $\boldsymbol{u} := (u_1, u_2)$, the pressure P, and the temperature T, gathered in

$\boldsymbol{U} := (\boldsymbol{u}, P, T)$, solutions of:

$$\begin{cases} (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nabla P - \nu\Delta\boldsymbol{u} - g_0\,(0.1\,T+1)\,\boldsymbol{e}_2 = \boldsymbol{0}, & \text{in } \Omega, \quad \text{(momentum)} \\ \nabla\cdot\boldsymbol{u} = 0, & \text{in } \Omega, \quad \text{(incompressibility)} \\ \boldsymbol{u}\cdot\nabla T - k_f\Delta T = 0, & \text{in } \Omega, \quad \text{(energy)} \end{cases} \tag{3.42}$$

with $g_0 = 9.81$ the gravitational constant and a thermal expansion coefficient fixed at 0.1. The parameters $\boldsymbol{\mu} = (\nu, k_f) \in \mathcal{M} = [0.01, 0.1]^2$ are respectively the viscosity and the thermal conductivity, so that the number of parameters is $p = 2$ in this test case.

Problem (3.42) is closed by a no-slip condition on the velocity and mixed thermal boundary conditions:

$$\begin{gathered} \boldsymbol{u} = \boldsymbol{0} \ \text{ on } \ \Gamma, \\ T = 1 \ \text{ on } \ \{x_1 = -1\}, \qquad T = -1 \ \text{ on } \ \{x_1 = 1\}, \qquad \frac{\partial T}{\partial n} = 0 \ \text{ on } \ \Gamma_{\text{ad}}, \end{gathered} \tag{3.43}$$

with $\Gamma_{\text{ad}} := \Gamma \cap \{x_2 = \pm 1\}$ the (adiabatic) top and bottom walls: the cavity is heated on the left and cooled on the right, which drives the flow by buoyancy.

The relevant quantity for this study is the Rayleigh number Ra [36], which increases as $\boldsymbol{\mu} = (\nu, k_f)$ decreases, with the flow becoming increasingly convection-dominated, and therefore increasingly difficult to solve, as Ra grows. In this section, we consider three parameter sets:

$$\boldsymbol{\mu}^{(1)} = (0.1, 0.1), \qquad \boldsymbol{\mu}^{(2)} = (0.05, 0.05), \qquad \boldsymbol{\mu}^{(3)} = (0.01, 0.01),$$

with increasing Rayleigh number from $\boldsymbol{\mu}^{(1)}$ to $\boldsymbol{\mu}^{(3)}$. For each $\boldsymbol{\mu}^{(j)}$, the reference solution $\boldsymbol{U}^{(j)}$ is computed with the standard FEM on an over-refined mesh ($h \approx 7 \cdot 10^{-3}$), using $\mathbb{P}_3^2 \times \mathbb{P}_2 \times \mathbb{P}_3$ elements (see Section 3.6.3). The solutions obtained are shown in Figure 3.28 for each parameter set $\boldsymbol{\mu}^{(j)}$.

3.6.2. Construction of the parametric PINN prior

We deploy a parametric PINN $\boldsymbol{U}_\theta := (\boldsymbol{u}_\theta, P_\theta, T_\theta)$ with $\boldsymbol{u}_\theta := (u_{1,\theta}, u_{2,\theta})$, based on [42], which takes $(\boldsymbol{x}, \boldsymbol{\mu}) \in \Omega \times \mathcal{M}$ as input. It is trained by minimizing a weighted sum of the residual losses associated with the three equations of (3.42), together with a loss enforcing the adiabatic condition on Γ_{ad}; the other boundary conditions of (3.43) are imposed exactly on the network output through a post-processing step.

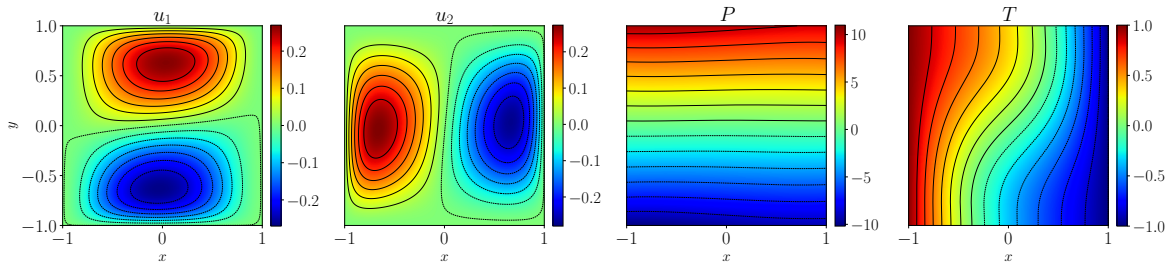
Architecture. Based on (2.42), the no-slip condition on the velocity is thus imposed exactly using the level-set function φ_1 defined by:

$$\varphi_1(\boldsymbol{x}) = (x_1 - 1)(x_1 + 1)(x_2 - 1)(x_2 + 1),$$

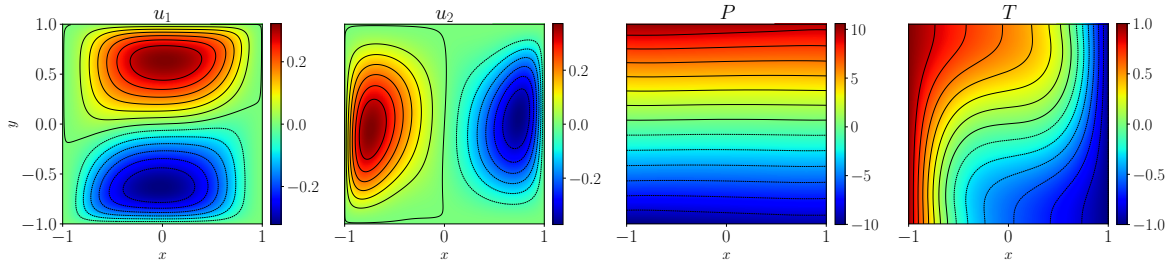
which vanishes on all of Γ. For the temperature, we use the level-set function φ_2 and the linear function g defined by:

$$\varphi_2(\boldsymbol{x}) = (x_1 - 1)(x_1 + 1), \quad \text{and} \quad g(\boldsymbol{x}) = 1 - (x_1 + 1),$$

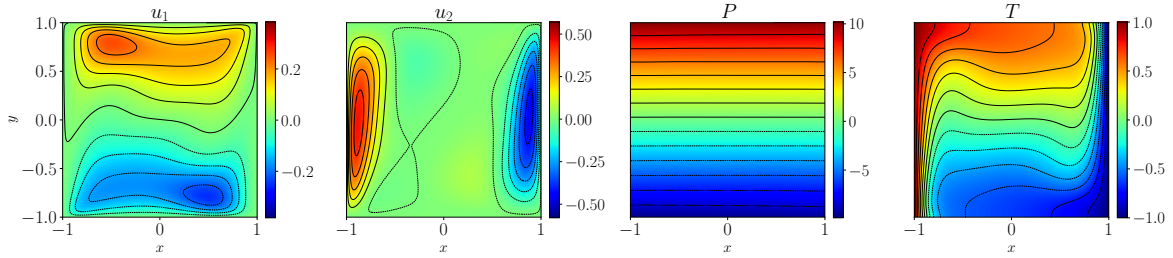
to impose $T = \pm 1$ on the walls $\{x_1 = \pm 1\}$ of (3.43). The remaining adiabatic condition on Γ_{ad} is enforced solely through the associated loss (3.45). No condition is imposed on the pressure P_θ, which is left free at the network output. A scheme of the network considered is given in Figure 3.29.



(a) Case of $\boldsymbol{\mu}^{(1)} = (0.1, 0.1)$, low Rayleigh number



(b) Case of $\boldsymbol{\mu}^{(2)} = (0.05, 0.05)$, medium Rayleigh number



(c) Case of $\boldsymbol{\mu}^{(3)} = (0.01, 0.01)$, high Rayleigh number

Figure 3.28.: Considering the *2D Navier–Stokes problem*. Reference solutions $\boldsymbol{U}^{(j)}$ for $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, and $\boldsymbol{\mu}^{(3)}$ (top to bottom), with increasing Rayleigh number.

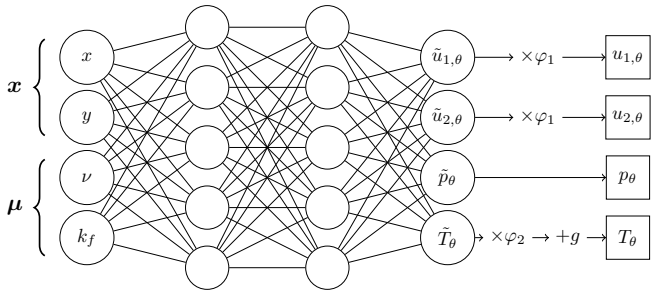


Figure 3.29.: Architecture of the parametric PINN $\boldsymbol{U}_\theta$ considered for the *2D Navier–Stokes problem*. The boundary conditions of (3.43), except for the adiabatic condition on Γ_{ad}, are imposed exactly through post-processing of the network output.

Loss and training. More precisely, following the minimization problem (2.36), we seek the optimal weights $\theta^\star$ solutions of:

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \left(\omega_{\text{mom}} \mathcal{J}_{\text{mom}}(\theta) + \omega_{\text{inc}} \mathcal{J}_{\text{inc}}(\theta) + \omega_{\text{ener}} \mathcal{J}_{\text{ener}}(\theta) + \omega_{\text{ad}} \mathcal{J}_{\text{ad}}(\theta) \right), \tag{3.44}$$

with ω_{mom}, ω_{inc}, ω_{ener}, and ω_{ad} the weights balancing the terms of the loss function. The three residual losses $\mathscr{J}_{\text{mom}}$, $\mathscr{J}_{\text{inc}}$, and $\mathscr{J}_{\text{ener}}$, associated respectively with the momentum, incompressibility, and energy equations of (3.42), are built following the residual loss (2.37), and defined by:

$$\begin{aligned}
\mathscr{J}_{\text{mom}}(\theta) &:= \int_{\mathcal{M}} \int_{\Omega} \left| (\boldsymbol{u}_\theta \cdot \nabla)\boldsymbol{u}_\theta + \nabla P_\theta - \nu \Delta \boldsymbol{u}_\theta - g_0 \left(0.1\, T_\theta + 1\right) \boldsymbol{e}_2 \right|^2 \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{\mu}, \\
\mathscr{J}_{\text{inc}}(\theta) &:= \int_{\mathcal{M}} \int_{\Omega} \left| \nabla \cdot \boldsymbol{u}_\theta \right|^2 \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{\mu}, \\
\mathscr{J}_{\text{ener}}(\theta) &:= \int_{\mathcal{M}} \int_{\Omega} \left| \boldsymbol{u}_\theta \cdot \nabla T_\theta - k_f \Delta T_\theta \right|^2 \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{\mu},
\end{aligned}$$

while the adiabatic loss $\mathscr{J}_{\text{ad}}$ imposes the homogeneous Neumann condition on Γ_{ad} of (3.43):

$$\mathscr{J}_{\text{ad}}(\theta) := \int_{\mathcal{M}} \int_{\Gamma_{\text{ad}}} \left| \frac{\partial T_\theta}{\partial n} \right|^2 \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{\mu}. \tag{3.45}$$

As for (2.40), these integrals are, in practice, approximated by a Monte Carlo method. The MLP and training hyperparameters are given in Table 3.32. The training loss evolution is shown in Figure 3.30.

Network - MLP	
layers	$40, 60, 60, 60, 40$
σ	sine

Training - with LBFGS			
lr	7e-3	N_{epochs}	10 000
decay	0.99	n_{switch}	3 000
N_{col}	40 000	N_{bc}	30 000

Loss weights			
ω_{mom}	1	ω_{inc}	1
ω_{ener}	1	ω_{ad}	2

Table 3.32.: Network, training parameters (Notation 3.8) and loss weights for $\boldsymbol{U}_\theta$ in the *2D Navier–Stokes problem.*

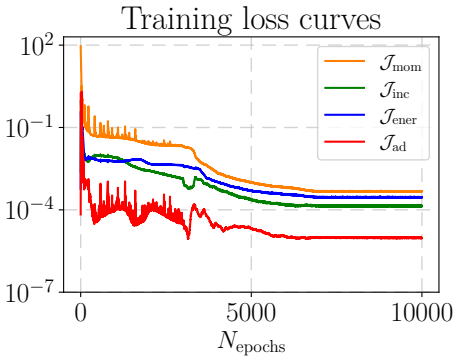
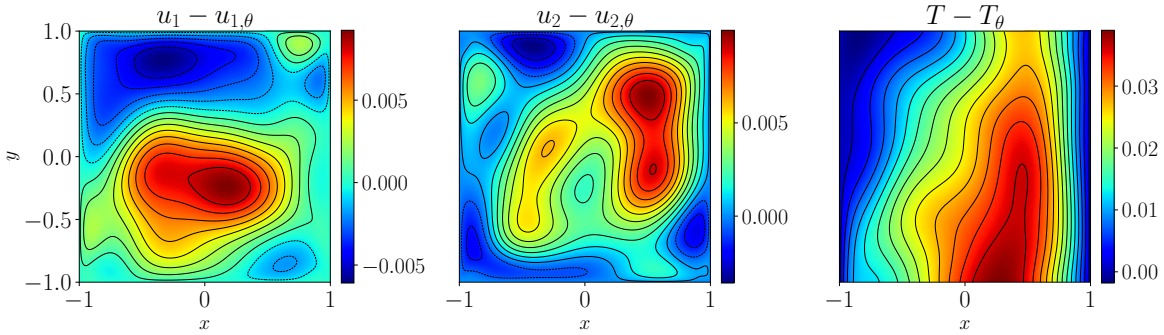


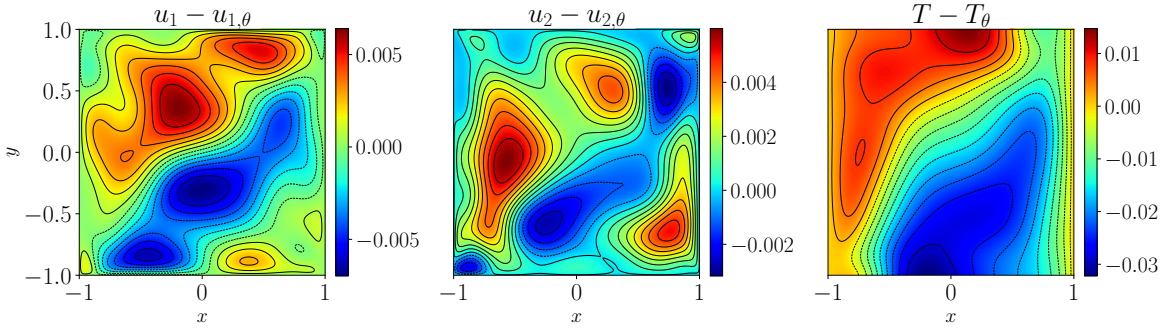
Figure 3.30.: Considering the *2D Navier–Stokes problem.* Evolution of the components $\mathscr{J}_{\text{mom}}$, $\mathscr{J}_{\text{inc}}$, $\mathscr{J}_{\text{ener}}$, and $\mathscr{J}_{\text{ad}}$ (3.44) during training of the parametric PINN $\boldsymbol{U}_\theta$.

The prediction of the parametric PINN $\boldsymbol{U}_\theta$ is then evaluated for each parameter set $\boldsymbol{\mu}^{(j)}$, $j \in \{1, 2, 3\}$, and compared to the reference solution $\boldsymbol{U}^{(j)}$ in Figure 3.31. The error

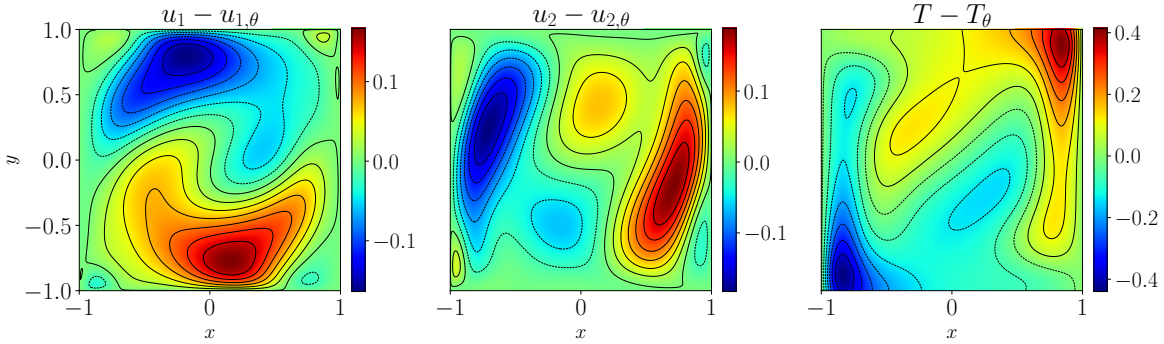
$\boldsymbol{U}^{(j)} - \boldsymbol{U}_\theta^{(j)}$ is observed to increase with the Rayleigh number, which is consistent with the fact that the flow becomes increasingly convection-dominated as $\boldsymbol{\mu}^{(j)}$ decreases.



(a) Case of $\boldsymbol{\mu}^{(1)} = (0.1, 0.1)$, low Rayleigh number



(b) Case of $\boldsymbol{\mu}^{(2)} = (0.05, 0.05)$, medium Rayleigh number



(c) Case of $\boldsymbol{\mu}^{(3)} = (0.01, 0.01)$, high Rayleigh number

Figure 3.31.: Considering the *2D Navier–Stokes problem*. Error $\boldsymbol{U}^{(j)} - \boldsymbol{U}_\theta^{(j)}$ between the reference solution $\boldsymbol{U}^{(j)}$ and the prediction of the parametric PINN $\boldsymbol{U}_\theta^{(j)}$, for $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, and $\boldsymbol{\mu}^{(3)}$ (top to bottom), with increasing Rayleigh number.

3.6.3. Finite element discretization

We first detail the discrete weak formulation of the problem, based on a Taylor–Hood finite element space, and then the Newton solver together with its two initialization strategies.

Discrete formulation. In this section and the next one, the parameter $\boldsymbol{\mu}$ is fixed, and its dependence is deliberately omitted to lighten the notation. For the finite element discretization (except for the reference solution), we consider the Taylor–Hood mixed finite element space [124] defined by:

$$X_h := [V_h^0]^2 \times Q_h \times W_h,$$

with V_h^0 the continuous Lagrange $\mathbb{P}_2$ space defined in (2.5) (homogeneous, since the no-slip condition holds on all of Γ), Q_h a continuous Lagrange $\mathbb{P}_1$ space for the pressure, and W_h a continuous Lagrange $\mathbb{P}_2$ space accounting for the non-homogeneous isothermal walls of (3.43).

Notation 3.25. *As for the previous 2D test cases, we consider a Cartesian mesh with N^2 nodes, so that the characteristic mesh size is $h = \frac{2\sqrt{2}}{N-1}$.*

The discrete weak problem then reads: find $\boldsymbol{U}_h := (\boldsymbol{u}_h, P_h, T_h) \in X_h$ such that, for all $(\boldsymbol{v}_h, q_h, w_h) \in X_h^0 := [V_h^0]^2 \times Q_h \times W_h^0$:

$$\begin{aligned}
&\int_\Omega (\boldsymbol{u}_h \cdot \nabla)\boldsymbol{u}_h \cdot \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} + \nu \int_\Omega \nabla \boldsymbol{u}_h : \nabla \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} \\
&\qquad\qquad - \int_\Omega P_h \,\nabla \cdot \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} - g_0 \int_\Omega (1 + 0.1\, T_h)\, \boldsymbol{e}_2 \cdot \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} = 0, \\
&\int_\Omega q_h \,\nabla \cdot \boldsymbol{u}_h \,\mathrm{d}\boldsymbol{x} + 10^{-4} \int_\Omega q_h \, P_h \,\mathrm{d}\boldsymbol{x} = 0, \\
&\int_\Omega (\boldsymbol{u}_h \cdot \nabla T_h)\, w_h \,\mathrm{d}\boldsymbol{x} + k_f \int_\Omega \nabla T_h \cdot \nabla w_h \,\mathrm{d}\boldsymbol{x} = 0,
\end{aligned} \tag{3.46}$$

where $W_h^0 \subset \{w \in H^1(\Omega),\ w|_{x_1=\pm 1} = 0\}$, and where the term $10^{-4} \int_\Omega q_h \, P_h \,\mathrm{d}\boldsymbol{x}$ is a standard pressure penalization, which forces the discrete pressure to have zero mean, i.e. $P_h \in L_0^2(\Omega)$.

Newton solver and initialization. Let $N_{\mathrm{dofs}} := \dim(X_h^0)$ denote the total number of degrees of freedom of the discrete space X_h. Denoting by $\boldsymbol{\beta}_h \in \mathbb{R}^{N_{\mathrm{dofs}}}$ the vector of degrees of freedom associated with $\boldsymbol{U}_h$ and by $F : \mathbb{R}^{N_{\mathrm{dofs}}} \to \mathbb{R}^{N_{\mathrm{dofs}}}$ the non-linear operator associated with (3.46), we solve $F(\boldsymbol{\beta}_h) = 0$ using a Newton method. Two types of solver initialization can then be considered:

- **Natural initialization**: a constant or componentwise linear function is used. For a fixed parameter set $\boldsymbol{\mu}^{(j)}$, $j \in \{1, 2, 3\}$, we thus consider:

  $$\boldsymbol{\beta}_h^{(0)} := (\mathbf{0}, \mathbf{0}, 0, \boldsymbol{T}_0),$$

  where, for $i = 1, \dots, N_{\mathrm{dofs}}$, $(\boldsymbol{T}_0)_i := g(\boldsymbol{x}^{(i)}) = 1 - (x_1^{(i)} + 1)$ (the affine function already introduced in Section 3.6.2), with $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})$ the coordinates of the i-th degree of freedom of X_h; in other words, the velocity and pressure components are initialized to zero, while the temperature is initialized by an affine function consistent with the isothermal boundary conditions of (3.43);

- **PINN initialization**: the prediction of the parametric PINN $\boldsymbol{U}_\theta$ (Section 3.6.2) is used, following an approach similar to that employed in other works to accelerate the convergence of a Newton solver using a neural network, whether a fully connected network [101] or an FNO [2]. For a fixed parameter set $\boldsymbol{\mu}^{(j)}$, we thus consider, for $i = 1, \dots, N_{\text{dofs}}$:

$$\left(\boldsymbol{\beta}_h^{(0)}\right)_i := \boldsymbol{U}_\theta\left(\boldsymbol{x}^{(i)}, \boldsymbol{\mu}^{(j)}\right),$$

  with $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})$ the coordinates of the i-th degree of freedom of X_h.

Remark 3.26. *Another classical strategy to facilitate the convergence of the Newton solver, in particular when Ra increases, is to use a continuation method [4]: a sequence of problems is solved for parameter sets $\boldsymbol{\mu}$ progressively approaching the target, initializing the Newton solver at each step with the solution obtained at the previous step. This approach is not explored here, where we restrict ourselves to the two initializations described above.*

3.6.4. Extending the additive approach to the Navier–Stokes system

We first write the enriched formulation of the problem and the associated Newton solver. We then show that, when the PINN prediction is interpolated exactly, the additive approach coincides with the standard FEM initialized by the PINN.

Enriched formulation. As in the previous section, a fixed parameter $\boldsymbol{\mu}$ is considered here, and its dependence is deliberately omitted to lighten the notation. The additive approach of Section 3.1 is then extended to the above setting by seeking the solution in the mixed space additively enriched by the PINN prior and defined by:

$$X_h^+ := \left\{\boldsymbol{U}_h^+ = \boldsymbol{U}_\theta + \boldsymbol{p}_h^+,\ \ \boldsymbol{p}_h^+ \in X_h^0\right\},$$

with $\boldsymbol{U}_\theta$ the PINN prediction (Section 3.6.2) and $\boldsymbol{p}_h^+ := (\boldsymbol{p}_{h,\boldsymbol{u}}^+, p_{h,P}^+, p_{h,T}^+) \in X_h^0$ the correction, the unknown of the variational formulation associated with the additive problem. The problem becomes: find $\boldsymbol{p}_h^+ \in X_h^0$ such that, for all $(\boldsymbol{v}_h, q_h, w_h) \in X_h^0$:

$$\begin{aligned}
&\int_\Omega \left((\boldsymbol{u}_\theta \cdot \nabla)\boldsymbol{u}_\theta + (\boldsymbol{u}_\theta \cdot \nabla)\boldsymbol{p}_{h,\boldsymbol{u}}^+ + (\boldsymbol{p}_{h,\boldsymbol{u}}^+ \cdot \nabla)\boldsymbol{u}_\theta + (\boldsymbol{p}_{h,\boldsymbol{u}}^+ \cdot \nabla)\boldsymbol{p}_{h,\boldsymbol{u}}^+\right) \cdot \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} \\
&\qquad + \nu\left(\int_\Omega \nabla\boldsymbol{u}_\theta : \nabla\boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} + \int_\Omega \nabla\boldsymbol{p}_{h,\boldsymbol{u}}^+ : \nabla\boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x}\right) - \int_\Omega \left(P_\theta + p_{h,P}^+\right)\nabla\cdot\boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} \\
&\qquad - g_0 \int_\Omega \left(1 + 0.1\left(T_\theta + p_{h,T}^+\right)\right)\boldsymbol{e}_2 \cdot \boldsymbol{v}_h \,\mathrm{d}\boldsymbol{x} = 0, \\
&\int_\Omega q_h \left(\nabla\cdot\boldsymbol{u}_\theta + \nabla\cdot\boldsymbol{p}_{h,\boldsymbol{u}}^+\right)\mathrm{d}\boldsymbol{x} + 10^{-4}\int_\Omega q_h\left(P_\theta + p_{h,P}^+\right)\mathrm{d}\boldsymbol{x} = 0, \\
&\int_\Omega \left(\boldsymbol{u}_\theta \cdot \nabla T_\theta + \boldsymbol{u}_\theta \cdot \nabla p_{h,T}^+ + \boldsymbol{p}_{h,\boldsymbol{u}}^+ \cdot \nabla T_\theta + \boldsymbol{p}_{h,\boldsymbol{u}}^+ \cdot \nabla p_{h,T}^+\right) w_h \,\mathrm{d}\boldsymbol{x} \\
&\qquad\qquad + k_f\left(\int_\Omega \nabla T_\theta \cdot \nabla w_h \,\mathrm{d}\boldsymbol{x} + \int_\Omega \nabla p_{h,T}^+ \cdot \nabla w_h \,\mathrm{d}\boldsymbol{x}\right) = 0.
\end{aligned} \tag{3.47}$$

Newton solver. Let $N_{\text{dofs}} = \dim(X_h^0)$ denote the total number of degrees of freedom of the discrete space X_h^0. The additive approach consists in solving the non-linear problem (3.47) for the correction $\boldsymbol{p}_h^+$, which is an element of X_h^0. Denoting by $\boldsymbol{\beta}_h^+ \in \mathbb{R}^{N_{\text{dofs}}}$ the vector of degrees of freedom associated with $\boldsymbol{p}_h^+$ and by $F_\theta : \mathbb{R}^{N_{\text{dofs}}} \to \mathbb{R}^{N_{\text{dofs}}}$ the non-linear operator associated with (3.47), we solve $F_\theta(\boldsymbol{\beta}_h^+) = 0$ using a Newton method starting from $\boldsymbol{\beta}_h^{+,(0)} = \mathbf{0}$. Since the PINN prediction is assumed to be relatively close to the exact solution, the correction $\boldsymbol{p}_h^+$ is expected to be small, and hence the number of Newton iterations needed to reach convergence is expected to be low.

Link with the PINN-initialized FEM. In practice, the terms of (3.47) involving the components of $\boldsymbol{U}_\theta$ are integrated exactly using quadrature rules, after interpolating $\boldsymbol{U}_\theta$ beforehand into a polynomial space (Section 3.4.1). As observed in the scalar linear case (Section 3.5.5.1, Figure 3.15), the degree of the polynomial approximation used for this integration must be high enough for the additive approach to be effective.

In the non-linear case considered here, we can even go further and show that, if $\boldsymbol{U}_\theta$ is interpolated exactly in X_h, the additive approach coincides exactly with the standard FEM initialized by the PINN. More precisely, let $\boldsymbol{U}_{\theta,h} \in \mathbb{R}^{N_{\text{dofs}}}$ denote the vector of degrees of freedom of this interpolation, which then coincides exactly with the vector $\boldsymbol{\beta}_h^{(0)}$ of the PINN initialization of Section 3.6.3. In this case, $\boldsymbol{U}_{\theta,h}$ and $\boldsymbol{\beta}_h^+$ are represented in the same discrete space, and the non-linear residual F_θ of the additive approach (3.47) coincides with the residual F of the standard FEM (3.46), evaluated at the point shifted by $\boldsymbol{U}_{\theta,h}$:

$$F_\theta(\boldsymbol{\beta}_h^+) = F\big(\boldsymbol{U}_{\theta,h} + \boldsymbol{\beta}_h^+\big), \qquad \forall\, \boldsymbol{\beta}_h^+ \in \mathbb{R}^{N_{\text{dofs}}}.$$

The Newton method is in fact invariant under such an affine change of variable; the first iteration of the additive approach, starting from $\boldsymbol{\beta}_h^{+,(0)} = \mathbf{0}$, gives:

$$F_\theta\big(\mathbf{0}\big) + F'_\theta\big(\mathbf{0}\big)\, \delta^{(1)} = 0,$$

which is then exactly that of the standard FEM initialized by the PINN (Section 3.6.3), defined by:

$$F(\boldsymbol{U}_{\theta,h}) + F'(\boldsymbol{U}_{\theta,h})\, \delta^{(1)} = 0.$$

By induction, the same holds for all subsequent iterations, and hence, in this particular case, the additive approach coincides exactly with the standard FEM initialized by the PINN. This is why, in what follows, we apply the same remark as in the scalar case (Figure 3.15), using an interpolation degree high enough for the additive approach to be effective.

3.6.5. Error estimates

As in the previous test cases, and in particular in Section 3.5.1.1, we study the convergence of the relative L^2 error as a function of h (Notation 3.25). Recall that, for each parameter set $\boldsymbol{\mu}^{(j)}$, the reference solution $\boldsymbol{U}^{(j)}$ is computed with the standard FEM on an over-refined mesh (Figure 3.28), and that the PINN prediction is denoted $\boldsymbol{U}_\theta^{(j)}$ (Section 3.6.2). We denote by $\boldsymbol{U}_h^{(j)}$ the solution obtained with the standard FEM

(Section 3.6.3) on the current mesh, and by $\boldsymbol{U}_{h,+}^{(j)}$ the solution obtained with the additive approach (Section 3.6.4).

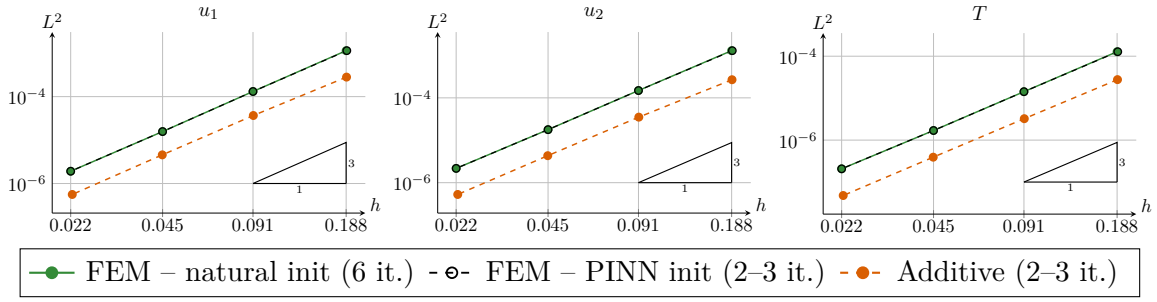
Unlike the previous scalar test cases, each of these solutions gathers four components. For each approach (standard FEM, additive approach, or PINN), a componentwise relative L^2 error is then defined as in (3.30) and (3.31). Since the pressure component P is not relevant for this comparison, we restrict the study in what follows to the relative errors on u_1, u_2, and T. For conciseness, these errors are denoted indifferently $e_h^{(j)}$, $e_\theta^{(j)}$, and $e_{h,+}^{(j)}$, without explicitly showing their dependence on the component considered. Figure 3.32 shows the convergence curves obtained for the three parameter sets $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, and $\boldsymbol{\mu}^{(3)}$. As explained in Section 3.6.3, two types of initialization are considered for the standard FEM: a natural initialization or a PINN initialization. The additive approach, for its part, is initialized with $\mathbf{0}$. The number of Newton iterations needed to reach convergence, when convergence occurs, is given in parentheses in the legend of each curve.

First, we see in Figure 3.32 that all curves converge with a convergence order of 3, which is consistent with the approximation order of the finite elements considered for the velocity $\boldsymbol{u}$ and the temperature T. We also note that the naturally initialized standard FEM does not converge for some meshes, in particular for $\boldsymbol{\mu}^{(2)}$ with the coarsest mesh ($N = 16$), and for $\boldsymbol{\mu}^{(3)}$ with all the meshes considered. This is consistent with the fact that the exact solution becomes increasingly convection-dominated as Ra grows, so that the natural initialization is then no longer close enough to the exact solution to guarantee convergence of the Newton solver. In contrast, the standard FEM initialized by the PINN and the additive approach appear to converge for all the meshes considered. We note that the errors obtained with the PINN-initialized standard FEM overlap with those of the natural initialization when convergence occurs, which is consistent with the fact that the Newton method converges to the same solution, regardless of the initialization, as soon as the latter is close enough to the exact solution.

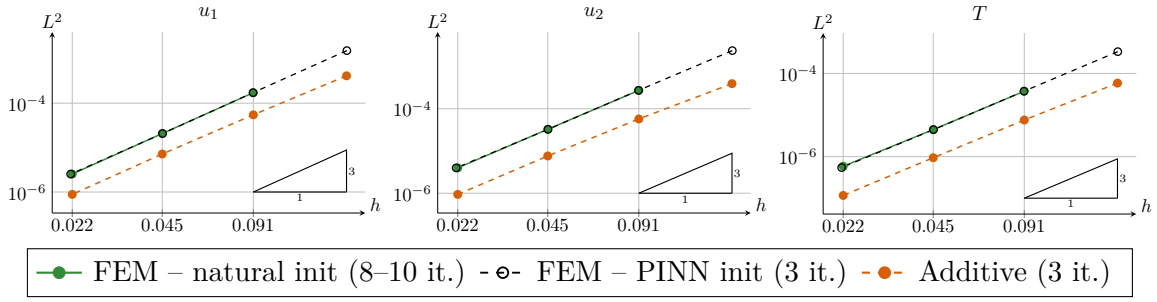
As in the other test cases, the additive approach appears more efficient than the standard FEM for the three parameter sets considered. However, it seems that the higher the Rayleigh number, and hence the lower the accuracy of the PINN prior (Figure 3.31), the less significant the gain from the additive approach becomes. In particular, for $\boldsymbol{\mu}^{(3)}$, the additive approach brings only a small improvement over the standard FEM. This is explained, in the same way, by the fact that the flow becomes increasingly convection-dominated as Ra grows, and that the parametric PINN $\boldsymbol{U}_\theta$ has more difficulty to correctly predict the solution in this regime.

In conclusion, this non-linear vector-valued case, more complicated than the previous test cases, falls completely outside the theoretical framework considered. However, the numerical results show that the additive approach can be effective even in this setting (to some extent). We have seen that, as the Rayleigh number increases, the accuracy of the PINN prior decreases (Figure 3.31), and that it is precisely in this case that the efficiency of the additive approach becomes much less significant. This analysis is in fact consistent with the results obtained in Section 3.5, where we saw that, as the accuracy of the prior decreases, and in particular the accuracy of its derivatives, the efficiency of the enriched approach also decreases. We can therefore hope to improve the results obtained for $\boldsymbol{\mu}^{(3)}$ by improving the accuracy of the prior, for example by considering a non-parametric PINN trained specifically for this parameter set, or by considering

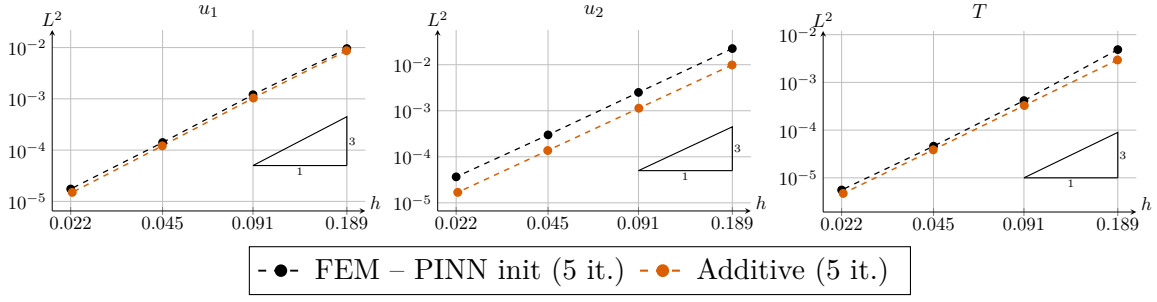
other neural network architectures, such as neural operators. Another direction would be to add to the training loss a term penalizing the norm of the resulting Newton correction, $\|[F'(\boldsymbol{U}_{\theta,h})]^{-1}F(\boldsymbol{U}_{\theta,h})\|^2$, in addition to the usual residual-based PINN loss. A more in-depth study should be considered to better understand the limitations of the additive approach in this context, and to identify the conditions under which it can be effective.



(a) Case of $\boldsymbol{\mu}^{(1)}$



(b) Case of $\boldsymbol{\mu}^{(2)}$



(c) Case of $\boldsymbol{\mu}^{(3)}$

Figure 3.32.: Considering the *2D Navier–Stokes problem* and the parametric PINN prior $\boldsymbol{U}_\theta$. Componentwise L^2 relative errors (u_1, u_2, and T) as a function of h, obtained with the naturally initialized standard FEM $e_h^{(j)}$ (solid lines), the standard FEM initialized by the PINN $\boldsymbol{U}_\theta$ (black dashed lines), and the additive approach $e_{h,+}^{(j)}$ (orange dashed lines), for $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, and $\boldsymbol{\mu}^{(3)}$. The number in parentheses in the legend indicates the number of Newton iterations needed for convergence (range observed over the four meshes). The points that do not appear on the curve of the naturally initialized standard FEM correspond to meshes for which the Newton method did not converge.

Having shown, using test cases both within and beyond the theoretical framework, that enriching a continuous space in FEM using a PINN prior can significantly reduce the number of dofs required to achieve a target accuracy, provided that this prior, and in particular its derivatives, is sufficiently accurate, we now turn to another way of coupling neural networks with mesh-based PDE solvers. Rather than enriching a standard finite element solution (or one based on the DG method) a posteriori with a fixed PINN prior, we introduce the `JAX` *version of* `ScimBa`*, a differentiable programming library whose modules for assembling the geometry, basis functions and DG/FE scheme are all designed to be differentiable.*

4 SciML library using differentiable tools

Contents

4.1. Elements of Differentiable Programming . . . 101
4.2. Geometry and integration modules . . . 104
4.2.1. Quadrature module . . . 105
4.2.2. Mapping module . . . 106
4.2.3. Mesh module . . . 109
4.3. Basis functions module . . . 117
4.4. DG variable and post-processing modules . . . 119
4.5. DG scheme modules . . . 122
4.5.1. Linear weak forms . . . 122
4.5.2. Numerical fluxes . . . 124
4.5.3. Boundary conditions . . . 125
4.5.4. DG scheme assembly . . . 127
4.6. Solve framework modules . . . 129
4.6.1. Newton solve . . . 129
4.6.2. Matrix-free Newton solve . . . 130
4.6.3. The solver as a non-linear approximation space . . . 131
4.7. Extension to FEM . . . 131
4.7.1. Continuous FE variables . . . 132
4.7.2. FE scheme assembly . . . 133
4.8. Extension to φ-FEM . . . 135
4.8.1. Level-set-based cell and face classification . . . 136
4.8.2. φ-FEM scheme assembly . . . 136
4.9. Learnable formalization of the Galerkin schemes . . . 140
4.10. Numerical validations . . . 144
4.10.1. 1D Poisson problem . . . 144
4.10.2. 1D Poisson problem with mixed boundary conditions . . . 147
4.10.3. 1D general elliptic system and convection-dominated regime . . . 150
4.10.4. 1D non-linear Poisson problem . . . 153
4.10.5. 2D Poisson problem in a square domain . . . 155
4.10.6. 2D Poisson problem in a 3-lobed shape domain . . . 157
4.10.7. 1D reaction-diffusion inverse problem . . . 161

The notations introduced in this chapter, together with those of Chapter 5, are recalled in Appendix A.1.3.

In this chapter, we present the different modules of the `ScimBa`[1] library (a SciML library), developed in `JAX` [26], which allows implementing differentiable numerical methods in the context of solving PDEs. This library is designed to make a number of its modules differentiable (see Section 4.1) and therefore learnable. Sections 4.2 to 4.6 detail the library's modules used to build a differentiable DG method, whose architecture is presented in Figure 4.1.

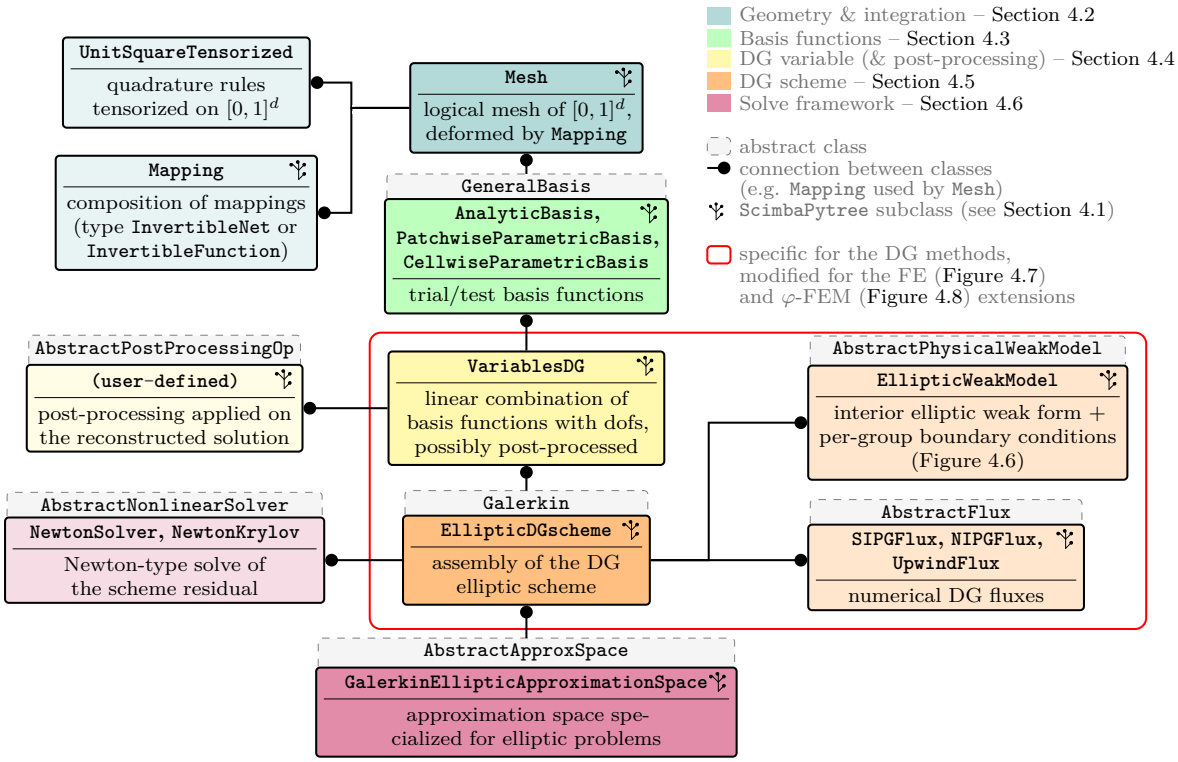


Figure 4.1.: Architecture of `ScimBa` for the DG method (Sections 4.2 to 4.6). The DG-specific boxes (red outline) are replaced in Figure 4.7 for the FE extension (Section 4.7) and in Figure 4.8 for the φ-FEM one (Section 4.8).

This notably includes the geometry (with mesh, mappings and numerical integration), the basis functions, the DG variables, the post-processing, as well as the DG schemes and their solve frameworks. We then present two extensions of this same architecture, in the context of the FE method (Section 4.7) and for φ-FEM (Section 4.8), both built by adding new variable and assembly modules on top of the geometry, basis and solve modules above, which are largely reused. Once DG, FEM and φ-FEM are all in place, we present, in Section 4.9, the formalism allowing differentiable learning on these Galerkin schemes, in particular by presenting the bi-level optimization principle. Finally, in Section 4.10, we present numerical validations of the library, checking that the DG, FEM and φ-FEM implementations recover the convergence orders expected

[1] https://www.scimba.org/

from theory, and that the learnable formalism can be used on an inverse problem. A minimal example putting these modules together, with a classical FE solver and a learned basis (the latter being studied in Chapter 5), is given in Appendix B.1. In this chapter, we do not discuss the merits of parameterizing specific pieces of these Galerkin schemes; we focus instead on the formalism that makes these modules differentiable (and therefore learnable). We discuss the benefit of doing so, particularly for the DG scheme, only in the following chapter (Chapter 5).

> **Remark 4.1.** *In Chapter 2, the three mesh-based methods (FEM, DG and φ-FEM) are presented on a mesh $\mathcal{T}_h$ composed of simplexes, following the standard references on which each presentation is based. This choice is not essential to the methods themselves: in this chapter and the next one (Chapters 4 and 5), the numerical schemes are implemented on a mesh of (possibly deformed) hypercubes, due to construction choices related to the* `ScimBa` *library (see Section 4.2.3). The constructions and results presented in these chapters therefore apply directly to this framework, by replacing $\mathbb{P}_k$ with $\mathbb{Q}_k$ (see Section 4.3).*

4.1. Elements of Differentiable Programming

Solving a variational PDE on an arbitrary domain requires derivatives at several levels simultaneously (differential operator, parameter optimization, etc.). These derivatives concern objects of different natures, in particular analytical functions or learnable networks, which in a classical framework would be handled by separate mechanisms. This section presents the principles of automatic differentiation, the `JAX` library that implements them, and the pytree mechanism used to make the modules of `ScimBa` differentiable.

Differentiable programming. Differentiable programming (see [25]) aims to unify the derivatives of these different objects by extending automatic differentiation beyond simple neural networks, i.e. to arbitrary programs (loops, branches, data structures, etc.).

Three approaches make it possible to compute derivatives numerically [19]. Symbolic differentiation, used by libraries such as `SymPy` [94], manipulates mathematical expressions to produce their derivative in closed form by mechanically applying known differentiation rules. It is exact, but limited to closed-form expressions, meaning it does not apply to arbitrary programs and can produce expressions much longer than the original. Finite differences, on the other hand, approximate the derivative, which makes them universally usable and simple to implement, but they introduce truncation error and require one function evaluation per dimension, which becomes prohibitive as soon as the number of parameters is large. They are therefore rarely used in machine learning, where one can count millions, or even billions, of dimensions. Automatic differentiation (AD or autodiff) combines the advantages of both: it is exact like symbolic differentiation and applicable to arbitrary programs like finite differences.

Automatic differentiation. Let us consider an arbitrary differentiable function $\mathrm{f} : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ (unrelated to the PDE source term) with n_1 and n_2 here denoting generic

input/output dimensions, and $J_f \in \mathbb{R}^{n_2 \times n_1}$ its Jacobian matrix. To efficiently compute the derivatives of the function f, AD relies on simple but fundamental mathematical principles from linear algebra, differential calculus, and graph theory. Computer programs can be viewed as Directed Acyclic Graphs (DAGs) [25] where each node corresponds to the result of an intermediate operation of the program, and the edges represent the dependencies of the current operations on the previous ones. The derivative of the entire program is then obtained by applying the chain rule along this graph, composing the derivatives of each elementary operation whose analytical expression is known.

This traversal of the graph can be performed in two directions (see [67]). Forward mode propagates derivatives from input to output via Jacobian-Vector Products (JVPs): for a given direction $\boldsymbol{v} \in \mathbb{R}^{n_1}$, it computes $J_f \cdot \boldsymbol{v}$ in a single pass. It is efficient when the number of inputs is small compared to the number of outputs. Backward mode, of which gradient backpropagation in machine learning is a special case, propagates derivatives from output to input via Vector-Jacobian Products (VJPs): for a given covector $\boldsymbol{u} \in \mathbb{R}^{n_2}$, it computes $\boldsymbol{u}^T J_f$. Its cost is proportional to that of the forward program (up to a small constant factor), regardless of the number of parameters, a result known as the Baur–Strassen theorem [18, 67], at the price of storing the intermediate values of the forward pass. AD is therefore close to optimal in terms of computation time: computing the full gradient of a scalar functional costs only a few evaluations of the functional itself, which is what makes the optimization of high-dimensional models viable. In our context, where the goal is to minimize a scalar functional with respect to a large number of parameters, backward mode is naturally the one favored.

AD as described so far assumes that the program is explicitly represented as a computational graph of elementary operations. Some operations, however, do not lend themselves to this direct representation: this happens when a quantity of interest is defined only implicitly, as the solution of an equation depending on some parameters, rather than by an explicit computation procedure. A typical example is an iterative solver (a Newton algorithm, for instance) whose output one wishes to differentiate with respect to its parameters. A first possibility is to differentiate through all the steps of the solver: the iterations are recorded in the computational graph and the chain rule is applied through each of them, which yields the so-called unrolled gradient. However, the graph then grows with the number of iterations (and so do the memory and the cost of the backward pass), which becomes prohibitive for a Newton algorithm applied to a large system. A second option relies on the implicit function theorem, which provides an analytical expression for the derivative of the solution with respect to these parameters from the equation satisfied by the solution alone, without differentiating through the solver's iterations. This approach, known as implicit differentiation, will be detailed and used in Section 4.9.

The `JAX` library. These principles are implemented in `JAX` [26], a `Python` library that stands apart from classical deep learning frameworks such as `PyTorch` [103] or `TensorFlow` [1] in its philosophy. `JAX` is not a neural network framework, but a composer of functional transformations applied to NumPy-like functions. Concretely, `JAX` exposes a set of primitive transformations that apply to pure functions and compose freely with one another. Notable examples include:

- `jax.grad`, `jax.vjp`, `jax.jvp`: computation of gradients, VJPs in backward mode and JVPs in forward mode;
- `jax.vmap`: automatic vectorization of a function over a batch of data;
- `jax.jit`: Just-In-Time compilation via Accelerated Linear Algebra (XLA) for execution on CPU, GPU or tensor processing unit (TPU);
- `jax.lax.scan`: efficient compilation and differentiation of fixed-length loops (without unrolling).

The composability of these different transformations is the key property of the framework: an expression such as `jax.jit(jax.vmap(jax.grad(f)))` is perfectly valid and produces a compiled, vectorized, and differentiated version of a function `f`. It is this feature that will later make it possible to uniformly handle differential operators and learnable parameters within a single program.

Pytrees and the `ScimBa` pytree mechanism. These transformations operate on pytrees, a term denoting any `Python` data structure whose leaves are `JAX` arrays. By default, `JAX` recognizes standard containers such as lists, tuples, and dictionaries as pytrees, but the mechanism is fully extensible: any `Python` class can be registered as a pytree by implementing a `tree_flatten` method, decomposing an instance into its dynamic `children` (the arrays `JAX` differentiates through and maps over) and its static `aux_data` (treated as a compile-time constant, receiving no gradient), together with a `tree_unflatten` method rebuilding an instance from these two parts. This makes it possible to differentiate complex objects such as neural networks, meshes, or mappings, by treating them as containers of `JAX` arrays, as long as they are registered as pytrees.

Writing this pair of functions, `tree_flatten`/`tree_unflatten`, manually for each class quickly becomes repetitive in a library where most objects (e.g. meshes, mappings, bases, variables, etc.) share the same need. `ScimBa` factors this pattern into a common base class `ScimbaPytree`, from which all these objects are derived and which provides a single, shared implementation of `tree_flatten`/`tree_unflatten`, reused by all subclasses, none of which writes it by hand. In the general case, a subclass does not need to declare anything: an attribute is placed in `children` as soon as it contains at least one `JAX` array, whether directly or not (e.g. nested in a list), and in `aux_data` otherwise (e.g. a scalar, a string), so that it becomes a compile-time constant. Neural networks themselves (MLPs, invertible networks, etc.) are no exception: they derive from `ScimbaModule`, itself a subclass of `ScimbaPytree`, so their weights, being `JAX` arrays, fall into `children` through the same automatic rule as any other differentiable attribute in the library.

Despite this automatic classification, two markers allow overriding it explicitly, one in each direction. A `dynamic(...)` marker places a field within `children` even when its current value carries no `JAX` array. Such a field is traced by `JAX` rather than treated as a compile-time constant: it can change from one call to the next (e.g. a loss history updated at each training step) without triggering a recompilation, whereas a changing value locked within `aux_data` would silently force one at every step. Conversely, a `static(...)` marker forces a field into `aux_data` even though it holds a `JAX` array, when that array must remain a compile-time constant rather than being traced.

Being placed within `children` simply indicates that a leaf is subject to `JAX` transformations (differentiation, vectorization); it says nothing about whether an optimizer should modify it. This second question, which concerns its role rather than its location (`children` or `aux_data`), is handled by a separate marker, `trainable(...)`. For example, `weight = trainable(True)` on an `MLP` declares that the `weight` field is optimized, whereas `False` freezes an entire subtree, regardless of the declarations of its own fields. This distinction is exploited when differentiating with respect to a `ScimbaPytree`. The object is first split by an automatic partition (`auto_partition`) into its active leaves, i.e. those declared `trainable(True)`, and a frozen part gathering all the others. The gradient, and hence the optimizer, only concerns the active leaves, and the frozen part is left untouched. A field can therefore be dynamic, i.e. traced by `JAX` so that changing its value does not trigger any recompilation, without being differentiated (e.g. a loss history).

In `ScimBa`, almost every module inherits from the `ScimbaPytree` class, as indicated by the tree icon in the different architecture figures (e.g. Figure 4.1), and is therefore fully differentiable and composable with the `JAX` transformations presented above. Most of them rely entirely on the automatic classification described above: the explicit markers `dynamic`, `static`, and `trainable` are only used when a field needs one of these behaviors spelled out. A few modules are, however, deliberate exceptions. For instance, the quadrature rules `UnitSquareTensorized` (Section 4.2.1) keep the points and weights fixed once and for all on the reference cell; they are never traced, batched or differentiated. Registering them as a pytree would therefore offer no advantage. Non-linear solver strategies `NewtonSolver` and `NewtonKrylov` (Sections 4.6.1 and 4.6.2) are configuration-based rather than state-based, and are therefore defined as simple, immutable `dataclass` objects, and are thus hashable. Unlike an ordinary `Python` object, which compares and hashes based on its identity (its memory address), two `dataclass` instances constructed with identical fields are considered equal and share the same hash. They can therefore be passed as non-differentiable, static arguments of compiled functions without triggering any unnecessary recompilation.

Having presented the principles of differentiable programming, the `JAX` library implementing them, and the pytree mechanism through which `ScimBa` objects are made differentiable, we now turn to the first concrete modules built on these foundations: the geometry and integration modules. A quadrature, defined once and for all on the reference element, is combined with a mapping, which may be learnable, and with the mesh it deforms, in order to integrate over a physical domain whose geometry can itself be learned.

4.2. Geometry and integration modules

Numerically integrating a function over the physical domain $\Omega \subset \mathbb{R}^d$ (d the space dimension) is a core operation of Galerkin methods, whose weak forms are sums of integrals over the cells and the faces of the mesh. This requires, on the one hand, a geometric description of Ω and, on the other hand, quadrature points and weights on the reference element, used to actually discretize the integral on this geometry. Classically, this is done by explicitly storing a mesh of the domain Ω (vertex coordinates,

connectivity, etc.) and applying, on each cell, a local quadrature rule together with the geometric transformation from a reference element to that cell. In `ScimBa`, this is done differently, precisely to allow the geometry itself to be learned: rather than storing the physical mesh explicitly, we keep a fixed logical mesh and derive the physical one, by pushing it forward through a differentiable mapping towards the target domain.

This design choice is what motivates splitting the geometric and integration machinery into three dedicated modules, introduced below. We start in Section 4.2.1 with the quadrature module, which allows the construction of quadrature points and weights on the reference element and its faces, independently of any geometric description. We then turn to how the geometry itself is represented implicitly through a mapping: in Section 4.2.2, we define the mapping module, which allows the transformation of points from the logical domain to the physical domain via a composition of analytical and/or learnable mappings; and in Section 4.2.3, we define the mesh module, which organizes the cells and faces of a fixed logical mesh and combines them with the quadratures and mappings defined previously to perform the numerical integration on the (never explicitly stored) physical mesh.

4.2.1. Quadrature module

The first tool required for the implementation of the `ScimBa` library is the quadrature module, which allows the construction of quadrature points and weights on the reference hypercube $\hat{K} := [0,1]^d$ and its faces. To do so, we consider well-known 1D quadrature rules (for example Gauss–Legendre or Chebyshev) on the interval $[0,1]$ to construct 1D quadrature points and weights. These 1D points and weights are then tensorized to construct volumetric and surface points and weights on $\hat{K}$ and its faces. These quadratures are implemented in the `UnitSquareTensorized` module.

Volumetric quadrature. For an arbitrary function $\mathrm{f} : \hat{K} \to \mathbb{R}$ (unrelated to the PDE source term), the volumetric integral of f over the reference element is approximated by the following quadrature formula:

$$\int_{\hat{K}} \mathrm{f}(\hat{\boldsymbol{\xi}})\, d\hat{\boldsymbol{\xi}} \approx \sum_{i=1}^{q^d} \hat{w}_i\, \mathrm{f}(\hat{\boldsymbol{\xi}}_i),$$

where q^d is the total number of volumetric quadrature points with q the number of 1D quadrature points per direction, $\hat{w}_i$ and $\hat{\boldsymbol{\xi}}_i$ are respectively the volumetric quadrature weights and points constructed by tensor product.

Surface quadrature. In the same way, `UnitSquareTensorized` defines surface integrals by constructing a tensor-product quadrature on each face direction of the reference element. More precisely, we define a reference face $\hat{F}^{(\ell)}$ for each direction $\ell \in \{1, \dots, d\}$, corresponding to the hyperplane of $\hat{K}$ where the ℓ-th coordinate is fixed to 0, in other words:

$$\hat{F}^{(\ell)} := \{\hat{\boldsymbol{\xi}} = (\hat{\xi}_1, \hat{\xi}_2, \dots, \hat{\xi}_d) \in \hat{K} : \hat{\xi}_\ell = 0\}.$$

Thus, for an arbitrary function $\mathrm{f} : \hat{F}^{(\ell)} \to \mathbb{R}$ (unrelated to the PDE source term), the surface integral of f over the reference face $\hat{F}^{(\ell)}$ (associated with direction ℓ) is

approximated by the following quadrature formula:

$$\int_{\hat{F}^{(\ell)}} \mathrm{f}(\hat{\boldsymbol{\xi}})\, d\hat{\boldsymbol{\xi}} \approx \sum_{i=1}^{q^{d-1}} \hat{w}_i^{\mathrm{surf}}\, \mathrm{f}(\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}}),$$

where q^{d-1} is the total number of surface quadrature points for a reference face (a number independent of the direction ℓ), $\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}}$ are the surface quadrature points and $\hat{w}_i^{\mathrm{surf}}$ the surface quadrature weights (also independent of ℓ).

4.2.2. Mapping module

Before turning to the mesh of the physical domain, we introduce the `Mapping` module allowing the definition of a composition of mappings. This composition, denoted $\mathcal{G}_\theta : \widetilde{\Omega} \to \Omega$, is used to transform the quadrature points from the logical domain $\widetilde{\Omega} := [0,1]^d$ to the physical domain Ω via $\boldsymbol{x} := \mathcal{G}_\theta(\tilde{\boldsymbol{\xi}})$ with $\tilde{\boldsymbol{\xi}} \in \widetilde{\Omega}$ a logical point and $\boldsymbol{x} \in \Omega$ its image in the physical domain.

The mapping $\mathcal{G}_\theta$ can be defined as an ordered composition of N_{maps} invertible mappings by:

$$\boldsymbol{x} = \mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}) := \mathcal{G}_{\theta,N_{\mathrm{maps}}} \circ \mathcal{G}_{\theta,N_{\mathrm{maps}}-1} \circ \cdots \circ \mathcal{G}_{\theta,1}(\tilde{\boldsymbol{\xi}}), \tag{4.1}$$

with θ learnable parameters and where each $\mathcal{G}_{\theta,j}$, $j \in \{1,\dots,N_{\mathrm{maps}}\}$, is an invertible mapping of one of the following two types:

- Analytical mapping (`InvertibleFunction` module): a differentiable function defined analytically (e.g. trigonometric or polynomial function) whose inverse is also defined analytically;
- Learnable mapping (`InvertibleNet` module): an invertible neural network (for example a normalizing flow network [46, 47]) whose parameters are learned from data or an optimization task.

The invertibility of the mappings is needed, among other things, to recover the logical point associated with a given physical point, for instance to find the cell containing it.

> **Notation 4.2.** *By abuse of notation, we consider that each of the N_{maps} mappings of the composition depends on the same parameters θ. In practice, each mapping can be defined as an independent neural network with its own parameters or even be defined as an analytical mapping without learnable parameters.*

Each mapping $\mathcal{G}_{\theta,j}$ therefore acts between two successive domains of an intermediate chain $\Omega_0 := \widetilde{\Omega},\ \Omega_1,\ \dots,\ \Omega_{N_{\mathrm{maps}}} := \Omega$, i.e. $\mathcal{G}_{\theta,j} : \Omega_{j-1} \to \Omega_j$. We denote by $\boldsymbol{\xi}_j \in \Omega_j$ the point obtained after applying the first $j \in \{1,\dots,N_{\mathrm{maps}}\}$ mappings of the composition:

$$\boldsymbol{\xi}_0 := \tilde{\boldsymbol{\xi}}, \qquad \boldsymbol{\xi}_j := \mathcal{G}_{\theta,j}(\boldsymbol{\xi}_{j-1}). \tag{4.2}$$

Note that, after applying the N_{maps} mappings, we have $\boldsymbol{x} = \boldsymbol{\xi}_{N_{\mathrm{maps}}} = \mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}) \in \Omega$.

In Example 4.3, we illustrate an example of a composition with a learnable mapping, together with an analytical alternative to it.

Example 4.3 (Composition with a learnable mapping, and an analytical alternative)**.** *Let us define the target domain* $\Omega = \{\varphi < 0\}$ *from the level-set function (Section 2.2.3):*

$$\varphi(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_c\|_2^2 - r^2 - \frac{40 \times 0.07^2 \, \|\boldsymbol{x} - \boldsymbol{x}_c\|_2^2}{\left(\|\boldsymbol{x} - \boldsymbol{x}_c\|_2^2 + 0.07^2\right)^2} (d_x^3 - 3\, d_x\, d_y^2),$$

with $\|\cdot\|_2$ *the Euclidean norm,* $r := 0.3$, $\boldsymbol{x}_c := (0.5, 0.5)$ *the center of the unit square, and* $(d_x, d_y) := \boldsymbol{x} - \boldsymbol{x}_c$*. Geometrically,* φ *describes a disk of radius* r *centered at* $\boldsymbol{x}_c$*, perturbed by a 3-fold-symmetric bump to create a 3-lobed shape.*
Consider two compositions of $N_{maps} = 3$ *invertible mappings allowing the parametrization of a domain* Ω *of complex shape from the logical domain* $\widetilde{\Omega}$*. The first composition* $\mathcal{G}_\theta = \mathcal{G}_{\theta,3} \circ \mathcal{G}_2 \circ \mathcal{G}_1$ *is defined as a composition of two analytical mappings* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *and one learnable mapping* $\mathcal{G}_{\theta,3}$*, while the second composition* $\mathcal{G} := \mathcal{G}_3 \circ \mathcal{G}_2 \circ \mathcal{G}_1$ *is defined as a composition of three analytical mappings. More precisely, we define the mappings as follows:*

- $\mathcal{G}_1 : \widetilde{\Omega} \to \mathbb{D}$ *is an analytical mapping (*`InvertibleFunction`*) sending the reference square onto the unit disk* $\mathbb{D} := \Omega_1 = \{\boldsymbol{\xi}_1 \in \mathbb{R}^2 : \|\boldsymbol{\xi}_1\|_2 \leq 1\}$ *via the Fernandez-Guasti formula [61]:*

$$\boldsymbol{\xi}_1 = \mathcal{G}_1(\tilde{\boldsymbol{\xi}}) := \begin{pmatrix} (2\tilde{\xi}_1 - 1)\sqrt{1 - \frac{(2\tilde{\xi}_2 - 1)^2}{2}} \\ (2\tilde{\xi}_2 - 1)\sqrt{1 - \frac{(2\tilde{\xi}_1 - 1)^2}{2}} \end{pmatrix},$$

  with $\boldsymbol{\xi}_0 = \tilde{\boldsymbol{\xi}} := (\tilde{\xi}_1, \tilde{\xi}_2) \in \Omega_0 = \widetilde{\Omega}$ *a logical point;*

- $\mathcal{G}_2 : \mathbb{D} \to \mathbb{D}_r(\boldsymbol{x}_c)$ *is an analytical, affine mapping (*`InvertibleFunction`*) recentering and rescaling the unit disk onto the disk* $\mathbb{D}_r(\boldsymbol{x}_c)$ *of radius* r *centered at* $\boldsymbol{x}_c$*, i.e. already at the scale and location of* Ω*, via* $\boldsymbol{\xi}_2 = \mathcal{G}_2(\boldsymbol{\xi}_1) := r\, \boldsymbol{\xi}_1 + \boldsymbol{x}_c$*;*

- *The last mapping encodes the 3-lobed deformation for* Ω *of a disk already at the right place and scale* $\mathbb{D}_r(\boldsymbol{x}_c)$*, either as a learnable mapping* $\mathcal{G}_{\theta,3}$ *(*`InvertibleNet`*), implemented as a normalizing flow composed of affine coupling layers [47], or as a closed-form analytical mapping* $\mathcal{G}_3$ *(*`InvertibleFunction`*), introduced below.*

Learning the mapping $\mathcal{G}_{\theta,3}$**.** *Separating the affine step* $\mathcal{G}_2$ *from* $\mathcal{G}_{\theta,3}$ *considerably eases the learning task: the network only has to represent the residual lobed deformation of a disk already correctly placed and scaled, rather than the translation and rescaling as well.*
The parameters θ *of the mapping* $\mathcal{G}_{\theta,3}$ *are learned by minimizing the following loss, which combines an interior loss and a weighted boundary loss:*

$$\mathcal{J}(\theta) := \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \|\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}_{data}^{(i)}) - \boldsymbol{x}_{data}^{(i)}\|_2^2 + 100 \cdot \frac{1}{N_{\partial data}} \sum_{i=1}^{N_{\partial data}} \|\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}_{\partial data}^{(i)}) - \boldsymbol{x}_{\partial data}^{(i)}\|_2^2,$$

where $(\tilde{\boldsymbol{\xi}}_{data}^{(i)}, \boldsymbol{x}_{data}^{(i)})$*,* $i \in \{1, \ldots, N_{data}\}$*, are pairs of interior points, respectively in the logical domain* $\widetilde{\Omega}$ *and the target domain* Ω*, and* $(\tilde{\boldsymbol{\xi}}_{\partial data}^{(i)}, \boldsymbol{x}_{\partial data}^{(i)})$*,* $i \in \{1, \ldots, N_{\partial data}\}$*, are pairs of points on* $\partial\widetilde{\Omega}$ *and* Γ*.*

The training data includes $N_{data} = 5\,000$ interior points drawn uniformly in the logical domain $\widetilde{\Omega}$ and $N_{\partial data} = 4 \times 300 = 1\,200$ points on the 4 boundaries of $\partial\widetilde{\Omega}$; the target data (interior and boundary) is generated using the closed-form mapping $\mathcal{G} = \mathcal{G}_3 \circ \mathcal{G}_2 \circ \mathcal{G}_1$ introduced below. The network $\mathcal{G}_{\theta,3}$ is a normalizing flow composed of 2 affine coupling layers, each equipped with 2 hidden layers of size 32 (for a total of 9 224 parameters) and a tanh *activation. The optimization is performed with a matrix-free energy natural gradient (ENG) method [98], over 100 epochs (batches of 512 data points). The training time is about 2 minutes and 30 seconds on an NVIDIA RTX 2000 GPU (8 GB VRAM).*

An analytical alternative. *Instead of learning the last mapping, it can also be built entirely analytically. The boundary of Ω has no closed form because it is defined implicitly by $\varphi = 0$ using the level-set function. We seek to approximate its radius ρ from $\boldsymbol{x}_c$, as a function of the polar angle, via a truncated Fourier series using cosine functions, whose coefficients are fitted by least squares on the values of ρ, computed at a discrete set of angles as the radial root of φ. Without going into detail here, $\mathcal{G}_3$ is then simply the mapping that stretches each point of $\mathbb{D}_r(\boldsymbol{x}_c)$ radially so that, on the boundary circle, this stretch reproduces the actual boundary of Ω up to the Fourier truncation, while decaying smoothly to no stretch at the center $\boldsymbol{x}_c$.*

Comparison of the two mappings. *Both target domains, and their meshes (Section 4.2.3), are shown side by side in Figure 4.2. Intermediate domains are also shown, as well as the training loss curves of $\mathcal{G}_{\theta,3}$.*

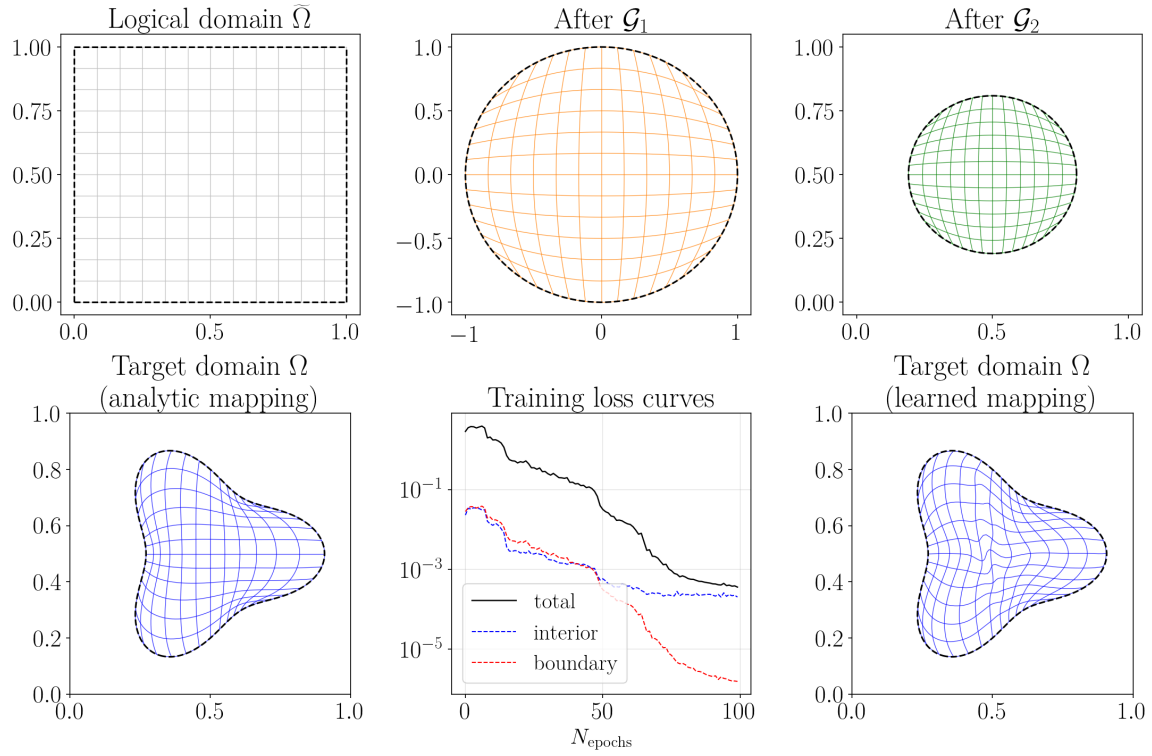


Figure 4.2.: Composition towards the 3-lobed target domain Ω (Example 4.3). Top – the mesh of the logical domain $\widetilde{\Omega}$, after the analytical mapping $\mathcal{G}_1$, and after the analytical mapping $\mathcal{G}_2$. Bottom – the resulting target domain, obtained either through the closed-form correction $\mathcal{G}_3$ (left) or through the learned correction $\mathcal{G}_{\theta,3}$ (right), and the training loss curves of $\mathcal{G}_{\theta,3}$ (middle). The boundary of each intermediate domain is represented in dashed lines (black).

When fitted directly to Γ, $\mathcal{G}_3$ reproduces the actual boundary of Ω with a mean error of the order of $1.8 \cdot 10^{-6}$, approximately three orders of magnitude smaller than the mean boundary error of $\mathcal{G}_{\theta,3}$, which is $1.1 \cdot 10^{-3}$. We compare the two mappings quantitatively in Section 4.10.6, through the convergence of a Poisson problem solved by FEM on the resulting (curved) meshes.

Jacobian of the mapping. In Section 4.2.3, we will see that the computation of the Jacobian of the composition of mappings $\mathcal{G}_\theta$ is necessary to transform the quadrature weights from the logical domain to the physical domain. More precisely, using the intermediate points $\boldsymbol{\xi}_0, \dots, \boldsymbol{\xi}_{N_{\text{maps}}}$ of (4.2), the Jacobian of the composition $\mathcal{G}_\theta$ (4.1) is computed by the chain rule, using the intermediate Jacobians of the composed mappings:

$$J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}}) = J_{\mathcal{G}_{\theta,N_{\text{maps}}}}(\boldsymbol{\xi}_{N_{\text{maps}}-1}) \cdot J_{\mathcal{G}_{\theta,N_{\text{maps}}-1}}(\boldsymbol{\xi}_{N_{\text{maps}}-2}) \cdot \dots \cdot J_{\mathcal{G}_{\theta,1}}(\boldsymbol{\xi}_0). \tag{4.3}$$

We also need the determinant of the Jacobian to transform the quadrature weights, simply defined as the product of the determinants of the intermediate Jacobians:

$$|\det J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})| = \prod_{j=1}^{N_{\text{maps}}} |\det J_{\mathcal{G}_{\theta,j}}(\boldsymbol{\xi}_{j-1})|.$$

Computing these Jacobians and their determinants at every quadrature point is costly, in particular when the composition involves many mappings. For a fixed mapping, they are therefore computed once and stored by default in the `Mesh` (as children of the pytree, see Section 4.1), and simply read afterwards. When the mapping is learnable, such a stored table would freeze the initial geometry, so they are recomputed at each evaluation instead.

4.2.3. Mesh module

We now define the `Mesh` module allowing the management of the cells and faces of the physical mesh $\mathcal{T}_h$ and the application of the quadratures and mappings defined previously. The goal of this module is to allow numerical integration of functions (residuals, bilinear forms, fluxes, etc.) over the physical domain Ω (potentially curved) by splitting the domain into cells (and faces) on which to apply a local quadrature. This requires indexing these cells/faces in a structured way so that we can navigate between them (neighborhood, topological affiliation), but also so that we can transport the quadrature points/weights from the reference element to the physical domain via the composition of mappings introduced in Section 4.2.2. The physical mesh $\mathcal{T}_h$ is never

stored explicitly: any necessary information is computed on the fly from the logical mesh $\widetilde{\mathcal{T}}_h$ and transferred to the physical mesh via the mapping $\mathcal{G}_\theta$. We successively describe the cells and the faces of this mesh, the physical normals, and the mapping chains used to transport the quadrature points and weights, from which we deduce the volumetric and surface integration formulas on the physical mesh. Most of these constructions are standard in finite element methods (change of variables, Nanson's formula, etc.). They are nevertheless detailed here because the additional mapping $\mathcal{G}_\theta$ makes the transport go through three spaces (reference, logical and physical) instead of two, so that the Jacobians must be composed and, when the mapping is learnable, recomputed at each evaluation.

Mesh cells. We then define a uniform mesh $\widetilde{\mathcal{T}}_h$ on the logical domain $\widetilde{\Omega} = [0,1]^d$, split into N_ℓ cells in the direction $\ell \in \{1,\dots,d\}$ where h represents the characteristic size of the mesh. We can then denote $\boldsymbol{N}_{\text{cells}}$ the tuple of the number of cells per direction and N_{cells} the total number of cells, defined by:

$$\boldsymbol{N}_{\text{cells}} := (N_1,\dots,N_d), \quad \text{and} \quad N_{\text{cells}} := \prod_{\ell=1}^{d} N_\ell.$$

A cell of the logical mesh $\widetilde{\mathcal{T}}_h$ is indexed either by a flat index $c \in \{0,\dots,N_{\text{cells}}-1\}$, used to vectorize over cells (`vmap` over a 1D array of indices), or by a multi-index $\bar{\boldsymbol{c}}(c) := (\bar{c}_1,\dots,\bar{c}_d)$ with $\bar{c}_\ell \in \{0,\dots,N_\ell - 1\}$, used to navigate geometrically (neighbors, shared faces).

Each cell $\tilde{K}_c$ (where c is the flat index of the cell) is then defined as a hypercube of dimension d and length $1/N_\ell$ in direction $\ell \in \{1,\dots,d\}$. The physical mesh $\mathcal{T}_h$ is then defined as the image of $\widetilde{\mathcal{T}}_h$ by the mapping $\mathcal{G}_\theta$ defined in Section 4.2.2:

$$\mathcal{T}_h := \mathcal{G}_\theta(\widetilde{\mathcal{T}}_h) = \{K_c = \mathcal{G}_\theta(\tilde{K}_c) : \tilde{K}_c \in \widetilde{\mathcal{T}}_h\}.$$

Remark 4.4. *Even though $\mathcal{G}_\theta$ may be an arbitrarily non-linear deformation, or even a learned one (Example 4.3), the resulting mesh $\mathcal{T}_h$ remains structured: i.e. we know exactly which cell is adjacent to which other one thanks to the inheritance from the fixed logical grid $\widetilde{\mathcal{T}}_h$. Besides this structured mesh, the library also provides block-structured meshes (one structured patch per macro-cell, adjacent patches being coupled through their shared interface faces) and fully unstructured meshes (built from an explicit list of cells and nodes). These modules were developed by other contributors to* `ScimBa` *and fall outside the scope of this manuscript, in which we only use the structured mesh deformed by $\mathcal{G}_\theta$.*

Mesh faces. Faces, on the other hand, are $(d-1)$-dimensional hypersurfaces, organized into d groups according to their normal axis. The faces of group $\ell \in \{1,\dots,d\}$ are perpendicular to that same direction, which means that each layer (corresponding to a fixed position on axis ℓ) contains $\prod_{j\neq\ell} N_j$ faces.

Since there are $N_\ell + 1$ layers in each group ℓ, we have in total $(N_\ell+1)\cdot\prod_{j\neq\ell} N_j$ faces in group ℓ, and thus the total number of faces (i.e. the sum over the groups) is given by:

$$N_{\text{faces}} := \sum_{\ell=1}^{d} (N_\ell + 1) \cdot \prod_{j\neq\ell} N_j.$$

In our numerical approximation methods (in particular for DG approaches), it is necessary to distinguish the interior and boundary faces of the mesh. Faces are therefore classified into two categories:

- Interior: faces shared by two cells (i.e. faces that are not on the boundary of the domain);
- Boundary: faces that are on the boundary of the domain (i.e. faces that are shared by only one cell).

Thus, in the same way as in the section presenting the DG approaches (Section 2.2.2), we denote by $\widetilde{\mathcal{F}}_h := \widetilde{\mathcal{F}}_h^I \cup \widetilde{\mathcal{F}}_h^B$ the set of all faces in the logical mesh $\widetilde{\mathcal{T}}_h$, with $\widetilde{\mathcal{F}}_h^I$ the set of interior faces and $\widetilde{\mathcal{F}}_h^B$ the set of boundary faces. We can then recover all the faces $\mathcal{F}_h$ of the physical mesh by applying the mapping $\mathcal{G}_\theta$ to the logical faces, in exactly the same way as for the cells.

Similarly to cells, a face is indexed either by a flat index $k \in \{0, \dots, N_{\text{faces}} - 1\}$, which runs through the groups one after the other, or by a multi-index $\bar{\boldsymbol{k}}(k) := (\ell;\, \bar{k}_1, \dots, \bar{k}_d)$ of size $d+1$, where $F_k = \mathcal{G}_\theta(\tilde{F}_k) \in \mathcal{F}_h$ with $\tilde{F}_k \in \widetilde{\mathcal{F}}_h$. The components $\bar{k}_j$ give the position of the face in each direction, with $\bar{k}_j \in \{0, \dots, N_j - 1\}$ for $j \neq \ell$ and $\bar{k}_\ell \in \{0, \dots, N_\ell\}$, and the leading component ℓ indicates the group of the face (i.e. the direction of its normal), since the same positions can occur in several groups.

Physical normals. Unlike the mapping Jacobian introduced in Section 4.2.2, physical normals are not used in the volumetric integration formulas below. However, they will be required to evaluate the numerical fluxes at the mesh interfaces in the DG scheme (both interior and at the boundary), as indicated in Section 2.2.2.2, as well as for the weak imposition of Neumann or Robin boundary conditions at the boundary, in both the DG and FE schemes.

In the logical domain, the (positive) normal of a face $\tilde{F}_k \in \widetilde{\mathcal{F}}_h$ of group ℓ is simply the canonical vector $\boldsymbol{e}_\ell = (0, \dots, 0, 1, 0, \dots, 0)^T$ with a 1 at position ℓ: the outward normal on $\tilde{F}_k$, relative to whichever of its (at most two) neighboring cells is considered, is then $\pm\boldsymbol{e}_\ell$. Thus, defining $K \in \mathcal{T}_h$ as the physical cell corresponding to a logical cell $\tilde{K} \in \widetilde{\mathcal{T}}_h$ neighboring $\tilde{F}_k$ (as introduced above), the physical outward unit normal $\boldsymbol{n}_K(\boldsymbol{x})$ at a point $\boldsymbol{x} = \mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}) \in F_k \subset \partial K$ is then obtained via Nanson's formula [38]:

$$\boldsymbol{n}_K(\boldsymbol{x}) = \pm \text{sign}(\det J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})) \frac{J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})^{-T} \boldsymbol{e}_\ell}{\| J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})^{-T} \boldsymbol{e}_\ell \|_2}, \tag{4.4}$$

with $\| \cdot \|_2$ the Euclidean norm and $J_{\mathcal{G}_\theta}$ the Jacobian (4.3) of the mapping $\mathcal{G}_\theta$. The sign of the determinant of the Jacobian is used to ensure that the normal is oriented outward (i.e. pointing outside the physical cell K), according to the mapping orientation.

The following Example 4.5 illustrates Nanson's formula (4.4) on the learnable mapping introduced earlier in this chapter (Example 4.3), and shows the resulting physical normals on the boundary of the target domain Ω.

Example 4.5 (Physical normals)**.** *Let us return to Example 4.3 with the learned mapping $\mathcal{G}_\theta = \mathcal{G}_{\theta,3} \circ \mathcal{G}_2 \circ \mathcal{G}_1$ towards the 3-lobed target domain Ω.*

Analytical sanity check. *Let us first verify (4.4) analytically on the first mapping $\mathcal{G}_1 : \widetilde{\Omega} \to \mathbb{D}$ alone, which sends the logical domain $\widetilde{\Omega} = [0,1]^2$ onto the unit disk $\mathbb{D}$, and check that we recover the classical outward normal to the unit circle. Consider the right face (group $\ell = 1$) of the logical domain $\widetilde{\Omega}$ defined by $\tilde{\xi}_1 = 1$ (with logical normal $\boldsymbol{e}_1$), and denote by $\tilde{K} \in \widetilde{\mathcal{T}}_h$ the corresponding boundary cell and by $K = \mathcal{G}_1(\tilde{K}) \in \mathcal{T}_h$ its physical image. Write $v := 2\tilde{\xi}_2 - 1 \in [-1,1]$ as a shorthand for the second logical coordinate. On this face, the Fernandez-Guasti mapping and its Jacobian read:*

$$\boldsymbol{x} = \mathcal{G}_1(\tilde{\boldsymbol{\xi}})\big|_{\tilde{\xi}_1=1} = \begin{pmatrix} \sqrt{1-\frac{v^2}{2}} \\ \frac{v}{\sqrt{2}} \end{pmatrix}, \quad \text{and} \quad J_{\mathcal{G}_1}(\tilde{\boldsymbol{\xi}})\big|_{\tilde{\xi}_1=1} = \begin{pmatrix} 2\sqrt{1-\frac{v^2}{2}} & -\dfrac{v}{\sqrt{1-\frac{v^2}{2}}} \\ -\sqrt{2}\,v & \sqrt{2} \end{pmatrix},$$

with the positive determinant, considering $v \in (-1,1)$, defined by:

$$\det J_{\mathcal{G}_1} = \frac{2\sqrt{2}\,(1-v^2)}{\sqrt{1-\frac{v^2}{2}}}.$$

A direct computation then gives:

$$J_{\mathcal{G}_1}(\tilde{\boldsymbol{\xi}})^{-T}\,\boldsymbol{e}_1\Big|_{\tilde{\xi}_1=1} = \frac{1}{2(1-v^2)}\,\boldsymbol{x},$$

i.e. $J_{\mathcal{G}_1}^{-T}\boldsymbol{e}_1$ is collinear with $\boldsymbol{x}$, with a positive scalar factor since $v \in (-1,1)$. Since $\boldsymbol{x}$ lies on the unit circle ($\|\boldsymbol{x}\|_2 = 1$) and $\det J_{\mathcal{G}_1} > 0$, Nanson's formula simplifies to:

$$\boldsymbol{n}_K(\boldsymbol{x}) = \frac{J_{\mathcal{G}_1}^{-T}\boldsymbol{e}_1}{\|J_{\mathcal{G}_1}^{-T}\boldsymbol{e}_1\|_2} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2} = \boldsymbol{x} = (x_1, x_2),$$

which is exactly the classical outward unit normal to the unit circle at a point $\boldsymbol{x}$.

Physical normals of the learnable mapping. *Let us now return to the full learnable mapping $\mathcal{G}_\theta$. In Figure 4.3, we illustrate the physical outward normals $\boldsymbol{n}_K$ of all boundary cells $K \in \mathcal{T}_h$, evaluated at their respective boundary faces; computed via Nanson's formula (4.4) from the logical normals $\pm\boldsymbol{e}_\ell$ and the Jacobian of the mapping $\mathcal{G}_\theta$ defined in (4.3).*

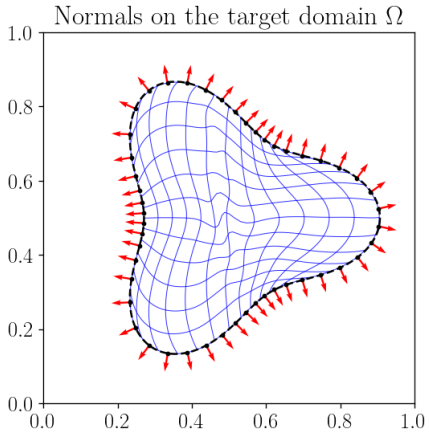


Figure 4.3.: Visualization of the physical normals of the learnable mapping $\mathcal{G}_\theta = \mathcal{G}_{\theta,3} \circ \mathcal{G}_2 \circ \mathcal{G}_1$ towards the 3-lobed target domain Ω (Examples 4.3 and 4.5).

Mapping chains. In this section, we have seen that the mesh $\mathcal{T}_h$ on the physical domain is obtained as the image of the mesh $\widetilde{\mathcal{T}}_h$ on the logical domain via the mapping $\mathcal{G}_\theta$. To numerically integrate a function over a physical cell or face, one must be able to transport the quadrature points and weights, defined on the reference element $\hat{K}$ (or the reference face $\hat{F}^{(\ell)}$ associated with the group ℓ), to the physical domain Ω. This transport is therefore done in two successive steps:

- a local affine transformation, specific to each cell (resp. face), which links the reference element to the corresponding cell (resp. face) of the logical mesh, which performs a mapping between two hypercubes of dimension d (resp. $d-1$);
- the global mapping $\mathcal{G}_\theta$, common to the whole mesh, which sends the logical mesh onto the physical one.

Thus, we define two mapping chains (volumetric and surface) to transform the quadrature points from the reference element (or the reference face) towards the physical domain. We denote $T_{\tilde{K}_c} : \hat{K} \to \tilde{K}_c \subset \widetilde{\Omega}$ the affine transformation that sends the reference element $\hat{K}$ onto the cell $\tilde{K}_c$ (associated with flat index c) of the logical mesh $\widetilde{\mathcal{T}}_h$. The transformation, as well as its inverse, is defined by:

$$\tilde{\boldsymbol{\xi}} = T_{\tilde{K}_c}(\hat{\boldsymbol{\xi}}) := \frac{\hat{\boldsymbol{\xi}} + \bar{\boldsymbol{c}}(c)}{\boldsymbol{N}_{\text{cells}}}, \quad \text{and} \quad \hat{\boldsymbol{\xi}} = T^{-1}_{\tilde{K}_c}(\tilde{\boldsymbol{\xi}}) := \tilde{\boldsymbol{\xi}} \cdot \boldsymbol{N}_{\text{cells}} - \bar{\boldsymbol{c}}(c),$$

where the division and multiplication are done component-wise. The volumetric quadrature points then pass through three successive spaces as illustrated in Figure 4.4. We can then denote the composition of the two volumetric transformations by:

$$T_{K_c} := \mathcal{G}_\theta \circ T_{\tilde{K}_c} : \hat{K} \to K_c \subset \Omega, \tag{4.5}$$

i.e. the passage from the reference element to the physical domain via the logical mesh.

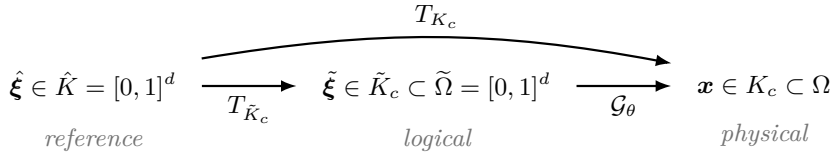


Figure 4.4.: Mapping chain for volumetric points from the reference element $\hat{K}$ to the physical cell $K_c \in \mathcal{T}_h$ via the logical cell $\tilde{K}_c \in \widetilde{\mathcal{T}}_h$.

Remark 4.6. *Classically, the physical domain Ω is meshed directly, without going through a logical domain: the mesh $\mathcal{T}_h$ is built by positioning each cell to fit the geometry of Ω. Only a single geometric transformation per element is therefore needed, directly from the reference element to the physical element. Generally speaking, there is no additional global mapping $\mathcal{G}_\theta$ and the mesh itself does all the work of conforming to Ω. In `ScimBa`, this is different because the logical mesh $\widetilde{\mathcal{T}}_h$ remains a fixed uniform mesh of the hypercube $\widetilde{\Omega} = [0,1]^d$ (never rebuilt or remeshed). This is precisely what allows differentiability and learning of the geometry without having to regenerate a mesh at each iteration. Two steps are therefore needed to reach the physical domain, which is what the composed transformation T_{K_c} does.*

For faces, we proceed in a similar way. We denote $T_{\tilde{F}_k} : \hat{F}^{(\ell)} \to \tilde{F}_k \subset \widetilde{\Omega}$ the affine transformation that sends the reference face $\hat{F}^{(\ell)}$ (associated with group ℓ) onto the face $\tilde{F}_k$ of the logical mesh $\widetilde{\mathcal{T}}_h$. Writing $\bar{\boldsymbol{k}}(k) = (\ell; \bar{k}_1, \dots, \bar{k}_d)$, only the positional part $(\bar{k}_1, \dots, \bar{k}_d)$ enters the transformation below, the group label ℓ itself only serving to identify which reference face $\hat{F}^{(\ell)}$ is being mapped. The transformation, as well as its inverse, is defined by:

$$\tilde{\boldsymbol{\xi}} = T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}) := \frac{\hat{\boldsymbol{\xi}} + (\bar{k}_1, \dots, \bar{k}_d)}{\boldsymbol{N}_{\text{cells}}}, \quad \text{and} \quad \hat{\boldsymbol{\xi}} = T^{-1}_{\tilde{F}_k}(\tilde{\boldsymbol{\xi}}) := \tilde{\boldsymbol{\xi}} \cdot \boldsymbol{N}_{\text{cells}} - (\bar{k}_1, \dots, \bar{k}_d),$$

where the division and multiplication are also done component-wise. The surface quadrature points (living in a hyperplane of the volumetric space) then pass through three successive spaces as illustrated in Figure 4.5. We can then denote the composition of the two surface transformations by:

$$T_{F_k} := \mathcal{G}_\theta \circ T_{\tilde{F}_k} : \hat{F}^{(\ell)} \to F_k \subset \Omega,$$

i.e. the passage from the reference face (associated with group ℓ) to the physical domain via the logical mesh.

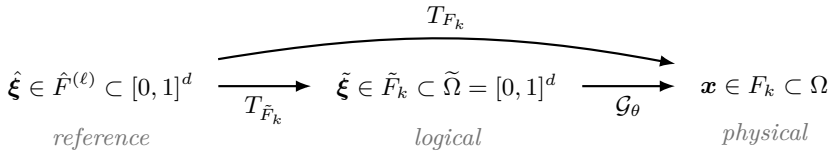


Figure 4.5.: Mapping chain for surface points from the reference face $\hat{F}^{(\ell)}$ to the physical face $F_k \in \mathcal{F}_h$ via the logical face $\tilde{F}_k \in \widetilde{\mathcal{F}}_h$.

Jacobians of the local affine transformations. In the same way as for the mapping $\mathcal{G}_\theta$, we will see later that we need to compute the Jacobians of the local transformations $T_{\tilde{K}_c}$ and $T_{\tilde{F}_k}$. Thus, by expanding the volumetric transformation component-wise, we have for the j-th component, $j \in \{1, \dots, d\}$:

$$(T_{\tilde{K}_c}(\hat{\boldsymbol{\xi}}))_j = \frac{\hat{\xi}_j + \bar{c}_j}{N_j},$$

with N_j the number of cells in direction j, i.e. the j-th component of $\boldsymbol{N}_{\text{cells}}$. Its Jacobian is therefore a constant diagonal matrix (with respect to $\hat{\boldsymbol{\xi}}$), as is its determinant, both defined by:

$$J_{T_{\tilde{K}_c}} := \operatorname{diag}\left(\frac{1}{N_1}, \frac{1}{N_2}, \dots, \frac{1}{N_d}\right), \quad \text{and} \quad |\det J_{T_{\tilde{K}_c}}| = \prod_{j=1}^{d} \frac{1}{N_j} = \frac{1}{N_{\text{cells}}}.$$

In the same way, by expanding the surface transformation component-wise, we have for the j-th component:

$$(T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}))_j = \frac{\hat{\xi}_j + \bar{k}_j}{N_j}.$$

However, the surface integral is over a $(d-1)$-dimensional manifold, so only the directions tangent to the reference face $\hat{F}^{(\ell)}$ matter. We define the tangential Jacobian as the $d \times (d-1)$ matrix obtained by removing the ℓ-th column of the (diagonal) Jacobian of $T_{\tilde{F}_k}$, that is:

$$J^{\mathrm{tan}}_{T_{\tilde{F}_k}} := \mathrm{diag}\left(\frac{1}{N_j}\right)_{j\neq\ell} \in \mathbb{R}^{d\times(d-1)}.$$

Since the columns are orthogonal, the Gram determinant (generalizing the notion of determinant to rectangular matrices) is:

$$|\det J^{\mathrm{tan}}_{T_{\tilde{F}_k}}| := \sqrt{\det\left((J^{\mathrm{tan}}_{T_{\tilde{F}_k}})^T J^{\mathrm{tan}}_{T_{\tilde{F}_k}}\right)} = \prod_{j\neq\ell}\frac{1}{N_j} = \frac{N_\ell}{N_{\mathrm{cells}}}.$$

Volumetric integration (d-dimensional). Equipped with these Jacobians, we can now perform the change of variables needed to integrate a function over a physical cell or face. Suppose here that we want to integrate a function $\mathrm{f}:\Omega\to\mathbb{R}$ (unrelated to the PDE source term) over a physical cell $K_c \in \mathcal{T}_h$. The change of variables on the physical cell K_c is done in two steps:

$$\begin{aligned}\int_{K_c}\mathrm{f}(\boldsymbol{x})\,d\boldsymbol{x} &\underbrace{=}_{\boldsymbol{x}=\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}})}\int_{\tilde{K}_c}\mathrm{f}(\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}))\,|\det J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})|\,d\tilde{\boldsymbol{\xi}},\\ &\underbrace{=}_{\tilde{\boldsymbol{\xi}}=T_{\tilde{K}_c}(\hat{\boldsymbol{\xi}})}\frac{1}{N_{\mathrm{cells}}}\int_{\hat{K}}\mathrm{f}(T_{K_c}(\hat{\boldsymbol{\xi}}))\,|\det J_{\mathcal{G}_\theta}(T_{\tilde{K}_c}(\hat{\boldsymbol{\xi}}))|\,d\hat{\boldsymbol{\xi}}.\end{aligned}$$

We can then discretize by quadrature on the reference element $\hat{K}$, as defined in Section 4.2.1. We obtain:

$$\int_{K_c}\mathrm{f}(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{i=1}^{q^d} w_{c,i}\,\mathrm{f}(T_{K_c}(\hat{\boldsymbol{\xi}}_i)),$$

where $w_{c,i} := \frac{1}{N_{\mathrm{cells}}}\,\hat{w}_i\,|\det J_{\mathcal{G}_\theta}(T_{\tilde{K}_c}(\hat{\boldsymbol{\xi}}_i))|$ are the corrected weights associated with the cell K_c and $T_{K_c}(\hat{\boldsymbol{\xi}}_i)$ are the physical volumetric quadrature points.

Surface integration ($(d-1)$-dimensional). Suppose now that we want to integrate a function $\mathrm{f}:\Omega\to\mathbb{R}$ over a physical face $F_k \in \mathcal{F}_h$. The change of variables on this physical face F_k is also done in two steps. Unlike the volumetric case, the physical surface element also involves Nanson's factor $\|J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})^{-T}\boldsymbol{e}_\ell\|_2$ in addition to the Jacobian determinant. As a consequence of Nanson's formula [38] (already used in (4.4) for the physical normal), we have:

$$\begin{aligned}\int_{F_k}\mathrm{f}(\boldsymbol{x})\,d\boldsymbol{x} &\underbrace{=}_{\boldsymbol{x}=\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}})}\int_{\tilde{F}_k}\mathrm{f}(\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}))\,|\det J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})|\,\|J_{\mathcal{G}_\theta}(\tilde{\boldsymbol{\xi}})^{-T}\boldsymbol{e}_\ell\|_2\,d\tilde{\boldsymbol{\xi}},\\ &\underbrace{=}_{\tilde{\boldsymbol{\xi}}=T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}})}\frac{N_\ell}{N_{\mathrm{cells}}}\int_{\hat{F}^{(\ell)}}\mathrm{f}(T_{F_k}(\hat{\boldsymbol{\xi}}))\,|\det J_{\mathcal{G}_\theta}(T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}))|\,\|J_{\mathcal{G}_\theta}(T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}))^{-T}\boldsymbol{e}_\ell\|_2\,d\hat{\boldsymbol{\xi}},\end{aligned}$$

with $\|\cdot\|_2$ the Euclidean norm and $\boldsymbol{e}_\ell$ the canonical vector associated with the group ℓ of the face F_k. Example 4.8 illustrates on a simple 2D example why Nanson's factor $\|J_{\mathcal{G}_\theta}^{-T}\boldsymbol{e}_\ell\|_2$, in addition to $|\det J_{\mathcal{G}_\theta}|$, is needed.

Notation 4.7. *The integrals over the faces F_k (physical), $\tilde{F}_k$ (logical) and $\hat{F}^{(\ell)}$ (reference) above are surface integrals, i.e. with respect to the $(d-1)$-dimensional surface measure on these faces, and not the d-dimensional volumetric measure used in the previous paragraph. For simplicity, we keep the same notation ($\boldsymbol{x}$, $\tilde{\boldsymbol{\xi}}$, $\hat{\boldsymbol{\xi}}$, $d\boldsymbol{x}$, $d\tilde{\boldsymbol{\xi}}$, $d\hat{\boldsymbol{\xi}}$) as in the volumetric case rather than introducing a dedicated surface-measure notation.*

We can then discretize by quadrature on the reference face $\hat{F}^{(\ell)}$, as defined in Section 4.2.1. We obtain:

$$\int_{F_k} \mathrm{f}(\boldsymbol{x})\, d\boldsymbol{x} \approx \sum_{i=1}^{q^{d-1}} w_{k,i}^{\mathrm{surf}}\, \mathrm{f}(T_{F_k}(\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}})),$$

where $w_{k,i}^{\mathrm{surf}} := \dfrac{N_\ell}{N_{\mathrm{cells}}}\, \hat{w}_i^{\mathrm{surf}}\, |\det J_{\mathcal{G}_\theta}(T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}}))|\, \|J_{\mathcal{G}_\theta}(T_{\tilde{F}_k}(\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}}))^{-T}\boldsymbol{e}_\ell\|_2$ are the corrected weights for the face F_k (associated with group ℓ) and $T_{F_k}(\hat{\boldsymbol{\xi}}_{\ell,i}^{\mathrm{surf}})$ are the physical surface quadrature points.

Example 4.8 (Nanson's factor)**.** *To illustrate the importance of Nanson's factor in the computation of surface integrals, consider a simple 2D test case with an anisotropic linear mapping. Take $\mathcal{G}_\theta(\tilde{\boldsymbol{\xi}}) = (L_1\tilde{\xi}_1, L_2\tilde{\xi}_2)$ with $L_1 \neq L_2$, which stretches the logical square $\widetilde{\Omega} = [0,1]^2$ into the rectangle $[0, L_1]\times[0, L_2]$. Then its Jacobian and determinant are:*

$$J_{\mathcal{G}_\theta} = \mathit{diag}(L_1, L_2), \quad \textit{and} \quad |\det J_{\mathcal{G}_\theta}| = L_1 L_2.$$

Consider a vertical face (belonging to group $\ell = 1$) of the logical domain (of length 1), for example the right face, i.e. $\tilde{\xi}_1 = 1$. After mapping, both coordinates are stretched, by L_1 and L_2 respectively, but only the tangential stretch L_2, along $\tilde{\xi}_2$, affects the length of the face, so this segment has length L_2. The correct scaling of this face is therefore L_2, but $|\det J_{\mathcal{G}_\theta}| = L_1 L_2 \neq L_2$, which means that the volumetric Jacobian overestimates by a factor L_1 (the stretch in the normal direction $\boldsymbol{e}_1$, which does not contribute to the area of the face).
Nanson's factor corrects this exactly because $J_{\mathcal{G}_\theta}^{-T}\boldsymbol{e}_1 = \boldsymbol{e}_1/L_1$, so $\|J_{\mathcal{G}_\theta}^{-T}\boldsymbol{e}_1\|_2 = 1/L_1$, and finally the correct scaling of the face is given by:

$$|\det J_{\mathcal{G}_\theta}| \cdot \|J_{\mathcal{G}_\theta}^{-T}\boldsymbol{e}_1\|_2 = L_1 L_2 \cdot \frac{1}{L_1} = L_2.$$

Having defined the logical mesh $\widetilde{\mathcal{T}_h}$, its image the physical mesh $\mathcal{T}_h$, together with the quadratures and mappings needed to numerically integrate over it, we now turn to the basis functions defined on each of its cells, which are used to build the approximation space of the PDE solution.

4.3. Basis functions module

Once the mesh has been defined, along with its mapping and the quadratures, in Section 4.2, we can introduce the scalar basis functions (trial or test) that are defined on each cell of the mesh. These functions are used to build the approximation space for the solutions of the PDE problem under consideration.

We seek to build basis functions living on each element K_c of the physical mesh $\mathcal{T}_h$. To this end, we consider two families of bases, namely analytical bases, which are built from polynomials (such as Lagrange, Taylor, etc.), and neural bases, which can either be shared across all cells (patchwise bases) or specific to each cell (cellwise bases). These three kinds of bases (analytical, patchwise and cellwise) are implemented in the classes `AnalyticBasis`, `PatchwiseParametricBasis` and `CellwiseParametricBasis`, which inherit from the interface of an abstract class `GeneralBasis` (Figure 4.1). We first describe the construction of polynomial bases, then that of neural bases, and finally the test functions.

Construction of a polynomial basis. In the classical case with analytical functions, we start from a reference basis $(\hat{\phi}_i)_{i=1}^{N_b}$, with $\hat{\phi}_i : \hat{K} \to \mathbb{R}$ defined on the reference element $\hat{K}$ and N_b the number of basis functions per cell (identical for all cells). We then use the geometric transformation $T_{K_c} : \hat{K} \to K_c$, defined in (4.5), to transfer this basis from the reference element $\hat{K}$ to the physical element $K_c \in \mathcal{T}_h$, by defining the i-th basis function on the cell K_c by:

$$\phi_{c,i} := \hat{\phi}_i \circ T_{K_c}^{-1} : K_c \to \mathbb{R}.$$

In practice, the inverse $T_{K_c}^{-1}$ is never formed explicitly: the basis is evaluated at points whose preimage in the reference cell is known (the quadrature points), and `ScimBa` then only needs to expand the inverse transformation around this point. Each basis can therefore also be evaluated directly in logical coordinates, which is necessary when the cell transformation has no closed-form inverse (e.g. for an unstructured mesh).

Remark 4.9. *In contrast to Chapter 3, since the reference element $\hat{K}$ considered here is a hypercube, the associated natural polynomial space is the space $\mathbb{Q}_k$ of polynomials of degree at most k in each variable separately (tensor product of 1D spaces), and not the space $\mathbb{P}_k$ of polynomials of total degree at most k (natural on a simplex). In particular, the number of basis functions per cell is $N_b = \dim \mathbb{Q}_k = (k+1)^d$.*

As discussed in Chapter 1, polynomial bases are simple, cheap to evaluate and well understood theoretically, but their approximation power is tied to the polynomial degree k and to the (fixed) geometry of the mesh, which can make them ill-suited to represent solutions with sharp features or complex shapes without resorting to very fine meshes or high degrees. Neural networks, on the other hand, are non-linear approximators whose parameters are learned rather than prescribed, which can make them better suited to represent such features, though this added flexibility comes at the cost of a harder (non-convex) training problem and weaker theoretical guarantees. This motivates the use of neural bases, in particular enriching the analytical basis with a learnable neural contribution, a question we will study in more depth in Chapter 5.

Construction of a neural basis. A neural basis is built from a network, parameterized by a vector θ of learnable parameters, which is combined with the analytical basis (see below). Unlike polynomial basis functions, it is not defined as the pullback of a single reference function via the inverse transformation $T_{K_c}^{-1}$, but is evaluated directly from the physical coordinates.

It is then possible to consider two types of neural bases: either we choose to consider a single network v_θ, shared by all cells, to enrich the analytical basis (so-called patchwise basis), or each physical cell K_c (of flat index c) has its own network $v_{c,\theta}$ (so-called cellwise basis).

Remark 4.10. *Note that in the cellwise case, this can become costly in memory and computation time if each cell has its own network. This type of basis is therefore only used on coarse meshes and with a limited number of learnable parameters per network.*

In both cases (patchwise or cellwise), the networks take the physical coordinates as input. Independently of the choice of basis type, the network's output can either be shared by all basis functions of the cell, or dedicated to a single one:

- shared output: $v_\theta : \mathbb{R}^d \to \mathbb{R}$ (resp. $v_{c,\theta} : \mathbb{R}^d \to \mathbb{R}$), the same value enriches every one of the N_b basis functions of the cell;
- dedicated output: $v_\theta : \mathbb{R}^d \to \mathbb{R}^{N_b}$ (resp. $v_{c,\theta} : \mathbb{R}^d \to \mathbb{R}^{N_b}$), with the i-th component used for the i-th basis function.

Both options are possible in `ScimBa`, but only the shared output is used in the numerical results of Chapter 5.

Notation 4.11. *In what follows, we denote a trial basis function, whether it is analytical, patchwise or cellwise, by $\phi^\theta_{c,i}$, using the same abuse of notation as for the mapping $\mathcal{G}_\theta$ (see Notation 4.2). This does not imply that the basis is learnable: in the analytical case, $\phi^\theta_{c,i} = \phi_{c,i}$ involves no parameter.*

For a finite family to form a basis of a local space on the physical cell K_c, it must be both linearly independent on K_c and span that space. In the polynomial case, these two properties are automatically satisfied, given that the polynomials (e.g. Lagrange or Taylor) are constructed such that they form a basis for $\mathbb{Q}_k$ with respect to the reference element $\hat{K}$, and that T_{K_c} acts only as an invertible change of variables.

For a neural basis, no such guarantee exists: there is no space fixed independently of θ that $\phi^\theta_{c,i}$ is meant to span, and no known basis simply being transported into new coordinates by an invertible transformation. Whether the N_b functions remain linearly independent on K_c depends entirely on the parameters θ learned during training, and is not guaranteed to hold throughout. This is particularly delicate in the patchwise case, where a single network v_θ must remain well-conditioned simultaneously on every cell of the mesh: training can drive it toward a locally degenerate behavior on some cells, making the local mass matrix (see Section 4.4) singular there, even while it keeps improving the loss globally. The cellwise case is less exposed to this specific failure mode, since each network only ever has to remain well-behaved on its own cell, though nothing prevents a cellwise network from degenerating on that cell either.

Several choices are possible to combine the network with the analytical basis $\phi_{c,i}$. A choice used in practice, which mitigates this loss of linear independence, is the multiplicative one:

$$\phi^{\theta}_{c,i}(\boldsymbol{x}) = m_\theta(\boldsymbol{x})\,\phi_{c,i}(\boldsymbol{x}),$$

and similarly for a cellwise basis, with $m_{c,\theta}$ in place of m_θ, where the multiplier m_θ (resp. $m_{c,\theta}$) is a function built from the network v_θ (resp. $v_{c,\theta}$). As soon as the multiplier does not vanish on K_c, the family $(m_\theta\,\phi_{c,i})_i$ remains linearly independent on K_c (and the local mass matrix remains well conditioned as long as it also stays away from zero). A more in-depth discussion of the choice of neural enrichment of the polynomial basis, and in particular of the construction of the multiplier, will be given in Chapter 5.

Remark 4.12. *The idea of enriching the basis with a neural contribution is closely related to the additive/multiplicative enrichment studied in Chapter 3. The key difference lies in when the network is trained: in Chapter 3, the prior u_θ is pretrained beforehand, independently of the numerical scheme, and only the remaining classical dofs are solved for once the prior is fixed. Here, by contrast, the parameters θ of the basis-enriching network are learned jointly with the Galerkin solve itself, by differentiating through the resolution of the scheme, which is the learnable Galerkin approach presented in Section 4.9 and developed in Chapter 5.*

Test functions. We consider that the test functions are built in the same way as the basis functions, that is, they can be analytical, patchwise or cellwise. We can choose to use the same test functions as the trial functions (Galerkin), or to use different test functions (Petrov–Galerkin), the test basis then being either analytical or of the same type as the trial basis. In what follows, the j-th test function associated with the physical cell K_c is denoted by $\psi^{\theta}_{c,j}$, regardless of the chosen basis type.

Having defined, on each physical cell, the (possibly learnable) basis functions used to represent the discrete solution, we now turn to how these basis functions are combined with the degrees of freedom to actually build this discrete solution, which is the role of the DG variable module presented next.

4.4. DG variable and post-processing modules

In this section, we focus on the discrete variable module `VariablesDG`, which is a central module of the library since it is the one that allows building the discrete DG solution from the linear degrees of freedom and the basis functions under consideration, defined on each cell of the physical mesh. We first describe how the discrete DG solution is reconstructed locally from the dofs and the basis functions (with their possibly learnable parameters), then how it can optionally be post-processed, and finally how an analytical function can conversely be projected onto the DG space.

Remark 4.13. *`ScimBa` is not restricted to scalar problems: it supports general systems of coupled equations (e.g. velocity, pressure and temperature for a Navier–Stokes system, with four unknowns in 2D). For simplicity of exposition, and to remain consistent throughout this chapter, we restrict ourselves here to the scalar case, by considering the elliptic PDE (2.1) and (2.2) with a single unknown u and a single source term f.*

Local discrete solution. In the previous sections, we saw that it was possible to parameterize in different ways a number of modules of the `ScimBa` library. In particular, in Section 4.2.2, we saw that the mapping $\mathcal{G}_\theta$ from the logical mesh to the physical mesh can be made learnable, and thus parameterize the physical mesh. In Section 4.3, we saw that it was possible to define as learnable the basis functions (patchwise or cellwise) used to build the discrete DG solution.

Notation 4.14. *As for the mapping and the basis functions, θ denotes in what follows the set of all learnable parameters of the model (mapping, basis functions and post-processing, defined below). This is an abuse of notation, since each type of module has its own parameters.*

For a physical cell $K_c \in \mathcal{T}_h$, we denote by $\phi^\theta_{c,i} : K_c \to \mathbb{R}$ the i-th (trial) basis function and by $\beta_{c,i}$ the linear degree of freedom associated with it, for $i \in \{1, \dots, N_b\}$. We can then locally define the discrete solution $w_{h,\theta} : \Omega \to \mathbb{R}$ reconstructed from the linear dofs and the basis functions on the physical cell K_c as:

$$w_{h,\theta}|_{K_c}(\boldsymbol{x}) := \sum_{i=1}^{N_b} \beta_{c,i}\, \phi^\theta_{c,i}(\boldsymbol{x}), \qquad \boldsymbol{x} \in K_c = \mathcal{G}_\theta(\tilde{K}_c), \tag{4.6}$$

where $\tilde{K}_c$ is a cell in the logical mesh $\widetilde{\mathcal{T}}_h$ and $\mathcal{G}_\theta$ is the mapping (learnable or not).

Beyond the mapping and the basis functions, the `VariablesDG` module allows for a third optional source of parameterization, acting this time on the reconstructed discrete variable $w_{h,\theta}$ itself rather than on the mesh or the basis: a local post-processing operator.

Local post-processing. The post-processing, denoted $\mathcal{P}_\theta$, is an operator that acts on the discrete variable $w_{h,\theta}$ reconstructed from the dofs, and may also depend on the position $\boldsymbol{x}$ at which it is evaluated. This operator can be an analytical function (linear or non-linear in $w_{h,\theta}$) or a neural network (non-linear).

Post-processing can be used to improve the quality of the discrete solution, for example by reducing oscillations or by imposing boundary conditions exactly (e.g. via a level-set function of $\boldsymbol{x}$ enforcing a Dirichlet condition). It is applied to $w_{h,\theta}$ to produce a post-processed solution $u_{h,\theta} : \Omega \to \mathbb{R}$ defined by:

$$u_{h,\theta}|_{K_c}(\boldsymbol{x}) := \mathcal{P}_\theta(w_{h,\theta}(\boldsymbol{x}),\ \boldsymbol{x}), \qquad \boldsymbol{x} \in K_c = \mathcal{G}_\theta(\tilde{K}_c), \tag{4.7}$$

where $w_{h,\theta}$ is given by (4.6).

Remark 4.15. *In `ScimBa`, the post-processing is defined by the abstract class* `AbstractPostProcessingOp` *(Figure 4.1). No concrete operator is provided by the library itself: each one is written by the user for the case at hand, as an algebraic or differential expression on the reconstructed solution $w_{h,\theta}$. In practice, $\mathcal{P}_\theta$ is implemented by a method* `formula(u)` *that receives $w_{h,\theta}$ not as a value sampled at $\boldsymbol{x}$, but as a functional* `ParamScalarFunction` *object (introduced in Section 4.5.1), so that the post-processing may itself depend on spatial derivatives of $w_{h,\theta}$ (e.g. its gradient) in addition to its pointwise value. The notation $\mathcal{P}_\theta(w_{h,\theta}(\boldsymbol{x}), \boldsymbol{x})$ is thus a shorthand for the operator $\mathcal{P}_\theta$ applied to the function $w_{h,\theta}$ and evaluated at $\boldsymbol{x}$.*

Beyond enforcing boundary conditions, the post-processing can also carry part of the difficulty of the approximation space. For instance, in one dimension, the function $(1+x)^2$ cannot be represented exactly by a piecewise affine space, whereas with $\mathcal{P}_\theta(w,x) = w^2$, that is $u_{h,\theta} = w_{h,\theta}^2$, the reconstructed solution $w_{h,\theta}$ only has to represent $1+x$, which does belong to the space. The post-processing thus plays for the output a role comparable to that of the mapping $\mathcal{G}_\theta$ for the input: the mapping does not only describe the geometry, it can also transform the inputs so as to simplify the problem, while the post-processing transforms the output of the scheme. Both act as changes of variables on either side of a space that remains linear in the dofs, an idea close in spirit to [24], where a linear operator is placed between two invertible networks. One may even hope, ultimately, to learn the mapping and the post-processing jointly, so that a small mesh is enough to approximate the solution.

Having described how the discrete solution $u_{h,\theta}$ is reconstructed and possibly post-processed from a given set of dofs, we now turn to the reverse operation: computing the dofs that best represent a known analytical function.

Projection onto the DG space. In addition to evaluating the reconstructed (and possibly post-processed) discrete DG solution $u_{h,\theta}$ defined in (4.7), the `VariablesDG` class allows implementing the projection of an analytical function $\mathrm{f} : \Omega \to \mathbb{R}$ (unrelated to the PDE source term) onto the DG space, with or without post-processing. This projection can be used for different applications, in particular:

- projecting the initial condition onto the DG space to fill in the degrees of freedom at the initial time (in the time-dependent setting, which is not detailed here);
- initializing the dofs before a learning/optimization phase (instead of starting with dofs at zero);
- comparing the discrete solution to the exact solution by projecting the known analytical solution to compute an error (for numerical verifications or unit tests).

Let us start with the classical case without post-processing, that is $\mathcal{P}_\theta(w, \boldsymbol{x}) = w$, so that $u_{h,\theta} = w_{h,\theta}$. For a physical cell $K_c \in \mathcal{T}_h$, we seek $\boldsymbol{\beta}_c := (\beta_{c,i})_{i=1}^{N_b}$ such that:

$$\sum_{i=1}^{N_b} \beta_{c,i} \int_{K_c} \phi_{c,i}^\theta(\boldsymbol{x})\, \psi_{c,j}^\theta(\boldsymbol{x})\, d\boldsymbol{x} = \int_{K_c} \mathrm{f}(\boldsymbol{x})\, \psi_{c,j}^\theta(\boldsymbol{x})\, d\boldsymbol{x}, \quad \forall j \in \{1, \dots, N_b\},$$

with $\psi_{c,j}^\theta$ the j-th test function associated with the physical cell K_c. This local linear system, whose matrix is the local mass matrix of the basis functions, is solved cell by cell to obtain the dofs $\boldsymbol{\beta}_c$ associated with the physical cell K_c, from which we reconstruct the discrete solution $u_{h,\theta} = w_{h,\theta}$ on K_c using the (trial) basis functions $\phi_{c,i}^\theta$.

In the case with post-processing, the projection is a bit more delicate, since the post-processing can be linear or non-linear. Let $\mathcal{P}_\theta$ be a local post-processing, acting on the reconstructed variable $w_{h,\theta}$ of (4.6) to produce the post-processed solution $u_{h,\theta}$ of (4.7). For a fixed physical cell $K_c \in \mathcal{T}_h$, we seek, as before, $\boldsymbol{\beta}_c = (\beta_{c,i})_{i=1}^{N_b}$ such that:

$$\int_{K_c} \left(\mathcal{P}_\theta(w_{h,\theta}(\boldsymbol{x}), \boldsymbol{x}) - \mathrm{f}(\boldsymbol{x})\right) \psi_{c,j}^\theta(\boldsymbol{x})\, d\boldsymbol{x} = 0, \quad \forall j \in \{1, \dots, N_b\},$$

with $w_{h,\theta}$ given by (4.6). If the post-processing is linear (more precisely, affine in $w_{h,\theta}$), we obtain a linear system on the dofs $\boldsymbol{\beta}_c$ that can be solved cell by cell, as in the classical case without post-processing. If the post-processing is non-linear (neural network or non-linear analytical function), we obtain a non-linear system on the dofs $\boldsymbol{\beta}_c$ that must be solved by an iterative Newton-type method (see Section 4.6.1).

Notation 4.16. *In this section, we have deliberately omitted the dependence of the degrees of freedom $\beta_{c,i}$ on θ: as soon as at least one of the mapping $\mathcal{G}_\theta$, the basis functions $\phi^\theta_{c,i}$ and the post-processing $\mathcal{P}_\theta$ is learnable, the calculation of the degrees of freedom, whether by projection (detailed above) or by solving the DG scheme (Section 4.5), amounts to solving a linear or non-linear system which itself depends on θ, so that $\beta_{c,i}$ must in practice be understood as $\beta_{c,i}(\theta)$. We keep omitting this dependence of the degrees of freedom on the learnable parameters throughout this chapter, and discuss it only in the dedicated Section 4.9.*

Having defined how the discrete DG solution $u_{h,\theta}$ is built from the dofs, the (possibly learnable) basis functions and post-processing, and how, conversely, an analytical function can be projected onto the DG space, we now turn to the module that assembles the DG scheme itself. Concatenating the local dof vectors $\boldsymbol{\beta}_c$ over all the cells of the mesh yields the global dof vector, which the DG scheme is assembled to solve for.

4.5. DG scheme modules

Having defined the mesh and the mappings (Section 4.2), the basis functions (Section 4.3) and the discrete DG variables (Section 4.4), we can now turn to the module that assembles the DG scheme in order to determine the approximate solution $u_{h,\theta}$, defined in (4.7), of the PDE (2.1). This assembly relies on three additional ingredients, which we first introduce separately: the linear weak forms (Section 4.5.1), which describe the PDE under consideration independently of any DG discretization, the numerical fluxes (Section 4.5.2), which define how information is communicated between neighboring cells across the mesh interfaces, and the boundary conditions (Section 4.5.3), which prescribe the behavior of the solution on Γ. We then present how these three modules are combined to assemble the residual of the DG scheme (Section 4.5.4).

4.5.1. Linear weak forms

We consider here the linear elliptic PDEs (with respect to the unknown u) introduced in (2.1) and (2.2). As recalled in Section 2.2.2, such a PDE, considered there with homogeneous Dirichlet boundary conditions for simplicity, admits a variational formulation (2.19) including the bilinear form and the linear form defined in (2.20) and (2.21). We focus here on the volume parts of these two forms, that is, on the terms integrated over the domain Ω and not over the interfaces $\mathcal{F}_h$. These actually coincide with the bilinear form a and the linear form l defined in Section 2.2.1 for the classical FEM approach, except that the usual gradient is here replaced by the broken gradient ∇_h. In the continuous FEM case, these forms are sufficient to fully define the discrete problem (2.6), whereas in the DG case they only constitute its volume part, to which

are added the interface terms associated with the choice of numerical fluxes presented in Section 4.5.2.

Weak form classes. In the `ScimBa` library, such a linear PDE is defined by specializing the class `EllipticWeakForm` (specific to elliptic problems), itself a specialization of the abstract class `AbstractWeakForm`, which expects two methods to be implemented, directly encoding:

- the bilinear form `bilinear_form(u, v)`, which returns the integrand of the volume part of the bilinear form $a_h^{\text{disc}}(u, v)$;
- the linear form `linear_form(v)`, which returns the integrand of the volume part of the linear form $l_h^{\text{disc}}(v)$.

Here, `u` and `v` are `ParamScalarFunction` objects (defined in the following paragraph) that expose, in addition to their pointwise evaluation, their spatial derivatives, allowing the user to write the forms in a purely functional manner, without ever explicitly manipulating quadrature points or weights.

During the assembly of the global residual (Section 4.5.4), `ScimBa` evaluates these integrands at each volume quadrature point for each cell (Section 4.2.1) and sums the values obtained, weighted by the quadrature weights, to obtain the integrals over Ω. The volume residual $r^{\Omega}(u, v)$ (the integrand of the volume terms) is the difference between the integrands returned by `bilinear_form` and `linear_form`, and is given by:

$$r^{\Omega}(u, v) := R\, u\, v + \boldsymbol{C} \cdot \nabla_h u\, v + \frac{1}{\mathrm{Pe}} D \nabla_h u \cdot \nabla_h v - f v. \tag{4.8}$$

In this expression, u and v are evaluated at a point $\boldsymbol{x}$, so that $r^{\Omega}(u, v)$ is understood as a function of $\boldsymbol{x}$. Once $u = u_{h,\theta}$ is the discrete solution, it also depends on the dofs $\boldsymbol{\beta}_h := (\boldsymbol{\beta}_c)_{c=0}^{N_{\text{cells}}-1}$, the concatenation of the local dof vectors $\boldsymbol{\beta}_c$ of all the cells, which are the linear unknowns of the scheme, and, whenever the scheme is learnable, on the non-linear parameters θ (basis functions, mapping, post-processing and, as detailed below, PDE fields). This is precisely what makes it possible to differentiate the weak form with respect to all of them. Since the integrand is written as a function of $\boldsymbol{x}$, $\boldsymbol{\beta}_h$ and θ, `JAX` differentiates it automatically, with respect to $\boldsymbol{\beta}_h$ to obtain the Jacobian of the Newton method (Section 4.6.1), and with respect to θ to differentiate through the solver (Section 4.9), without ever deriving these derivatives by hand.

The `LaplacianWeakForm` specialization of the `EllipticWeakForm` class is provided for the Poisson problem; it corresponds to the volume residual (4.8) with $R = 0$, $\boldsymbol{C} = \boldsymbol{0}$, $D = 1$ and $\mathrm{Pe} = 1$.

Functional composition. This functional writing relies on the class `ParamFunction`, from which `ParamVecFunction` and `ParamMatrixFunction` directly inherit. The classes `ParamScalarFunction` and `ParamFieldFunction` are themselves specializations of `ParamVecFunction` (a scalar being treated as a vector of size one, while the size of a field corresponds to the spatial dimension). Such an object simply encapsulates a function evaluable by `JAX` (a closure), and overloads the usual operators (addition, multiplication, dot product, matrix product, etc.) as well as the differential operators

(gradient, divergence, etc.), so that combining two `ParamFunction` objects produces a new `ParamFunction` without evaluating anything immediately. The differential operators are themselves obtained by automatic differentiation, using `JAX`, of the encapsulated function. Thus, when writing the weak forms, no quadrature is performed yet: one obtains a new `ParamFunction` representing the integrand, which is only effectively evaluated at assembly time (Section 4.5.4), at each quadrature point.

PDE fields. A weak form can depend on variable coefficients called fields of the PDE. In particular, for the PDE (2.1), these fields are the coefficients R, $\boldsymbol{C}$, D and Pe of the operator $\mathcal{L}$ defined in (2.2), as well as the source term f. These fields can simply be declared as analytical functions, but they can also be learnable, that is, functions parameterized by a set of parameters θ that are optimized during model training. This latter case typically corresponds to an inverse problem setting, where one of the PDE fields (for example a diffusion or reaction coefficient) is unknown and must be identified from observation data. This is illustrated by the inverse-problem validation of Section 4.10.7 (using the formalism of Section 4.9), where the reaction coefficient R is learnable and parameterized by a neural network. In that case, the residual depends on θ not only through the discrete solution $u_{h,\theta}$, but also directly through the field, for instance through $R_\theta(\boldsymbol{x})$ in the term $R\,u\,v$ of (4.8).

4.5.2. Numerical fluxes

As recalled in Section 2.2.2, and still restricting ourselves to homogeneous Dirichlet boundary conditions, the richness of DG methods lies in the choice of numerical fluxes $\hat{u}_h$ and $\hat{\boldsymbol{\varsigma}}_h^D$ used to replace the traces of the solution u and of the diffusive flux $D\boldsymbol{\varsigma}$, which are ambiguous on either side of an interface between two discontinuous cells. In `ScimBa`, this choice is encapsulated in the abstract class `AbstractFlux` (common to all fluxes), whose interface we present here. The classes `SIPGFlux` and `NIPGFlux` are concrete specializations of it, respectively implementing the SIPG and NIPG diffusive fluxes introduced in Section 2.2.2.2; `UpwindFlux` implements the upwind convective flux of Section 2.2.2.3. For readability, we detail below the case of the diffusive fluxes only: the upwind flux, which can be combined with either of them, acts in the same way, its contribution (2.30) simply being added to the face residuals defined below.

While the weak form `EllipticWeakForm` of Section 4.5.1 encodes the volume part (the integrals over Ω) of $a_h^{\mathrm{disc}}(u,v)$, this module encodes the interface terms (the surface integrals over $\mathcal{F}_h$ and $\mathcal{F}_h^I$), which allow neighboring cells to communicate across the mesh. The faces of the mesh are separated into two categories: the interior faces $\mathcal{F}_h^I$ shared by two cells, and the boundary faces $\mathcal{F}_h^B$ adjacent to a single cell, meaning that the set of faces is the disjoint union $\mathcal{F}_h = \mathcal{F}_h^I \cup \mathcal{F}_h^B$. Concretely, the abstract class `AbstractFlux` enforces the implementation of two methods:

- `__call__`, called on an interior face $F \in \mathcal{F}_h^I$, which returns the two associated face contributions, to be added respectively to the entries of the global residual $\boldsymbol{R}_h$ associated with the left cell K_L and the right cell K_R adjacent to F (see (4.11));
- `boundary_call`, called on a boundary face $F \in \mathcal{F}_h^B$, which returns the single face contribution to be added to the entries of the global residual $\boldsymbol{R}_h$ associated with

the cell adjacent to F. As detailed in Section 4.5.3, this method is only ever invoked to impose a Dirichlet condition, with Neumann and Robin conditions being treated independently of the numerical flux.

These two methods then directly implement the surface residuals $r^{\mathcal{F}_h^I}$ (the interior residual) and $r^{\mathcal{F}_h^B}$ (the boundary residual), defined by:

$$r^{\mathcal{F}_h^I}(u,v) := \frac{1}{\mathrm{Pe}}\Big(\{\hat{u}_h - u\}[\![D\nabla_h v]\!] - [\![\hat{\boldsymbol{\varsigma}}_h^D]\!]\{v\} + [\![\hat{u}_h - u]\!]\cdot\{D\nabla_h v\} - \{\hat{\boldsymbol{\varsigma}}_h^D\}\cdot[\![v]\!]\Big), \quad (4.9)$$

and

$$r^{\mathcal{F}_h^B}(u,v) := \frac{1}{\mathrm{Pe}}\Big([\![\hat{u}_h - u]\!]\cdot\{D\nabla_h v\} - \{\hat{\boldsymbol{\varsigma}}_h^D\}\cdot[\![v]\!]\Big), \quad (4.10)$$

with $\{\cdot\}$ and $[\![\cdot]\!]$ the average and jump operators of (2.15) and (2.16). These are the integrands of the surface integrals of (2.20), where the integral over $\mathcal{F}_h = \mathcal{F}_h^I \cup \mathcal{F}_h^B$ is split between interior and boundary faces. Rather than separately exposing the abstract fluxes $\hat{u}_h$ and $\hat{\boldsymbol{\varsigma}}_h^D$ of Section 2.2.2, `ScimBa` directly implements, for each choice of flux, the resulting values of $r^{\mathcal{F}_h^I}$ and $r^{\mathcal{F}_h^B}$, without ever constructing $\hat{u}_h$ or $\hat{\boldsymbol{\varsigma}}_h^D$ as intermediate objects. For SIPG, for instance, this amounts to substituting the SIPG fluxes of (2.22) to obtain the closed form (2.23).

As for the weak forms of Section 4.5.1, the two methods of `AbstractFlux` only return the integrand of the surface integral as a `ParamFunction` object, in a purely functional manner, and not the integral itself. The integrals over $\mathcal{F}_h$ and $\mathcal{F}_h^I$ of (2.20) are obtained as for the volume integrals, by summing these integrands over the surface quadrature points of each face F (Section 4.2.1), weighted by the surface quadrature weights, and then over the faces. This double summation is the subject of the assembly of the global residual, presented in Section 4.5.4.

4.5.3. Boundary conditions

As introduced in Section 2.1, the PDE (2.1) can be equipped with Dirichlet, Neumann or Robin boundary conditions on Γ. For simplicity, only the homogeneous Dirichlet case was considered in Section 2.2.2; the general case, detailed in what follows, extends this to (non-homogeneous) Dirichlet, Neumann and Robin conditions.

Boundary condition classes. A boundary condition is defined in `ScimBa` independently of the scheme considered (DG or otherwise). Two families of concrete conditions exist, according to whether they prescribe the solution or its flux:

- `Dirichlet`, prescribing the solution itself;
- `Neumann` and `Robin`, both specializations of `AbstractBoundaryWeakForm`, itself a specialization of `AbstractWeakForm` (Section 4.5.1), prescribing the diffusive flux.

The `Mesh` class (Section 4.2.3) allows the boundary Γ to be divided into different boundary groups, through its attribute `boundary_groups`. The name `"boundary"` designates the entire boundary Γ, while an individual side can be selected by `"west"`/`"east"` (and `"south"`/`"north"` in 2D). A boundary condition (Dirichlet, Neumann or Robin) can then be defined completely independently on each boundary group, a group without any condition being left free.

Imposition in the DG scheme. This split into two distinct families matters: being a specialization of `AbstractWeakForm`, a `Neumann` or `Robin` condition directly implements the same `bilinear_form(u, v)`/`linear_form(v)` interface as an interior weak form (only now integrated over the boundary group carrying the condition, rather than over Ω). Both are thus integrated like any other `AbstractBoundaryWeakForm` (the interface is generic enough that a user-defined condition, e.g. a Nitsche penalty, is handled identically), and contribute additional terms to the boundary residual $r^{\mathcal{F}_h^B}$ (4.10) independently of the choice of numerical flux, since a boundary term of the weak form does not involve $\hat{u}_h$ or $\hat{\boldsymbol{\varsigma}}_h^D$ at all.

A `Dirichlet` condition, by contrast, prescribes no such term: it is a constraint on the solution, not on the weak form, and is treated differently by each scheme (see Section 4.7 for the continuous FE scheme). In DG, it is imposed through the numerical flux (Section 4.5.2): the prescribed boundary value enters through $\hat{u}_h$, generalizing the homogeneous case of (2.22) and (2.25), so that `boundary_call` compares the trace of the solution with this value.

The physical model. We now introduce a new class `EllipticWeakModel`, a specialization of `AbstractPhysicalWeakModel`, which brings together everything needed to describe the PDE problem, independently of the discretization: the internal weak form from Section 4.5.1, as well as, for each boundary group, the corresponding boundary condition. In particular, it holds, for each boundary group, either a `Dirichlet` condition or, for a `Neumann` or `Robin` condition, its own boundary weak form as introduced above. Figure 4.6 summarizes this hierarchy of weak forms and boundary conditions.

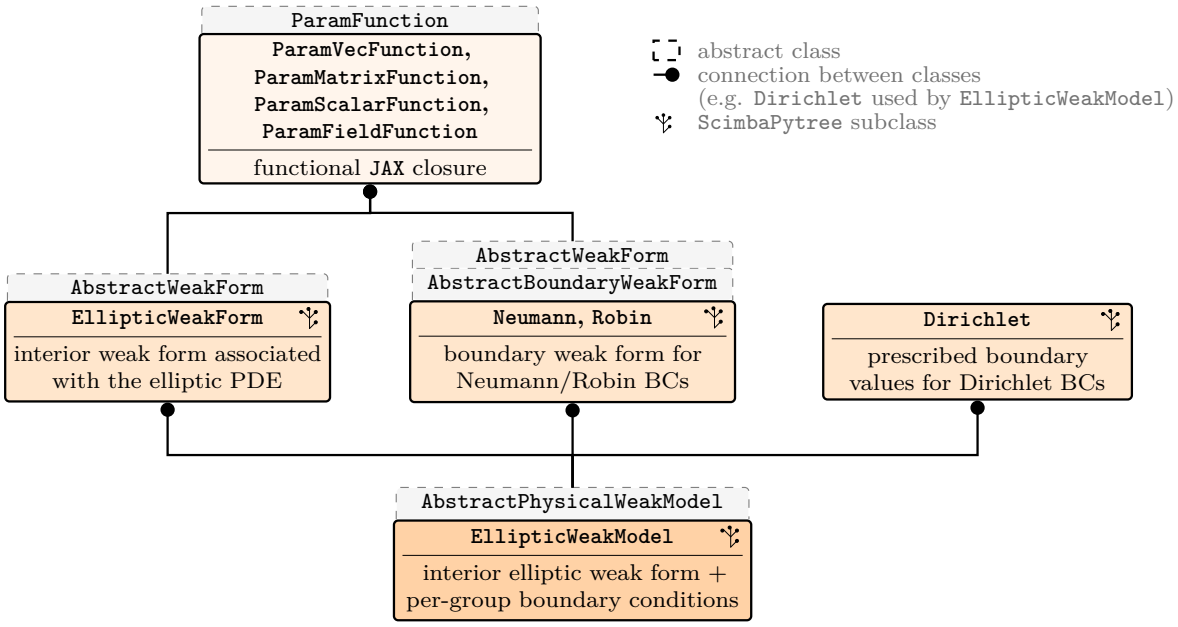


Figure 4.6.: Class hierarchy of `EllipticWeakModel`, detailing the weak-model box of Figure 4.1 and Figure 4.7. `Neumann` and `Robin` are boundary weak forms sharing the `bilinear_form`/`linear_form` interface of the elliptic interior weak form, while `Dirichlet` constrains the solution directly.

4.5.4. DG scheme assembly

We can now combine the previous modules to assemble the residual of the DG scheme, implemented in the class `EllipticDGscheme` (Figure 4.1), which takes as arguments a physical model `EllipticWeakModel` (Section 4.5.3), a discrete variable `VariablesDG` (Section 4.4) and a numerical flux, which is an instance of a subclass of `AbstractFlux` (Section 4.5.2). `EllipticDGscheme` itself specializes the abstract class `Galerkin`, which factors out the solving machinery presented in Section 4.6 and is shared by every scheme in the library, regardless of how its degrees of freedom are organized, in particular the continuous FE scheme of Section 4.7 and the φ-FEM scheme of Section 4.8. A concrete scheme need only provide its own residual assembly, as detailed in the remainder of this section.

Structure of the discrete residual. Recall that $\boldsymbol{\beta}_h$, introduced in Section 4.5.1, is the global vector of dofs of the discrete variable $u_{h,\theta}$. As in Notation 4.16, we omit the dependence of $\boldsymbol{\beta}_h$, as well as that of $\boldsymbol{R}_h$, A_h and $\boldsymbol{b}_h$ below, on the (here fixed) learnable parameters θ, writing for instance $\boldsymbol{\beta}_h$ rather than $\boldsymbol{\beta}_h(\theta)$. We study this dependence, and how to differentiate through it, only later, in Section 4.9.

The total residual of the DG scheme is then the array $\boldsymbol{R}_h(\boldsymbol{\beta}_h)$, of the same shape as $\boldsymbol{\beta}_h$, indexed by cell and then by local basis function. Each entry of the residual corresponds to a single pair (physical cell, local test function). For a physical cell $K_c \in \mathcal{T}_h$ and local test function $\psi^\theta_{c,j}$ (Section 4.3), the corresponding entry of the residual is given by:

$$\begin{aligned}\boldsymbol{R}_h(\boldsymbol{\beta}_h)_{c,j} := \int_{K_c} r^{\Omega}(u_{h,\theta}, \psi^\theta_{c,j}) + \sum_{F \in \mathcal{F}_h^I,\, F \subset \partial K_c} \int_F r^{\mathcal{F}_h^I}(u_{h,\theta}, \psi^\theta_{c,j}) \\ + \sum_{F \in \mathcal{F}_h^B,\, F \subset \partial K_c} \int_F r^{\mathcal{F}_h^B}(u_{h,\theta}, \psi^\theta_{c,j}),\end{aligned} \tag{4.11}$$

where r^{Ω}, $r^{\mathcal{F}_h^I}$ and $r^{\mathcal{F}_h^B}$ are the volume and surface residuals defined in (4.8) to (4.10), the test function $\psi^\theta_{c,j}$ being extended by zero outside K_c in the face terms, and where $u_{h,\theta}$ is the discrete DG function (possibly post-processed) defined in (4.7), reconstructed from the dofs $\boldsymbol{\beta}_h$. On a given boundary face, $r^{\mathcal{F}_h^B}$ further receives the additional terms of a Neumann or Robin boundary weak form, whenever its boundary group carries one (Section 4.5.3). Assembling the scheme therefore amounts to evaluating r^{Ω}, $r^{\mathcal{F}_h^I}$ and $r^{\mathcal{F}_h^B}$ at each (volumetric or surface) quadrature point and summing the weighted values, as explained above. We now detail how `ScimBa` organizes this evaluation and summation in practice.

Three summation passes. Concretely, the assembly proceeds in three independent passes, each corresponding to a traversal of all the cells or all the faces of a given type:

- a volume pass, which evaluates `bilinear_form(u, v)` and `linear_form(v)` at each volumetric quadrature point of each cell and sums the values obtained, weighted by the quadrature weights, to directly fill the entry of the residual associated with each pair (physical cell, local test function);

- a pass over the interior faces $\mathcal{F}_h^I$, which evaluates `__call__` at each surface quadrature point of each face and sums in the same way, to obtain the pair of contributions to be added to the entries of the residual associated with K_L and K_R (the left and right cells of the current internal face);
- a pass over the boundary faces $\mathcal{F}_h^B$, similar to the previous one but evaluating, depending on the boundary condition attached to the face's boundary group, either `boundary_call` (`Dirichlet`) or the `bilinear_form`/`linear_form` of the boundary weak form (`Neumann`, `Robin`), and adding the resulting contribution only to the entries of the residual associated with the single adjacent cell.

These three passes rely on the same principle of summation weighted by quadrature weights (Section 4.2.1), whether over the volume of a cell or over the surface of a face, and are interchangeable in their execution order, since they only add contributions to the same residual array. We now explain how `ScimBa` avoids recomputing the trial-side quantities for each test function, and then what this structure implies for the resulting residual $\boldsymbol{R}_h(\boldsymbol{\beta}_h)$ defined in (4.11).

Vectorizing over the test functions. Filling the N_b entries of a cell K_c one test function $\psi_{c,j}^\theta$ at a time would recompute $u_{h,\theta}$ and its derivatives at each quadrature point N_b times. `ScimBa` avoids this by exploiting the fact that the residuals r^Ω, $r^{\mathcal{F}_h^I}$ and $r^{\mathcal{F}_h^B}$ are linear in their test argument v. They are evaluated once, for a generic test field $\psi_c^\theta := \sum_{j=1}^{N_b} \zeta_j \, \psi_{c,j}^\theta$ with free coefficients $\boldsymbol{\zeta} \in \mathbb{R}^{N_b}$, and then differentiated with respect to $\boldsymbol{\zeta}$ by forward-mode automatic differentiation (see Section 4.1). By linearity, the j-th derivative is exactly $r(u_{h,\theta}, \psi_{c,j}^\theta)$, so the N_b entries of the cell are obtained in a single pass. The same principle applies to the face passes, with $2N_b$ coefficients for an interior face, one block for each adjacent cell.

Affine and non-affine residuals. In (4.8), the volume residual $r^\Omega(u, v)$ is affine in u as long as the fields R, $\boldsymbol{C}$, D, Pe of the operator $\mathcal{L}$ defined in (2.2), as well as the source term f, are themselves independent of u. If, moreover, the post-processing $\mathcal{P}_\theta$ is affine in $w_{h,\theta}$, so that $u_{h,\theta}$ is affine in $\boldsymbol{\beta}_h$, the assembled residual is itself affine in $\boldsymbol{\beta}_h$:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h) = A_h \, \boldsymbol{\beta}_h - \boldsymbol{b}_h,$$

for a matrix $A_h \in \mathbb{R}^{N_{\text{dofs}} \times N_{\text{dofs}}}$ and a right-hand side $\boldsymbol{b}_h \in \mathbb{R}^{N_{\text{dofs}}}$, with $N_{\text{dofs}} := N_{\text{cells}} \cdot N_b$ the total number of dofs. Determining the DG solution $u_{h,\theta}$ then reduces to solving the linear system $A_h \, \boldsymbol{\beta}_h = \boldsymbol{b}_h$.

Beyond this linear setting, nothing prevents these fields from depending on the pointwise value of the solution u itself (e.g. a reaction coefficient $R(u)$), in which case $r^\Omega(u, v)$, and therefore $\boldsymbol{R}_h(\boldsymbol{\beta}_h)$, is no longer affine. The same holds if the post-processing is non-linear. In this general case, determining the DG solution $u_{h,\theta}$ amounts to finding a zero $\boldsymbol{\beta}_h$ of $\boldsymbol{R}_h$, that is, to solving:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h) = \boldsymbol{0}. \tag{4.12}$$

Having shown how `ScimBa` assembles the residual $\boldsymbol{R}_h(\boldsymbol{\beta}_h)$ of the DG scheme, whether affine or not in $\boldsymbol{\beta}_h$, we now turn to how this residual is actually solved for $\boldsymbol{\beta}_h$, which is the role of the solve framework modules presented next.

4.6. Solve framework modules

We saw at the end of Section 4.5.4 that, once the scheme is assembled, determining the DG solution $u_{h,\theta}$ amounts to finding the vector of dofs $\boldsymbol{\beta}_h$ such that the residual $\boldsymbol{R}_h$ of the scheme vanishes (4.12), where $\boldsymbol{R}_h$ is potentially non-linear in $\boldsymbol{\beta}_h$ (as soon as a field of the PDE depends on the solution or the post-processing is non-linear). In this section, we still consider θ fixed (Notation 4.16), so that we are only interested in how `ScimBa` solves (4.12) to obtain $\boldsymbol{\beta}_h$. The case where θ is no longer fixed, that is, where one wishes to differentiate the solution $\boldsymbol{\beta}_h(\theta)$ with respect to θ, is the subject of Section 4.9. We first present how `ScimBa` solves (4.12) via Newton's method, in a dense (Section 4.6.1) then a matrix-free (Section 4.6.2) variant, before showing how, once differentiated, this resolution defines a non-linear approximation space (Section 4.6.3). Both solvers inherit from a common abstract base class, `AbstractNonlinearSolver` (Figure 4.1).

4.6.1. Newton solve

In the general (non-linear) case, we solve (4.12) using the Newton–Raphson method: starting from an initial iterate $\boldsymbol{\beta}_h^{(0)}$ (by default $\boldsymbol{\beta}_h^{(0)} = \mathbf{0}$), we build the sequence $\big(\boldsymbol{\beta}_h^{(n)}\big)_{n\geq 0}$, indexed by the iteration number n, defined by:

$$\boldsymbol{\beta}_h^{(n+1)} = \boldsymbol{\beta}_h^{(n)} - J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big)^{-1}\boldsymbol{R}_h\big(\boldsymbol{\beta}_h^{(n)}\big), \qquad J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h) := \frac{\partial \boldsymbol{R}_h}{\partial \boldsymbol{\beta}_h}(\boldsymbol{\beta}_h), \tag{4.13}$$

where the Jacobian $J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big) \in \mathbb{R}^{N_{\text{dofs}}\times N_{\text{dofs}}}$ is obtained by automatic differentiation of the assembled residual using `JAX`. Concretely, the inverse $J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big)^{-1}$ is never formed explicitly: the Newton increment $\boldsymbol{\delta}_h^{(n)} := \boldsymbol{\beta}_h^{(n+1)} - \boldsymbol{\beta}_h^{(n)}$ solves the linear system:

$$J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big)\,\boldsymbol{\delta}_h^{(n)} = -\boldsymbol{R}_h\big(\boldsymbol{\beta}_h^{(n)}\big), \tag{4.14}$$

which is cheaper and more numerically stable than forming the inverse. In the affine case of Section 4.5.4, $J_{\boldsymbol{R}_h} = A_h$ does not depend on $\boldsymbol{\beta}_h$, and a single Newton step (4.13) suffices to obtain the exact solution, regardless of the initial iterate. In the non-linear case, several steps are generally required to approach a zero of $\boldsymbol{R}_h$. This dense solver is implemented in `ScimBa` by the class `NewtonSolver`.

As in a standard implementation of Newton's method, the iterations stop when a convergence criterion is satisfied, such as $\|\boldsymbol{R}_h\big(\boldsymbol{\beta}_h^{(n)}\big)\|_2 < \text{tol}$, where $\|\cdot\|_2$ denotes the Euclidean norm and tol a tolerance, or when a maximum number of iterations is reached, both being chosen by the user; in the non-linear case, the returned iterate is then not necessarily close to a zero of $\boldsymbol{R}_h$.

Efficient assembly of the Jacobian. The Jacobian $J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big)$ is where the automatic differentiation of `JAX` (Section 4.1) is used the most. Differentiating $\boldsymbol{R}_h$ naively, as a single function of the full dof vector $\boldsymbol{\beta}_h$, would trace the entire assembly loop over cells and faces inside the automatic-differentiation graph, so that building $J_{\boldsymbol{R}_h}\big(\boldsymbol{\beta}_h^{(n)}\big)$ would cost of the order of N_{dofs} evaluations of the residual. `ScimBa` avoids this by exploiting the structure of (4.11): each entry $\boldsymbol{R}_h(\boldsymbol{\beta}_h)_{c,j}$ is a sum of local contributions, one per cell or face, and differentiation, being linear, commutes with this sum.

The N_b entries associated with a cell K_c, already obtained in a single pass by the vectorization over the test functions of Section 4.5.4, only depend on the local dofs $\boldsymbol{\beta}_c$, and those associated with an interior face on the dofs of its two adjacent cells. `ScimBa` differentiates each of these small functions with respect to the dofs it depends on, vectorizes these differentiations over all cells and faces, and scatters the resulting blocks into $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)$ following the same connectivity as the residual itself. The traced graph thus has the size of a single block and is reused unchanged across the whole mesh, so that the cost of compiling the Jacobian assembly does not grow with N_{dofs}, and the remaining cost, evaluating this graph once per cell and face, scales only linearly in N_{dofs}.

Even though `ScimBa` builds $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)$ this efficiently, the dense solver just described still explicitly forms and factorizes the resulting $N_{\text{dofs}} \times N_{\text{dofs}}$ matrix at each Newton step. It is simple and robust, but cannot be used for large-scale problems: we therefore introduce a matrix-free variant of Newton's method.

4.6.2. Matrix-free Newton solve

We now turn to the matrix-free alternative to the dense solver of Section 4.6.1, which solves the same linear system (4.14) by a Krylov method accessing $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)$ only through matrix-vector products.

Matrix-free variant. `ScimBa` proposes a matrix-free variant of Newton's method, implemented by the class `NewtonKrylov`, in which this linear system is solved by a Krylov method [113] that only accesses the Jacobian through matrix-vector products $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)\boldsymbol{\delta}_h^{(n)}$, themselves obtained by automatic differentiation, in particular through JVPs (Section 4.1), without ever constructing $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)$ itself. This approach is much more memory-efficient and allows solving problems with a large number of dofs, at the cost of a more complex implementation and a less robust convergence.

Choice of linear solver. Several Krylov methods are available and implemented in `ScimBa` to solve this system at each Newton step: the conjugate gradient method [71], valid only for a symmetric positive definite operator (the case of pure SIPG diffusion), as well as the generalized minimal residual (GMRES) method [112] and BiCGStab [128], both valid for an arbitrary operator and therefore required as soon as the Jacobian is no longer symmetric. Their number of iterations is governed by the conditioning of $J_{\boldsymbol{R}_h}\left(\boldsymbol{\beta}_h^{(n)}\right)$, which the preconditioners presented below aim to improve.

Preconditioning. To accelerate the convergence of these Krylov methods, `ScimBa` proposes several matrix-free preconditioners. The main one, block-Jacobi, exploits the DG structure: each diagonal block $\partial \boldsymbol{R}_h|_{K_c}/\partial \boldsymbol{\beta}_c$ (associated with physical cell K_c, and obtained exactly as in Section 4.6.1) only couples the dofs internal to K_c, so it suffices to invert these small blocks independently of one another. On a uniform Cartesian mesh (identity mapping), all blocks coincide, and a single one is computed, inverted, and replicated across all cells. `ScimBa` also proposes more advanced preconditioners (randomized Nyström [64], Chebyshev polynomial, damped Gauss–Seidel sweeps), suited to other regimes.

Batch of solves. An advantage of `JAX` is that the solver itself can be vectorized with `jax.vmap`. Given a function that builds the scheme from what differs between the problems (for instance the parameter $\boldsymbol{\mu}$ of the PDE) and solves it, this vectorization turns it into a single compiled program that solves a whole batch of problems in parallel, the mesh, the basis and the variables being built once and shared. An example is given in Section 4.10.5.

4.6.3. The solver as a non-linear approximation space

The two solvers presented above (dense and matrix-free) are independent of any particular scheme: they take as arguments generic residual and Jacobian assembly functions (or their action, in the matrix-free case), supplied by the scheme (Section 4.5.4). This separation allows `ScimBa` to expose the resolution of the DG scheme in a form directly reusable by the rest of the library, through the class `GalerkinEllipticApproximationSpace`. Its two main methods are:

- `get_intermediate_values`, which solves the problem (4.12) (by the dense or the matrix-free Newton solver) and returns the obtained dof vector;
- `create_variables`, which reconstructs, from these dofs, the discrete functions $u_{h,\theta}$ (Section 4.4) as `ParamFunction` objects, directly usable in a loss or by another module of the library.

Once the solver is part of the model, $u_{h,\theta}$ is a function of $\boldsymbol{x}$ and of the learnable parameters θ, just like the output of a neural network is a function of its weights. It thus depends non-linearly on θ, and defines a non-linear approximation space, unlike a classical Galerkin space, which is linear in the dofs. Since the solver can be differentiated with respect to θ (Section 4.9), this space can be optimized like any neural approximation space.

`GalerkinEllipticApproximationSpace` inherits from `AbstractApproxSpace`, the interface common to all models of the library (e.g. neural network). Whatever the underlying model, it exposes the same methods `get_intermediate_values` and `create_variables`, which allows the rest of the library (loss, optimizer) to manipulate it without having to know its nature. As a `ScimbaPytree` (Section 4.1), this space exposes to `JAX` every learnable parameter θ it carries (weights of a network enriching the basis functions, mapping parameters, etc.), which therefore remains differentiable through the call to the solver. Its use in a training is illustrated in Appendix B.1.

Having shown how `ScimBa` solves the DG scheme for a fixed θ and packages this resolution into a reusable approximation space, we now turn to two extensions of this same architecture: continuous FEM (Section 4.7), then φ-FEM (Section 4.8), which reuses most of the continuous FE stack.

4.7. Extension to FEM

Unlike the previous chapters, where continuous FEM (Section 2.2.1) was introduced first, and where DG (Section 2.2.2) was then presented as its extension allowing

communication between discontinuous cells, the `ScimBa` library is built in the opposite direction: each module described so far, namely the geometry, basis function, DG variable, DG scheme and solve framework modules (Sections 4.2 to 4.6), works cell by cell, which is exactly the DG setting.

In the previous sections, the DG case is therefore presented as the general setting, of which continuous FEM turns out to be a special case: obtained not by rewriting any of the modules above, but by adding a single continuity constraint that identifies the dofs shared by adjacent cells. Figure 4.7 illustrates this by highlighting the similarity between the DG and FE modules, showing what needs to be replaced in the architecture of `ScimBa` (Figure 4.1) to go from one to the other. The modules specific to FE are described next, with Section 4.7.1 focusing on the dof identification, and Section 4.7.2 on the assembly of the FE scheme.

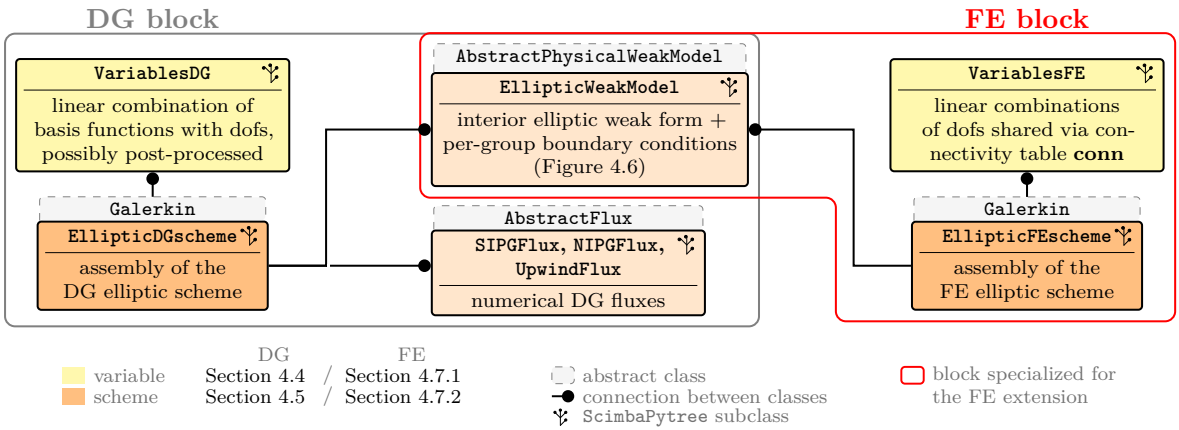


Figure 4.7.: Extension of Figure 4.1 to the continuous FE scheme (Section 4.7). All modules other than the DG-specific ones are reused unmodified. `EllipticFEscheme` and `EllipticDGscheme` inherit from a common abstract base `Galerkin`, and `VariablesFE` replaces `VariablesDG`.

Remark 4.17. *Unlike the mesh $\mathcal{T}_h$ of Section 2.2.1 and Chapter 3, made of simplexes, the FE extension of* `ScimBa` *reuses, without modification, the hypercube mesh structure of Section 4.2 (the same* `Mesh` *class, built from the images of a logical grid of hypercubes deformed by the mapping $\mathcal{G}_\theta$). Its local basis therefore spans $\mathbb{Q}_k$ rather than $\mathbb{P}_k$, as in simplex-based libraries such as* `FEniCS`*. This is a deliberate design choice, made precisely so that the entire geometric stack (mesh, mapping, quadrature) is shared without change between the DG and FE schemes.*

4.7.1. Continuous FE variables

We now describe the `VariablesFE` module, which replaces `VariablesDG` and makes the reconstructed solution continuous by sharing the dofs of adjacent cells through a connectivity table.

From independent to shared dofs. Recall, from Section 4.4, that in the DG setting the discrete solution (4.6) is reconstructed cell by cell from independent local dofs

$\boldsymbol{\beta}_c = (\beta_{c,i})_{i=1}^{N_b}$, concatenated into a global vector $\boldsymbol{\beta}_h = (\boldsymbol{\beta}_c)_{c=0}^{N_{\text{cells}}-1} \in \mathbb{R}^{N_{\text{cells}} \cdot N_b}$ (Section 4.5), with no relation imposed between the dofs of two neighboring cells. Hence the discontinuities that the numerical fluxes of Section 4.5.2 are precisely there to weakly reconcile. The `VariablesFE` module keeps the reconstruction formula (4.6) unchanged, but modifies the way $\beta_{c,i}$ relates to the global dof array. It builds, once and for all at construction time and from the mesh topology alone, a connectivity table:

$$\mathbf{conn} : \{0, \dots, N_{\text{cells}} - 1\} \times \{1, \dots, N_b\} \longrightarrow \{1, \dots, N_{\text{dofs}}\},$$

which maps each pair (cell, local shape function) to the global index of the mesh node carrying that shape function. The total number of dofs is then given by:

$$N_{\text{dofs}} := \prod_{\ell=1}^{d} (N_\ell k + 1),$$

considering the logical grid, where N_ℓ denotes the number of cells per direction $\ell \in \{1, \dots, d\}$ defined in Section 4.2.3. Whenever a node is shared by several cells, i.e. it lies on one of their common faces, **conn** assigns it the same global index in each of them. The local dof is then simply read from the global dof vector $\boldsymbol{\beta}_h := (\beta_I)_{I=1}^{N_{\text{dofs}}} \in \mathbb{R}^{N_{\text{dofs}}}$, through this table:

$$\beta_{c,i} = \beta_I, \quad I = \mathbf{conn}(c, i), \quad c \in \{0, \dots, N_{\text{cells}} - 1\}, \quad i \in \{1, \dots, N_b\}, \tag{4.15}$$

in place of the cell-independent array $\boldsymbol{\beta}_c$ of the DG case. The continuity of $u_{h,\theta}$ (4.7) is thus imposed directly at the data-structure level, via the connectivity table, at no computational overhead (no term to assemble, unlike a weak penalty).

Indeed, let K_L, K_R be two adjacent cells sharing a face $F \in \mathcal{F}_h^I$. By (4.15), the nodes of K_L and K_R lying on F share the same entries of $\boldsymbol{\beta}_h$: the dofs associated with them are therefore identical in both cells. Now the $\mathbb{Q}_k$ shape functions attached to these nodes are precisely the Lagrange interpolants of the face F. Having the same dofs and the same shape functions on F, $u_{h,\theta}|_{K_L}$ and $u_{h,\theta}|_{K_R}$ therefore coincide at every point of F. The reconstructed solution is thus continuous over all of Ω (and no longer only cell by cell), i.e. $u_{h,\theta} \in C^0(\Omega)$, as long as the basis functions are themselves continuous across the faces (which is automatic for analytical and patchwise bases, and must be enforced by construction in the cellwise case, see Chapter 5). In the analytical case without post-processing, $u_{h,\theta} \in V_h$, the continuous Lagrange FE space of (2.4), adapted here to $\mathbb{Q}_k$ elements in line with Remark 4.17.

Notation 4.18. *The symbol N_{dofs} denotes, for both classical FEM (Section 2.2.1) and the continuous FE scheme of `ScimBa` presented in this section, the number of dofs of the space V_h. The DG scheme (Section 4.5.4), by contrast, reuses this same symbol for a different quantity, $N_{cells} \cdot N_b$, which is the total number of independent dofs (N_b per cell, without any continuity constraint) and therefore the dimension of the space V_h^{disc} (2.13). The context should make it clear which meaning is intended.*

4.7.2. FE scheme assembly

The `EllipticFEscheme` class assembles the residual of the continuous FE scheme, taking as arguments the same building blocks as `EllipticDGscheme` (Section 4.5.4): a

physical model `EllipticWeakModel` (Section 4.5.3), which gathers the interior weak form `EllipticWeakForm` (Section 4.5.1) and the boundary conditions on each boundary, together with a discrete variable `VariablesFE` in place of `VariablesDG`. Unlike `EllipticDGscheme`, it takes no numerical flux.

One or two passes instead of three. Since $u_{h,\theta}$ is now continuous, the broken gradient ∇_h of the volume residual r^{Ω} (4.8) coincides with the ordinary gradient ∇, and there is no longer a jump to reconcile across an interior face. Thus, the residual $r^{\mathcal{F}_h^I}$ (4.9) on interior faces simply disappears. So does the entire pass over $\mathcal{F}_h^I$, as well as the numerical flux module of Section 4.5.2. Of the three summation passes (see Section 4.5.4) that assemble the DG residual (4.11), at most two therefore remain: a volume pass, evaluating the same `bilinear_form`/`linear_form` methods of Section 4.5.1, and, if the boundary carries Neumann or Robin conditions, a pass over the corresponding faces of $\mathcal{F}_h^B$ (see Section 4.5.3), the only ones summed over in (4.16) below. The boundary residual $r^{\mathcal{F}_h^B}$ (4.10) is no longer evaluated via the `flux` object and its `boundary_call` method, as in the DG method, but directly from the `bilinear_form`/`linear_form` of the boundary condition itself.

In practice, the volume pass differs slightly from that of DG: instead of writing each pair (physical cell, local test function) directly into its own entry of $\boldsymbol{R}_h$, as in (4.11), the contribution of physical cell K_c and local test function $\psi_{c,j}^{\theta}$ is added to the entry of the global node $\mathbf{conn}(c,j)$, using the dof identification (4.15). Each free node of global index I, $I \in \{1,\dots,N_{\text{dofs}}\}$ (excluding Dirichlet nodes), thus accumulates the contributions of all the cells to which it belongs:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h)_I := \sum_{(c,j)\,:\,\mathbf{conn}(c,j)=I} \left(\int_{K_c} r^{\Omega}(u_{h,\theta}, \psi_{c,j}^{\theta}) + \sum_{F\in\mathcal{F}_h^B,\, F\subset\partial K_c} \int_F r^{\mathcal{F}_h^B}(u_{h,\theta}, \psi_{c,j}^{\theta}) \right). \tag{4.16}$$

Strong imposition of Dirichlet conditions. A Dirichlet condition is treated differently from a Neumann or Robin condition. In DG, it is imposed weakly, in the same way as interior jumps, via a ghost state substituted into the numerical flux (Section 4.5.3). `VariablesFE`, by contrast, identifies each dof with a global mesh node via the connectivity table $\mathbf{conn}$ (4.15): for nodes lying on a Dirichlet boundary, `EllipticFEscheme` imposes Dirichlet conditions strongly, in the classical way. In other words, for each node of global index I lying on the Dirichlet boundary, the corresponding entry of (4.16) is replaced by:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h)_I = \beta_I - g\big(\boldsymbol{x}^{(I)}\big),$$

where g is the prescribed Dirichlet data and $\boldsymbol{x}^{(I)}$ the physical position of the node of global index I. In this way, the corresponding dof is fixed to the prescribed value, and the search space is restricted to the functions of V_h satisfying it, that is, to V_h^0 (2.5) when $g = 0$, rather than being penalized towards it as in the DG method. No penalty parameter or face integral is therefore required.

Shared architecture with DG. The two schemes `EllipticDGscheme` (Section 4.5.4) and `EllipticFEscheme` share a common abstract base class, `Galerkin`, which factors

out what does not depend on how the dofs are organized, in particular the resolution by Newton's method of Section 4.6, dense or matrix-free. A `Galerkin` scheme subclass therefore only provides its own residual assembly, namely (4.16) for continuous FEM, and (4.11) for DG.

As a result, the continuous FE extension only requires the dof identification of (4.15), the removal of the interior-face pass and the strong imposition of Dirichlet conditions, the other modules being unchanged (see Figure 4.7).

Having shown how continuity between elements is obtained, in `ScimBa`, from a connectivity table and a lighter assembly on top of the DG stack, we now turn to a second extension, this time towards unfitted geometries, namely the φ-FEM approach.

4.8. Extension to φ-FEM

We restrict this section to the setting of the Poisson problem with a homogeneous Dirichlet boundary condition (2.31), exactly as in Section 2.2.3 and as in the founding reference [53]. Unlike the continuous FE extension of Section 4.7, which treats the elliptic operator $\mathcal{L}$ of Section 2.1 using the generic `bilinear_form`/`linear_form` methods of the weak form `EllipticWeakForm` (Section 4.5.1), the stabilization terms $\mathcal{S}_h/\mathcal{S}_h^{\text{rhs}}$ of the φ-FEM scheme (2.34) are specific to the PDE under consideration. The class `PoissonDirichletPhiFEMscheme`, which implements this scheme, therefore takes as argument a physical model, `PoissonDirichletPhiFEMWeakModel`, specific to the φ-FEM scheme. Like `EllipticWeakModel` of Section 4.5.3, it inherits from `AbstractPhysicalWeakModel`.

This φ-FEM extension reuses most of the continuous FE stack presented in Section 4.7, in particular the same continuous variable class `VariablesFE` and its connectivity table **conn** (4.15). What fundamentally changes is that the boundary Γ of Ω is no longer fitted by the mesh, which is now a background mesh $\mathcal{T}_h^{\mathcal{O}}$ built with the `Mesh` class of Section 4.2.3. Consequently, only a subset of its cells, referred to as active cells, must take part in the assembly, which requires an additional geometric selection step, detailed in Section 4.8.1. The computational mesh $\mathcal{T}_h$, defined in (2.32), therefore consists only of the active cells, as presented in Section 2.2.3. On each active cell $K_c \in \mathcal{T}_h$, the field $w_{h,\theta}$ to which the dofs $\boldsymbol{\beta}_h$ are associated is reconstructed by `VariablesFE` exactly as in (4.6). The solution is then obtained by multiplying it by φ, which is implemented as the `LevelSetMultiplier` post-processing of `VariablesFE` (Section 4.4):

$$u_{h,\theta} := \varphi\, w_{h,\theta}, \tag{4.17}$$

as in Section 2.2.3, with φ in place of its interpolant φ_h. The assembly of the φ-FEM scheme, where this choice is discussed, and the associated physical model are detailed in Section 4.8.2. Figure 4.8 illustrates the architecture of the φ-FEM scheme, highlighting its similarities with the FE scheme presented in Section 4.7.

Remark 4.19. *The extension to the non-homogeneous case can be considered by modifying the reconstruction (4.17) into $u_{h,\theta} := \varphi\, w_{h,\theta} + g$, where g is a lifting function of the boundary data defined on Ω_h, in the same way as for strongly imposing boundary conditions in PINNs (Section 2.3.2). The terms of the scheme involving $u_{h,\theta}$, in particular the stabilization, would then have to be adapted accordingly.*

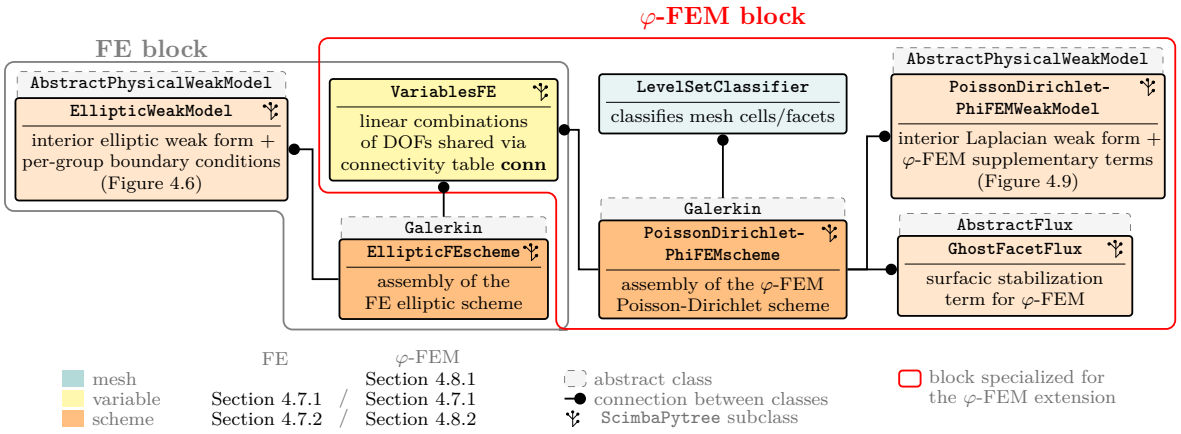


Figure 4.8.: Extension of Figure 4.7 to the φ-FEM scheme (Section 4.8). `VariablesFE` and the abstract base `Galerkin` are reused unmodified. `LevelSetClassifier` is the only new module, while `PoissonDirichletPhiFEMWeakModel` and `GhostFacetFlux` are specializations of existing modules.

4.8.1. Level-set-based cell and face classification

The `LevelSetClassifier` class is responsible for building, once and for all at scheme construction time and from the sole knowledge of φ, the sets of cells and faces required for assembly, illustrated in Figure 2.2. Each cell $K_c \in \mathcal{T}_h^{\mathcal{O}}$ is labeled `INTERIOR`, `EXTERIOR` or `CUT` according to the sign of φ at its vertices (respectively all negative, all positive, or of mixed signs). The `INTERIOR` and `CUT` cells together form the computational mesh $\mathcal{T}_h \subset \mathcal{T}_h^{\mathcal{O}}$ (active cells), while the `CUT` cells alone form $\mathcal{T}_h^{\Gamma}$ (cells cut by the boundary Γ). The interior faces of $\mathcal{T}_h^{\mathcal{O}}$ are labeled similarly from the labels of their two neighboring cells, which yields the stabilized faces $\mathcal{F}_h^{\Gamma}$, namely the interior faces of $\mathcal{T}_h$ touching a cut cell, as well as the faces of $\partial\Omega_h$, namely the faces separating an active cell from a purely exterior cell. These are precisely the sets on which the additional terms of the φ-FEM scheme (2.33) act.

Notation 4.20. *In FE and DG, $\mathcal{T}_h$ is the mesh built with the* `Mesh` *class of Section 4.2.3. In φ-FEM, this class is used for the background mesh $\mathcal{T}_h^{\mathcal{O}}$, and, as in Section 2.2.3, $\mathcal{T}_h$ only denotes the active subset of its cells. Thus, $\mathcal{T}_h$ is always the computational mesh on which the scheme is built.*

4.8.2. φ-FEM scheme assembly

Once the active cell and face sets of Section 4.8.1 are available, the φ-FEM scheme can be assembled. We first detail the assembly of the residual itself, before turning to the physical model that packages it, to the interface shared with the DG and FE schemes, and finally to the current limitation of this implementation regarding shape optimization.

Assembly of the residual. In the setting of the φ-FEM scheme, the assembled residual combines three terms, (I) to (III), the last of which splits into a volume half and a facet half, which makes four assembly passes, against one or two for continuous FEM (4.16). For each node of global index $I \in \{1, \dots, N_{\text{dofs}}^{\mathcal{O}}\}$, with $N_{\text{dofs}}^{\mathcal{O}}$ the total number of dofs of `VariablesFE`, i.e. those of the whole background mesh $\mathcal{T}_h^{\mathcal{O}}$ (a size fixed by the mesh alone, which does not depend on φ), the entry $\boldsymbol{R}_h(\boldsymbol{\beta}_h)_I$ accumulates the contributions of all the active cells to which the node belongs (none for inactive nodes, see below):

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h)_I := \sum_{(c,j):\, \mathbf{conn}(c,j)=I} \Bigg[\underbrace{\int_{K_c} r^{\Omega}(u_{h,\theta}, \varphi\psi_{c,j}^{\theta})}_{\text{(I)}} \underbrace{- \sum_{\substack{F \in \mathcal{F}_h \\ F \subset \partial K_c \cap \partial\Omega_h}} \int_F \frac{\partial}{\partial \boldsymbol{n}}(u_{h,\theta})\, \varphi\psi_{c,j}^{\theta}}_{\text{(II)}} + \underbrace{\mathcal{S}_h(w_{h,\theta}, \psi_{c,j}^{\theta}) - \mathcal{S}_h^{\text{rhs}}(\psi_{c,j}^{\theta})}_{\text{(III)}} \Bigg], \tag{4.18}$$

with $u_{h,\theta}$ defined by (4.17), and where $\mathcal{S}_h$ and $\mathcal{S}_h^{\text{rhs}}$ are the stabilization terms of the scheme defined in (2.34), evaluated here on $w_{h,\theta}$ and $\psi_{c,j}^{\theta}$. Since $\psi_{c,j}^{\theta}$ is supported on K_c, the sums defining $\mathcal{S}_h$ and $\mathcal{S}_h^{\text{rhs}}$ reduce to the contribution of K_c itself, if $K_c \in \mathcal{T}_h^{\Gamma}$, and to that of those of its faces which belong to $\mathcal{F}_h^{\Gamma}$.

Remark 4.21. *The assembly of the scheme defined above (4.18) involves in particular the Laplacian $\Delta(\varphi w_{h,\theta})$, via $\mathcal{S}_h$ defined in (2.34), whose computation relies on an analytical evaluation of φ, exactly at the point $\boldsymbol{x}$ requested by the quadrature, whereas Section 2.2.3 interpolates φ on the background mesh into a φ_h (of degree at least k) before using it in the scheme. This choice is not an additional approximation on the* `ScimBa` *side, but on the contrary a simplification made possible by automatic differentiation: since φ is an arbitrary* `JAX` *function (traceable, hence differentiable), the operators of* `ParamFunction` *allow differentiating directly through φ, so that the product-rule differentiation formulas for $\nabla(\varphi w)$ and $\Delta(\varphi w)$ never need to be written by hand. The price of this simplification is that the theoretical framework of Theorem 2.12, which assumes φ replaced by its interpolant φ_h, does not apply as is to this analytical variant of the implementation.*

Term (I) – Volume term (unchanged). Term (I) of (4.18) is, for each active cell $K_c \in \mathcal{T}_h$, exactly the interior weak form of the `LaplacianWeakForm` model (a specific case of `EllipticWeakForm`) evaluated on the trial and test functions, already multiplied by φ. It is obtained through the same call to the `bilinear_form`/`linear_form` methods, simply restricted to the active cells $\mathcal{T}_h$ rather than to the whole background mesh $\mathcal{T}_h^{\mathcal{O}}$.

Term (II) – Exterior flux term. Term (II) is carried by a simple boundary weak form `ExteriorFluxForm` that inherits from `AbstractBoundaryWeakForm`: this is the natural term left by integration by parts on Ω_h, which, unlike continuous FEM, is not canceled by the homogeneous Dirichlet condition. Unlike the boundary faces of DG or FE (Section 4.5.3), which have a single valid neighbor carrying the dofs, a face of $\partial\Omega_h$ has two concrete neighbors (one active, the other purely exterior): it is therefore the active neighbor that carries the dofs and orients the outward normal.

Term (III) – Stabilization terms. Term (III) of (4.18), including $\mathcal{S}_h$ and $\mathcal{S}_h^{\mathrm{rhs}}$, splits into a volume half, over the cut cells $\mathcal{T}_h^\Gamma$, and a surface half, over the stabilized faces $\mathcal{F}_h^\Gamma$. These two halves are carried by two distinct mechanisms:

- The volume part of $\mathcal{S}_h$, combined with $-\mathcal{S}_h^{\mathrm{rhs}}$, over the cut cells $\mathcal{T}_h^\Gamma$, is an integral over a single cell at a time:

$$\tau h^2 \int_{K_c} \left[\Delta(\varphi w_{h,\theta}) + f\right] \Delta(\varphi \psi_{c,j}^\theta), \qquad K_c \in \mathcal{T}_h^\Gamma.$$

  This term therefore fits the same pattern as an ordinary weak form, and is carried by a dedicated weak form, `GhostVolumeForm`, inheriting from `AbstractWeakForm` and using the same source term f as the interior weak form. This form is assembled cell by cell exactly like the interior term (I), but over $\mathcal{T}_h^\Gamma$ instead of $\mathcal{T}_h$;

- The second part of $\mathcal{S}_h$, acting on the stabilized faces $\mathcal{F}_h^\Gamma$:

$$\tau h \int_F \left[\frac{\partial}{\partial \boldsymbol{n}}(\varphi w_{h,\theta})\right] \left[\frac{\partial}{\partial \boldsymbol{n}}(\varphi \psi_{c,j}^\theta)\right], \qquad F \in \mathcal{F}_h^\Gamma.$$

  This part instead couples two cells at a time, so that it falls outside the scope of `AbstractWeakForm` (used for the two previous terms). Its expression is then written as a dedicated numerical flux, `GhostFacetFlux`, which reuses the same two-cell `AbstractFlux` structure as the fluxes of DG methods (e.g. `SIPGFlux` of Section 4.5.2).

Inactive nodes. Some nodes of **conn**, touched only by `EXTERIOR` cells (Section 4.8.1), do not appear in any of the terms (I) to (III) of (4.18). Now, since the φ-FEM scheme reuses without modification the `VariablesFE` class built on the background mesh $\mathcal{T}_h^{\mathcal{O}}$ (Section 4.7), these inactive nodes nonetheless occupy a fully-fledged entry in the complete dof vector $\boldsymbol{\beta}_h$, and hence an entry in the residual and a row in the Jacobian. Without intervention, these would remain devoid of any equation, leaving the system underdetermined. The scheme lifts this indeterminacy by fixing these dofs directly to zero: the corresponding entry of the residual becomes $\beta_I = 0$, the corresponding row of the Jacobian that of the identity, and the Jacobian-vector product is then obtained automatically by automatic differentiation. The resulting system is thus square and well-posed, with no additional penalty parameter involved.

Physical weak model. Figure 4.9 illustrates the architecture of the physical model of the φ-FEM scheme, which is implemented by `PoissonDirichletPhiFEMWeakModel`. This class inherits from `AbstractPhysicalWeakModel` and imposes the structure of the weak form of the φ-FEM scheme.

First, the model requires that the interior weak form of the model be a Laplacian, i.e. a `LaplacianWeakForm`. Next, since the φ-FEM scheme defined in (2.33) contains additional terms compared to the standard weak form, the physical model must also include the two weak forms `ExteriorFluxForm` (boundary) and `GhostVolumeForm` (interior), which both specialize `AbstractWeakForm`. These two weak forms are then

assembled by the φ-FEM scheme exactly like the interior weak form, but over the appropriate cell/face sets (Section 4.8.1).

The φ-FEM scheme is thus entirely defined by its physical model, which imposes the structure of the weak form and of the boundary conditions, and by the background mesh $\mathcal{T}_h^{\mathcal{O}}$ and the level-set function φ, which define the active cell/face sets. Because of the homogeneous Dirichlet conditions imposed in this section, the physical model defines no `Neumann`- or `Robin`-type boundary weak form, and no explicit `Dirichlet` term either, since the homogeneous Dirichlet condition is imposed structurally via the level-set function.

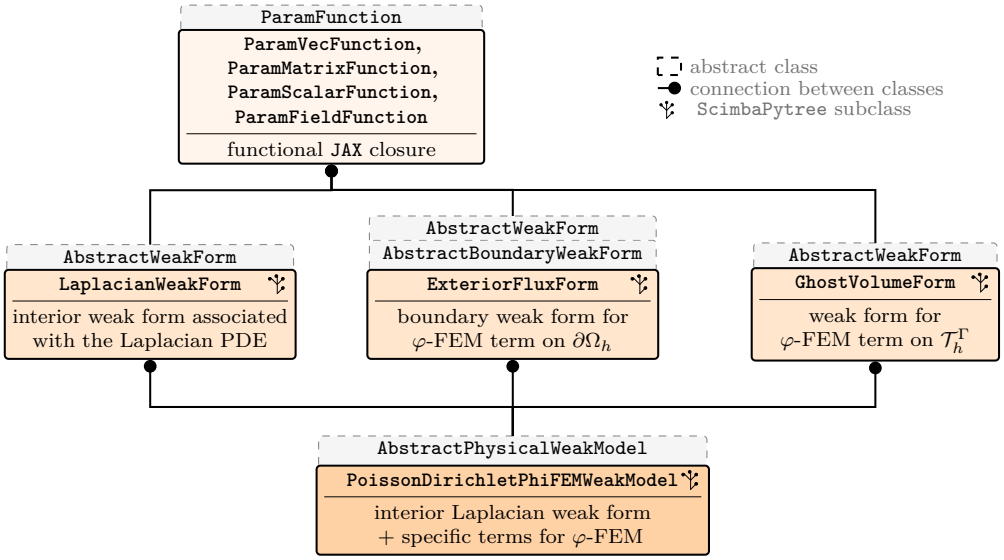


Figure 4.9.: Class hierarchy of `PoissonDirichletPhiFEMWeakModel`, detailing the weak-model box of Figure 4.8. `ExteriorFluxForm` is a boundary weak form and `GhostVolumeForm` an interior one, both sharing the `bilinear_form`/`linear_form` interface of the Poisson interior weak form.

Shared interface and restriction of the evaluation to Ω_h. Beyond the four assembly passes presented above (volume over $\mathcal{T}_h$, exterior flux over $\partial\Omega_h$, volume penalization over $\mathcal{T}_h^\Gamma$ and facet penalization over $\mathcal{F}_h^\Gamma$), no other module of Sections 4.2 to 4.6 or of Section 4.7 needs to be modified: quadrature, mapping and basis remain strictly shared with the DG and FE schemes, whose complete architecture is illustrated in Figure 4.1. Like `EllipticDGscheme` and `EllipticFEscheme`, `PoissonDirichletPhiFEMscheme` also inherits from the same abstract class `Galerkin` (Figure 4.8), so that the Newton–Raphson iterations and their dense/matrix-free variants of Section 4.6 are, once again, reused as is.

On the other hand, the evaluation of the solution must be restricted to Ω_h (and not to the whole background mesh), since $u_{h,\theta} = \varphi\, w_{h,\theta}$ only makes sense where $w_{h,\theta}$ has actually been constrained by the assembly, i.e. on the active cells only. This restriction is handled by the scheme itself (the pointwise evaluation of $u_{h,\theta}$ and the error norms are computed on the active cells only), without any change to the solving framework.

Towards shape optimization. A natural use case of φ-FEM, not yet validated numerically in `ScimBa`, is shape optimization: φ itself then becomes a learned or parameterized object, describing a domain Ω that evolves during the optimization. The current implementation, however, runs into a limitation related to `JAX` compilation: at scheme construction time, `LevelSetClassifier` computes concrete `NumPy` arrays of active cells and faces, whose size depends on how the boundary Γ crosses the background mesh. Now, `jax.jit` traces and compiles an XLA program specialized to the exact shape of its inputs: any different array size therefore forces a recompilation. If φ changes from one iteration to the next, these sets must be recomputed on the `NumPy` side, which triggers a full recompilation of the scheme at every iteration rather than a simple update of the dofs. This recompilation cost, currently prohibitive for a shape-optimization procedure, is left for future work.

Having presented the FEM and φ-FEM extensions of the `ScimBa` library, both built, like the DG scheme itself, on the same `Galerkin` architecture, we now turn to the case where θ is no longer fixed, that is, to the formalism making all three schemes learnable, in particular by presenting the bi-level optimization principle.

4.9. Learnable formalization of the Galerkin schemes

A classical application of differentiable frameworks such as the one built in this chapter is inverse problems, where one seeks to recover an unknown field of the PDE (for example its diffusion or reaction coefficient) from observation data, as illustrated on the DG scheme in Section 4.10.7. In this case, the learnable parameters θ are those of a parametrization of the unknown field (for instance a neural network).

In this section, we consider more generally a Galerkin problem (DG, FEM or φ-FEM) with learnable parameters θ, which are not limited to the fields of the PDE: they can also gather learnable components of the solver itself. The parameter θ, of size N_θ, may thus denote the parameters of a PDE field as above, the weights of a network enriching the patchwise or cellwise basis functions $\phi^\theta_{c,i}$ (Section 4.3), the parameters of the mapping $\mathcal{G}_\theta$ transforming the logical mesh (Section 4.2.2), or those of a learnable post-processing $\mathcal{P}_\theta$ (Section 4.4). Several components can even be parameterized simultaneously, for instance both a mapping and a post-processing, in which case θ gathers all their parameters. Whatever the parametrization, the goal is the same: to learn θ by minimizing a loss that involves the approximate solution $u_{h,\theta}$, with a gradient-based optimizer. Since $u_{h,\theta}$ is obtained by solving the Galerkin scheme, this requires differentiating through the solver itself, which is the purpose of this section.

Remark 4.22. *The formalism developed below relies only on the generic residual equation $\boldsymbol{R}_h(\boldsymbol{\beta}_h) = \mathbf{0}$ (4.12) shared by the three schemes (Sections 4.5, 4.7 and 4.8), and therefore covers DG, FEM and φ-FEM alike. In practice, however, only DG and FEM are currently available as learnable schemes in `ScimBa`: the natural learnable use case of φ-FEM, shape optimization with a learned level-set φ, still runs into the `JAX` recompilation issue discussed in Section 4.8, which is left for future work.*

All these parametrizations actually share the same structure: they make the approximate solution $u_{h,\theta}$, defined in (4.7), depend on the learnable parameters θ (either directly

or through the dofs). The previous sections, in particular Sections 4.5.4, 4.7.2 and 4.8.2, presented the resolution of each Galerkin scheme while treating these parameters as fixed, which allowed us so far to omit the explicit dependence of the dof vector $\boldsymbol{\beta}_h$ on θ (Notation 4.16). This is precisely the dependence we now need to make explicit, since the learning of θ goes through it. Solving the Galerkin scheme amounts to determining the global dof vector $\boldsymbol{\beta}_h(\theta) \in \mathbb{R}^{N_{\text{dofs}}}$, where N_{dofs} denotes the total number of dofs of the scheme. Since the scheme depends on θ, the dofs obtained by solving it (Section 4.6) also depend on these learnable parameters θ.

We first formulate the resulting bi-level optimization problem, then derive the gradient of the loss with respect to θ by implicit differentiation and the adjoint method, and finally summarize the corresponding steps.

Bi-level optimization problem. Let $\mathcal{J}$ denote the loss functional measuring the quality of the approximate solution $u_{h,\theta}$ (for instance the PINN loss of Example 4.23). This loss depends on θ in two ways: directly, and through the dof vector $\boldsymbol{\beta}_h(\theta)$, via the reconstruction of the approximate solution $u_{h,\theta}$ defined in (4.7). Since $\boldsymbol{\beta}_h(\theta)$ is itself determined by solving the scheme for given parameters θ, the learning problem is a bi-level optimization problem, which we write in constrained form:

$$\theta^\star = \operatorname*{argmin}_\theta \mathcal{J}(\boldsymbol{\beta}_h(\theta);\theta) \quad \text{s.t.} \quad \boldsymbol{R}_h(\boldsymbol{\beta}_h(\theta);\theta) = \mathbf{0}, \tag{4.19}$$

with $\boldsymbol{R}_h$ the assembled residual of the scheme, of size N_{dofs}; for DG, FEM and φ-FEM respectively, it is given by (4.11), (4.16) and (4.18).

Example 4.23 (PINN loss)**.** *An example already encountered in the previous chapters is the PINN loss, defined in (2.36). As in the presentation of the classical approximation methods (Section 2.2), we work here on a fixed parameter $\boldsymbol{\mu}$ and omit the dependence on it for conciseness. Rather than plugging a neural network (e.g. a dense network) directly into the loss, we plug in the approximate solution $u_{h,\theta}$, reconstructed from the dofs $\boldsymbol{\beta}_h(\theta)$ and the learnable parameters θ. If, for instance, no training data is considered ($\omega_{data} = 0$), this loss reads:*

$$\mathcal{J}(\boldsymbol{\beta}_h(\theta);\theta) = \omega_r \mathcal{J}_r(\boldsymbol{\beta}_h(\theta);\theta) + \omega_b \mathcal{J}_b(\boldsymbol{\beta}_h(\theta);\theta),$$

with ω_r and ω_b weights balancing the different terms of the loss. This loss consists of the residual loss:

$$\mathcal{J}_r(\boldsymbol{\beta}_h(\theta);\theta) = \int_\Omega \left|\mathcal{L}\big(u_{h,\theta}(\boldsymbol{x});\boldsymbol{x}\big) - f(\boldsymbol{x})\right|^2 \mathrm{d}\boldsymbol{x},$$

and the boundary loss:

$$\mathcal{J}_b(\boldsymbol{\beta}_h(\theta);\theta) = \int_\Gamma \left|u_{h,\theta}(\boldsymbol{x}) - g(\boldsymbol{x})\right|^2 \mathrm{d}\boldsymbol{x},$$

with g the Dirichlet boundary condition.

The constraint thus states that, for given parameters θ, the dofs $\boldsymbol{\beta}_h(\theta)$ solve the residual equation of the scheme:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h(\theta);\theta) = \mathbf{0}. \tag{4.20}$$

As soon as one of the fields of the PDE depends on u or the post-processing is non-linear, this residual becomes non-linear in the dofs and must be solved using the Newton–Raphson method (Section 4.6.1). Conversely, when neither of these two situations occurs, the residual is affine in the dofs and reads:

$$\boldsymbol{R}_h(\boldsymbol{\beta}_h(\theta);\theta) = A_h(\theta)\boldsymbol{\beta}_h(\theta) - \boldsymbol{b}_h(\theta),$$

with $A_h(\theta)$ and $\boldsymbol{b}_h(\theta)$ defined in Section 4.5.4 (themselves depending on θ). Its Jacobian $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta) = A_h(\theta)$ is then independent of $\boldsymbol{\beta}_h$, and a single Newton step suffices to solve (4.20).

At each iteration of the optimization over θ, the scheme must be solved to obtain the corresponding dofs $\boldsymbol{\beta}_h(\theta)$, which makes the optimization problem (4.19) more complex than a classical optimization problem on θ alone.

Computing the gradient of $\mathcal{J}$ with respect to θ. To solve the bi-level optimization problem (4.19) with a gradient-based optimizer, we need to compute the gradient of the loss $\mathcal{J}$ with respect to the parameters θ. By the chain rule, it is given by:

$$\frac{d\mathcal{J}}{d\theta} = \underbrace{\left.\frac{\partial \mathcal{J}}{\partial \theta}\right|_{\boldsymbol{\beta}_h}}_{\text{(I)}} + \underbrace{\frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}_h} \cdot \frac{d\boldsymbol{\beta}_h}{d\theta}}_{\text{(II)}}. \tag{4.21}$$

Term (I) – Direct dependence on θ. This term is the partial derivative of $\mathcal{J}$ with respect to θ with $\boldsymbol{\beta}_h$ held fixed. In our case, $\mathcal{J}$ depends directly on θ (through the basis functions, the mapping or the post-processing involved in the evaluation of $u_{h,\theta}$). This term is computed by automatic differentiation (backward) of the loss with respect to θ, "forgetting" the path through $\boldsymbol{\beta}_h$.

Term (II) – Dependence through the dofs. This term is the partial derivative of $\mathcal{J}$ with respect to $\boldsymbol{\beta}_h$, multiplied by the total derivative of $\boldsymbol{\beta}_h$ with respect to θ. Computing this term, and in particular $d\boldsymbol{\beta}_h/d\theta$, is more delicate, as it requires determining how the dofs of the scheme change when the learnable parameters vary.

Rather than computing this term explicitly, which would be costly, it can be computed efficiently by implicit differentiation using the adjoint method, the mechanism announced in Section 4.1 for differentiating the output of an iterative solver (here, the Newton solver of Section 4.6.1).

Implicit differentiation and the adjoint method. To compute the gradient of the loss $\mathcal{J}$ with respect to the parameters θ, we must therefore take into account the dependence of $\boldsymbol{\beta}_h$ on θ. To do so, we start by differentiating the constraint (4.20) with respect to θ:

$$\frac{\partial \boldsymbol{R}_h}{\partial \boldsymbol{\beta}_h}(\boldsymbol{\beta}_h;\theta)\,\frac{d\boldsymbol{\beta}_h}{d\theta} + \frac{\partial \boldsymbol{R}_h}{\partial \theta}(\boldsymbol{\beta}_h;\theta) = \boldsymbol{0},$$

which, with the Jacobian $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta) = \partial \boldsymbol{R}_h/\partial \boldsymbol{\beta}_h(\boldsymbol{\beta}_h;\theta)$, can be rewritten as:

$$\frac{d\boldsymbol{\beta}_h}{d\theta} = -J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)^{-1}\,\frac{\partial \boldsymbol{R}_h}{\partial \theta}(\boldsymbol{\beta}_h;\theta).$$

Substituting into the chain rule (4.21), we obtain:

$$\frac{d\mathcal{J}}{d\theta} = \frac{\partial \mathcal{J}}{\partial \theta}\bigg|_{\boldsymbol{\beta}_h} - \underbrace{\frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}_h} \cdot J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)^{-1}}_{\boldsymbol{\lambda}_h^T} \frac{\partial \boldsymbol{R}_h}{\partial \theta}(\boldsymbol{\beta}_h;\theta).$$

We saw in Section 4.6.1 that the inverse $J_{\boldsymbol{R}_h}^{-1}$ is never formed explicitly, and that applying $J_{\boldsymbol{R}_h}^{-1}$ to a vector of size N_{dofs} actually amounts to solving a linear system. Here, we do not want to apply $J_{\boldsymbol{R}_h}^{-1}$ to a single vector, but to an $N_{\text{dofs}} \times N_\theta$ matrix, that is, to its N_θ columns one by one. Computing term (II) therefore requires N_θ linear system solves, which is prohibitive as soon as θ is high-dimensional, as is the case for neural networks. What matters here is that we do not actually need $d\boldsymbol{\beta}_h/d\theta$ in full: we are only interested in its product with the vector $\partial\mathcal{J}/\partial\boldsymbol{\beta}_h$. Introducing the adjoint variable $\boldsymbol{\lambda}_h$ defined by:

$$J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)^T \boldsymbol{\lambda}_h = \left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{\beta}_h}\right)^T,$$

the total gradient of the loss with respect to the parameters θ then reads:

$$\frac{d\mathcal{J}}{d\theta} = \frac{\partial \mathcal{J}}{\partial \theta}\bigg|_{\boldsymbol{\beta}_h} - \boldsymbol{\lambda}_h^T \frac{\partial \boldsymbol{R}_h}{\partial \theta}(\boldsymbol{\beta}_h;\theta).$$

This requires a single system solve, regardless of the dimension of θ. The remaining product $\boldsymbol{\lambda}_h^T\, \partial \boldsymbol{R}_h/\partial\theta$ is then obtained by automatic differentiation, as a VJP, of the assembly of the residual $\boldsymbol{R}_h$ with respect to θ.

Summary of the steps. Table 4.1 summarizes the different steps to compute the gradient $d\mathcal{J}/d\theta$ of the loss with respect to the parameters θ, for the DG, FEM or φ-FEM solver alike. Note that the total cost amounts to several Newton steps plus a single linear system solve, instead of the N_θ linear system solves that would be required to compute $d\boldsymbol{\beta}_h/d\theta$ explicitly. This gradient can then be used by any gradient-based optimizer, as in the inverse problem of Section 4.10.7.

Step	Method
Computing $\boldsymbol{\beta}_h$: $\boldsymbol{R}_h(\boldsymbol{\beta}_h;\theta) = \mathbf{0}$	several Newton steps
Computing $\partial\mathcal{J}/\partial\theta$ and $\partial\mathcal{J}/\partial\boldsymbol{\beta}_h$	VJP on the loss
Computing $\boldsymbol{\lambda}_h$ (adjoint): $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)^T\boldsymbol{\lambda}_h = (\partial\mathcal{J}/\partial\boldsymbol{\beta}_h)^T$	one linear solve
Computing $\boldsymbol{\lambda}_h^T\, \partial\boldsymbol{R}_h/\partial\theta$	VJP on the assembly of $\boldsymbol{R}_h$

Table 4.1.: Steps to compute the gradient of $\mathcal{J}$ with respect to θ using implicit differentiation and the adjoint method.

Remark 4.24. *The forward solve and the adjoint solve of Table 4.1 in fact involve the same Jacobian $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)$, up to a transposition for the adjoint system (whose matrix-vector products are available at the same cost through a VJP), so the same matrix-free/Krylov solver (Section 4.6.2) used for the forward solve can be reused for the adjoint solve, without ever explicitly forming $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)$.*

Note that the bi-level problem is not solved by alternating optimization, in which θ would be held fixed while $\boldsymbol{\beta}_h$ is updated in turn. At every iteration on θ, the scheme is solved again (Section 4.6.1) to obtain the corresponding $\boldsymbol{\beta}_h(\theta)$, and this dependence is fully accounted for in the gradient $d\mathcal{J}/d\theta$ obtained above, so that $\boldsymbol{\beta}_h$ is always a deterministic function of θ, never a free variable.

Having shown how to differentiate the schemes with respect to the learnable parameters θ using implicit differentiation and the adjoint method, we now turn to numerical validations of the library. We first check, for a fixed θ, that the DG, FEM and φ-FEM implementations recover the convergence orders expected from theory, then we validate the learnable formalism itself on an inverse problem.

4.10. Numerical validations

This section proposes a series of numerical test cases, of increasing complexity, aimed at validating the implementation of the DG, FEM, and φ-FEM schemes of `ScimBa` presented in this chapter (Sections 4.5, 4.7 and 4.8). We consider four 1D test cases: the Poisson problem with homogeneous Dirichlet boundary conditions in Section 4.10.1, the same problem with mixed boundary conditions in Section 4.10.2, a general elliptic system in a convection-dominated regime in Section 4.10.3, and a non-linear Poisson problem in Section 4.10.4. We then consider two 2D test cases: the Poisson problem with homogeneous Dirichlet boundary conditions on a square domain, in Section 4.10.5, which also includes a comparison of the vectorized resolution of a batch of parameter instances with a sequential loop, then the same problem on a non-convex 3-lobed domain in Section 4.10.6. For each of these six test cases, we verify the convergence orders expected by theory, in L^2 norm and in semi-H^1 norm (broken for the DG schemes), and we illustrate the solutions and errors obtained on a representative example; for the four 1D test cases, we also study the conditioning of the associated linear system. Finally, rather than validating a forward discretization, a seventh and last test case validates the learnable formalism of Section 4.9 built on top of these schemes, on an inverse problem recovering an unknown reaction coefficient (Section 4.10.7).

These test cases follow, in spirit, those of Chapter 3 (Section 3.5), but differ from them on several points specific to the `ScimBa` framework introduced in this chapter. On the one hand, the logical mesh considered is structured, made of hypercubic cells (Section 4.2.3) rather than simplexes as in Chapter 3, which used `FEniCS`, leading us to consider local $\mathbb{Q}_k$ bases (Section 4.3) rather than $\mathbb{P}_k$. On the other hand, the first three 1D test cases and the first 2D test case also compare the DG scheme (Sections 2.2.2 and 4.5) to FEM (Sections 2.2.1 and 4.7), while the second 2D test case, which does not involve DG, compares FEM on meshes deformed by an analytical or a learned mapping with φ-FEM (Sections 2.2.3 and 4.8).

4.10.1. 1D Poisson problem

In this section, we return to the test case of Section 3.5.2, which deals with a 1D Poisson problem with homogeneous Dirichlet conditions. We consider the parameter $\boldsymbol{\mu}^{(1)} = (0.3, 0.2, 0.1)$ as defined in Section 3.5.2.2 and solve the problem on the interval

$\Omega = (0, 1)$. In Figures 4.10 and 4.11, we check the convergence orders of the two DG methods presented in Section 2.2.2.2 (SIPG and NIPG) and of the classical FEM, respectively, for Lagrange polynomials of degree $k \in \{1, 2, 3, 4\}$. For the three methods, we consider uniform 1D meshes with $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$ cells and compute the errors in L^2 norm and semi-H^1 norm (broken for the DG methods), as well as the condition number of the associated linear system matrix. Finally, Figure 4.12 compares the solutions obtained with the SIPG, NIPG and FEM methods, together with the associated absolute errors, for $k = 1$ and $N_{\text{cells}} \in \{10, 80\}$.

DG methods – SIPG and NIPG. For these two methods, the penalty parameter η_F entering κ is chosen differently depending on the method, owing to their distinct stability properties. For SIPG, the coercivity of the bilinear form a_h^{sipg} is only guaranteed for η_F sufficiently large, with a minimal threshold depending on the polynomial degree k and the dimension d. Following [116], we thus choose $\eta_F = k(k+1)d = k(k+1)$ for SIPG. For NIPG, on the other hand, the stability of a_h^{nipg} is guaranteed for any $\eta_F > 0$, independently of the polynomial degree [111], a property that is one of the historical motivations for the method compared to SIPG. In our experiments, we therefore simply take $\eta_F = 1$ for NIPG.

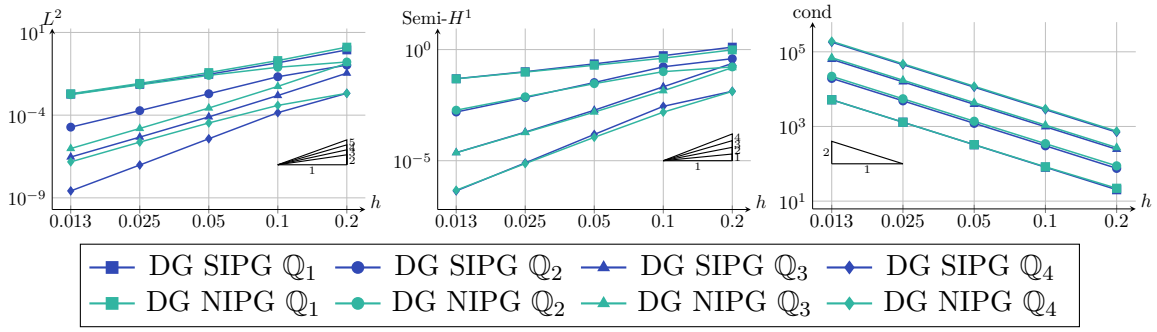


Figure 4.10.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, for the DG SIPG and NIPG methods with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Middle – Broken semi-H^1 relative error. Right – Condition number.

The results of Figure 4.10 show, as expected from the theory [9], an optimal slope k in the broken semi-H^1 norm for both DG methods (SIPG and NIPG) and for all degrees tested. In the L^2 norm, the two methods behave differently. SIPG systematically reaches the optimal order $k + 1$, as established in [9] (see Theorem 2.10). By contrast, NIPG is expected to reach only the suboptimal order k with the penalty considered here [9, 111], the optimal order $k + 1$ being guaranteed by this theory only for a strengthened penalty choice. Our numerical results show, however, that NIPG actually reaches the optimal order $k + 1$ for odd degrees ($k \in \{1, 3\}$), while remaining stuck at the suboptimal order k for even degrees ($k \in \{2, 4\}$), despite a standard penalty. This parity phenomenon is in fact a known result [85], specific to the 1D case on a uniform mesh: the authors show that, for odd degrees, the discontinuous part of the DG solution possesses an additional orthogonality property (related to the parity of the basis functions on a uniform mesh) that does not exist for even degrees, and which allows exactly one order to be gained in

the L^2 norm, without needing to strengthen the penalty as suggested by [111].

Finally, we observe that the condition number of the linear system matrix grows as h^{-2} for both methods and for all degrees tested, as shown by the alignment of the slopes in Figure 4.10. This is consistent with the theoretical analysis of [35], established for SIPG as well as for NIPG with a standard penalty. At fixed h, the condition number also increases with the degree k (the curves are ordered from $k = 1$ to $k = 4$), a classical behavior for high-order methods, independent of the flux choice.

Classical FEM. The results of Figure 4.11 confirm the optimal convergence rates expected for the classical FEM (Theorem 2.6): a slope k in the semi-H^1 norm and $k + 1$ in the L^2 norm, for all degrees tested. At fixed degree and mesh, the errors obtained are moreover of the same order of magnitude as those of the DG methods, except for even degrees, where NIPG remains comparatively less accurate, owing to the parity phenomenon already discussed above. As for the DG methods (Figure 4.10), the condition number grows as h^{-2} (see [55]), and at fixed h its level increases with the degree k. We note, however, that at fixed degree and mesh, the condition number obtained with FEM remains slightly lower than that of the DG methods (for instance, by a factor of about 2 for $k = 1$ and $N_{\text{cells}} = 80$). Moreover, at fixed N_{cells}, the FEM matrix has fewer degrees of freedom ($N_{\text{cells}} \cdot k + 1$ versus $N_{\text{dofs}} = N_{\text{cells}} \cdot N_b$ for DG, with $N_b = k + 1$ in 1D) and includes no face penalty term. FEM thus achieves an accuracy comparable to that of the DG methods while solving a smaller, better-conditioned linear system.

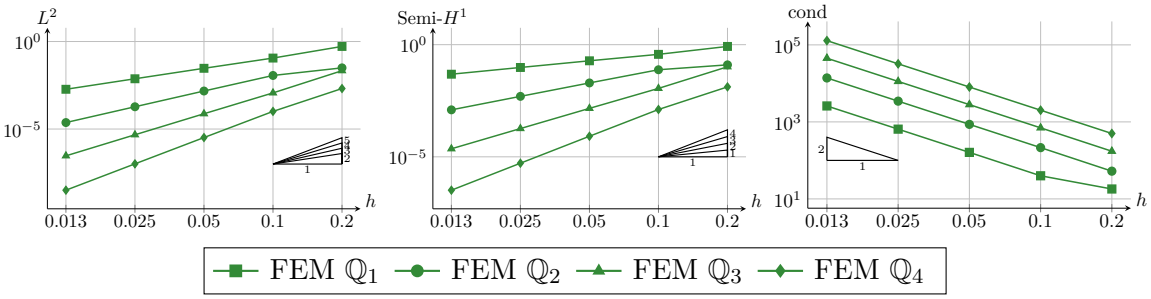


Figure 4.11.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, for the classical FEM with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Middle – Semi-H^1 relative error. Right – Condition number.

Comparison of the DG and FEM methods. Figure 4.12 illustrates, for $k = 1$ and $\boldsymbol{\mu}^{(1)}$, the solutions (top row) and the associated absolute errors (bottom row) obtained with the SIPG, NIPG and FEM methods (from left to right), for $N_{\text{cells}} \in \{10, 80\}$. As expected, the FEM solution is continuous, whereas the two DG solutions display visible jumps between cells for the coarsest mesh ($N_{\text{cells}} = 10$); these jumps are more pronounced for NIPG than for SIPG, consistent with the weaker face penalty retained for NIPG ($\eta_F = 1$) compared to SIPG ($\eta_F = k(k + 1) = 2$ for $k = 1$). For all three methods, the absolute error oscillates within each cell, with an amplitude that decreases uniformly over the whole domain as the mesh is refined from $N_{\text{cells}} = 10$ to $N_{\text{cells}} = 80$, consistent with the optimal convergence rates observed in Figures 4.10 and 4.11.

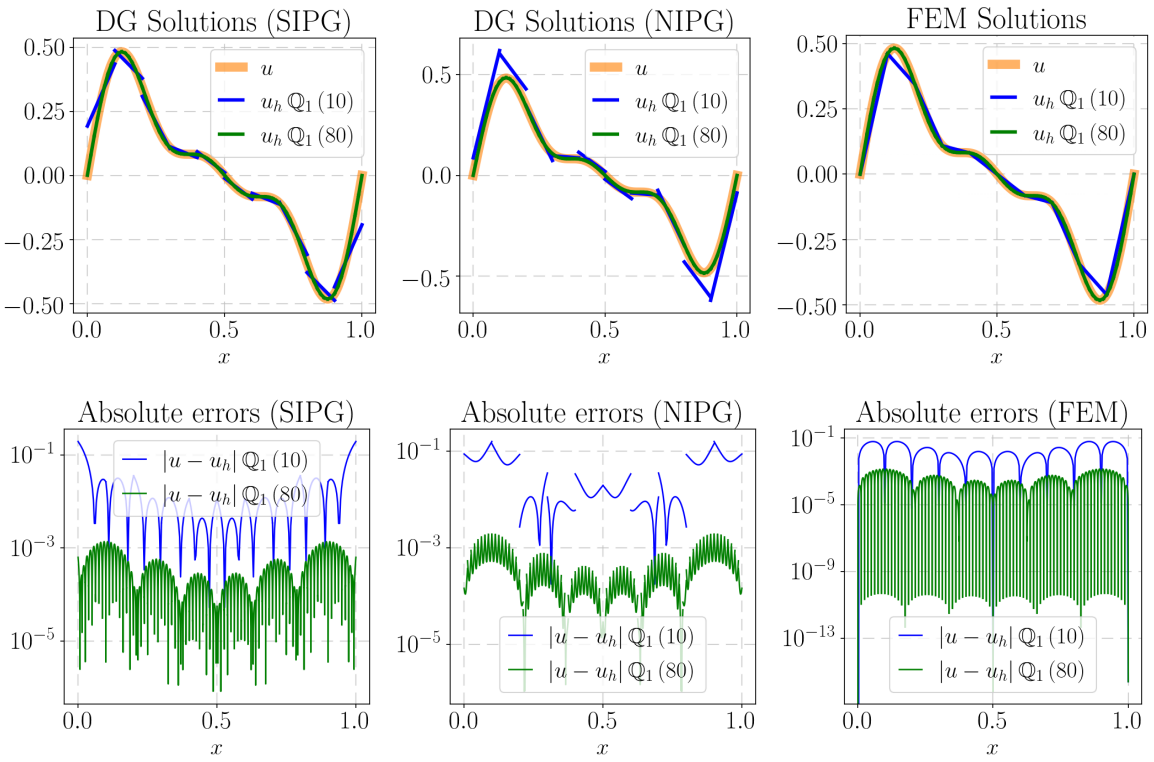


Figure 4.12.: Considering the *1D Poisson problem* with $\boldsymbol{\mu}^{(1)}$ and $k = 1$, for $N_{\text{cells}} \in \{10, 80\}$. Top – Solutions. Bottom – Absolute errors. In each row, from left to right – DG SIPG, DG NIPG and the classical FEM.

4.10.2. 1D Poisson problem with mixed boundary conditions

In this section, we return to the test case of Section 4.10.1, namely the 1D Poisson problem with the same parameter $\boldsymbol{\mu}^{(1)} = (0.3, 0.2, 0.1)$ solved on the same interval $\Omega = (0, 1)$, but replacing the homogeneous Dirichlet condition imposed at $x = 1$ with a non-homogeneous Neumann condition $\partial_x u(1) = 2\pi(\mu_1 + 2\mu_2 + 3\mu_3)$, fixed by the derivative of the exact solution at $x = 1$. The homogeneous Dirichlet condition is kept at $x = 0$ so that the problem remains well posed, since a Neumann condition at both endpoints would leave the problem defined only up to an additive constant. This test case therefore allows us to validate, in addition to the Dirichlet boundary condition already tested in Section 4.10.1, the application of a Neumann boundary condition, as presented in Section 4.5.3, using the `EllipticWeakModel` class, specifically with a `Neumann` condition (a specialization of `AbstractBoundaryWeakForm`, see Figure 4.6).

As in the previous case, we check in Figure 4.13 (DG SIPG and NIPG) and Figure 4.14 (FEM) the convergence orders for Lagrange polynomials of degree $k \in \{1, 2, 3, 4\}$, on the same uniform meshes ($N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$) and with the same L^2, semi-H^1 (broken for DG) norms, as well as the condition number of the linear system matrix. Finally, Figure 4.15 compares the solutions obtained with the SIPG, NIPG and FEM methods, together with the associated absolute errors, for $k = 1$ and $N_{\text{cells}} \in \{10, 80\}$.

DG methods – SIPG and NIPG. As in Section 4.10.1, the penalty parameter η_F is chosen in the same way for both methods: $\eta_F = k(k + 1)$ for SIPG, and $\eta_F = 1$ for

NIPG. Replacing, at $x = 1$, the homogeneous Dirichlet condition with a Neumann condition only changes the treatment of this single boundary face (Section 4.5.3): the Neumann condition is imposed by directly adding the associated boundary weak form, unlike the Dirichlet condition, which is still imposed via the numerical flux at $x = 0$.

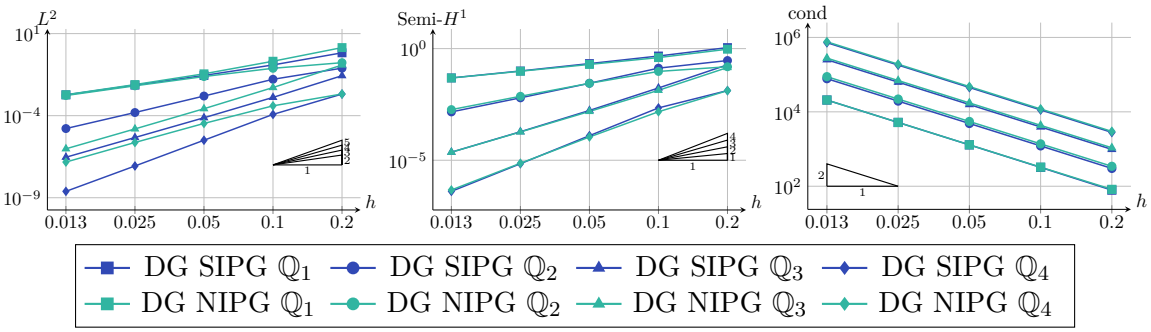


Figure 4.13.: Considering the *1D Poisson problem with mixed BC*, for $\boldsymbol{\mu}^{(1)}$, for the DG SIPG and NIPG methods with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Middle – Broken semi-H^1 relative error. Right – Condition number.

The results of Figure 4.13 show that this change of boundary condition does not degrade the convergence orders obtained in Figure 4.10: we recover an optimal slope k in the broken semi-H^1 norm for SIPG and NIPG and for all degrees tested, as well as the optimal order $k + 1$ in the L^2 norm for SIPG. For NIPG, we also recover the same parity phenomenon already discussed in Section 4.10.1: the optimal order $k + 1$ is reached in the L^2 norm for odd degrees ($k \in \{1, 3\}$), while the order remains stuck at the suboptimal value k for even degrees ($k \in \{2, 4\}$). Replacing one of the two homogeneous Dirichlet conditions with a Neumann condition therefore does not change this phenomenon, which is specific to the internal structure of the uniform mesh rather than to the nature of the imposed boundary conditions.

The condition number grows as h^{-2} for both methods and all degrees tested, in agreement with [35]. At fixed degree, the two DG methods have very close condition numbers (within about 13%). At fixed h and k, it is also systematically about 4 times higher than in the purely Dirichlet case of Figure 4.10. A spectral argument explains this factor: replacing a Dirichlet constraint with a Neumann one at one endpoint frees the slowest mode, whose wavelength doubles (a quarter-wave instead of a half-wave). The smallest eigenvalue is thus four times smaller while the largest is unchanged, as confirmed by comparing the two test cases.

Classical FEM. The results of Figure 4.14 confirm that the classical FEM achieves, as in the purely Dirichlet case (Theorem 2.6), the optimal convergence orders (a slope k in the semi-H^1 norm and $k+1$ in the L^2 norm) for all degrees tested, with no degradation linked to replacing, at $x = 1$, the Dirichlet condition with a Neumann condition. As for DG, the condition number grows as h^{-2} and increases with the degree k at fixed h. At fixed h and k, it nevertheless remains slightly lower than that obtained with the DG methods (by a factor of about 2 for $k = 1$ and $N_{\text{cells}} = 80$), as in the purely Dirichlet

case of Section 4.10.1. We also recover, relative to this same purely Dirichlet case, the roughly fourfold increase in condition number already noted above for DG.

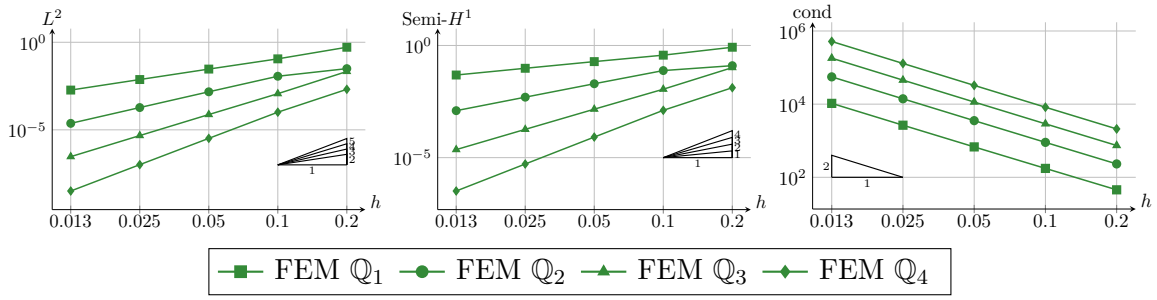


Figure 4.14.: Considering the *1D Poisson problem with mixed BC*, for $\boldsymbol{\mu}^{(1)}$, for the classical FEM with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Middle – Semi-H^1 relative error. Right – Condition number.

Comparison of the DG and FEM methods. Figure 4.15 illustrates, for $k = 1$ and $\boldsymbol{\mu}^{(1)}$, the solutions (top row) and the associated absolute errors (bottom row) obtained with the SIPG, NIPG and FEM methods (from left to right), for $N_{\text{cells}} \in \{10, 80\}$.

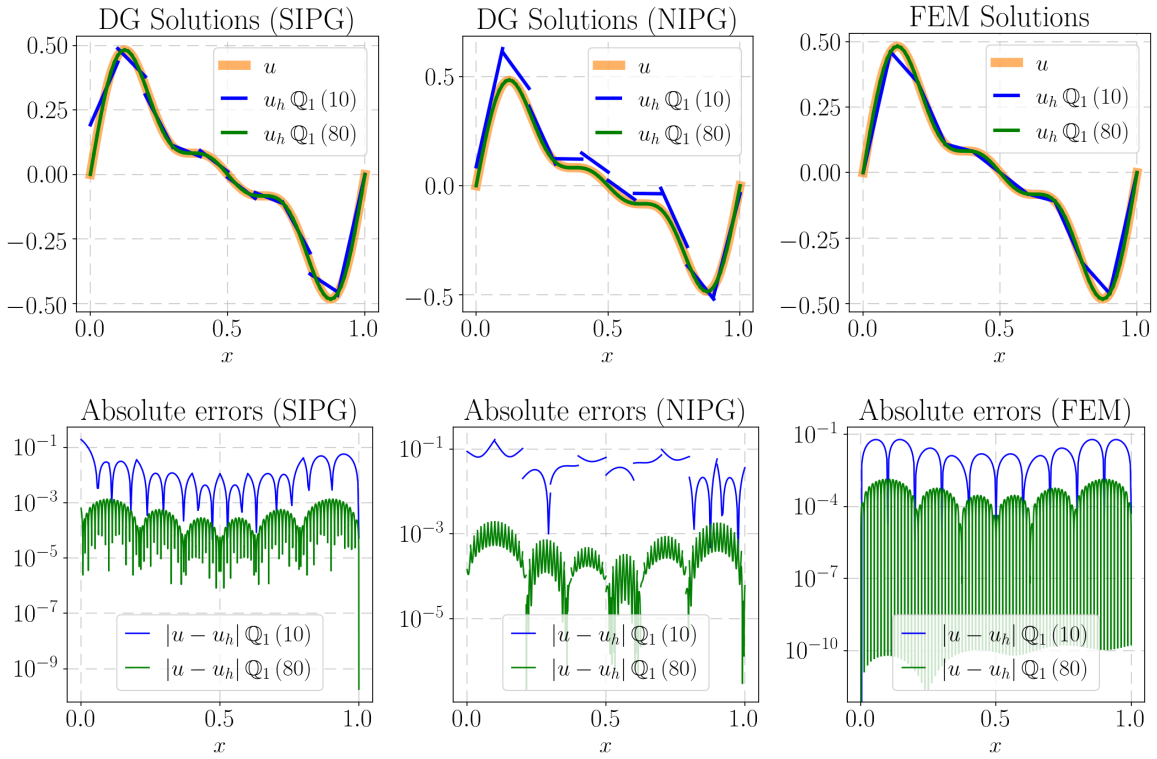


Figure 4.15.: Considering the *1D Poisson problem with mixed BC*, for $\boldsymbol{\mu}^{(1)}$, $k = 1$ and $N_{\text{cells}} \in \{10, 80\}$. Top – Solutions. Bottom – Absolute errors. In each row, from left to right – DG SIPG, DG NIPG and the classical FEM.

As in the purely Dirichlet case, the FEM solution is continuous, whereas the two DG solutions show visible jumps between cells for the coarsest mesh ($N_{\text{cells}} = 10$), more

pronounced for NIPG than for SIPG, which is consistent with the weaker face penalty used for NIPG ($\eta_F = 1$) than for SIPG ($\eta_F = k(k+1) = 2$ for $k = 1$). For all three methods, the absolute error oscillates within each cell with an amplitude that decreases uniformly over the whole domain upon refining from $N_{\text{cells}} = 10$ to $N_{\text{cells}} = 80$, including near the Neumann condition at $x = 1$, which therefore does not introduce any visible local degradation of accuracy; a behavior consistent with the optimal convergence orders observed in Figures 4.13 and 4.14.

4.10.3. 1D general elliptic system and convection-dominated regime

In this section, we revisit the test case from Section 3.5.3, which deals with a slightly more general elliptic problem coupling diffusion and convection. More precisely, we consider, as in Section 3.5.3.1, a convection-dominated regime in 1D with homogeneous Dirichlet boundary condition, for the parameter $\boldsymbol{\mu}^{(2)} = (1.5, 90)$, i.e. a Péclet number $\text{Pe} = 90$, and solve the problem on the interval $\Omega = (0, 1)$. In this section, we also consider a new parameter $\boldsymbol{\mu}^{(3)} = (1.5, 500)$, whose Péclet number $\text{Pe} = 500$ lies outside the interval $[10, 100]$ covered in the previous chapter, in order to explore an even more strongly convection-dominated regime.

As recalled in Remark 2.8 for DG methods, the purely volumetric treatment of the convective term via the fluxes presented in Section 2.2.2.2, adequate in a diffusion-dominated regime, provides no control over the jumps of the discrete solution and can degrade the convergence order in a convection-dominated regime, unless a dedicated Upwind numerical flux is added (Section 2.2.2.3). We therefore compare here, considering only the SIPG method, and for Lagrange polynomials of degree $k \in \{1, 2, 3, 4\}$, a purely volumetric treatment and a treatment supplemented by this Upwind flux (2.30), for $\text{Pe} = 90$ (Figure 4.16) and then for $\text{Pe} = 500$ (Figure 4.17); we also compare these DG results with the classical FEM for $\text{Pe} = 90$ (Figure 4.18), before a final qualitative comparison of the solutions obtained by the different methods (Figure 4.19) for $\text{Pe} = 90$.

DG methods – Pe = 90. The exact solution u defined in (3.37) exhibits a boundary layer near $x = 1$, of characteristic width of order $1/\text{Pe}$. As long as the uniform mesh does not resolve this layer, i.e. for a size $h > 1/\text{Pe}$, we find ourselves in a pre-asymptotic regime. In Figure 4.16, this is what is observed on the coarsest meshes, for SIPG as well as for SIPG with the addition of the Upwind flux, and for all degrees tested: the oscillations localized in the layer (visible for $N_{\text{cells}} = 16$ in Figure 4.19, left column) then dominate the global error norms and mask the asymptotic convergence order, which is only recovered once the mesh is sufficiently refined to resolve the boundary layer, from $N_{\text{cells}} = 128$ onward (Figure 4.19, middle column).

Once this pre-asymptotic regime is exceeded, the broken semi-H^1 norm reaches the optimal order k for both SIPG and SIPG Upwind, and for all degrees tested. In the L^2 norm, however, a marked difference is observed between the two treatments for even degrees ($k \in \{2, 4\}$): the purely volumetric treatment of the convective term plateaus at the suboptimal order k, whereas adding the upwind flux restores the optimal order $k+1$. For odd degrees ($k \in \{1, 3\}$), both treatments appear to converge toward the optimal order $k + 1$. This degradation, for SIPG alone and in the convection-dominated regime, is consistent with the mechanism described in Remark 2.8, and its correction by adding the upwind flux is consistent with [10]; its apparent dependence on the parity of the

degree, on the other hand, is reminiscent of the same type of behavior already observed for NIPG in Sections 4.10.1 and 4.10.2, without knowing whether the two phenomena share exactly the same mechanism. The condition number of the linear system matrix (Figure 4.16, right) follows the same transition: it varies non-monotonically as long as the boundary layer is not resolved, then grows as h^{-2} once the asymptotic regime is reached, as in the purely diffusive case of Section 4.10.1.

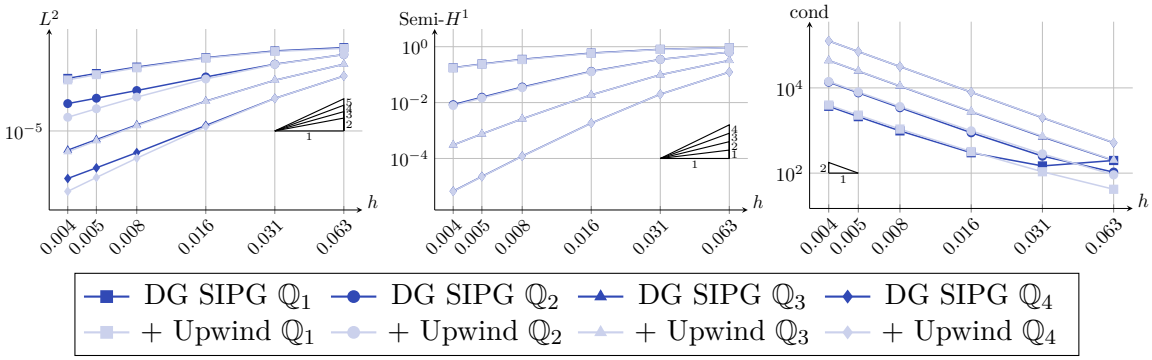


Figure 4.16.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$ (Pe = 90), for the DG SIPG method with the convective term treated in volumetric form, and with an upwind interface flux, with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{16, 32, 64, 128, 192, 256\}$. Left – L^2 relative error. Middle – Broken semi-H^1 relative error. Right – Condition number.

DG methods – Pe = 500. For $\boldsymbol{\mu}^{(3)}$, the characteristic width of the boundary layer is nearly 6 times smaller than for $\boldsymbol{\mu}^{(2)}$, which explains the larger and finer set of meshes retained here (N_{cells} going up to 768, versus 256 for Pe = 90). The pre-asymptotic regime described above is thereby markedly amplified: in Figure 4.17, the error even increases in places as the mesh is refined, as long as it remains too coarse to resolve the layer, and, for SIPG alone, the condition number, rather than increasing, transiently decreases. It is only on the finest meshes that both recover the expected asymptotic behavior.

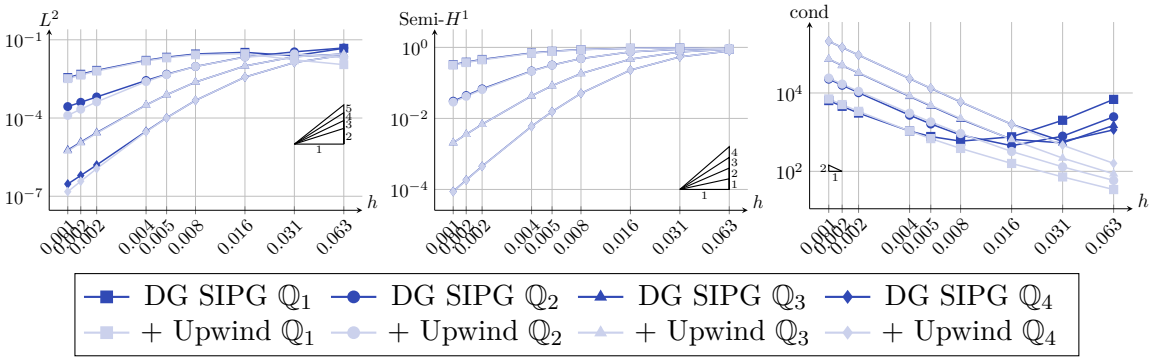


Figure 4.17.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(3)}$ (Pe = 500), for the DG SIPG method with the convective term treated in volumetric form, and with an upwind interface flux, with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{16, 32, 64, 128, 192, 256, 512, 640, 768\}$. Left – L^2 relative error. Middle – Broken semi-H^1 relative error. Right – Condition number.

Once this regime is reached, we recover, qualitatively, the same observations as for Pe = 90: an optimal slope k in the broken semi-H^1 norm for both SIPG and SIPG Upwind and for all degrees tested, the same gap between the two treatments of the convective term in the L^2 norm for even degrees (a plateau at the suboptimal order k for the volumetric treatment alone, restored to $k+1$ by adding the upwind flux), and a growth of the condition number as h^{-2} on the finest meshes.

Classical FEM – Pe = 90. Unlike DG methods, the classical FEM has no stabilization mechanism dedicated to the convective term here: it is simply assembled from the standard weak formulation (2.6) (without a discontinuous part).

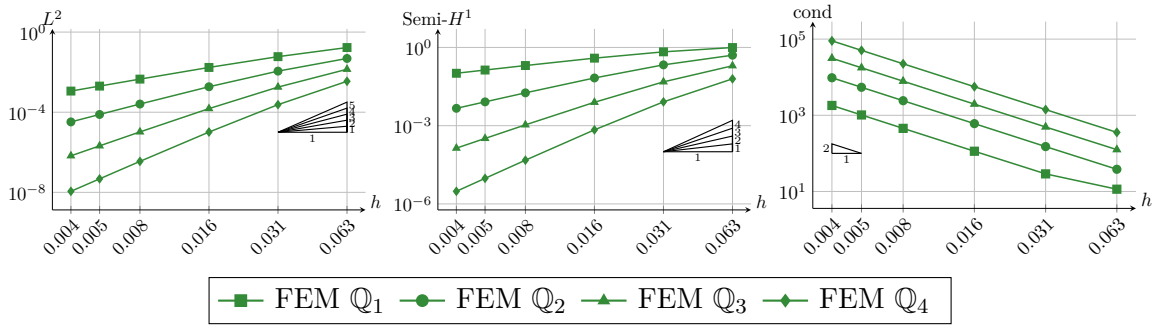


Figure 4.18.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$ (Pe = 90), for the classical FEM with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1,2,3,4\}$, and $N_{\text{cells}} \in \{16, 32, 64, 128, 192, 256\}$. Left – L^2 relative error. Middle – Semi-H^1 relative error. Right – Condition number.

In Figure 4.18, we observe that, for $N_{\text{cells}} = 16$, the $\mathbb{Q}_1$ solution shows a clear overshoot near the boundary layer (Figure 4.19, right column), characteristic of the instability of unstabilized Galerkin-type schemes in a convection-dominated regime. The classical FEM nevertheless recovers, for every degree $k \in \{1,2,3,4\}$ tested, the fully optimal convergence orders $k+1$ in the L^2 norm and k in the semi-H^1 norm, with no equivalent of the even-degree L^2 degradation observed above for SIPG without the upwind flux; the condition number likewise grows as h^{-2} on the finest meshes, as in the DG case (Figure 4.16). This result, obtained here with `ScimBa`, is consistent with the one obtained in the previous chapter for this same problem with `FEniCS` (Section 3.5.3 and Figures 3.7 and 3.9a): the same conclusions therefore hold for both implementations, for the degrees and norm considered there.

Comparison of the DG and FEM methods. Figure 4.19 illustrates the solutions (top row) and the associated absolute errors (bottom row) obtained with the DG SIPG method for $N_{\text{cells}} \in \{16, 128\}$ (left and middle columns) and with the classical FEM for $N_{\text{cells}} = 16$ (right column). For $N_{\text{cells}} = 16$, the figure highlights the contrast between the clear overshoot of the FEM solutions and the better-contained oscillations of the DG solutions.

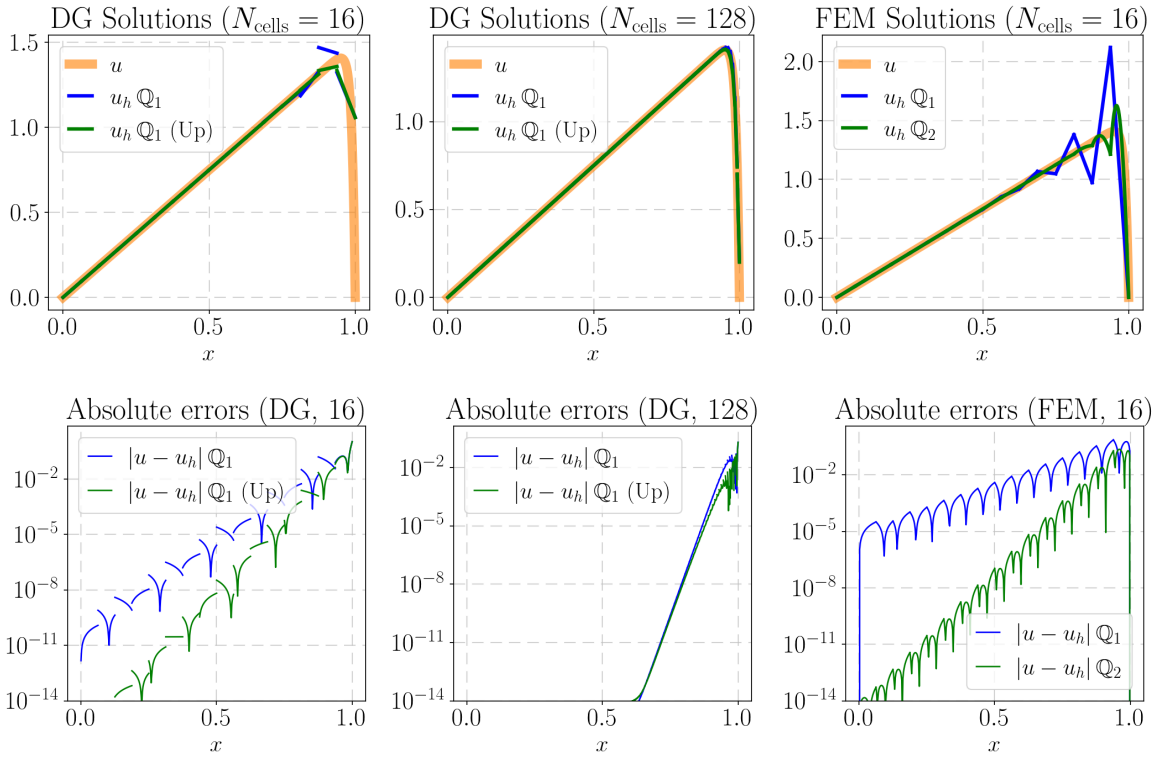


Figure 4.19.: Considering the *1D elliptic problem* with $\boldsymbol{\mu}^{(2)}$ (Pe = 90). Top – Solutions. Bottom – Absolute errors. Left – DG SIPG method with $\mathbb{Q}_1$ basis, without and with the upwind flux, for $N_{\text{cells}} = 16$. Middle – Same for $N_{\text{cells}} = 128$. Right – Classical FEM with $\mathbb{Q}_1$ and $\mathbb{Q}_2$ bases, for $N_{\text{cells}} = 16$.

4.10.4. 1D non-linear Poisson problem

In this section, we depart from the linear framework of the operator (2.2) to consider a non-linear (and non-parametric) diffusion problem with homogeneous Dirichlet conditions in 1D:

$$\begin{cases} -\partial_x\big(D(u)\,\partial_x u\big) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases}$$

on $\Omega = (0,1)$, with the diffusion coefficient $D(u) = 1 + u^2$ depending here on the solution itself. We consider the manufactured solution defined by $u(x) = \sin(\pi x)$, from which the right-hand side f is deduced by direct computation.

Since the DG residual (4.12) is no longer affine in the degrees of freedom $\boldsymbol{\beta}_h$, solving it requires the Newton–Raphson method presented in Section 4.6.1 (dense variant, with a maximum of 20 iterations), rather than the direct solution of a single linear system as in the linear case of Section 4.10.1. Only the SIPG flux (Section 2.2.2.2) is studied here, reusing exactly the same penalty κ as in the linear case, with $\eta_F = k(k+1)$, but now evaluating the diffusion coefficient at the value of the current Newton iterate on the face, rather than at a fixed field $D(x)$ independent of the solution. We therefore check, in Figure 4.20, the convergence orders obtained with SIPG for $k \in \{1,2,3,4\}$ on the same uniform meshes ($N_{\text{cells}} \in \{5,10,20,40,80\}$) as in the linear case, then compare, in Figure 4.21, the DG solution obtained for $k = 1$ with $N_{\text{cells}} \in \{10,80\}$, to

the manufactured solution.

DG methods – SIPG. The results of Figure 4.20 confirm, for all tested degrees ($k \in \{1, 2, 3, 4\}$), the optimal convergence orders already observed for SIPG in the linear case (Figure 4.10): a slope k in the broken semi-H^1 norm and $k+1$ in the L^2 norm. The non-linearity of the diffusion coefficient, resolved by Newton's method, therefore degrades neither the accuracy nor the convergence order here. As in the linear case (Figure 4.10), the condition number of the system's Jacobian matrix grows as h^{-2} (Figure 4.20, right).

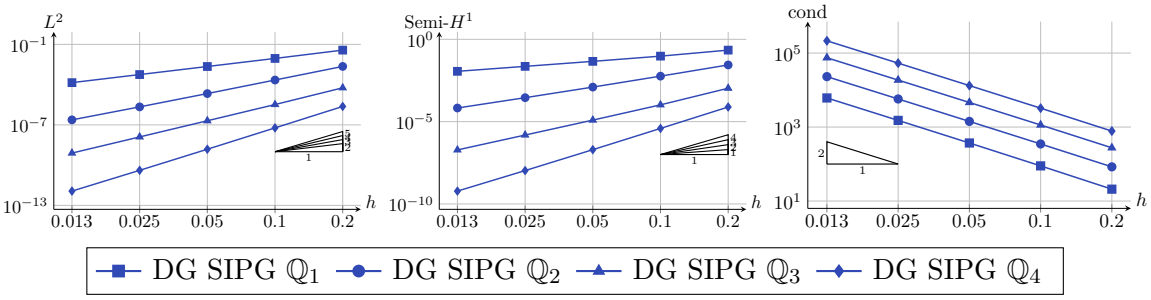


Figure 4.20.: Considering the *1D non-linear Poisson problem*, for the DG SIPG method with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2, 3, 4\}$, and $N_{\text{cells}} \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Middle – Broken semi-H^1 relative error. Right – Condition number.

We now compare, in Figure 4.21, the DG SIPG solution obtained for $k = 1$ with the manufactured solution $u(x) = \sin(\pi x)$, for $N_{\text{cells}} = 10$ and $N_{\text{cells}} = 80$, together with the corresponding absolute errors. The DG solution faithfully reproduces u, with the jumps between cells and the oscillating pointwise error already encountered in the linear case (Figure 4.12), of amplitude decreasing uniformly over Ω as the mesh is refined. This behavior is consistent with the optimal convergence orders recovered in Figure 4.20, despite the non-linearity of the problem.

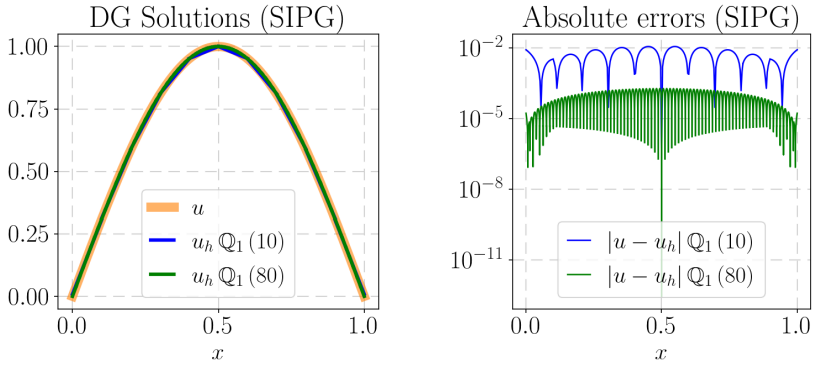


Figure 4.21.: Considering the *1D non-linear Poisson problem*, $k = 1$ and $N_{\text{cells}} \in \{10, 80\}$. Left – Solutions obtained with the DG SIPG method, compared with the analytical solution. Right – Absolute errors.

4.10.5. 2D Poisson problem in a square domain

In this section, we return to the 2D Poisson problem of Section 3.5.5, in the low-frequency case of Section 3.5.5.1, with homogeneous Dirichlet conditions. We consider the parameter $\boldsymbol{\mu}^{(1)} = (0.05, 0.22)$ as defined in that section and solve the problem on the square $\Omega = (-0.5\pi, 0.5\pi)^2$. We thus consider a 2D mesh with $\boldsymbol{N}_{\text{cells}} = (N_1, N_2)$ cells, where N_1 and N_2 are the number of cells in the x_1 and x_2 directions respectively, and we choose here $N_1 = N_2$ (i.e. a total of $N_{\text{cells}} = N_1^2$ cells). Unlike in Sections 4.10.1 to 4.10.4, the physical domain Ω is no longer identical to the logical domain $\widetilde{\Omega} = (0,1)^2$, which requires defining the physical mesh $\mathcal{T}_h$ as the image, by an analytical affine mapping (and no longer the identity), of the logical mesh $\widetilde{\mathcal{T}}_h$.

Owing to the change in dimension, the linear system resulting from the discretization of the 2D Poisson problem is much larger than in the 1D case, and direct factorization quickly becomes prohibitive. We therefore restrict ourselves here to Lagrange polynomials of degree $k \in \{1, 2\}$, and solve the linear system by matrix-free conjugate gradient (see Section 4.6.2) rather than by direct factorization. In Figure 4.22, we check the convergence orders of the SIPG method presented in Section 2.2.2.2 for Lagrange polynomials of degree $k \in \{1, 2\}$. In Figure 4.23, we likewise check the convergence orders of the classical FEM for the same degrees. Then, in Figure 4.24, we compare the SIPG and FEM solutions to the exact solution for $k = 1$. Finally, in Table 4.2, we go beyond the validation of the discretization and compare the vectorized resolution of a batch of parameter instances with a sequential loop.

DG methods – SIPG. The penalty parameter η_F entering κ (see Section 2.2.2.2) is chosen, for this 2D test case, in the same way as for its 1D counterpart (Section 4.10.1), based on [116]. We thus choose for SIPG $\eta_F = k(k+1)d = 2k(k+1)$.

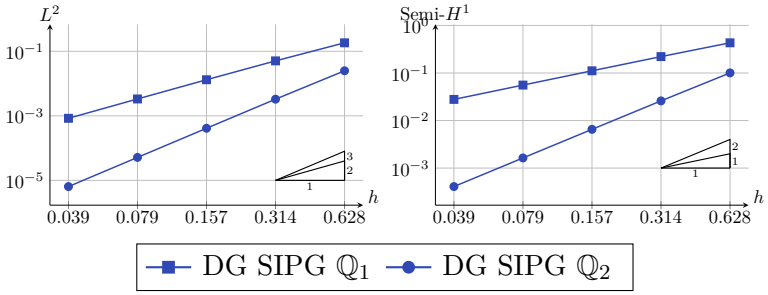


Figure 4.22.: Considering the *2D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, for the DG SIPG method with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2\}$, and $N_1 = N_2 \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Right – Broken semi-H^1 relative error.

The results of Figure 4.22 confirm, already for the coarsest mesh ($N_1 = N_2 = 5$), the optimal convergence orders expected by the theory [9] for SIPG: a slope k in the broken semi-H^1 norm and $k+1$ in the L^2 norm, for $k \in \{1, 2\}$. Consistently with the 1D case of Section 4.10.1, where SIPG already reached the optimal orders for all degrees, we observe no loss of accuracy here. The parity phenomenon observed in that section for NIPG, which relies on the result of [85] specific to the 1D setting on a uniform mesh, is not concerned, since NIPG is not studied in 2D.

Classical FEM. The results of Figure 4.23 likewise confirm the optimal convergence orders of the classical FEM (Theorem 2.6) for $k \in \{1, 2\}$, at an error level comparable to that of SIPG for a fixed mesh and degree.

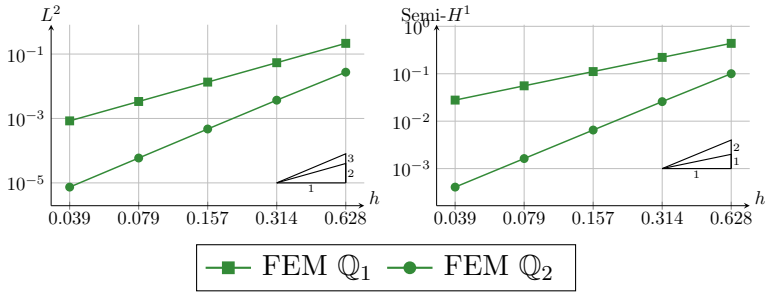


Figure 4.23.: Considering the *2D Poisson problem* with $\boldsymbol{\mu}^{(1)}$, for the classical FEM with $\mathbb{Q}_k$ Lagrange bases, $k \in \{1, 2\}$, and $N_1 = N_2 \in \{5, 10, 20, 40, 80\}$. Left – L^2 relative error. Right – Semi-H^1 relative error.

Comparison of the DG and FEM methods. Figure 4.24 compares, for $k = 1$ and $N_{\text{cells}} = 10 \times 10$, the SIPG and FEM solutions to the exact solution, together with the associated errors.

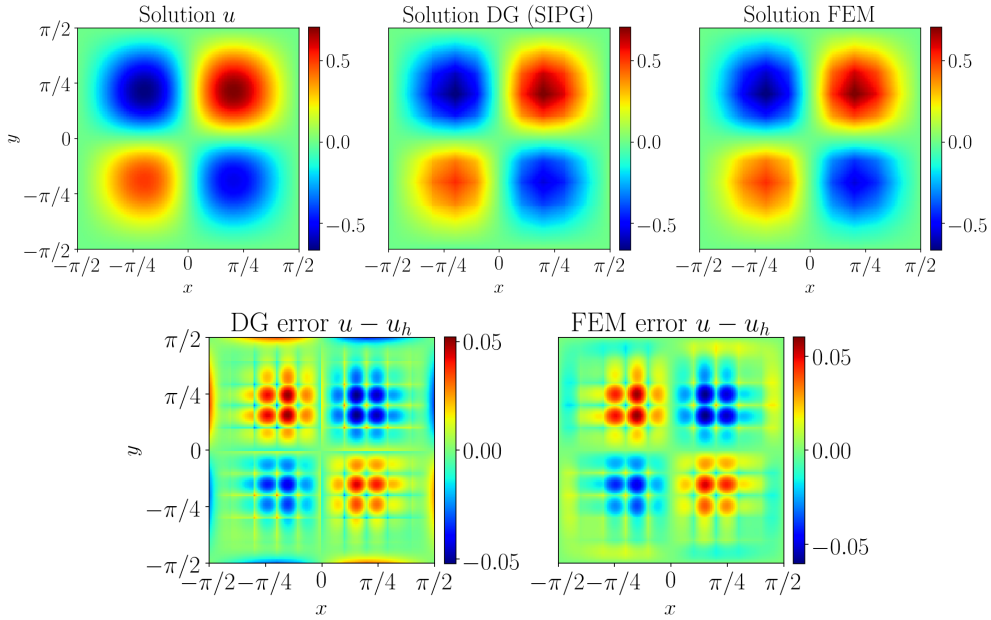


Figure 4.24.: Considering the *2D Poisson problem* with $\boldsymbol{\mu}^{(1)}$ and $k = 1$, for $N_1 = N_2 = 10$. Top – Exact solution and solutions obtained with the DG SIPG method and the classical FEM. Bottom – Corresponding errors $u - u_h$.

In Figure 4.24, we observe that, despite the coarse mesh, the two discrete solutions remain visually rather close to the exact solution, which is consistent with the convergence orders already observed for the coarsest mesh ($N_1 = N_2 = 5$) in Figures 4.22 and 4.23. The error fields, for their part, exhibit the same oscillating pattern, concentrated in the regions where the solution varies most rapidly (inside the four lobes), and of comparable amplitude for the two methods.

Batch of parameters. The problem above is solved for a single value of the parameter $\boldsymbol{\mu}$. We step slightly outside the framework of this section by considering here a set $\mathcal{D} \subset \mathcal{M}$ of N_μ parameter instances, with $\mathcal{M} = [-0.5, 0.5]^2$ as in Section 3.5.5, since in a parametric setting it is the cost of solving many problems, rather than one, that matters. An advantage of `JAX` (Section 4.1) is that the solver itself can be vectorized: `jax.vmap` solves the N_μ problems in parallel, in a single compiled program. The mesh, the basis and the variables are shared by all the problems, and only the source term f, hence the weak form, depends on $\boldsymbol{\mu}$. Table 4.2 compares this approach to a sequential loop over the N_μ values, in which the solve is compiled once, so that the comparison only measures the parallel execution (the code of the two versions is given in Appendix B.2). We consider SIPG with $k = 2$ and $N_1 = N_2 = 10$ (i.e. $N_{\text{dofs}} = 900$), with the N_μ values of $\boldsymbol{\mu}$ drawn uniformly in $\mathcal{M}$. The times are the wall-clock times of the whole resolution of the batch, from the construction of the weak form to the Newton–Krylov solve, once the compilation is done. They are averaged over 3 calls and measured on an NVIDIA RTX 2000 GPU (8 GB VRAM).

N_μ	Compiled loop	`jax.vmap`	Gain
1	0.009	0.009	1.0
4	0.028	0.019	1.5
16	0.095	0.021	4.6
64	0.372	0.040	9.3
256	1.487	0.118	12.6

Table 4.2.: Considering the *2D Poisson problem* for SIPG with $k = 2$ and $N_1 = N_2 = 10$. Computation time for N_μ values of $\boldsymbol{\mu}$, with a compiled sequential loop and with `jax.vmap`, and gain of `jax.vmap` over the loop. Times in seconds.

In Table 4.2, the vectorized solve gives the same result as the compiled loop, up to round-off (relative difference of order 10^{-15}). The gain is close to one for a single problem, and grows with the number of problems, up to a factor 12.6 for $N_\mu = 256$ (1.49 s for the compiled loop against 0.12 s). The time of `jax.vmap` varies little up to $N_\mu = 16$ (0.009 s for $N_\mu = 1$ against 0.021 s), then increases and tends towards a linear growth, whereas that of the loop is roughly proportional to N_μ from the start.

4.10.6. 2D Poisson problem in a 3-lobed shape domain

In this section, we consider again the non-convex 3-lobed domain Ω introduced in Example 4.3, along with the two mappings $\mathcal{G} = \mathcal{G}_3 \circ \mathcal{G}_2 \circ \mathcal{G}_1$ and $\mathcal{G}_\theta = \mathcal{G}_{\theta,3} \circ \mathcal{G}_2 \circ \mathcal{G}_1$ constructed there, to compare three ways of solving a Poisson problem with homogeneous Dirichlet conditions on Ω: the classical FEM on the curved mesh obtained via the analytical mapping $\mathcal{G}$, the classical FEM on the curved mesh obtained via the learned mapping $\mathcal{G}_\theta$, and φ-FEM (Sections 2.2.3 and 4.8) using directly the level-set function φ defining the physical domain Ω (without a mapping). The three schemes use the same $\mathbb{Q}_1$ Lagrange basis and share the same exact solution defined by:

$$u(\boldsymbol{x}) = \varphi(\boldsymbol{x})\, \exp(x_1) \sin(2\pi x_2),$$

which vanishes exactly on $\Gamma = \{\varphi = 0\}$ by construction, regardless of the scheme used to approximate it, and with $f := -\Delta u$ the corresponding source term.

For the three approaches, we define the same logical mesh $\widetilde{\mathcal{T}}_h$ with $\boldsymbol{N}_{\text{cells}} = (N_1, N_2)$ cells on $(0,1)^2$, where $N_1 = N_2$ is the number of cells per mesh direction. The physical meshes, shown in Figure 4.25 for $N_1 = N_2 = 12$, are defined differently depending on the scheme used:

- For FEM, this logical mesh is deformed by the two mappings $\mathcal{G}$ and $\mathcal{G}_\theta$ to obtain respectively the physical meshes:

$$\mathcal{T}_{h,\mathcal{G}} := \mathcal{G}(\widetilde{\mathcal{T}}_h), \quad \text{and} \quad \mathcal{T}_{h,\mathcal{G}_\theta} := \mathcal{G}_\theta(\widetilde{\mathcal{T}}_h).$$

  In both cases, each cell of the logical mesh is thus simply deformed into a curved cell in the physical mesh, and the logical mesh and the physical meshes have the same number of cells, namely $N_{\text{cells}} = N_1 \times N_2$;

- For φ-FEM, the logical mesh is used as is to define the background Cartesian mesh $\mathcal{T}_h^{\mathcal{O}}$, which contains $N_{\text{cells}}^{\mathcal{O}} := N_1 \times N_2$ cells. The physical mesh $\mathcal{T}_{h,\varphi}$ is defined by (2.32) as the set of active cells of $\mathcal{T}_h^{\mathcal{O}}$, and we denote by N_{cells} its number of cells (the number of active cells), so that $N_{\text{cells}} \leq N_{\text{cells}}^{\mathcal{O}}$. For $N_1 = N_2 = 12$, we obtain for example $N_{\text{cells}}^{\mathcal{O}} = 144$ background cells, of which only 64 are active (see Figure 4.25).

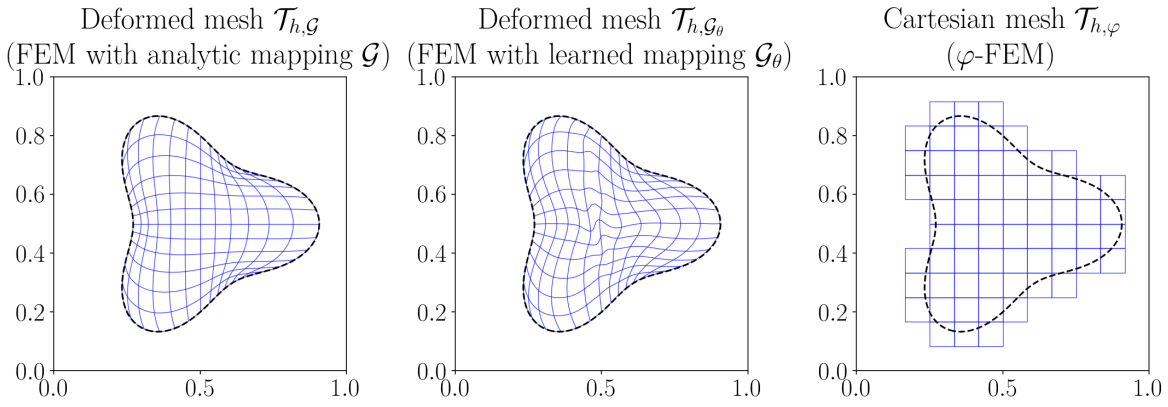


Figure 4.25.: Considering the *2D Poisson problem on the 3-lobed domain* and $N_1 = N_2 = 12$. Left – Physical mesh $\mathcal{T}_{h,\mathcal{G}}$ obtained via the analytical mapping $\mathcal{G}$. Middle – Physical mesh $\mathcal{T}_{h,\mathcal{G}_\theta}$ obtained via the learned mapping $\mathcal{G}_\theta$. Right – Computational physical mesh $\mathcal{T}_{h,\varphi}$ (active cells) associated with φ-FEM. The dashed black line is the boundary $\Gamma = \{\varphi = 0\}$.

Unlike the logical mesh and the physical mesh associated with φ-FEM, the physical meshes associated with FEM are not uniform: the mesh size h is defined there as the largest diameter among all cells of the physical mesh, and is therefore specific to each mapping. For example, considering $N_1 = N_2 = 12$ as in Figure 4.25, the physical mesh $\mathcal{T}_{h,\mathcal{G}}$ associated with the analytical mapping $\mathcal{G}$ has a mesh size h of the order of $1.33 \cdot 10^{-1}$, compared to a diameter of $4.08 \cdot 10^{-2}$ for the smallest cell. This is why, unlike in the previous test cases, we choose to compare the results of the three schemes

as a function of the number of cells N_{cells} of the physical mesh, rather than the mesh size h. We study, in Figure 4.26, the convergence in L^2 and semi-H^1 norms of the three schemes, then compare, in Figure 4.27, the solution and error fields obtained for $N_1 = N_2 = 12$.

Remark 4.25. *For the φ-FEM approach, following the founding article [53], the errors in L^2 norm and H^1 semi-norm are computed on Ω_h, the domain covered by the current mesh $\mathcal{T}_{h,\varphi}$ (with $\Omega \subset \Omega_h$), rather than on Ω itself.*

Convergence of the three schemes. Figure 4.26 shows the convergence in L^2 and semi-H^1 norms of the three schemes, for $N_1 = N_2 \in \{12, 24, 48, 96, 192\}$, with a distinct behavior for each of them. Unlike the previous convergence curves, we have chosen to represent the errors as a function of the number of cells N_{cells} of the physical mesh (active cells for φ-FEM), rather than the mesh size h. For φ-FEM, a direct computation on the logical mesh gives $h \sim N_{\text{cells}}^{-1/2}$, which allows the convergence orders in h established in the previous test cases to be transposed directly: we thus observe the optimal orders for both norms considered (1 in L^2 norm and 0.5 in H^1 semi-norm). For the two FEM schemes using a mapping, an equivalent relation is obtained just as directly, since the mappings $\mathcal{G}$ and $\mathcal{G}_\theta$ are fixed and thus independent of the mesh resolution, which explains the optimal orders observed for the analytical mapping. The FEM with the learned mapping $\mathcal{G}_\theta$, on the other hand, is limited by the geometry error of the learned mapping (average boundary error $\approx 1.1 \cdot 10^{-3}$), set by the quality of the network's training (Example 4.3). In L^2 norm, its error decreases over the first refinements, then flattens (around $9 \cdot 10^{-3}$) once the discretization error becomes comparable to the error induced by this geometric defect, rather than continuing to decrease as for the two other methods. In H^1 semi-norm, no clear plateau is visible over the range of meshes considered, although the observed orders already fall slightly below the expected 0.5: since this norm converges twice as slowly as the L^2 norm (order 0.5 versus 1 in N_{cells}), the discretization error there remains larger, and the geometric defect of the learned mapping probably does not yet dominate.

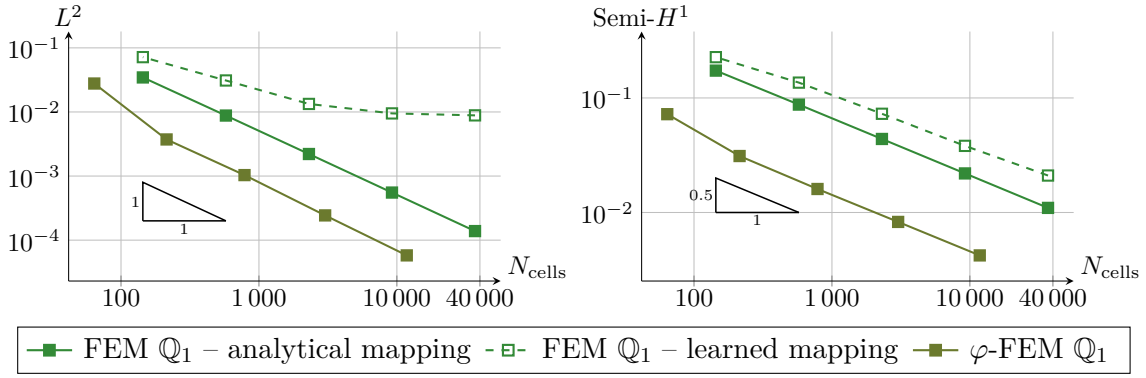


Figure 4.26.: Considering the *2D Poisson problem on the 3-lobed domain*, for the classical FEM with the analytical mapping $\mathcal{G}$, the classical FEM with the learned mapping $\mathcal{G}_\theta$, and φ-FEM, all with $\mathbb{Q}_1$ Lagrange bases, and $N_1 = N_2 \in \{12, 24, 48, 96, 192\}$. Left – L^2 relative error as a function of N_{cells}. Right – Semi-H^1 relative error as a function of N_{cells}.

Solution fields. Figure 4.27 shows, for the three schemes and $N_1 = N_2 = 12$, the exact solution, the approximate solution, and the pointwise error. The FEM solutions with the analytical mapping and with φ-FEM are visually very close to the exact solution. The FEM solution with the learned mapping $\mathcal{G}_\theta$ approximates the solution relatively well, except in the central region of the domain, where a degradation of the solution is observed, exactly where the cells appear to be most distorted by the learned mapping (see Figure 4.25 and Figure 4.2).

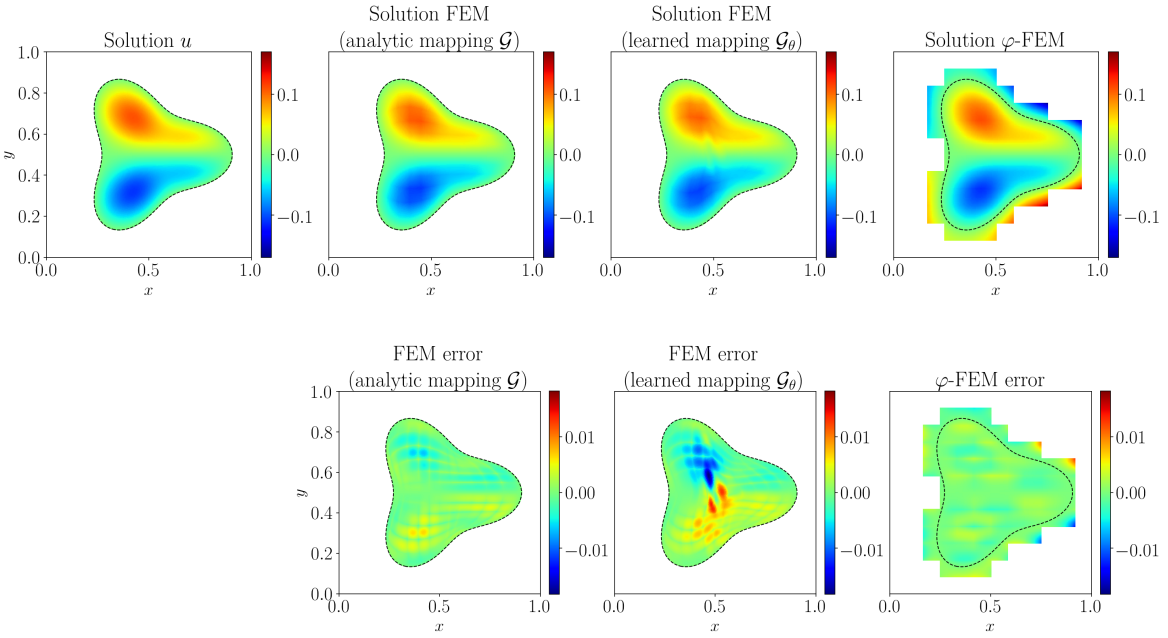


Figure 4.27.: Considering the *2D Poisson problem on the 3-lobed domain* with $N_1 = N_2 = 12$ and $k = 1$, for the classical FEM with the analytical mapping $\mathcal{G}$, the classical FEM with the learned mapping $\mathcal{G}_\theta$, and φ-FEM. Top – Exact solution u and approximate solutions, on a common color scale. Bottom – Corresponding errors $u - u_h$, on a common color scale. The φ-FEM fields are displayed on the whole computational domain Ω_h. The dashed black line is the boundary $\Gamma = \{\varphi = 0\}$.

The error fields confirm this observation. For the analytical mapping, the error shows a pattern typical of FEM schemes, with cells clearly discernible and a relatively small error amplitude. The learned mapping shows this same pattern, but with a larger amplitude (the three error fields share the same color scale), and exhibits an additional, localized error at the center of the geometry. The excess error of this approach therefore does indeed come from a localized geometric defect of $\mathcal{G}_\theta$, rather than a uniform loss of accuracy over the whole domain. For φ-FEM, the error is more diffuse over the domain, without a marked interpolation pattern, and of an amplitude comparable to that of the analytical mapping. However, the largest values appear to occur at the cells cut by the domain boundary, outside the domain Ω. These values therefore contaminate not only the visualization of the error on Ω, but also the global error in L^2 norm and H^1 semi-norm obtained in Figure 4.26, which is computed on $\Omega_h \supset \Omega$ rather than on Ω itself, in accordance with Remark 4.25.

4.10.7. 1D reaction-diffusion inverse problem

Unlike the previous test cases of this section, which all validate the forward discretizations (DG, FEM and φ-FEM) for a fixed θ, this last test case validates the learnable formalism of Section 4.9 itself: it solves a real bi-level optimization problem (4.19), in which the DG scheme is embedded and differentiated at every optimization step via implicit differentiation and the adjoint method. As noted there, only DG and FEM are currently available as learnable schemes in `ScimBa`; we consider here the DG case, on the following inverse problem.

We consider the one-dimensional reaction-diffusion problem on $\Omega = (0,1)$ with non-homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\partial_{xx} u + Ru = f, & \text{in } \Omega, \\ u = 2, & \text{on } \Gamma, \end{cases}$$

corresponding to (2.1) to (2.2) with $\boldsymbol{C} = \boldsymbol{0}$, $D = 1$ and $\mathrm{Pe} = 1$. The reaction coefficient R is unknown and must be recovered from $N_{\text{data}} = 60$ measurements $\left\{\left(\boldsymbol{x}_{\text{data}}^{(i)}, u_{\text{data}}^{(i)}\right)\right\}_{i=1}^{N_{\text{data}}}$, with $\boldsymbol{x}_{\text{data}}^{(i)}$ drawn uniformly on Ω and $u_{\text{data}}^{(i)} = u\left(\boldsymbol{x}_{\text{data}}^{(i)}\right)$ noise-free point-evaluations of the exact solution $u(x) = 2 + \sin(2\pi x)$; the source term f is set accordingly for the true coefficient $R(x) = 10\,x(1-x)$.

Learnable setup. The coefficient R is parameterized by an MLP, denoted R_θ, with 2 hidden layers of 16 neurons and a tanh activation (for a total of $N_\theta = 321$ parameters), plugged into the DG scheme as a learnable field of the PDE, on a fixed mesh of $N_{\text{cells}} = 40$ elements with a $\mathbb{Q}_1$ Lagrange basis and a SIPG flux (Section 2.2.2.2). The DG scheme is solved at every optimization step to obtain $u_{h,\theta}$ (Section 4.6); only the data-fitting term $\mathcal{J}_{\text{data}}$, defined as in (2.39), is minimized:

$$\mathcal{J}(\theta) = \mathcal{J}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| u_{h,\theta}\left(\boldsymbol{x}_{\text{data}}^{(i)}\right) - u_{\text{data}}^{(i)} \right|^2,$$

following the minimization problem (4.19). At each iteration, the gradient $d\mathcal{J}/d\theta$ is obtained by implicit differentiation and the adjoint method (Section 4.9), which reuses the Jacobian of the forward DG solve (here the matrix of the linear DG system, a single Newton step being enough, see Section 4.6.1) and requires a single additional linear system solve, regardless of the $N_\theta = 321$ parameters of R_θ. The optimization is performed with a matrix-free ENG method [98] over 100 epochs.

Training convergence and recovered fields. Figure 4.28 shows the training loss curve, together with the DG solution $u_{h,\theta}$ and the learned coefficient R_θ against their references. The training loss converges within a few epochs, and both $u_{h,\theta}$ and R_θ closely match u and R: the final L^2 errors are $6.77 \cdot 10^{-4}$ for $u_{h,\theta}$ and $3.92 \cdot 10^{-2}$ for R_θ, the latter naturally larger since R is only recovered indirectly, through the $N_{\text{data}} = 60$ point measurements of the solution u it produces. The training time is about 11 seconds on an NVIDIA RTX 2000 GPU (8 GB VRAM). This confirms, in the context of a specific inverse problem, the validity of the two-level formalism as implemented in `ScimBa`.

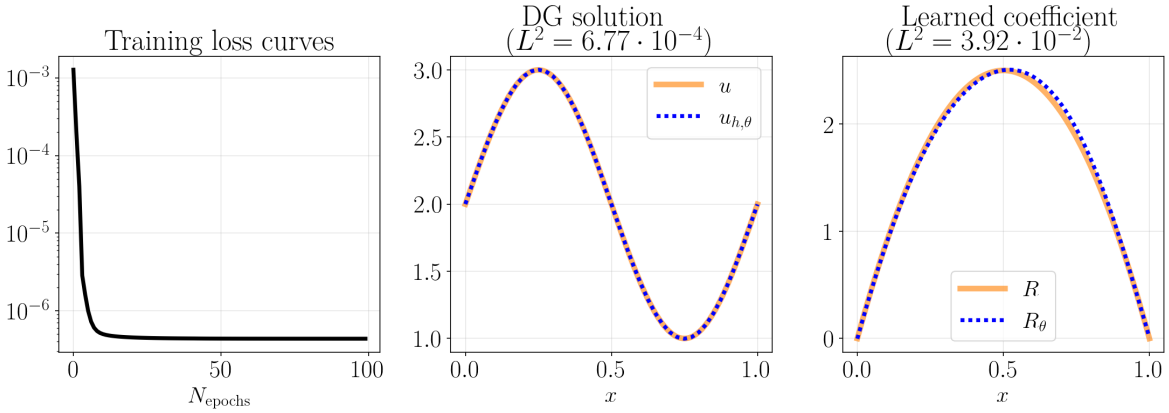


Figure 4.28.: Considering the *1D inverse problem* of learning a reaction coefficient R_θ from $N_{\text{data}} = 60$ observations of u, with the DG SIPG method, a $\mathbb{Q}_1$ Lagrange basis and $N_{\text{cells}} = 40$. Left – training loss curve. Middle – DG solution $u_{h,\theta}$ against the exact solution u. Right – learned coefficient R_θ against the true coefficient R.

Having checked that the DG, FEM and φ-FEM implementations of `ScimBa` *recover the expected convergence orders on this range of test cases, and that the learnable formalism recovers an unknown coefficient on an inverse problem, we now turn, in Chapter 5, to exploiting the learnable formalism of Section 4.9 in earnest. Whereas the inverse problem of Section 4.10.7 illustrated it on a single learnable field, the next chapter learns the approximation bases themselves. We first show, through a dimensional study, that classical Galerkin bases become increasingly costly as the space dimension grows, unlike network-based approximations, which motivates learned bases combining both.*

5 Learnable Galerkin methods

Contents

5.1. Dimensional study 164
5.1.1. Classical approximation methods (FEM, DG) 165
5.1.2. Learnable approximation methods (data-fitting, PINN) 168
5.1.3. Limits of classical bases and motivation for neural enrichment . 170
5.2. Non-parametric enrichment (PINN loss) – 1D test case 171
5.2.1. Finite element method – Patchwise basis 174
5.2.2. Finite element method – Cellwise basis 198
5.2.3. Discontinuous Galerkin – Patchwise basis 202
5.2.4. Discontinuous Galerkin – Cellwise basis 206
5.2.5. Comparison of the schemes, conclusion and limitations 210
5.3. Non-parametric enrichment (PINN loss) – 3D test case 212
5.4. Parametric enrichment (neural operator) – 1D test case 218
5.4.1. Linear transport problem 221
5.4.2. Turning point problem 227
5.4.3. Comparison, conclusion and limitations 232

The notations introduced in this chapter, together with those of Chapter 4, are recalled in Appendix A.1.3.

In this chapter, we return to the implementation of the `ScimBa` library described in Chapter 4, and in particular to the learnable DG and FEM schemes formalized in Section 4.9. We focus on a point raised in the introduction (Chapter 1), namely the lack of expressivity of classical approximation spaces. On a first set of problems, we investigate whether these spaces can be enriched with neural bases to obtain better approximations. Among the modules of the library that we use, the one for basis functions (Section 4.3) is of particular interest here, since it is the one parametrized by neural networks. The classical basis is multiplied by a function depending on a network, in the spirit of the multiplicative enrichment of Section 3.2 (see Remark 5.6). All the results of this chapter are obtained with the `ScimBa` library and performed in double precision (`float64`). To reproduce these results, the code is available on GitHub[1], and a minimal example is given in Appendix B.1.

In Section 5.1, we begin by motivating such an enrichment with a dimensional study. On a manufactured solution, we compare, as the dimension d increases, the number of

[1] https://github.com/flecourtier/thesis-learnable-methods

degrees of freedom that classical methods (FEM and DG) need to reach a given error with the number of weights of a neural network (data fitting and PINN).

We then study the enrichment of a classical Galerkin basis by a network. In Section 5.2, we first consider the non-parametric case, where a network is trained for a fixed value of the parameter $\boldsymbol{\mu}$, on a 1D advection-diffusion problem, for continuous finite elements and DG, with a patchwise or a cellwise basis, and we discuss the choice of the multiplier that enriches the Lagrange basis. The aim is to check that, for a fixed number of degrees of freedom and leaving aside the training cost, the learned basis reduces the error compared with the classical basis, and to identify the hyperparameters that influence this result the most. In Section 5.3, we extend this study to a 3D problem, where we also compare the cost with that of the classical method.

Finally, we turn to the parametric setting. In Section 5.4, a single network is trained for a whole family of parameters, so that the learned basis can be used for values of $\boldsymbol{\mu}$ never seen during training, without any new training. We study two problems of increasing difficulty, a linear transport problem (Section 5.4.1), then a turning point problem (Section 5.4.2) in which the layer moves with the parameter. For both, we compare a basis shared by all the values of $\boldsymbol{\mu}$ with a basis that depends on $\boldsymbol{\mu}$. For the linear transport problem, we also consider a basis that depends on $\boldsymbol{\mu}$ only through the transport field of the PDE, which is encoded by a second network.

Remark 5.1. *In the sections on enriched bases, Galerkin methods with a classical basis and with a network-enriched basis are compared by computing the relative errors in L^2 norm for a fixed number of degrees of freedom N_{dofs}. As in Chapter 3, the gain is defined as the ratio between the error of the classical method and that of the enriched method. The gain defined in this way is computed at a fixed number of degrees of freedom and does not account for the training cost, which is not systematically studied. It is discussed qualitatively in Section 5.2.5 and compared with that of the classical method in Section 5.3. The number of weights N_θ of the network is counted separately.*

5.1. Dimensional study

Before turning to the learnable Galerkin schemes formalized in Section 4.9, we first wish to justify one of the motivations of this manuscript, namely the interest of enriching classical Galerkin bases with neural bases. To do so, we begin with a dimensional study that illustrates the fact that classical Galerkin bases (FEM, DG) become increasingly costly, in number of degrees of freedom and hence in memory, as the spatial dimension d increases. More precisely, this phenomenon, known as the curse of dimensionality, makes the number of degrees of freedom required to reach a given accuracy grow exponentially with d [28]. Conversely, a neural network approximation can take advantage of the structure of the solution [15].

To illustrate this, we consider different values of the dimension d and the following manufactured solution on the unit hypercube $\Omega = [0,1]^d$:

$$u(\boldsymbol{x}) = \sin\left(2\pi \sum_{i=1}^{d} x_i\right), \tag{5.1}$$

which is a plane wave in the diagonal direction $\boldsymbol{x} \mapsto \sum_{i=1}^{d} x_i$, whose profile is always the same sinusoid, over a range of length d (it thus carries d periods). We then consider

the d-dimensional Poisson problem with non-homogeneous Dirichlet condition on Γ, defined by:

$$-\Delta u = f \text{ in } \Omega, \qquad u = g \text{ on } \Gamma, \tag{5.2}$$

with f the source term corresponding to the manufactured solution (5.1) and g the corresponding Dirichlet condition on the boundary Γ.

This manufactured solution is deliberately favorable to networks. It varies only along a single direction of space, the diagonal direction, so that it is enough to align the weights of the first layer with this direction for the network to have only a 1D profile to represent. A Cartesian mesh, by contrast, is aligned with the axes and therefore does not see this direction, which implies that it must resolve the oscillations in the d directions. This advantage nevertheless remains tied to the structure of the solution, since for more general functions the curse of dimensionality also persists for neural networks [132], as it does for PINN training [74]. The study thus does not claim to show that networks escape the curse of dimensionality in general.

The study unfolds in three stages. In Section 5.1.1, we first solve problem (5.2) with the two classical Galerkin methods, namely the FEM and the DG, in order to quantify how the number of dofs needed to reach a given accuracy grows with d. We then approximate, in Section 5.1.2, the solution to this same problem with a neural network, trained either by data-fitting or with the PINN method, and we compare the number of weights and the accuracy reached with those of the classical methods. Finally, in Section 5.1.3, we discuss the limits of classical bases and those of networks, which allows us to motivate the following sections.

5.1.1. Classical approximation methods (FEM, DG)

We consider problem (5.2), which we solve with two classical Galerkin methods, the FEM and the DG, both implemented in `ScimBa` (Sections 4.5 and 4.7) from the same weak form of the Poisson problem. The DG method uses the SIPG flux (Section 2.2.2) with the penalty $\eta_F = k(k+1)d$ as in Section 4.10.1, the Dirichlet condition being imposed weakly. For the FEM (Section 2.2.1), it is imposed strongly on the boundary nodes and no numerical flux is needed.

Both schemes are solved by the matrix-free Newton solver of Section 4.6.2, each iteration of which solves the linearized system with a Krylov solver (conjugate gradient) applied to the matrix-vector product computed by JVP, without ever assembling the dense Jacobian matrix of size $N_{\text{dofs}} \times N_{\text{dofs}}$. In practice, since problem (5.2) is linear, a single Newton iteration suffices. The condition number is estimated without forming the matrix, by applying the Lanczos algorithm [113] to the same matrix-vector product, which gives an estimate of the extreme eigenvalues of the Jacobian and hence, the operator being symmetric positive definite, of its condition number. For the FEM, it is computed on the interior block (the identity rows of the strongly imposed boundary dofs being removed), and for the DG on the whole operator.

In Figures 5.1 and 5.2, we study the dependence on the dimension, for $d \in \{1, 2, 3, 4\}$, and Lagrange polynomials $\mathbb{Q}_k$ of degree $k = 1$, i.e. $N_b = (k+1)^d = 2^d$ basis functions per cell. We consider uniform meshes of $\boldsymbol{N}_{\text{cells}} = (N_1, \ldots, N_d)$ cells, where each dimension has the same number of subdivisions $N_1 = \cdots = N_d$, and we denote by N_{cells} the total number of cells. For a fixed number N_1, the number of dofs associated with the DG is

$N_{\text{dofs}} = N_{\text{cells}} \cdot N_b = (2N_1)^d$, whereas that associated with the FEM is $N_{\text{dofs}} = (N_1+1)^d$. In each of the two figures, we plot the L^2 relative error and the H^1 semi-norm as well as the condition number of the linear system matrix as a function of the characteristic mesh size $h := \frac{\sqrt{d}}{N_1}$, i.e. the diagonal of a cell.

Table 5.1 summarizes these results by giving, to reach a fixed L^2 relative error of $6 \cdot 10^{-2}$ (a large error, but close to the best accuracy that we could reach in dimension 4, see Remark 5.2), the associated number of cells N_1 per direction, number of dofs N_{dofs}, mesh size h and condition number. N_1 is obtained by interpolating the convergence curves, rounded up to the next integer, and N_{dofs} and h are then deduced from it by the formulas above.

Remark 5.2. *The results are obtained on an NVIDIA RTX 2000 (8 GB VRAM), on which we ran into memory limits at the finest meshes. Some of them could probably have been avoided with options offered by* `ScimBa` *or with a better implementation of some parts of the pipeline, so that we do not claim that a FEM or DG code cannot go beyond the meshes obtained here. Memory necessarily becomes a limit at some point in high dimension, however, since the number of dofs grows as* N_1^d*. In dimension* $d = 4$*, we did not go below an error of about* $5 \cdot 10^{-2}$ *for either method, which motivates the choice of the threshold of* $6 \cdot 10^{-2}$ *in Table 5.1.*

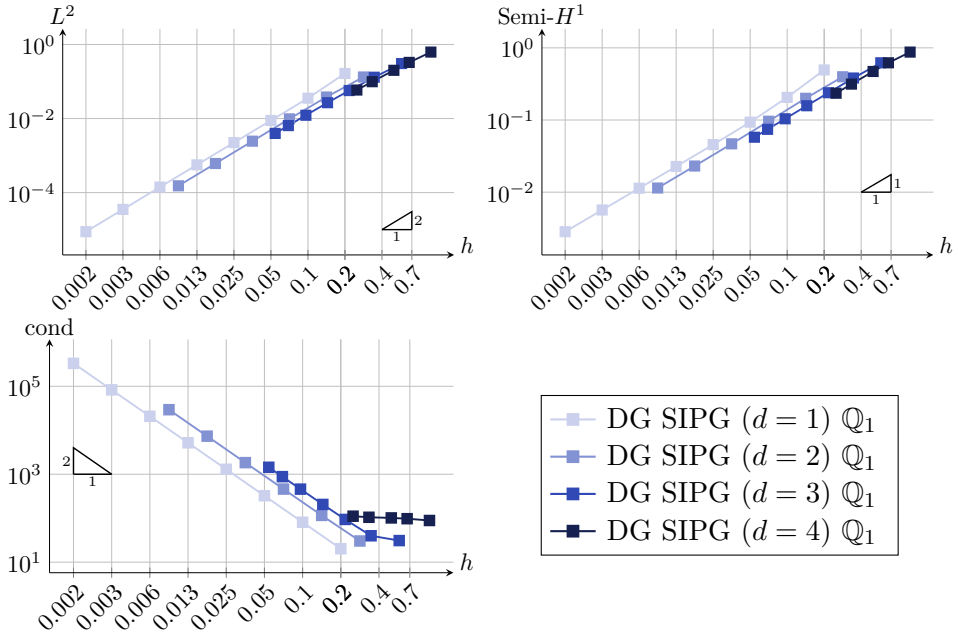


Figure 5.1.: Considering the *d-dimensional Poisson problem* for $d \in \{1, 2, 3, 4\}$, for the DG SIPG method with $\mathbb{Q}_1$ Lagrange bases and uniform meshes with N_1 cells per direction. Top left – L^2 relative error. Top right – Broken semi-H^1 relative error. Bottom – Condition number.

The convergence curves of Figures 5.1 and 5.2 recover, for both methods, the optimal orders expected for $\mathbb{Q}_1$ elements, namely a slope of 2 in L^2 norm and 1 in H^1 semi-norm (broken for the DG). For $d \in \{1, 2\}$, these slopes are observed from the coarsest meshes. For $d \in \{3, 4\}$, on the other hand, the first points are pre-asymptotic, but the slopes begin to stabilize on the finest meshes.

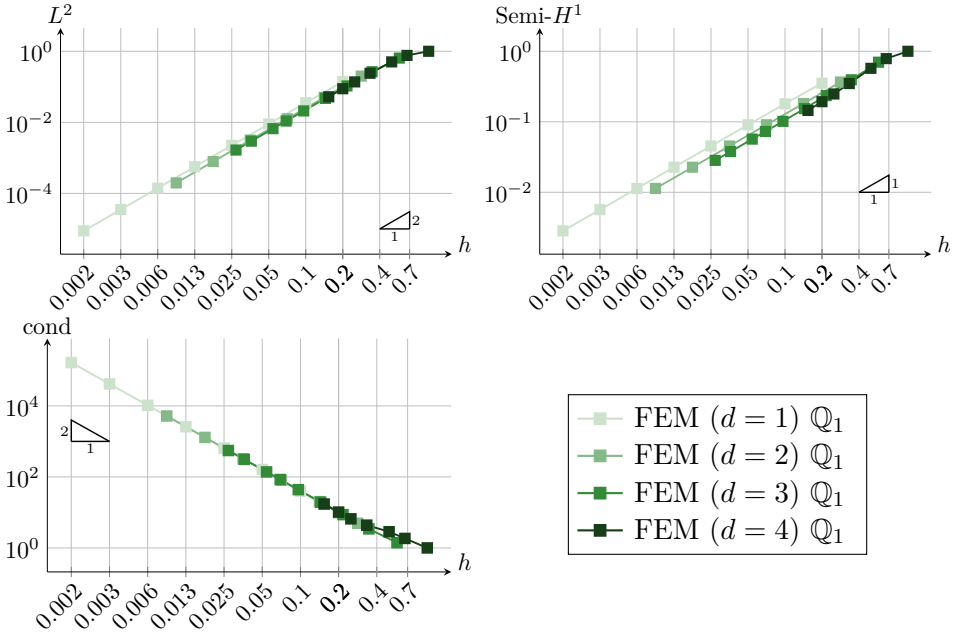


Figure 5.2.: Considering the *d-dimensional Poisson problem* for $d \in \{1, 2, 3, 4\}$, for the classical FEM with $\mathbb{Q}_1$ Lagrange bases and uniform meshes with N_1 cells per direction. Top left – L^2 relative error. Top right – Semi-H^1 relative error. Bottom – Condition number.

	FEM				**DG**			
d	N_1	$N_{\mathbf{dofs}}$	h	**cond**	N_1	$N_{\mathbf{dofs}}$	h	**cond**
1	8	9	$1.25 \cdot 10^{-1}$	$2.31 \cdot 10^{1}$	8	16	$1.25 \cdot 10^{-1}$	$5.03 \cdot 10^{1}$
2	10	121	$1.41 \cdot 10^{-1}$	$1.69 \cdot 10^{1}$	8	256	$1.77 \cdot 10^{-1}$	$7.11 \cdot 10^{1}$
3	11	1 728	$1.57 \cdot 10^{-1}$	$1.53 \cdot 10^{1}$	8	4 096	$2.17 \cdot 10^{-1}$	$8.89 \cdot 10^{1}$
4	13	38 416	$1.54 \cdot 10^{-1}$	$1.51 \cdot 10^{1}$	8	65 536	$2.5 \cdot 10^{-1}$	$(1.11 \cdot 10^{2})$

Table 5.1.: Considering the *d-dimensional Poisson problem*, for the classical FEM and the DG SIPG method with $\mathbb{Q}_1$ Lagrange bases. Number of cells per direction N_1, number of dofs N_{dofs}, mesh size h and condition number needed to reach an L^2 relative error of $6 \cdot 10^{-2}$, interpolated from the convergence curves. The condition number in parentheses (DG, $d = 4$) is a pre-asymptotic artifact and not a reliable estimate.

We see in Figure 5.2 that, in every dimension, the condition number associated with the FEM (interior block) follows the classical law h^{-2}, where $h = \sqrt{d}/N_1$ is the diameter of a cell. With this choice of h, the curves of the different dimensions are almost superposed, which is specific to the uniform Cartesian meshes and the $\mathbb{Q}_1$ elements considered here. For the DG in Figure 5.1, the slopes at $d \in \{1, 2, 3\}$ correctly follow h^{-2}, whereas at $d = 4$ the condition number remains almost constant (between 89 and 111) over the whole range tested, up to $N_1 = 8$. This is a pre-asymptotic behavior, so that the condition number in parentheses in Table 5.1 should not be read as a reliable estimate of the condition number of the DG at $d = 4$.

In Table 5.1, we observe that to reach the same L^2 relative error of $6 \cdot 10^{-2}$ the number of dofs grows exponentially with d for both methods. Between $d = 1$ and $d = 4$, it goes from 9 to 38 416 for the FEM and from 16 to 65 536 for the DG. The number of cells N_1 per direction, for its part, seems relatively stable with d, with a constant value of 8 for the DG against a small growth from 8 to 13 for the FEM. For equal accuracy, the DG seems to require about twice as many dofs as the FEM although it uses a smaller or equal number of cells per direction. The condition number, for its part, remains moderate in every dimension (between 15 and 23 for the FEM, between 50 and 89 for the DG for $d \leq 3$), and does not grow with d as the number of dofs does.

5.1.2. Learnable approximation methods (data-fitting, PINN)

We consider the same problem (5.2), now approximated by an MLP (see Notation 2.13) u_θ with 3 hidden layers of the same width, equal to 16, 24 or 32, and tanh activation. The number of weights N_θ (biases included) plays here a role similar to that of N_{dofs} for the FEM and the DG. For a fixed d, it depends only on the width and grows linearly with d, with a slope equal to the width, since d only enters the first layer. For example, for a width of 16, the number of weights is $N_\theta = 593$, 609, 625 and 641 for $d = 1, 2, 3$ and 4 respectively. Unlike the classical methods presented in Section 5.1.1, the number of weights N_θ thus does not grow exponentially with d, but linearly.

We compare here two ways of training this network. The first (data-fitting) consists in considering only the data loss defined by:

$$\mathcal{J}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| u_\theta(\boldsymbol{x}_{\text{data}}^{(i)}) - u(\boldsymbol{x}_{\text{data}}^{(i)}) \right|^2,$$

estimated on $N_{\text{data}} = 10^4$ data points drawn uniformly in Ω and redrawn at each epoch. This method uses the exact solution and serves only to measure the approximation capacity of the network alone, independently of the difficulty of optimizing the loss of the Poisson problem. The second is the PINN method (Section 2.3.1), which minimizes:

$$\mathcal{J}(\theta) = \omega_r \mathcal{J}_r(\theta) + \omega_b \mathcal{J}_b(\theta),$$

where $\mathcal{J}_r$ is the residual of the PDE (5.2) and $\mathcal{J}_b$ the boundary loss, estimated by Monte Carlo on $N_{\text{col}} = 10^4$ points drawn in Ω (i.e. the same number as for the data-fitting) and on $N_{\text{bc}} = 10^4$ points drawn uniformly on Γ, all redrawn at each epoch, with the weights $\omega_r = 1$ and $\omega_b = 10$. The choice of these numbers of points, kept fixed as d increases, is discussed in Remark 5.3.

Remark 5.3. *The numbers of points $N_{col} = 10^4$ on Ω and $N_{bc} = 10^4$ on Γ are kept fixed in this study as d increases. The N_{bc} points are split equally among the $2d$ faces of the hypercube, which gives very different budgets depending on the dimension. In $d = 1$, the boundary reduces to two points (0 and 1), so that each is simply drawn 5 000 times and increasing N_{bc} would be of no interest. For Ω, the 10^4 points form a very dense sample of the segment. In $d = 4$, on the other hand, on a Cartesian grid they would correspond to only $\sqrt[4]{10^4} = 10$ points per direction in Ω, and the 1 250 points on each of the 8 faces, which have dimension 3, to about 11 points per direction. These budgets then become small, and could limit the accuracy of the network.*

Both trainings use the ENG optimizer [98] of `ScimBa`, with a number of epochs $N_{\text{epochs}} = 50d$ (i.e. 50 to 200) and a Gram matrix regularization equal to the default value of the optimizer (10^{-6}) for the data-fitting and to 10^{-4} for the PINN. The PINN needs a larger regularization because, from $d = 2$, its training diverges or stalls with the default value, whereas the data-fitting keeps it. To account for the non-convex nature of the training, each pair of dimension and width considered is trained with 10 different initializations, and the network retained for each training is the one with the lowest loss over the epochs. The results are summarized in Table 5.2, which gives for each pair the mean and the minimum of the L^2 relative error obtained over the 10 initializations, as well as the final loss of the network that reached the minimum.

		Data-fitting			**PINN**		
		mean	**min**		**mean**	**min**	
d	**width**	**error**	**error**	$\mathcal{J}_{\text{data}}$	**error**	**error**	$\mathcal{J}$
1	16	$8.1 \cdot 10^{-5}$	$3.9 \cdot 10^{-5}$	$7.7 \cdot 10^{-10}$	$1.3 \cdot 10^{-5}$	$1.3 \cdot 10^{-7}$	$8.3 \cdot 10^{-9}$
	24	$9.3 \cdot 10^{-5}$	$3.7 \cdot 10^{-5}$	$6.9 \cdot 10^{-10}$	$1.2 \cdot 10^{-5}$	$1.9 \cdot 10^{-7}$	$8.8 \cdot 10^{-9}$
	32	$1.4 \cdot 10^{-4}$	$8.4 \cdot 10^{-5}$	$3.5 \cdot 10^{-9}$	$8.9 \cdot 10^{-6}$	$4.1 \cdot 10^{-7}$	$2 \cdot 10^{-9}$
2	16	$7 \cdot 10^{-5}$	$4.2 \cdot 10^{-5}$	$6.7 \cdot 10^{-10}$	$2.8 \cdot 10^{-5}$	$1.5 \cdot 10^{-5}$	$1.5 \cdot 10^{-7}$
	24	$9.7 \cdot 10^{-5}$	$4.7 \cdot 10^{-5}$	$1 \cdot 10^{-9}$	$2.7 \cdot 10^{-5}$	$1.4 \cdot 10^{-5}$	$1.7 \cdot 10^{-7}$
	32	$1.2 \cdot 10^{-4}$	$3.9 \cdot 10^{-5}$	$5.5 \cdot 10^{-10}$	$2.6 \cdot 10^{-5}$	$9.7 \cdot 10^{-6}$	$7.5 \cdot 10^{-8}$
3	16	$9.7 \cdot 10^{-5}$	$4.9 \cdot 10^{-5}$	$1 \cdot 10^{-9}$	$8.5 \cdot 10^{-5}$	$1.3 \cdot 10^{-5}$	$1.4 \cdot 10^{-7}$
	24	$1.6 \cdot 10^{-4}$	$4.5 \cdot 10^{-5}$	$6.6 \cdot 10^{-10}$	$3.1 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$	$1.6 \cdot 10^{-7}$
	32	$1.7 \cdot 10^{-4}$	$4.7 \cdot 10^{-5}$	$7.9 \cdot 10^{-10}$	$4 \cdot 10^{-5}$	$2.7 \cdot 10^{-5}$	$2.2 \cdot 10^{-7}$
4	16	$1 \cdot 10^{-4}$	$4.2 \cdot 10^{-5}$	$3.5 \cdot 10^{-10}$	$2.2 \cdot 10^{-5}$	$5.2 \cdot 10^{-6}$	$8.4 \cdot 10^{-8}$
	24	$1.1 \cdot 10^{-4}$	$3.3 \cdot 10^{-5}$	$3.5 \cdot 10^{-10}$	$2.4 \cdot 10^{-5}$	$1.4 \cdot 10^{-5}$	$9.4 \cdot 10^{-8}$
	32	$9.3 \cdot 10^{-5}$	$4.6 \cdot 10^{-5}$	$5.8 \cdot 10^{-10}$	$2.7 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	$1 \cdot 10^{-7}$

Table 5.2.: Considering the *d-dimensional Poisson problem*, for MLPs of depth 3, different widths and tanh activation, trained with the ENG optimizer, by data-fitting or with the PINN loss. L^2 relative error over 10 seeds – mean, and minimum together with the final loss of the corresponding seed.

The results of Table 5.2 show that, for the data-fitting, the mean L^2 relative error remains between $7 \cdot 10^{-5}$ and $2 \cdot 10^{-4}$ for all pairs of dimension and width considered, and the minimum between $3 \cdot 10^{-5}$ and $8 \cdot 10^{-5}$, without visible degradation as d goes from 1 to 4. At $d = 4$, a network of width 16 (that is, $N_\theta = 641$ weights) thus reaches on average 10^{-4}, whereas the FEM and the DG require 38 416 and 65 536 dofs for only $6 \cdot 10^{-2}$ (Table 5.1). The network can therefore represent this solution in high dimension with a few hundred weights, which is an advantage in terms of memory. As for the accuracy reached, increasing the width of the network improves nothing, the accuracy even degrading slightly for $d \leq 3$. The final loss $\mathcal{J}_{\text{data}}$ of the best network does not degrade either with d, remaining between $4 \cdot 10^{-10}$ and $4 \cdot 10^{-9}$ for all pairs of dimension and width. This stagnation of the error and of the loss, independent of d and of the width, suggests that the accuracy is not limited by the capacity of the network but

more probably by the optimization itself.

The PINN, which does not know the exact solution, reaches L^2 errors of the same order of magnitude, or even smaller. This does not mean that the PINN is intrinsically better, since the two trainings are limited by the optimization and do not use equal hyperparameters. A possible explanation is that the residual loss involves the second derivatives of u_θ, and thus constrains the network more than the values at the sampling points. The mean lies between $9 \cdot 10^{-6}$ and $9 \cdot 10^{-5}$, i.e. a factor 6 to 15 lower than with the data-fitting at $d = 1$ and a factor 1 to 5 lower for $d \geq 2$. Here too, the width has no systematic effect on the accuracy. The gap between the mean and the minimum over the 10 initializations is most marked at $d = 1$, where the minimum goes down to 10^{-7} for a mean of the order of 10^{-5}. This gap then narrows with d, and at $d = 4$ the mean and the minimum are of the same order. The final loss $\mathcal{J}$ goes, for its part, from an order of 10^{-9} to 10^{-8} at $d = 1$ to an order of 10^{-7} for $d \geq 2$, without further degradation between $d = 2$ and $d = 4$. As for the data-fitting, the PINN thus does not degrade notably as d increases beyond $d = 2$, with only a few hundred weights.

5.1.3. Limits of classical bases and motivation for neural enrichment

The previous results confirm, for the problem considered, the interest of neural networks. In particular, they show that networks are able to represent the solution with a number of weights N_θ that grows linearly with d, whereas the number of dofs N_{dofs} of classical bases grows exponentially. Nevertheless, it is important to emphasize that the values of N_{dofs} and N_θ are not comparable. Indeed, for an equivalent number of dofs and weights, the computational costs are not equivalent, the former being that of a linear solve while the latter is that of a non-convex optimization. Moreover, for networks, the quality of this optimization depends on the initialization and on the hyperparameters (optimizer, number of epochs, Gram matrix regularization, number of collocation points, network width, etc.), over which we have no a priori control.

This study thus shows that networks present a significant advantage in terms of memory cost. On the NVIDIA RTX 2000 graphics card (8 GB VRAM) used, whose memory limits are discussed in Remark 5.2, we did not obtain better than relative errors of about $5 \cdot 10^{-2}$ and $6 \cdot 10^{-2}$ with the FEM and the DG at $d = 4$, with 38 416 and 65 536 dofs (Figures 5.1 and 5.2), whereas a network with 641 weights could reach an error below 10^{-4} (Table 5.2). However, this memory advantage of neural networks comes with a drawback. Galerkin methods such as the FEM or the DG have theoretical guarantees that make it possible to control the error, which is found in the results with the orders observed in Figures 5.1 and 5.2. For neural networks, no such control exists. Since the optimization is non-convex, nothing guarantees that the minimum reached is close to the global minimum, nor that a small value of the loss systematically translates into a small error on the solution.

The cost of Galerkin methods in high dimension is explained by the exponential growth with d of the number of dofs of local bases on a Cartesian mesh, which makes the systems to solve larger and larger and more costly. This is why existing methods already seek to limit this cost by adapting the bases (see Chapter 1). The most natural one consists in using adaptive meshes, whose cells, with local support, concentrate where the solution needs them [3]. Another one consists in using bases that take into

account the physics of the problem, such as Trefftz methods, whose basis functions are local solutions of a homogeneous problem and which are widely used for waves [97]. In both cases, the adaptation relies on information provided to the method, namely an error estimator for the mesh or the physics of the problem for the basis, and not on a learned representation of the structure of the solution.

Classical Galerkin methods, even adapted, thus provide guarantees on the error, but their basis remains built from the mesh or from the physics of the PDE and their cost grows strongly with the dimension. Neural networks, conversely, can learn the structure of the solution with few weights, here its dependence on the diagonal direction alone, without it being provided to them, but without offering these guarantees. These two types of approaches are thus complementary, which motivates the development of hybrid methods, which combine the error control of classical bases and the ability of networks to capture the structure of the solution in order to limit costs in high dimension.

In this chapter, this hybridization takes the form of an enrichment, which consists in combining a classical Galerkin basis, which retains the variational framework (the solution being the Galerkin solution in the enriched space), with a neural basis in charge of capturing the structure of the solution. This dimensional study motivates the enrichment through the cost of classical bases in high dimension. The test cases of the following sections, in dimensions 1 and 3, are preliminary studies of this principle. They address another limit of classical bases, their lack of expressivity, since the solution presents a local structure (a boundary layer) that these bases capture poorly.

Having shown, on a manufactured solution with a one-dimensional structure, that neural networks can be advantageous to capture the solution as the dimension d increases, with a number of weights growing only linearly with d where classical bases suffer from the curse of dimensionality, we now turn to the enrichment of a classical Galerkin basis by a network. It aims at combining this ability with the structure of Galerkin methods, namely the variational formulation and the resolution of the scheme in the space generated by the basis. As a first step, we test this principle in low dimension, starting with a 1D advection-diffusion problem for which the network is trained for a fixed value of the parameter $\boldsymbol{\mu}$.

5.2. Non-parametric enrichment (PINN loss) – 1D test case

The objective of this section is to see whether the classical approximation bases (FEM, DG) can be improved by enriching them with a neural network (Section 4.3). In other words, we hope that, for a fixed number of degrees of freedom N_{dofs}, the solution $u_{h,\theta}$ computed in the enriched finite element space, reconstructed from its degrees of freedom by (4.6) (with no post-processing), is more accurate than the classical solution u_h. To this end, we consider problem (2.1) in its advection-diffusion form, in dimension $d = 1$:

$$\begin{cases} -\dfrac{1}{\mathrm{Pe}}\,\partial_{xx}u + (\mu_1 + \mu_2 x)\,\partial_x u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases} \tag{5.3}$$

with $\Omega = (0,1)$, Γ its boundary, a constant source term $f = 1$ and $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathbb{R}^2$ a fixed parameter. This is the operator (2.2) with $R = 0$, $D = 1$, $\mathrm{Pe} = 20$ and an affine convection $C(x, \boldsymbol{\mu}) = \mu_1 + \mu_2 x$. In this section, we consider the problem for three fixed parameter values:

$$\boldsymbol{\mu}^{(1)} = (0.8, 0.8), \qquad \boldsymbol{\mu}^{(2)} = (1.0, 1.0), \qquad \boldsymbol{\mu}^{(3)} = (1.2, 1.2). \tag{5.4}$$

Remark 5.4. *For this test case, no analytical solution is available. To estimate the errors, we therefore consider for each $\boldsymbol{\mu}^{(j)}$ a reference solution $u(\cdot, \boldsymbol{\mu}^{(j)})$, obtained by classical finite elements applied to (5.3) on a very fine uniform mesh of $N_{cells}^{ref} := 128$ cells (i.e. $h_{ref} := 1/128$) with Lagrange polynomials $\mathbb{Q}_{k_{ref}}$ of degree $k_{ref} := 4$. It is computed once for each $\boldsymbol{\mu}^{(j)}$ and used to measure the error of all the methods (FEM and DG, classical and learned bases). Its relative L^2 error, estimated by comparison with finer finite element solutions, is of the order of 10^{-8} or below depending on $\boldsymbol{\mu}$. The errors of the learned bases that come close to this level (for the finest configurations) are thus limited by the reference, and the corresponding gains are lower bounds.*

This study is called non-parametric because, in this section, a network is trained for a fixed value of the parameter $\boldsymbol{\mu}$, and the learned basis is only valid for this value. This approach, studied here and in Section 5.3, contrasts with the following Section 5.4, where a single network is trained for a whole family of parameters. The optimization is carried out through the solver, as presented in Section 4.9, that is, at each iteration the scheme is assembled and solved to obtain the degrees of freedom $\boldsymbol{\beta}_h(\theta)$ associated with the current network. Throughout this section, we only consider PINNs, that is, the loss without data of Example 4.23, for a fixed $\boldsymbol{\mu}$:

$$\mathcal{J}(\theta) = \omega_r \mathcal{J}_r(\theta) + \omega_b \mathcal{J}_b(\theta), \tag{5.5}$$

with $\mathcal{J}_r$ the residual loss:

$$\mathcal{J}_r(\theta) = \int_\Omega \left| -\tfrac{1}{\mathrm{Pe}}\, \partial_{xx} u_{h,\theta}(x) + (\mu_1 + \mu_2 x)\, \partial_x u_{h,\theta}(x) - f \right|^2 \mathrm{d}x,$$

and $\mathcal{J}_b$ the boundary loss:

$$\mathcal{J}_b(\theta) = \int_\Gamma \left| u_{h,\theta}(x) \right|^2 \mathrm{d}x,$$

which, in dimension 1, reduces to the two boundary points, whereas $\mathcal{J}_r$ is estimated by Monte Carlo on N_{col} collocation points drawn randomly in Ω and redrawn at each epoch, with $\omega_r = 1$ and $\omega_b = 10$. For the finite element method, the Dirichlet condition is imposed by the scheme, so that $\mathcal{J}_b$ vanishes up to numerical precision, whereas for DG it is only imposed weakly, and $\mathcal{J}_b$ is not zero a priori. The reference solution is therefore never used in the training, only in the evaluation of the errors.

Recall that $u_{h,\theta}$ is the finite element (or DG) solution reconstructed on each cell K_c from its degrees of freedom $\boldsymbol{\beta}_h(\theta)$ and the basis functions $\phi_{c,i}^{\theta}$ ((4.6)), that is, the solution in the space generated by the basis. Minimizing the PINN loss $\mathcal{J}(\theta)$ therefore amounts to looking for the best basis, in the sense of this loss, under the constraint that $u_{h,\theta}$ is the Galerkin solution in the space it generates, which reads:

$$\min_\theta \mathcal{J}(\boldsymbol{\beta}_h(\theta); \theta) \quad \text{s.t.} \quad \boldsymbol{R}_h(\boldsymbol{\beta}_h(\theta); \theta) = \mathbf{0},$$

that is the bi-level problem (4.19). It is this constraint that `ScimBa` handles. It solves the scheme at each iteration, and differentiates through this solve by implicit differentiation and the adjoint method (Section 4.9). The resulting training is summarized in Algorithm 5.1. A minimal implementation of this training, together with the classical solver for comparison, is given in Appendix B.1.

Algorithm 5.1 Training of an enriched Galerkin (FEM or DG) space with the PINN loss, for a gradient-based optimizer such as Adam. For the ENG optimizer, see Remark 5.5.

Input: mesh, degree k, parameter $\boldsymbol{\mu}$, initial weights θ, ω_r, ω_b, N_{col}, N_{epochs}

1: **for** $n = 1, \dots, N_{\text{epochs}}$ **do**
2: Build the basis $\phi^{\theta}_{c,i} = m_\theta \, \phi_{c,i}$ ▷ solver
3: Assemble $A_h(\theta)$ and $\boldsymbol{b}_h(\theta)$ ▷ solver
4: Solve $A_h(\theta)\, \boldsymbol{\beta}_h(\theta) = \boldsymbol{b}_h(\theta)$ for $\boldsymbol{\beta}_h(\theta)$ ▷ solver
5: Evaluate $u_{h,\theta}$ and the loss $\mathcal{J}(\theta) = \omega_r \mathcal{J}_r(\theta) + \omega_b \mathcal{J}_b(\theta)$
6: Solve the adjoint system $J^T_{\boldsymbol{R}_h} \boldsymbol{\lambda}_h = (\partial \mathcal{J}/\partial \boldsymbol{\beta}_h)^T$ ▷ backward
7: Compute $d\mathcal{J}/d\theta = \partial \mathcal{J}/\partial \theta - \boldsymbol{\lambda}_h^T \, \partial \boldsymbol{R}_h/\partial \theta$ by VJP ▷ backward
8: Update θ with the optimizer (e.g. Adam)
9: **end for**

Output: weights θ

Remark 5.5. *The backward steps of Algorithm 5.1 correspond to an optimizer based on the gradient of the loss, such as Adam. The ENG optimizer used for the experiments of this section works differently. It forms the Jacobian of the degrees of freedom with respect to θ by implicit differentiation, with N_θ direct solves of the linearized system (one per weight). The optimizer then builds from it the Gram matrix of the residuals to compute the search direction, and chooses the step by a line search that solves the scheme for several trial steps.*

We also note that, since the enriched basis functions are built from the local basis of the mesh, the learned space depends on the mesh and on the polynomial degree, so that a training is necessary for each triplet $(\boldsymbol{\mu}, k, N_{\text{cells}})$, which we will keep in mind when interpreting the results. This is a feasibility study on a simple 1D test case, which should make it possible to identify the parameters with the most influence (polynomial degree, number of cells, network capacity, type of enrichment), and not to propose at this stage an efficient solver.

We study four variants of this approach, presented in the following subsections, one for each pair (scheme, type of basis). We begin in Section 5.2.1 with the patchwise basis with continuous finite elements, where a single network is used for the whole domain, so as to automatically preserve the continuity of the basis. Then, in Section 5.2.2, we move to the cellwise basis, where one network per cell is used, constructed so that the basis remains continuous at the cell boundaries. Finally, we consider the DG method in Sections 5.2.3 and 5.2.4, first with a single network (patchwise basis) and then with one network per cell (cellwise basis). Since the interfaces are treated weakly by the SIPG flux (Section 2.2.2), no continuity constraint is necessary. The four variants are then compared in Section 5.2.5, which also gathers the conclusions and the limitations of this study. In all four cases, the learned basis is the product of the local Lagrange

basis $\phi_{c,i}$ by a multiplier built from MLPs with scalar output (a single one for the whole domain in the patchwise case, one per cell in the cellwise case), shared by all the basis functions of the cell (Section 4.3). For a patchwise basis, we have:

$$\phi^{\theta}_{c,i}(x) = m_\theta(x)\,\phi_{c,i}(x), \tag{5.6}$$

where m_θ is built from the single network v_θ, and for a cellwise basis:

$$\phi^{\theta}_{c,i}(x) = m_{c,\theta}(x)\,\phi_{c,i}(x), \tag{5.7}$$

where $m_{c,\theta}$ is built from the network $v_{c,\theta}$ of the physical cell K_c. Only the scheme (continuous finite elements or DG) and the number of networks therefore change.

Remark 5.6. *The product $m_\theta\,\phi_{c,i}$ is reminiscent of the multiplicative space of Section 3.2, where a prior u_θ, that does not vanish, multiplies the finite element space. The difference is that m_θ is not pretrained and then frozen, but learned through the solver, and that its positivity, which will be enforced by construction following the study of the choice of the multiplier (Section 5.2.1), plays a role analogous to the non-vanishing condition on the prior, enforced in Section 3.2.1 by a lifting of the problem. The additive counterpart, studied in Section 3.1 and compared with the multiplicative one in Section 3.3, has not been tested in the present setting.*

5.2.1. Finite element method – Patchwise basis

In this section, we consider the continuous finite element method (Section 2.2.1) whose local Lagrange basis $\phi_{c,i}$ is enriched by a single neural network v_θ for the whole domain (patchwise basis), using a multiplier m_θ, as defined in (5.6). The goal is to show that the error can be reduced by several orders of magnitude by enriching the classical basis with a learned multiplier, and to identify the parameters that most influence the final accuracy. To this end, the analysis is carried out in four steps. In Section 5.2.1.1, we start with a first study considering a simple affine multiplier. In Section 5.2.1.2, we then study the influence, one at a time, of the polynomial degree, the number of cells and the network width on the quality of the learned basis, still considering this same affine multiplier. Next, in Section 5.2.1.3, a more in-depth study is carried out on the choice of the multiplier, where three types of multipliers (affine, positive, positive with bounded weights) are compared at a fixed budget, in order to identify an efficient and robust choice of multiplier, whose conclusion and limitations are given in Section 5.2.1.4. Finally, in Section 5.2.1.5, we revisit the first study with the retained multiplier.

Remark 5.7. *Since the results depend on the initialization of the network, each configuration is trained with 20 different seeds. In the first study and in the single-axis studies, we report the best seed, selected as follows. Among the seeds whose best loss over the epochs (which may differ from the loss at the final epoch) is below a threshold arbitrarily set to 10^{-3}, we retain the one with the lowest L^2 error with respect to the reference solution (see the limitations in Section 5.2.5). The number of seeds below the threshold is also reported, and a dash means that none reached it. In the study on the choice of the multiplier, we instead consider all the seeds, without restricting ourselves to the best one or to those below the threshold.*

5.2.1.1. A first study on the enrichment of bases

In this first study, we choose for the multiplier m_θ of (5.6) the simplest form, which is affine in the output of the network v_θ:

$$m_\theta(x) = 2 + 0.1\, v_\theta(x), \tag{5.8}$$

with v_θ an MLP shared by all the cells, with tanh activation and 2 hidden layers of width 16 (i.e. $N_\theta = 321$ weights). This choice of multiplier guarantees that m_θ is almost constant at initialization. Indeed, since the network weights are initialized to small values, $0.1v_\theta$ is close to 0 and m_θ is close to 2. The constant 2 does not change the space generated by the basis, since multiplying all the basis functions by a constant leaves it unchanged, so that only the relative amplitude $0.1/2 = 5\%$ of the perturbation matters, which is chosen small. The enriched space is then almost the classical FE space, and the training therefore starts from the classical FE solution.

The network v_θ is trained through the solver (Section 4.9) with the PINN loss introduced in (5.5), for each value of $\boldsymbol{\mu}$ considered, using the ENG optimizer [98] of `ScimBa`. The training lasts $N_{\text{epochs}} = 300$ epochs, with $N_{\text{col}} = 6\,000$ collocation points drawn randomly in Ω and redrawn at each epoch, the boundary term being evaluated at the two endpoints of Ω, and the retained network is the one with the lowest loss (best loss) over the epochs. Once v_θ is trained, we solve the FE scheme in the enriched space to obtain $u_{h,\theta}$, whose relative L^2 error is evaluated with respect to the reference solution (Remark 5.4) and compared with that of the classical FE solution u_h, at the same number of degrees of freedom N_{dofs} (Remark 5.1). As specified in Remark 5.7, each configuration is trained with 20 seeds. The 1D computations are run on the NVIDIA RTX 2000 graphics card (8 GB VRAM) already used in Section 5.1.

This first study has two parts. First, we quantify the gain provided by the learned basis for the three values of $\boldsymbol{\mu}$, by varying the polynomial degree k and the number of cells N_{cells}. Second, we focus on $\boldsymbol{\mu}^{(1)}$ and $N_{\text{cells}} = 4$ to compare more closely the solutions and the pointwise errors of the classical and learned bases, and then look at the learned multiplier m_θ and at the resulting enriched basis functions.

Gains achieved with the learned basis. For the three values of $\boldsymbol{\mu}$ defined in (5.4), we compare the L^2 errors between the reference solution u and the standard FE solution u_h, as well as the enriched solution $u_{h,\theta}$, for two polynomial degrees $k = 3$ and $k = 4$ and $N_{\text{cells}} \in \{2, 4, 8\}$ (Tables 5.3 and 5.4).

Tables 5.3 and 5.4 show that the learned basis, associated with the best seed, reduces the error by several orders of magnitude in the cases where at least one seed is below the threshold. Over all the configurations for which a result is reported, the gain of the best seed ranges from 159 to more than 200 000. Regarding the influence of $\boldsymbol{\mu}$, we note that the gain tends to decrease as $\boldsymbol{\mu}$ increases. The larger the parameter is, the stronger the convection and the steeper the boundary layer, which can make the multiplier m_θ harder to represent and to optimize, the network and the training budget being identical for all the configurations.

Regarding the number of cells, for the smallest $N_{\text{cells}} = 2$, the number of seeds below the threshold is low, even zero for $k = 3$ with $\boldsymbol{\mu}^{(2)}$ and $\boldsymbol{\mu}^{(3)}$, which means that the training is harder for these configurations. Conversely, the more the number of cells

			classical	learned basis (best seed)			
$\boldsymbol{\mu}$	N_{cells}	N_{dofs}	error	error	gain	loss	/ 20
$\boldsymbol{\mu}^{(1)}$	2	7	$1.95 \cdot 10^{-1}$	$1.22 \cdot 10^{-3}$	159	$1.24 \cdot 10^{-4}$	2
	4	13	$5.11 \cdot 10^{-2}$	$6.48 \cdot 10^{-6}$	7 886	$6.61 \cdot 10^{-7}$	5
	8	25	$8.64 \cdot 10^{-3}$	$3.32 \cdot 10^{-8}$	260 552	$1.1 \cdot 10^{-9}$	15
$\boldsymbol{\mu}^{(2)}$	2	7	$2.61 \cdot 10^{-1}$	–	–	–	0
	4	13	$7.15 \cdot 10^{-2}$	$4.42 \cdot 10^{-5}$	1 619	$1.35 \cdot 10^{-6}$	5
	8	25	$1.44 \cdot 10^{-2}$	$5.44 \cdot 10^{-8}$	263 912	$1.11 \cdot 10^{-9}$	11
$\boldsymbol{\mu}^{(3)}$	2	7	$3.42 \cdot 10^{-1}$	–	–	–	0
	4	13	$9.08 \cdot 10^{-2}$	$1.51 \cdot 10^{-4}$	600	$3.53 \cdot 10^{-5}$	6
	8	25	$2.09 \cdot 10^{-2}$	$3.5 \cdot 10^{-7}$	59 803	$1.56 \cdot 10^{-9}$	8

Table 5.3.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 3$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the affine multiplier. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

			classical	learned basis (best seed)			
$\boldsymbol{\mu}$	N_{cells}	N_{dofs}	error	error	gain	loss	/ 20
$\boldsymbol{\mu}^{(1)}$	2	9	$9.16 \cdot 10^{-2}$	$4.1 \cdot 10^{-5}$	2 236	$4.02 \cdot 10^{-7}$	8
	4	17	$1.7 \cdot 10^{-2}$	$7.14 \cdot 10^{-8}$	237 925	$1.12 \cdot 10^{-9}$	20
	8	33	$1.63 \cdot 10^{-3}$	$7.8 \cdot 10^{-9}$	208 913	$1 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(2)}$	2	9	$1.24 \cdot 10^{-1}$	$1.83 \cdot 10^{-4}$	677	$8.85 \cdot 10^{-6}$	3
	4	17	$2.76 \cdot 10^{-2}$	$5.55 \cdot 10^{-7}$	49 802	$1.67 \cdot 10^{-9}$	11
	8	33	$3.29 \cdot 10^{-3}$	$1.57 \cdot 10^{-8}$	209 922	$1.11 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(3)}$	2	9	$1.54 \cdot 10^{-1}$	$8.31 \cdot 10^{-4}$	186	$2.65 \cdot 10^{-4}$	2
	4	17	$3.92 \cdot 10^{-2}$	$2.19 \cdot 10^{-6}$	17 892	$1.59 \cdot 10^{-8}$	8
	8	33	$5.57 \cdot 10^{-3}$	$2.96 \cdot 10^{-8}$	187 761	$1.13 \cdot 10^{-9}$	18

Table 5.4.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 4$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the affine multiplier. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

increases, the larger the number of seeds below the threshold, in particular for $k = 4$ and $N_{\text{cells}} = 8$ where almost all the seeds produce a loss below the threshold set to 10^{-3}. Moreover, we note that the gain seems to increase with the number of cells but may plateau for some configurations. For example, for $\boldsymbol{\mu}^{(1)}$ and $k = 4$, the gain decreases slightly when going from $N_{\text{cells}} = 4$ to $N_{\text{cells}} = 8$, which can be explained by three factors. First, the reported gain is that of the best seed among the 20 tested, which implies that

the results are sensitive to the variability of the training, and may fluctuate from one configuration to another. Second, the classical error decreases strongly with N_{cells}, by about a factor 10 for this same case. The classical FE solution therefore represents the boundary layer better, while the error of the learned basis is already of the order of 10^{-8}. The enrichment thus remains worthwhile, but the gain, which is a ratio of errors, increases less. Finally, we recall that the errors are computed from a fixed reference solution (see Remark 5.4), which may itself create a floor for the error of the learned basis, and thus limit the gain.

Finally, we note that the best losses reached (of the order of 10^{-9}) seem to concentrate on the finest meshes, which can be explained by the fact that the classical basis is richer and the enriched space is therefore richer, which makes the training of the multiplier easier.

Comparison of the classical and learned bases. We now focus only on the parameter $\boldsymbol{\mu}^{(1)}$ and compare the standard FE solution u_h and the enriched solution $u_{h,\theta}$ with the reference solution u, for the polynomial degrees $k \in \{3, 4\}$ and $N_{\text{cells}} = 4$. For each configuration, Figure 5.3 shows the solutions and the absolute errors obtained with the best seed. Figure 5.4 shows the learned multiplier m_θ and the resulting enriched basis functions.

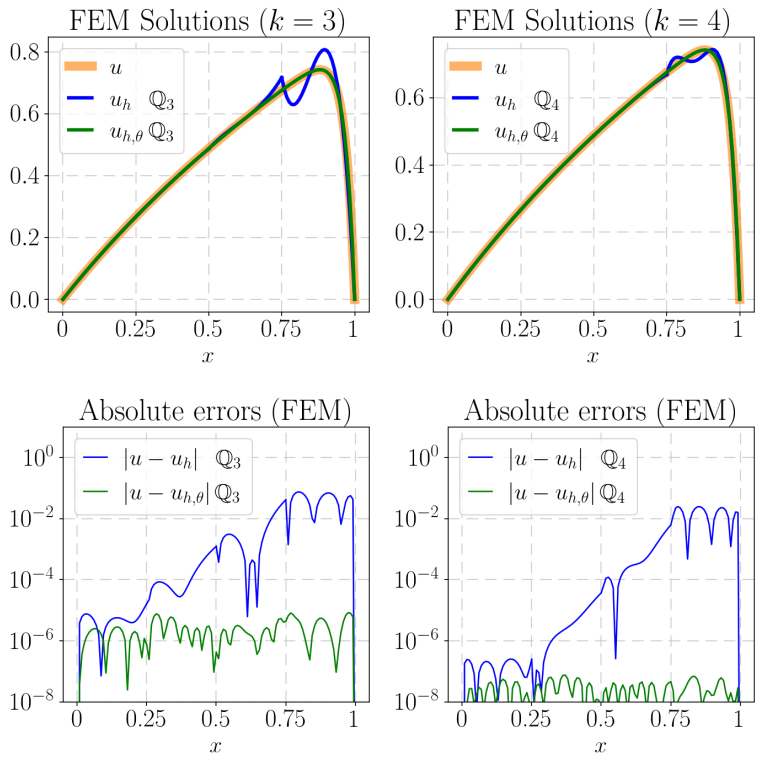


Figure 5.3.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k \in \{3, 4\}$, $N_{\text{cells}} = 4$ and the affine multiplier. Top – Classical FEM solution u_h and learned-basis solution $u_{h,\theta}$ (best seed, see Remark 5.7) compared with the reference solution u. Bottom – Corresponding absolute errors. In each row, from left to right – $k = 3$ and $k = 4$.

Figure 5.3 shows, as expected in the strong convection regime, that the classical FE solution u_h exhibits oscillations in the boundary layer region for $k \in \{3, 4\}$, whereas the enriched solution $u_{h,\theta}$ is indistinguishable from the reference solution u at the scale of the figure, which is consistent with the results of Tables 5.3 and 5.4. We also note a clear improvement of the pointwise absolute errors, which are at most about 10^{-5} ($k = 3$) and 10^{-7} ($k = 4$) for the enriched solution, against about $7 \cdot 10^{-2}$ and $2 \cdot 10^{-2}$ for the classical solution.

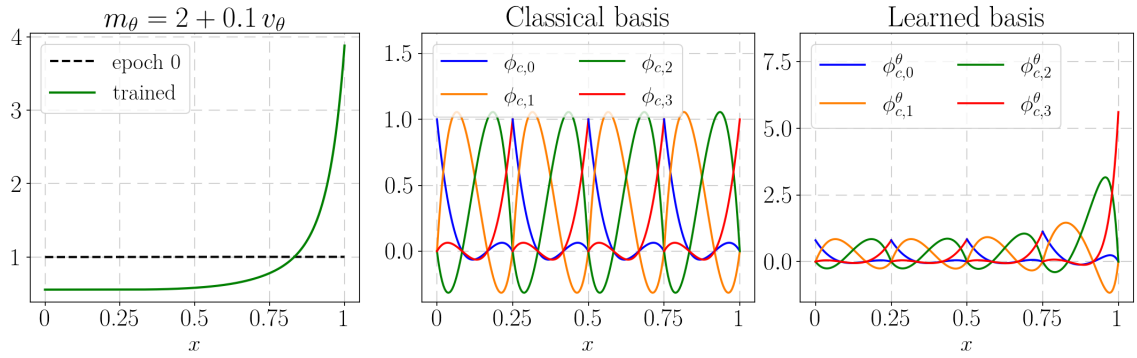


Figure 5.4.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N_{\text{cells}} = 4$ and the affine multiplier. Left – Learned multiplier (L^2 normalized), at initialization and after training (best seed, see Remark 5.7). Middle – Classical Lagrange basis functions. Right – Resulting enriched basis functions.

Figure 5.4 is plotted for $\boldsymbol{\mu}^{(1)}$, $k = 3$ and $N_{\text{cells}} = 4$. It shows the learned affine multiplier (normalized by its L^2 norm, at initialization and after training) and the enriched basis functions, compared with the classical Lagrange functions. As expected, the multiplier is constant at initialization. After training, it is almost constant on the first half of the domain, but grows strongly near $x = 1$, that is, where the boundary layer is located.

Since the multiplier is a single function defined on the whole of Ω, the local functions associated with the same node in two neighboring cells take the same value at this node, which preserves the continuity of the basis. The enriched basis functions roughly keep the shape of the Lagrange functions in the first three cells, but are strongly amplified in the last cell. In other words, the learned basis concentrates its approximation capacity in the boundary layer, whereas the position of the latter is only provided to the network by the PDE residual in the loss.

5.2.1.2. Single-axis capacity study

In the first study, we showed that the learned basis could provide gains of several orders of magnitude on the L^2 error with respect to the classical FE basis. Here we take up the affine multiplier (5.8) and study more closely the approximation capacity of the learned basis by varying three independent levers, namely the polynomial degree k, the number of cells N_{cells} and the width of the network v_θ, its depth remaining fixed at 2 hidden layers in all the sweeps. Unlike the sweep over $\boldsymbol{\mu}$ of the first study, $\boldsymbol{\mu}$ is fixed here at $\boldsymbol{\mu}^{(1)} = (0.8, 0.8)$, the most favorable of the three cases tested above, so that the

trend observed on the studied lever is not confounded with optimization failures specific to a harder $\boldsymbol{\mu}$. Each sweep below fixes the two other levers at a common reference configuration ($k = 3$, $N_{\text{cells}} = 10$, width 16, the depth always being 2) and varies the third one, so the reference point, trained only once, is the same in the three sweeps. As before, each configuration is trained with 20 random seeds as defined in Remark 5.7. The training is identical to that of the previous section and lasts $N_{\text{epochs}} = 300$ epochs, with $N_{\text{col}} = 6\,000$ collocation points drawn randomly in Ω, the boundary term being evaluated at the two endpoints.

Polynomial degree. In Table 5.5, we fix $N_{\text{cells}} = 10$ and the network width at 16 (depth 2), and we vary the polynomial degree $k \in \{1, \dots, 5\}$ (i.e. N_{dofs} from 11 to 51).

		classical	**learned basis (best seed)**			
k	N_{dofs}	**error**	**error**	**gain**	**loss**	**/ 20**
1	11	$1.18 \cdot 10^{-1}$	–	–	–	0
2	21	$2.41 \cdot 10^{-2}$	$3.32 \cdot 10^{-7}$	72 697	$1.97 \cdot 10^{-9}$	6
3	31	$4.36 \cdot 10^{-3}$	$1.22 \cdot 10^{-8}$	356 610	$1.03 \cdot 10^{-9}$	18
4	41	$6.71 \cdot 10^{-4}$	$5.22 \cdot 10^{-9}$	128 394	$9.98 \cdot 10^{-10}$	19
5	51	$8.9 \cdot 10^{-5}$	$6.69 \cdot 10^{-9}$	13 316	$1.02 \cdot 10^{-9}$	20

Table 5.5.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $N_{\text{cells}} = 10$, the affine multiplier with v_θ of width 16 and depth 2, and k varying. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

Table 5.5 shows that the classical error is divided by about 1 300 between $k = 1$ and $k = 5$. For $k = 1$, no seed is below the threshold and no result is reported. From $k = 2$, the learned basis gives gains between 13 000 and 360 000, and the maximal gain is reached at $k = 3$ and then decreases. This decrease comes from the fact that the error of the learned basis stops decreasing once it reaches the accuracy of the reference solution (Remark 5.4), while the classical error keeps decreasing with k, so that the gain stops growing. Moreover, the number of seeds below the threshold increases with k, which shows that the richer the bases, the easier the training.

Number of cells. In Table 5.6, we fix $k = 3$ and the network width at 16 (depth 2), and we vary the number of cells N_{cells} between 2 and 15 (i.e. N_{dofs} from 7 to 46). Table 5.6 shows that the classical error is divided by about 170 between $N_{\text{cells}} = 2$ and $N_{\text{cells}} = 15$. No seed is below the threshold for $N_{\text{cells}} = 2$. From $N_{\text{cells}} = 3$ to $N_{\text{cells}} = 10$, the gain increases with the number of cells, from 1 300 to 360 000, and then fluctuates for $N_{\text{cells}} \in \{12, 15\}$, the error of the learned basis being then of the order of the accuracy of the reference solution (Remark 5.4). The number of seeds below the threshold also increases with N_{cells}, from 7 out of 20 for $N_{\text{cells}} = 3$ to 18–20 out of 20 for $N_{\text{cells}} \geq 10$.

		classical	learned basis (best seed)			
N_{cells}	N_{dofs}	error	error	gain	loss	/ 20
2	7	$1.95 \cdot 10^{-1}$	–	–	–	0
3	10	$9.16 \cdot 10^{-2}$	$7.22 \cdot 10^{-5}$	1 270	$6.31 \cdot 10^{-6}$	7
4	13	$5.11 \cdot 10^{-2}$	$7.8 \cdot 10^{-6}$	6 556	$3.73 \cdot 10^{-8}$	7
6	19	$1.93 \cdot 10^{-2}$	$1.36 \cdot 10^{-7}$	141 542	$1.07 \cdot 10^{-9}$	11
8	25	$8.64 \cdot 10^{-3}$	$3.32 \cdot 10^{-8}$	260 552	$1.1 \cdot 10^{-9}$	15
10	31	$4.36 \cdot 10^{-3}$	$1.22 \cdot 10^{-8}$	356 610	$1.03 \cdot 10^{-9}$	18
12	37	$2.41 \cdot 10^{-3}$	$1.27 \cdot 10^{-8}$	189 425	$1.06 \cdot 10^{-9}$	20
15	46	$1.12 \cdot 10^{-3}$	$3.2 \cdot 10^{-9}$	349 400	$9.75 \cdot 10^{-10}$	18

Table 5.6.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, the affine multiplier with v_θ of width 16 and depth 2, and N_{cells} varying. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

Remark 5.8. *For $N_{cells} \in \{2, 4, 8\}$, the configurations are identical to those of the first study, with the same seeds and the same collocation points, but the results are not exactly reproduced (for $N_{cells} = 4$, 7 seeds are below the threshold here against 5 in the first study). The non-deterministic nature of the training on GPU, where the order of the parallel floating-point additions varies from run to run, is amplified by the optimization when the trajectory is unstable. The numbers of seeds below the threshold and the gains should therefore be read up to this variability.*

Network width. In Table 5.7, we fix here $k = 3$ and $N_{\text{cells}} = 10$, and we vary the width of the network v_θ (of depth 2) between 4 and 28, (i.e. N_θ from 33 to 897).

classical	
N_{dofs}	error
31	$4.36 \cdot 10^{-3}$

		learned basis (best seed)			
width	N_θ	error	gain	loss	/ 20
4	33	$2.69 \cdot 10^{-8}$	162 294	$1.15 \cdot 10^{-9}$	17
6	61	$2.38 \cdot 10^{-8}$	183 199	$1.11 \cdot 10^{-9}$	20
8	97	$1.94 \cdot 10^{-8}$	224 592	$1.08 \cdot 10^{-9}$	19
10	141	$2.77 \cdot 10^{-8}$	157 198	$1.12 \cdot 10^{-9}$	19
16	321	$1.22 \cdot 10^{-8}$	356 610	$1.03 \cdot 10^{-9}$	18
20	481	$1.47 \cdot 10^{-8}$	297 045	$1.09 \cdot 10^{-9}$	19
28	897	$1.04 \cdot 10^{-8}$	418 434	$1.04 \cdot 10^{-9}$	18

Table 5.7.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N_{\text{cells}} = 10$ ($N_{\text{dofs}} = 31$) and the affine multiplier, with the width of v_θ varying (depth 2). Left – Classical FEM: number of degrees of freedom and L^2 relative error. Right – Learned basis (best seed): number of weights of v_θ, L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

For this lever, the classical error is constant since the polynomial degree and the number of cells are fixed. Table 5.7 shows that the network width has much less influence than the two previous levers. For widths from 4 to 28, the gain of the best seed remains between 160 000 and 420 000, and one can note a slight trend towards an increase of the gain with the width, which is however not monotonic. The number of seeds below the threshold is relatively stable, between 17 and 20 out of 20 for all the widths. The error also seems relatively stable, between 10^{-8} and $3 \cdot 10^{-8}$, with a very slight trend to decrease with the width. These errors are of the order of the accuracy of the reference solution (Remark 5.4), so that this study cannot show an influence of the width below this level.

5.2.1.3. Choice of the multiplier parametrization

In the previous study, we saw that the determining levers seem to be the polynomial degree k and the number of cells N_{cells}, whereas the network capacity does not seem to be really limiting for this test case. The goal is now to determine, for equal resources, which combination of these levers is the most efficient. We therefore place ourselves here at a fixed budget, by fixing the number of degrees of freedom at $N_{\text{dofs}} = 25$ and by constraining the number of weights N_θ to stay within a given range ($134 \leq N_\theta \leq 151$), and then we vary the split of this budget between the polynomial degree k and the number of cells N_{cells} on the one hand, and the width and the depth of the network on the other hand. Each configuration is trained on 20 seeds, as defined in Remark 5.7, with a common training of 100 epochs and $N_{\text{col}} = 5\,000$ collocation points, different from that of the previous studies. The gains obtained here are therefore not directly comparable with those of these studies.

We start by taking up the affine multiplier proposed in (5.8) and we begin by identifying the configurations that work best at a fixed budget. However, we observe that a non-negligible share of the trainings does not reach the threshold set on the loss, and in particular that some of them explode (see Remark 5.9). To understand these failures, we analyze the shape and the sign of the learned multipliers. This analysis leads us to propose two other parametrizations, a positive multiplier and then a positive multiplier whose network weights are bounded, studied at the same budget. A final analysis then compares the three multipliers, both on the gain obtained with respect to the classical finite elements and on the number of trainings that explode.

> **Remark 5.9.** *In what follows, we say that a training "explodes" when its loss history contains a* `NaN`*. Since the retained network is the one with the lowest loss obtained during training, a training that explodes still has a learned multiplier, which can be analyzed like the others. Finally, we say that a training "beats" the classical finite elements when its gain, as defined in Remark 5.1, is greater than* 1.05*. Unless stated otherwise, the trainings that explode are included in the counts of those that beat the classical finite elements, with their best checkpoint. Only Figure 5.20 and Table 5.8 count the non-exploded trainings.*

Affine multiplier. Figure 5.5 summarizes the results of the budget study for the affine multiplier, which amounts to 400 trainings in total. For each combination of the polynomial degree, the number of cells and the shape of the network, it gives the error

of the classical finite elements and the median error (and that of the best seed) among the seeds below the threshold. It also provides the number of seeds that go below the threshold and of those that beat the classical finite elements (Remark 5.9). Figure 5.6 then represents the best loss as a function of the gain of each training, distinguishing their polynomial degree k, to see to what extent the loss informs about the quality of the learned basis.

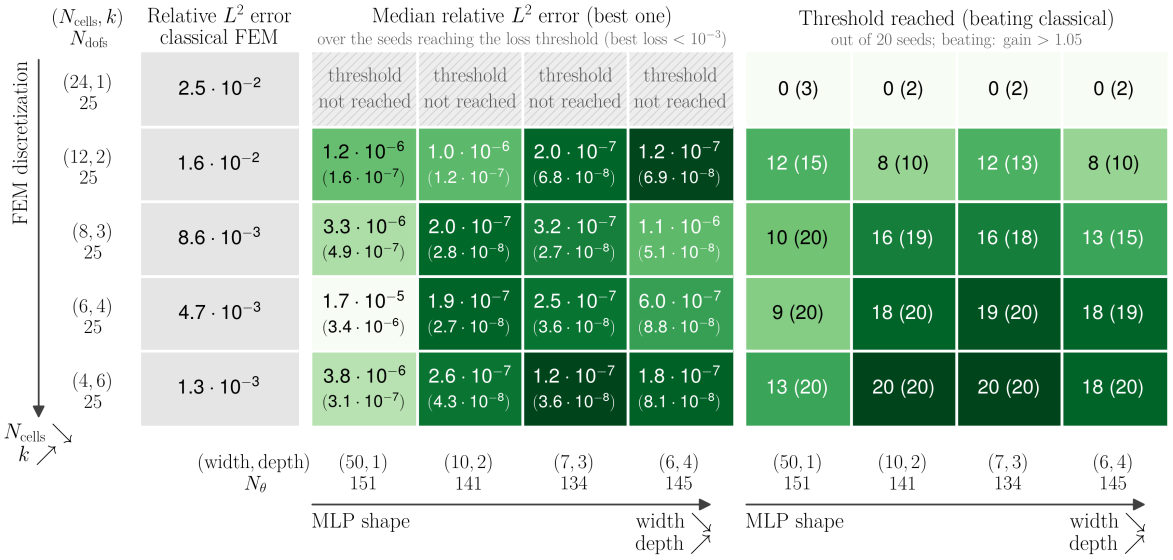


Figure 5.5.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $N_{\text{dofs}} = 25$, $134 \leq N_\theta \leq 151$ and the affine multiplier, with the FEM discretizations (N_{cells}, k) in rows and the MLP shapes $(\text{width}, \text{depth})$ in columns. Left – Classical FEM L^2 relative error. Middle – Median (and best seed) L^2 relative error of the learned basis over the seeds that reached the loss threshold. Right – Number of seeds that reached the loss threshold, out of 20, and, in parentheses, number of seeds that beat the classical FEM (see Remark 5.9).

In Figure 5.5, we first see that the number of seeds that beat the classical finite elements is always greater than or equal to the number of seeds that reach the threshold. Indeed, for this test case, reaching the threshold seems to imply beating the classical finite elements. On the other hand, the converse is false, a seed can beat the classical finite elements without reaching the arbitrarily fixed threshold.

Moreover, we note that the results depend strongly on the polynomial degree k. For $k = 1$, no seed reaches the threshold, whatever the shape of the network, and this despite a relatively high number of cells ($N_{\text{cells}} = 24$), and only 9 of the 80 seeds beat the classical finite elements. Subsequently, the number of seeds that reach the threshold, as well as that of the seeds that beat the classical finite elements, seems to increase with k, a high polynomial degree associated with a small number of cells seeming to favor reaching the threshold. Regarding the shape of the network, the $(50, 1)$ MLP configuration is the one for which the fewest seeds reach the threshold, the other configurations seeming roughly equivalent to each other from a depth of 2. As for the errors, the median error of the seeds that reach the threshold is lower than that of the classical finite elements by several orders of magnitude, whatever the configuration,

and the same holds for the error of the best seed, given in parentheses. The median error is moreover higher for the $(50, 1)$ network than for the other shapes, which give comparable results. The error of the best seed is for its part of the order of 10^{-8} to 10^{-7} from a depth of 2, with few differences between the degrees, which is the order of the accuracy of the reference solution (Remark 5.4). These observations globally go in the direction of the previous remarks, a polynomial degree $k \geq 3$ and a depth of at least 2 seeming to be good choices. Depth 4 however seems a bit harder to train, with slightly fewer seeds reaching the threshold and slightly higher errors than for depths 2 and 3, notably for $k = 3$ and $k = 4$.

Figure 5.6 specifies the link between the best loss of each configuration and the associated gain. Out of the 400 trainings, 288 (72 %) beat the classical finite elements, 11 have a constant gain and 101 do worse, while 230 reach the threshold. No training below the threshold does worse than the classical finite elements, whereas 58 trainings that do not reach the threshold beat them nonetheless, which confirms that a high loss does not necessarily mean a bad error. We also observe a correlation between the loss and the gain. The trainings that do not reach the threshold have a gain of the order of 10^2 at most, whereas those that reach it generally have much higher gains, which increase as the loss decreases, down to a loss of about 10^{-9} for the highest gains.

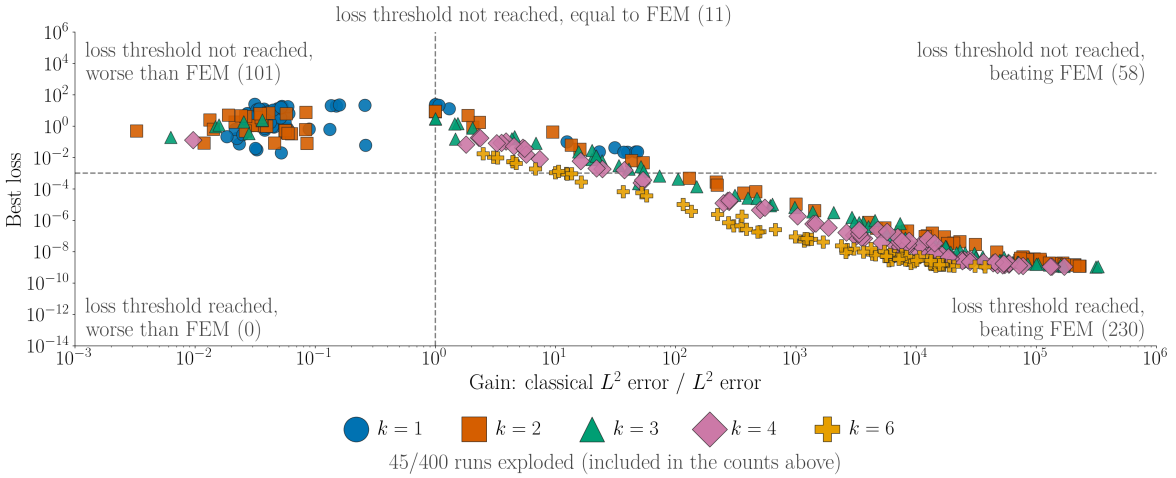


Figure 5.6.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, the affine multiplier and the 400 runs of Figure 5.5. Best loss as a function of the gain with respect to the classical FEM, the color and marker indicating the polynomial degree k. The dashed lines separate the runs according to whether they reached the loss threshold and whether they are worse than, equal to or beating the classical FEM (see Remark 5.9), and the number of runs in each case is indicated, as well as, under the legend, the number of runs that exploded (included in these counts).

We also find the effect of the degree, since the trainings that do worse than the classical finite elements are almost only of low degree. They are mostly $k = 1$ and $k = 2$, with a few $k = 3$ and a single $k = 4$, while no training with $k = 6$ is among them. The trainings with $k = 1$ are moreover all above the loss threshold. Furthermore, the highest gains are obtained for the intermediate degrees $k = 2$, 3 and 4, the gains obtained for $k = 6$ being slightly lower, at equal loss.

This figure reports, for each of the 400 trainings (i.e. the 20 seeds of each configuration), the best loss obtained during training, but it does not allow us to distinguish the trainings that explode, of which there are 45, that is, those whose loss history contains a `NaN` (Remark 5.9). Since the retained network is the one with the lowest loss obtained before the explosion, these trainings nevertheless have a point on the figure, mixed with those of the other trainings. We set these cases aside for now and, on the 355 remaining trainings, we seek to categorize the learned multipliers according to whether they allow beating the classical finite elements, doing worse or recovering the same error. To do so, we classify the 355 trainings that did not explode according to the shape (Remark 5.10) of their multipliers m_θ (constant, step, that is, a very steep variation, or other) and according to their signs (positive, negative or changing sign over the domain).

Remark 5.10. *The classification according to the shape and the sign is made on the multiplier m_θ evaluated on a uniform grid of 200 points of Ω, for the network associated with the best loss obtained during each training. We denote by $m_i := m_\theta(x_i)$ the value of m_θ at the grid point $x_i := (i-1)/199$, for $i = 1, \dots, 200$. We say that m_θ is constant if:*

$$\max_i m_i - \min_i m_i < 10^{-3} \max_i |m_i|,$$

that is, if its variation is negligible compared with its own size. We say that the multiplier is a step if:

$$\max_i |m_{i+1} - m_i| \geq 0.35 \left(\max_i m_i - \min_i m_i \right),$$

that is, if its largest variation between two consecutive grid points (i.e. over 0.5% of the domain) represents at least 35% of its total amplitude. These criteria are not optimal, and a few trainings are misclassified, while remaining close to the constant or step cases. This does not change the observations that follow.

Figure 5.7 gives the distribution of the trainings between these cases and, for each of them, the proportion of those that do worse than, as well as, or better than the classical finite elements. Figure 5.8 then shows an example of a learned multiplier m_θ for a training in each of these cases. This training is chosen as the one whose gain is the median gain among the trainings of the same shape and the same sign, this gain being indicated above each profile.

We observe that the shape and the sign of m_θ almost entirely determine the outcome of a training, that is, whether it beats the classical finite elements, recovers their error or does worse. The 9 trainings for which m_θ is constant (first column) give the same error as the classical finite elements, which is expected since a constant multiplier does not modify the finite element space. The 8 trainings for which m_θ is a step (second column) never beat the classical finite elements. The interpretation is that a step in m_θ makes the functions of the enriched space quasi-discontinuous, which the degrees of freedom cannot correct. The 59 trainings for which m_θ changes sign (last row), 4 of which are steps, never beat the classical finite elements. Indeed, every function of the enriched space vanishes where m_θ vanishes, whereas the exact solution is strictly positive inside the domain, since the source term is positive and the solution is zero on the boundary. The enriched space therefore cannot correctly approximate the solution in the neighborhood of this point.

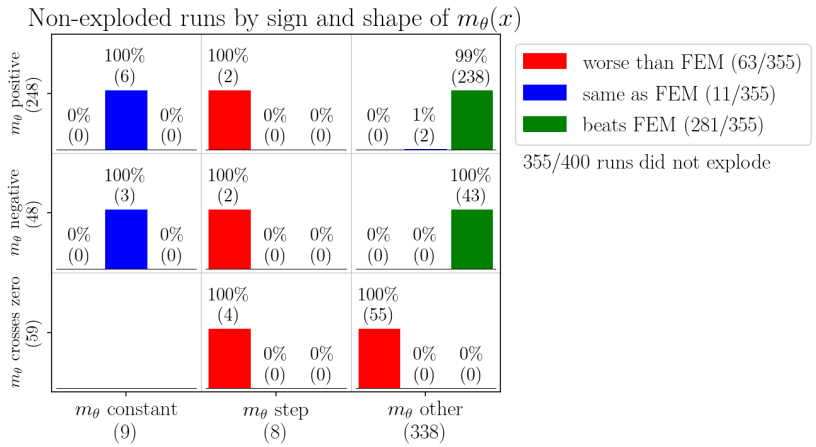


Figure 5.7.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the affine multiplier. Distribution of the 355 runs of Figure 5.5 that did not explode, according to the shape (columns) and the sign (rows) of the learned multiplier. For each case, the bars give the percentage (and the number) of runs that are worse than, the same as, or beating the classical FEM, and the legend gives the total number of runs for each outcome.

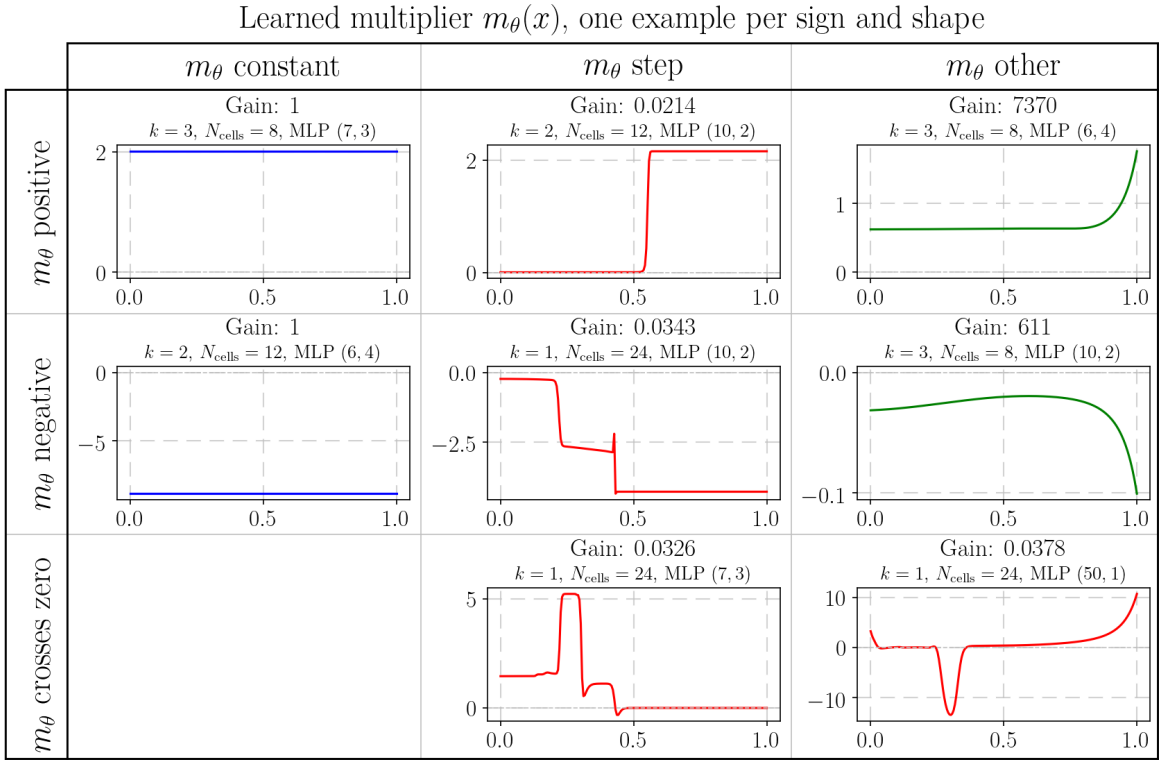


Figure 5.8.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the affine multiplier. Learned multiplier m_θ for one run per shape (columns) and sign (rows), chosen as the run of median gain among the non-exploded runs of the case. The configuration (k, N_{cells}), the MLP shape and the gain of each run are given above each plot, and the color indicates the outcome with respect to the classical FEM.

Conversely, a multiplier that is negative everywhere is not a problem, since changing the sign of all the basis functions does not modify the space. The 43 trainings of this type (excluding steps and constants) all beat the classical finite elements, as do 238 of the 240 trainings whose multiplier is positive and non-trivial. The 2 remaining trainings, whose multiplier is nonetheless positive and non-trivial, only do as well as the classical finite elements. These are smooth multipliers that stayed close to their initial value, the training having made practically no progress (best loss of the order of 3 and 22, far above the threshold). The profiles of Figure 5.8 are consistent with these observations and with the criteria of Remark 5.10: the constant multipliers are flat, the steps are abrupt jumps, and the only profiles that beat the classical finite elements (in green) are smooth and of constant sign. In total, out of the 74 trainings that do not beat the classical finite elements, 11 do as well (9 are constant and 2 are of another shape) and 63 do worse (8 are steps and 55 change sign without being steps). The trainings that do worse than the classical finite elements are therefore those whose multiplier is a step or changes sign. Now, imposing that m_θ be strictly positive is easy by construction, and this is what we test in the next paragraph.

Positive multiplier. In view of the previous analysis, we propose to constrain the multiplier to be strictly positive by construction, by parametrizing it as:

$$m_\theta(x) = \exp\big(2\tanh(v_\theta(x))\big), \tag{5.9}$$

where v_θ is the output of the network. As for the affine multiplier, the network weights being initialized to small values, v_θ is close to 0 at initialization and m_θ is close to $\exp(0) = 1$. The multiplier is therefore roughly constant at initialization, which amounts to starting from the classical finite element basis. This parametrization immediately rules out negative multipliers or multipliers changing sign, the latter being responsible for most of the trainings that do worse than the classical finite elements with the affine multiplier. The budget study is identical to the previous one, with the same configurations and the same seeds, that is, 400 trainings in total. Figures 5.9 and 5.10 summarizes its results, in the same way as Figures 5.5 and 5.6 for the affine multiplier.

In Figure 5.9, most of the observations made for the affine multiplier still hold. The number of seeds that beat the classical finite elements remains greater than or equal to that of the seeds that reach the threshold, the latter increases with the polynomial degree k (no seed for $k = 1$), the median errors and those of the best seeds are lower than that of the classical finite elements by several orders of magnitude, and the $(50, 1)$ configuration remains the one that gives the highest errors. The main difference concerns $k = 1$ where, although no seed reaches the threshold, many more seeds beat the classical finite elements than with the affine multiplier (29 out of 80 against 9).

Figure 5.10 also leads to the same observations as for the affine multiplier. The number of trainings that beat the classical finite elements is comparable (290 out of 400 against 288), no training below the threshold does worse than them, but some of those that do not reach the threshold beat them nonetheless, and the gain is correlated with the loss. The trainings that do worse than the classical finite elements are moreover almost only of low degree, while the highest gains are obtained for $k = 2$, 3 and 4.

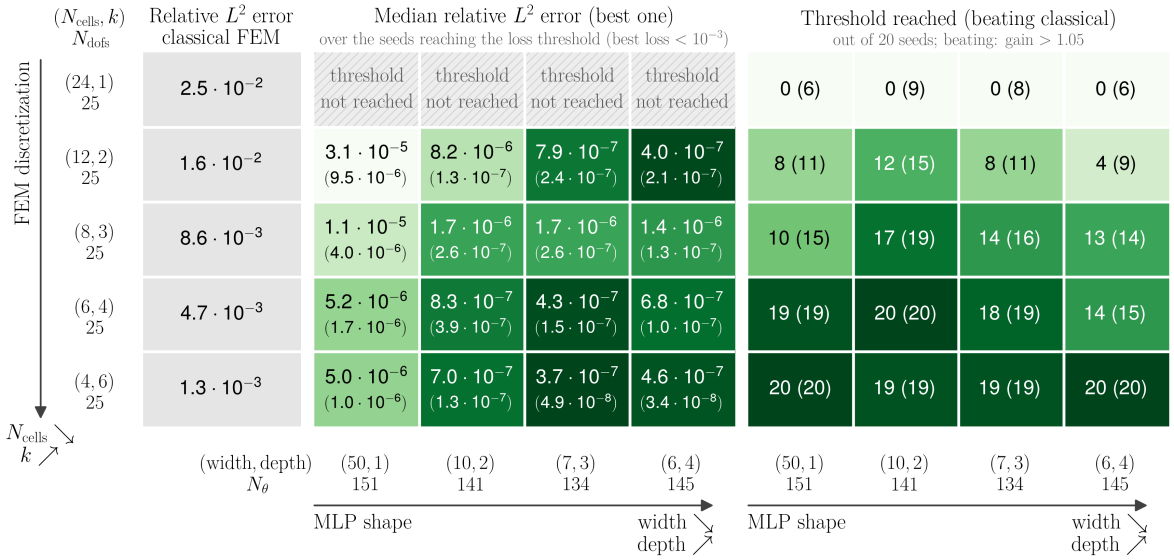


Figure 5.9.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $N_{\text{dofs}} = 25$, $134 \leq N_\theta \leq 151$ and the positive multiplier, with the FEM discretizations (N_{cells}, k) in rows and the MLP shapes (width, depth) in columns. Left – Classical FEM L^2 relative error. Middle – Median (and best seed) L^2 relative error of the learned basis over the seeds that reached the loss threshold. Right – Number of seeds that reached the loss threshold, out of 20, and, in parentheses, number of seeds that beat the classical FEM (see Remark 5.9).

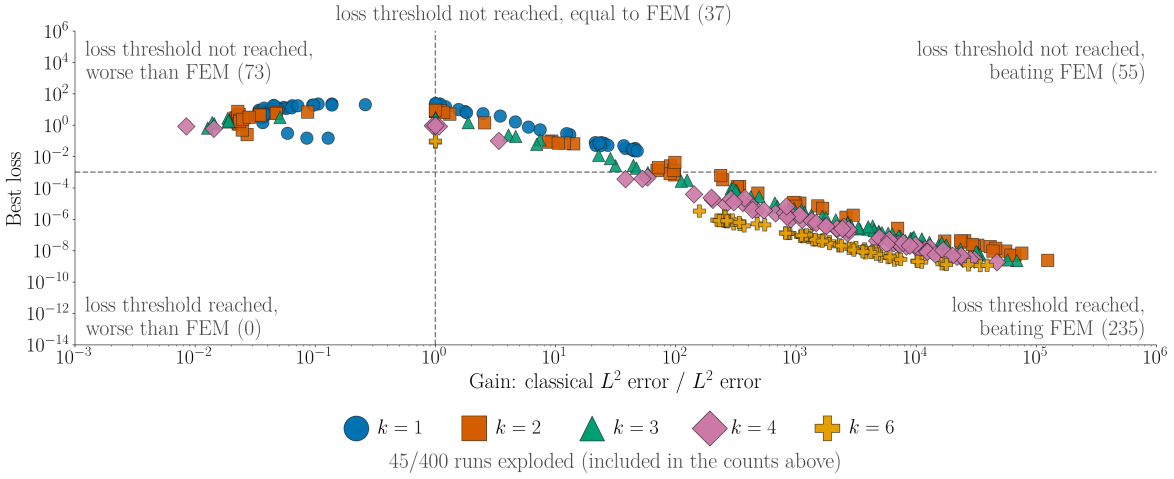


Figure 5.10.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, the positive multiplier and the 400 runs of Figure 5.9. Best loss as a function of the gain with respect to the classical FEM, the color and marker indicating the polynomial degree k. The dashed lines separate the runs according to whether they reached the loss threshold and whether they are worse than, equal to or beating the classical FEM (see Remark 5.9), and the number of runs in each case is indicated, as well as, under the legend, the number of runs that exploded (included in these counts).

As for the affine multiplier, 45 trainings explode (of which 27 for $k = 1$). We set them aside for now and, on the 355 remaining trainings, we classify the learned multipliers according to their shape (constant, step or other), the criteria being those of Remark 5.10. The sign no longer needs to be considered since m_θ is positive by construction. Figure 5.11 gives the distribution of the trainings between these shapes and, for each of them, the proportion of those that do worse than, as well as, or better than the classical finite elements. Figure 5.12 then shows an example of a learned multiplier for each shape, chosen as for the affine multiplier.

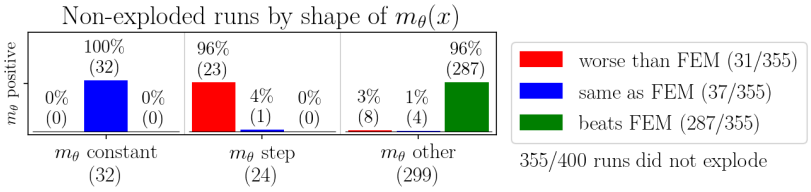


Figure 5.11.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the positive multiplier. Distribution of the 355 runs of Figure 5.9 that did not explode, according to the shape of the learned multiplier (columns), which is positive by construction (hence the single row). For each case, the bars give the percentage (and the number) of runs that are worse than, the same as, or beating the classical FEM, and the legend gives the total number of runs for each outcome.

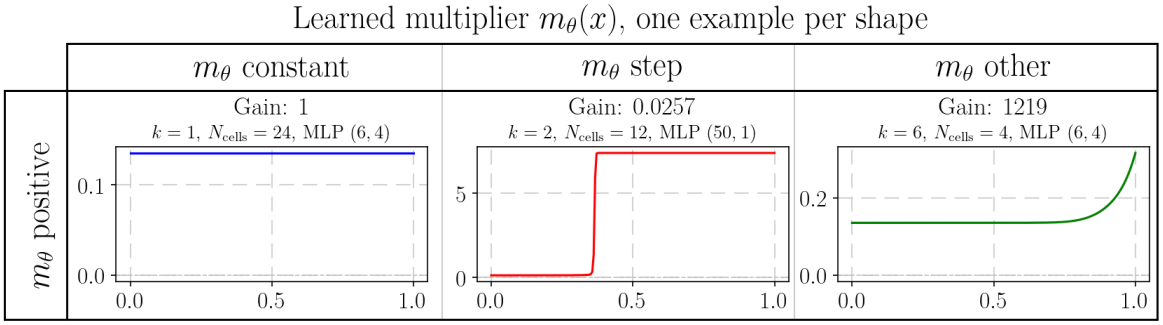


Figure 5.12.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the positive multiplier. Learned multiplier m_θ for one run per shape (columns), chosen as the run of median gain among the runs of the case, none of which is a step (hence the empty second column). The configuration (k, N_{cells}), the MLP shape and the gain of each run are given above each plot, and the color indicates the outcome with respect to the classical FEM.

As for the affine multiplier, the shape of m_θ almost entirely determines the outcome of a training. The 32 trainings for which m_θ is constant (first column) all give the same error as the classical finite elements. Among the 24 trainings for which m_θ is a step (second column), 23 do worse than the classical finite elements and only one does as well. Finally, 287 of the 299 trainings of another shape (last column) beat the classical finite elements. The 12 remaining ones (8 that do worse and 4 that do as well) are trainings misclassified by the criteria of Remark 5.10, which are in reality close to the

first two categories. The first 8 are steps whose transition spreads over a few grid points and the other 4 are almost constant. For the multipliers of another shape, m_θ stays almost constant, close to its lower bound e^{-2} (where tanh is saturated), over most of the domain, and only varies near $x = 1$, where the boundary layer lies.

In total, up to the 12 misclassified trainings, the non-exploded trainings that do not beat the classical finite elements are those whose multiplier is constant or is a step. The positive multiplier therefore rules out sign changes, which concerned 59 trainings with the affine multiplier, but not steps, which are more numerous here (24 against 8) and remain the main cause of the trainings that do worse than the classical finite elements. These steps are also those observed just before the explosion of the trainings, as shown by Figure 5.13 for all the exploded trainings and by Figure 5.14 for one of them.

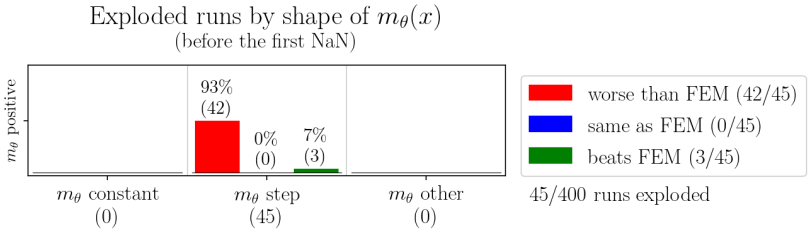


Figure 5.13.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the positive multiplier. Distribution of the 45 runs of Figure 5.9 that exploded, according to the shape of the last multiplier before the first `NaN` (columns), which is positive by construction (hence the single row), the number of runs of each shape being given below the columns. For each case, the bars give the percentage (and the number) of runs that are worse than, the same as, or beating the classical FEM, and the legend gives the total number of runs for each outcome.

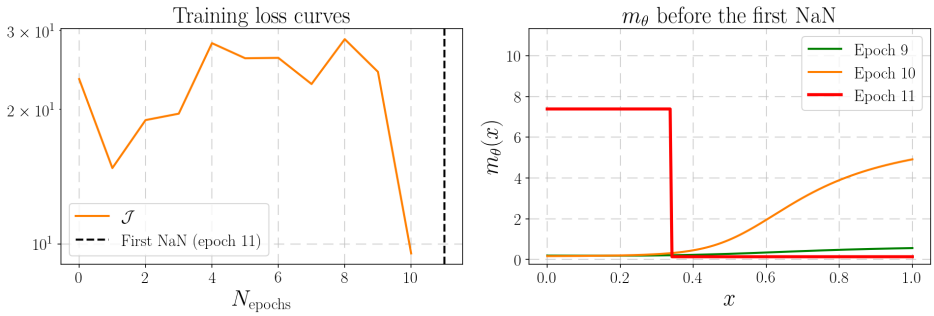


Figure 5.14.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, the positive multiplier, $k = 1$, $N_{\text{cells}} = 24$ and the MLP $(10, 2)$ (seed 4), chosen as the run of median gain (0.042) among the 45 exploded runs, whose first `NaN` is at epoch 11. Left – Training loss $\mathcal{J}$ as a function of the number of epochs, the dashed line marking the first `NaN`. Right – Learned multiplier m_θ at the last three epochs before the first `NaN` (epochs 9 to 11, as indicated in the legend), the red curve being the last one, which is a step.

For the 45 trainings that explode, the multiplier just before the appearance of the first `NaN` is in all cases a step (Figure 5.13). Among them, 42 do worse than the classical

finite elements, and the other 3 beat them thanks to the best checkpoint kept before the explosion. Figure 5.14 illustrates this behavior on one of these trainings, chosen as the one whose gain is the median. The multiplier m_θ is smooth at epoch 10, then abruptly becomes a step at epoch 11, just before the loss becomes `NaN`. This sudden change is not specific to the multiplier, it stems from an abnormally large update of the weights by the optimizer. A likely explanation is that we use the ENG optimizer with the default parameters of `ScimBa`, in particular a regularization of the Gram matrix (10^{-6}, the default value of ENG) that may be too small. This regularization serves to make the Gram matrix invertible, which the optimizer must invert to correct the descent direction. When it is too small, the matrix may remain almost non-invertible and the corrected direction become very large. The line search, which chooses the step size solely from the value of the loss, can then accept an update that strongly displaces the weights of the network.

To avoid this problem, a first solution would be to tune the parameters of the optimizer, for example with a stronger regularization. However, the problem runs deeper. If an update that is too large has led here to this step in the multiplier, nothing guarantees that the same holds for each exploded training, nor that the passage from a smooth multiplier to a step is each time due to the optimizer. Rather than correcting the optimizer, we therefore seek to directly prevent the formation of steps, which we do by bounding the weights of the network.

Positive multiplier with bounded weights. We take up the multiplier defined in (5.9), which constrains the multiplier to remain positive. In the previous paragraph, we saw that the steps in the multiplier m_θ were the only cause of the trainings that do worse than the classical finite elements. A step in m_θ requires a large Lipschitz constant of v_θ, which, for an MLP with tanh activation, is bounded by the product of the spectral norms of the weight matrices of its layers [127, Proposition 1] and thus generally requires large weights. At initialization, the weights are small and the network is smooth, but the optimizer can make some weights grow in a few epochs, as in the previous example (Figure 5.14). For this reason, we bound here the weights of the network, in order to limit the stiffness of v_θ and thus avoid steps in m_θ. To do so, we bound the norm of the weight matrix W of each layer, output layer included (the biases are not concerned), through a soft clipping at a multiple (here four times, a value retained after a few preliminary tests and not optimized) of its norm at initialization. Concretely, at each evaluation of the network, W is replaced by $\mathrm{softclip}(W)$, with:

$$\mathrm{softclip}(W) := W\left(1+\left(\frac{\|W\|_F}{4\,\|W^{(0)}\|_F}\right)^4\right)^{-1/4}, \tag{5.10}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and $W^{(0)}$ the value of W at initialization (drawn at random, so the bound depends on the seed).

As seen in Figure 5.15, as long as $\|W\|_F$ remains small compared with $4\,\|W^{(0)}\|_F$, $\mathrm{softclip}(W)$ is almost equal to W, which explains why the network starts like the unbounded one. When $\|W\|_F$ grows, the norm of $\mathrm{softclip}(W)$ approaches $4\,\|W^{(0)}\|_F$ without ever reaching it, whatever the value of W updated by the optimizer. This bound is a constraint on the form of the network, not a penalization of the loss. It changes neither the expression of the multiplier, nor the loss, nor the number of weights.

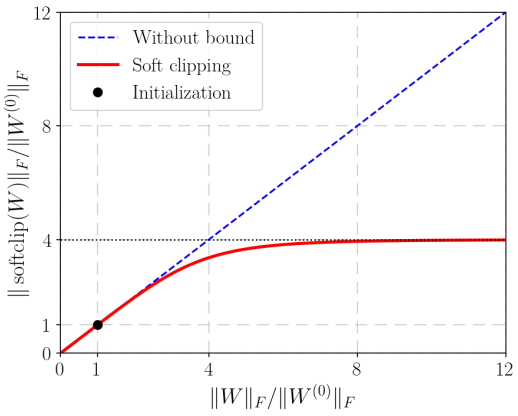


Figure 5.15.: Soft clipping of a weight matrix W of a layer of the network. Frobenius norm of softclip(W), the matrix used in the forward pass, as a function of the Frobenius norm of the stored matrix W, both divided by the initial norm $\|W^{(0)}\|_F$. The blue dashed line is the case without bound and the dotted line is the bound 4. The black dot marks the initialization.

The study carried out next is identical to the previous ones, with the same configurations and the same seeds, i.e. 400 trainings in total. Figures 5.16 and 5.17 summarizes its results, in the same way as Figures 5.9 and 5.10 for the positive multiplier.

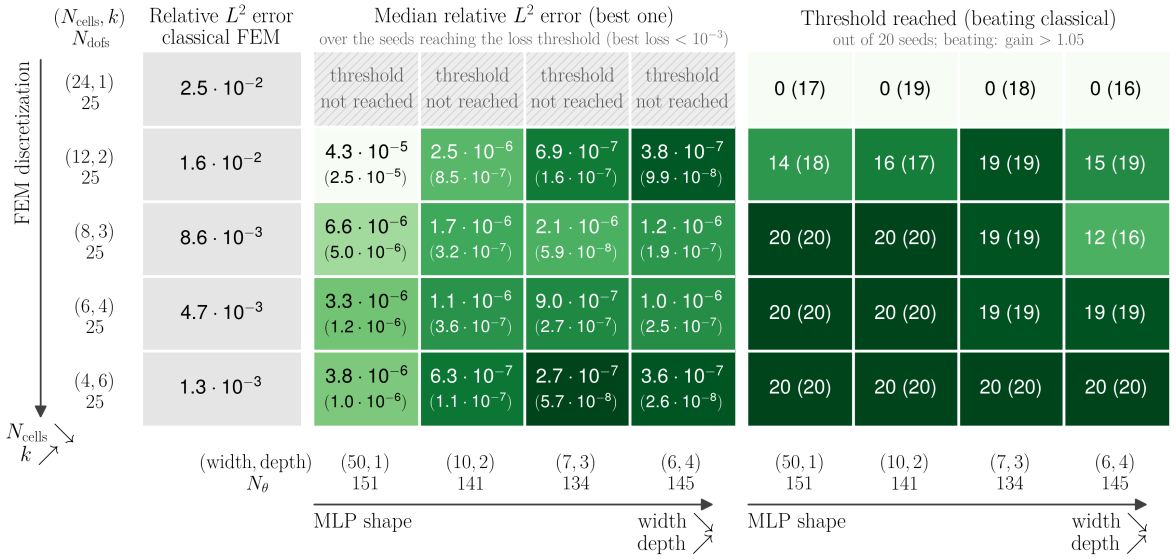


Figure 5.16.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $N_{\text{dofs}} = 25$, $134 \leq N_\theta \leq 151$ and the positive multiplier with bounded network weights, with the FEM discretizations (N_{cells}, k) in rows and the MLP shapes (width, depth) in columns. Left – Classical FEM L^2 relative error. Middle – Median (and best seed) L^2 relative error of the learned basis over the seeds that reached the loss threshold. Right – Number of seeds that reached the loss threshold, out of 20, and, in parentheses, number of seeds that beat the classical FEM (see Remark 5.9).

In Figure 5.16, we find most of the observations made for the previous multiplier. The main difference concerns $k = 1$ where, although no seed reaches the threshold, 70 seeds out of 80 beat the classical finite elements, against 29 with the positive multiplier (and 9 with the affine multiplier). From $k = 2$, more seeds also reach the threshold.

Figure 5.17 shows that no training does worse than the classical finite elements anymore, whereas 73 did with the positive multiplier (Figure 5.10), and that none explodes. The number of trainings that beat the classical finite elements increases markedly (376 out of 400 against 290), as does the number of those that reach the threshold. The remaining trainings do as well as the classical finite elements, and are fewer than with the positive multiplier (24 against 37).

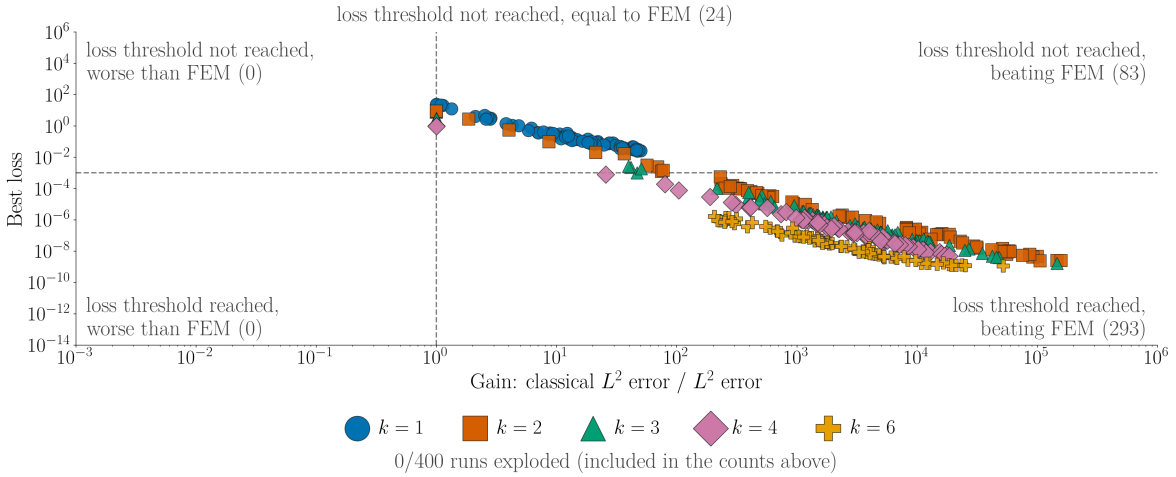


Figure 5.17.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, the positive multiplier with bounded network weights and the 400 runs of Figure 5.16. Best loss as a function of the gain with respect to the classical FEM, the color and marker indicating the polynomial degree k. The dashed lines separate the runs according to whether they reached the loss threshold and whether they are worse than, equal to or beating the classical FEM (see Remark 5.9), and the number of runs in each case is indicated, as well as, under the legend, the number of runs that exploded (included in these counts).

As for the previous multipliers, we can also analyze the shape of the learned multipliers. Since no training exploded, the 400 trainings are sorted according to the shape of m_θ (constant, step or other), its sign being positive by construction. Figure 5.18 gives the distribution of the trainings among these shapes and, for each of them, the proportion of those that do worse than, as well as, or better than the classical finite elements. Figure 5.19 shows an example of learned multiplier for each shape, chosen as for the previous multipliers.

The main result is that no learned multiplier is a step anymore, whereas there were 24 with the positive multiplier and 8 with the affine multiplier. The bound on the weights thus fulfills its role, which explains why no training does worse than the classical finite elements or explodes. The 22 trainings for which m_θ is constant, against 32 with the positive multiplier, all recover the classical error, as expected. Finally, among the 378 trainings of another shape, 376 beat the classical finite elements and the remaining 2,

of degree $k = 1$, are almost constant (without being classified as such by the criteria, see Remark 5.10) and also do as well.

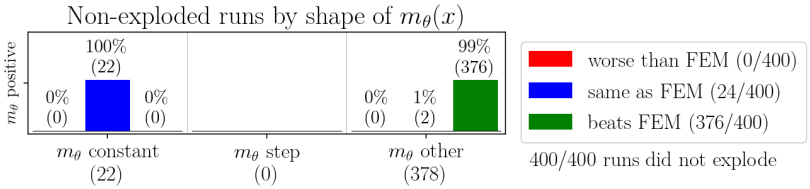


Figure 5.18.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the positive multiplier with bounded network weights. Distribution of the 400 runs of Figure 5.16 (none of which exploded), according to the shape of the learned multiplier (columns), which is positive by construction (hence the single row), no run being a step (hence the empty second column). For each case, the bars give the percentage (and the number) of runs that are worse than, the same as, or beating the classical FEM, and the legend gives the total number of runs for each outcome.

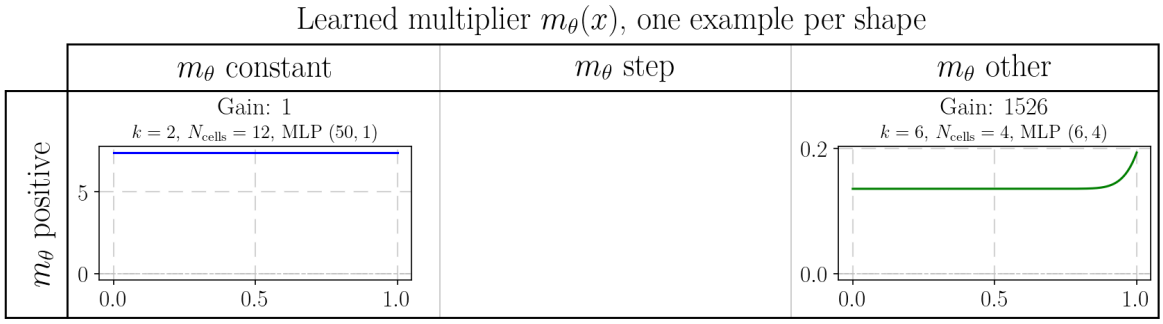


Figure 5.19.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and the positive multiplier with bounded network weights. Learned multiplier m_θ for one run per shape (columns), chosen as the run of median gain among the non-exploded runs of the case. The configuration (k, N_{cells}), the MLP shape and the gain of each run are given above each plot, and the color indicates the outcome with respect to the classical FEM.

Comparison of the multipliers. We now compare the three multipliers. Figure 5.20 shows, for each multiplier, the distribution of the gain of the trainings that beat the classical finite elements, as a number of trainings per gain interval (on a logarithmic scale). The hatched parts correspond to negative multipliers (for the affine multiplier only). Unlike in the previous paragraphs, only the non-exploded trainings are counted among those that beat the classical finite elements, their number being recalled in the legend. Table 5.8 summarizes these results by counting, for each multiplier, the number of trainings that exploded, the number of those that beat the classical finite elements, and the number of those whose gain exceeds 10^3, then 10^5, with their proportion among the 400 trainings of each study.

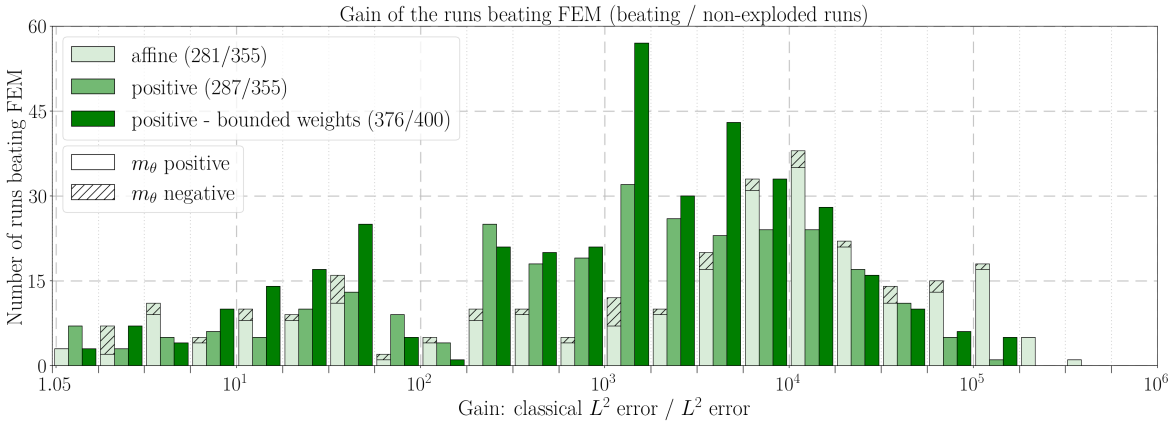


Figure 5.20.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $N_{\text{dofs}} = 25$ and $134 \leq N_\theta \leq 151$, for the affine multiplier (5.8), the positive multiplier (5.9) and the positive multiplier with bounded network weights (5.10). Number of runs beating the classical FEM as a function of their gain (see Remark 5.1). The legend gives, for each multiplier, the number of runs beating the classical FEM out of the number of non-exploded runs. The hatched parts correspond to runs for which the learned multiplier is negative.

In Figure 5.20, we see that the three distributions spread over several orders of magnitude, and that the trainings whose multiplier is negative are a minority in each interval. The legend shows that the positive multiplier with bounded weights counts almost 100 more trainings that beat the classical finite elements, since the problematic cases (explosions, steps) have disappeared. This translates into bars that are higher than or at the same level as the others for this multiplier in most gain intervals, with a marked peak around 10^3. In contrast, the affine multiplier is the only one to count a non-negligible number of trainings beyond 10^5, which the positive multiplier hardly allows. These high gains seem to come from the fact that the affine multiplier is the least constrained. For both positive multipliers, with or without bounded weights, the values of $m_\theta = \exp(2 \tanh v_\theta)$ are themselves bounded, which restricts the set of functions that the network can represent. This removes the failures, but may also prevent reaching some of the best gains.

multiplier	**exploded**	**beat FEM**	**gain** $\geq 10^3$	**gain** $\geq 10^5$
affine	45 (11 %)	281 (70 %)	188 (47 %)	**24 (6 %)**
positive	45 (11 %)	287 (72 %)	163 (41 %)	1 (<1 %)
positive - bounded weights	**0 (0 %)**	**376 (94 %)**	**228 (57 %)**	5 (1 %)

Table 5.8.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$ and 400 runs per multiplier. The rows are the affine multiplier, the positive multiplier and the positive multiplier with bounded network weights. For each multiplier, number of runs (and share of the 400 runs) that exploded, and, among the non-exploded runs, that beat the classical FEM (see Remark 5.9) and whose gain is larger than 10^3 or than 10^5. In bold: best result of each column.

In Table 5.8, we see that the affine and positive multipliers have the same number of exploding trainings (45, i.e. 11%) and comparable numbers of trainings that beat the classical finite elements (281 and 287, i.e. about 70%). The affine multiplier nevertheless yields the highest gains, with 188 trainings whose gain exceeds 10^3, against 163, and above all 24 trainings above 10^5, against 1 for the positive multiplier. The positive multiplier with bounded weights is by far the most robust. No training explodes, and 376 trainings out of 400 (94%) beat the classical finite elements. Moreover, 228 trainings (57%) have a gain above 10^3, against 188 and 163. It counts, however, only 5 trainings above 10^5, between the 24 of the affine multiplier and the single training of the positive multiplier. The highest gains thus remain those of the affine multiplier, but at the price of many failures.

5.2.1.4. Conclusion on the choice of the multiplier

The studies of this section show that the multiplicative enrichment of the basis can reduce the L^2 error by several orders of magnitude at a fixed number of degrees of freedom, with gains exceeding 10^5 for the best seeds (lower bounds when the error of the learned basis is close to the accuracy of the reference solution, see Remark 5.4). The polynomial degree k and the number of cells N_{cells} are the decisive levers, whereas the width of the network has little visible influence on this test case. For a fixed budget, the analysis of the shape and sign of the learned multipliers makes it possible to understand the outcome of a training. A constant multiplier gives back the error of the classical finite elements, whereas a multiplier that changes sign or that is a step does worse, and can lead the training to explode. This first motivates a positive multiplier, which removes the sign changes, then a bound on the weights of the network, which removes the steps. With the latter, no training explodes or does worse than the classical finite elements, and 376 trainings out of 400 beat them, against about 70 % for the two other multipliers.

This choice nevertheless has several limitations. First, none of the three parametrizations avoids the trainings for which m_θ is constant (9, 32 and 22 for the affine, positive and positive with bounded weights multipliers). These trainings are not failures in the sense that they do worse than the finite elements, but they bring nothing, since, the multiplier being constant, the enriched space is the same as the classical finite element space. Nothing in the construction makes it possible to exclude them, and the bound on the weights does not avoid them either. Avoiding them would require understanding first why these trainings lead to a constant multiplier, which we have not investigated. Next, the degree $k = 1$ remains out of reach of the loss threshold for the three multipliers despite the gains obtained, which is unfortunate since it is an inexpensive case and the one where the classical finite elements have the largest error. A possible reason is that, for $\mathbb{Q}_1$, the second derivative inside the cells only comes from the multiplier m_θ, which limits the ability of the strong residual to be reduced; this remains to be verified. A loss based on a weak (variational) residual, as in VPINNs [82], could be considered as a perspective.

Furthermore, the positive multiplier, whose values are bounded by construction, restricts the set of functions that the network can represent, with or without bounded weights. The affine multiplier, although less reliable, thus reaches the highest gains (24 trainings above 10^5, against 1 for the positive multiplier and 5 for its version with

bounded weights). Moreover, the bound on the weights depends on a factor 4 fixed once and for all after only a few preliminary trials, and it prevents the steps from forming without addressing why the optimizer produces them.

Finally, the conclusions of this comparison of multipliers are limited by its framework. They were obtained for a single value of the parameter $\boldsymbol{\mu}^{(1)}$, which says nothing about the generalization to other values of $\boldsymbol{\mu}$, and the justifications (for example the argument on the sign change) rely on the positivity of the exact solution. The budget is moreover fixed to $N_{\text{dofs}} = 25$, with 100 epochs and $N_{\text{col}} = 5\,000$ collocation points in Ω, redrawn at each epoch. The next study (Section 5.2.1.5), which revisits the first study with the retained multiplier, partly lifts these limitations by considering the three values of $\boldsymbol{\mu}$ and the budget of the first study.

5.2.1.5. Revisiting the first study with the retained multiplier

In this study, we repeat the first study of the affine case (Section 5.2.1.1) with the multiplier retained at the end of the study of the choice of the parametrization (Section 5.2.1.3), namely the positive multiplier with bounded weights. The multiplier is thus the one defined in (5.9), the weights of each layer of the network v_θ being bounded by the softclip operator with factor 4 (5.10). Everything else is identical to the first study. The results are thus directly comparable to those of the affine multiplier, with the same number of degrees of freedom and the same training budget, the latter being that of the first study.

As for the first study, we quantify the gain brought by the learned basis for the three values of $\boldsymbol{\mu}$ defined in (5.4), varying the polynomial degree k and the number of cells N_{cells}, and we compare the results to those of the affine multiplier. In Tables 5.9 and 5.10, we thus compare the L^2 errors of the standard finite element solution u_h and of the enriched solution $u_{h,\theta}$ with respect to the reference solution u, for the three values of $\boldsymbol{\mu}$, $k \in \{3, 4\}$ and $N_{\text{cells}} \in \{2, 4, 8\}$.

Tables 5.9 and 5.10 show that the learned basis, associated with the best seed, reduces the error by several orders of magnitude in all the configurations for which at least one seed goes below the threshold. The gain of the best seed ranges from 195 to 104 590. The two configurations for which no seed reaches the threshold are the same as with the affine multiplier, namely $k = 3$ and $N_{\text{cells}} = 2$ for $\boldsymbol{\mu}^{(2)}$ and $\boldsymbol{\mu}^{(3)}$ (Table 5.3). As with the affine multiplier, the gain of the best seed generally decreases as $\boldsymbol{\mu}$ increases, the stronger convection making the boundary layer steeper. The only exception concerns $k = 4$ and $N_{\text{cells}} = 8$, where the gains increase slightly with $\boldsymbol{\mu}$ (19 306, 20 704 and 24 601).

The main difference with the affine multiplier concerns the number of seeds that reach the threshold, which is higher in almost all the configurations (Tables 5.9 and 5.10, compared with Tables 5.3 and 5.4). At least 14 seeds out of 20 reach it as soon as at least one does, whereas this number went down to 2 with the affine multiplier. This gain in reliability is particularly clear on coarse meshes: for $N_{\text{cells}} = 2$, this number is between 17 and 20, against 2 to 8 with the affine multiplier.

On the other hand, the best gains are lower for fine meshes. They are comparable to those of the affine multiplier for $N_{\text{cells}} = 2$, then the gap widens with the number of cells, becoming clear for $N_{\text{cells}} = 8$. This is in line with the remark made in the study of the choice of the multiplier (Figure 5.20), according to which the positive multiplier,

whose values are bounded, restricts the set of functions that the network can represent and can prevent reaching the best gains.

			classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	$N_{\mathbf{cells}}$	$N_{\mathbf{dofs}}$	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	7	$1.95 \cdot 10^{-1}$	$9.97 \cdot 10^{-4}$	195	$1.28 \cdot 10^{-4}$	18
	4	13	$5.11 \cdot 10^{-2}$	$7.29 \cdot 10^{-6}$	7 014	$5.79 \cdot 10^{-8}$	18
	8	25	$8.64 \cdot 10^{-3}$	$1.04 \cdot 10^{-7}$	82 991	$2.58 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(2)}$	2	7	$2.61 \cdot 10^{-1}$	–	–	–	0
	4	13	$7.15 \cdot 10^{-2}$	$3.81 \cdot 10^{-5}$	1 877	$2.4 \cdot 10^{-6}$	17
	8	25	$1.44 \cdot 10^{-2}$	$2.9 \cdot 10^{-7}$	49 452	$6.48 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(3)}$	2	7	$3.42 \cdot 10^{-1}$	–	–	–	0
	4	13	$9.08 \cdot 10^{-2}$	$1.23 \cdot 10^{-4}$	740	$1.16 \cdot 10^{-5}$	17
	8	25	$2.09 \cdot 10^{-2}$	$1.41 \cdot 10^{-6}$	14 838	$4.66 \cdot 10^{-8}$	14

Table 5.9.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 3$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the positive multiplier with bounded weights. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

			classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	$N_{\mathbf{cells}}$	$N_{\mathbf{dofs}}$	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	9	$9.16 \cdot 10^{-2}$	$3.45 \cdot 10^{-5}$	2 657	$3.89 \cdot 10^{-7}$	20
	4	17	$1.7 \cdot 10^{-2}$	$1.62 \cdot 10^{-7}$	104 590	$6.83 \cdot 10^{-9}$	18
	8	33	$1.63 \cdot 10^{-3}$	$8.44 \cdot 10^{-8}$	19 306	$2.33 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(2)}$	2	9	$1.24 \cdot 10^{-1}$	$1.94 \cdot 10^{-4}$	638	$8.01 \cdot 10^{-6}$	17
	4	17	$2.76 \cdot 10^{-2}$	$8.23 \cdot 10^{-7}$	33 592	$3.22 \cdot 10^{-9}$	18
	8	33	$3.29 \cdot 10^{-3}$	$1.59 \cdot 10^{-7}$	20 704	$4.4 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(3)}$	2	9	$1.54 \cdot 10^{-1}$	$7.25 \cdot 10^{-4}$	213	$1.27 \cdot 10^{-4}$	17
	4	17	$3.92 \cdot 10^{-2}$	$4.09 \cdot 10^{-6}$	9 571	$8.28 \cdot 10^{-8}$	20
	8	33	$5.57 \cdot 10^{-3}$	$2.26 \cdot 10^{-7}$	24 601	$6.43 \cdot 10^{-9}$	19

Table 5.10.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 4$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the positive multiplier with bounded weights. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

5.2.2. Finite element method – Cellwise basis

In this section, we consider the continuous finite element method (Section 2.2.1) whose local Lagrange basis $\phi_{c,i}$ is enriched by a neural network $v_{c,\theta}$ specific to each cell K_c (cellwise basis), by means of a multiplier $m_{c,\theta}$, as defined in (5.7). The objective is to see whether, with one network per cell, one can obtain gains of the same order as those of the patchwise case. This gives the basis more local freedom, at the price of a number of weights N_θ that grows with the number of cells (Section 4.3). Unlike the patchwise case (Section 5.2.1), where a single network shared by all the cells automatically guarantees the continuity of the basis, nothing here imposes that the networks of two neighboring cells give the same value to a shared degree of freedom. Continuity must therefore be imposed by construction, which is done by modulating the output of each network by a level-set function local to the cell, which vanishes on its boundary.

As in the patchwise case, the networks are trained with the PINN loss (5.5) for a fixed value of $\boldsymbol{\mu}$. We do not repeat here all the studies of the patchwise case, and limit ourselves to the gain brought by the learned basis (Section 5.2.1.5), with the positive multiplier with bounded weights, which was the most reliable in the study of the parametrization (Section 5.2.1.3).

We denote by φ_c the level-set function local to the cell K_c, defined by:

$$\varphi_c(x) := 4\,\hat{\xi}\,(1-\hat{\xi}), \qquad \hat{\xi} = T_{K_c}^{-1}(x) \in \hat{K} = [0,1], \tag{5.11}$$

where $\hat{\xi}$ is the coordinate of x in the reference element $\hat{K}$, $T_{K_c} : \hat{K} \to K_c$ being the transformation that maps the reference element onto the cell K_c, defined in (4.5). The mesh being uniform here and the global mapping $\mathcal{G}_\theta$ the identity, T_{K_c} is affine and $\hat{\xi}$ is simply the relative position of x in K_c. The function φ_c vanishes on the boundary of K_c, remains between 0 and 1 and equals 1 at its center.

Remark 5.11. *Unlike the level-set of the φ-FEM method (Section 2.2.3), for which $\Omega = \{\varphi < 0\}$, the functions φ_c, local to each cell K_c, are strictly positive inside it.*

The multiplier of the cell K_c is then defined by:

$$m_{c,\theta}(x) = \exp\Big(2\,\varphi_c(x)\,\tanh\big(v_{c,\theta}(x)\big)\Big), \qquad x \in K_c, \tag{5.12}$$

where φ_c is the local level-set defined in (5.11) and $v_{c,\theta}$ is the MLP with scalar output specific to the cell K_c, which takes the physical coordinate x as input. We choose to build these networks identically on each cell, with a tanh activation and 2 hidden layers of width 6 (i.e. 61 weights per cell, and thus $N_\theta = 61\,N_{\text{cells}}$ in total). This is the positive multiplier (5.9) of the patchwise case, up to the factor φ_c.

The multiplier (5.12) defined for the cellwise case has two properties. First, it is strictly positive and bounded, just as in the patchwise case. Second, it equals 1 on the boundary of K_c whatever the network. It is thus continuous at the interfaces between cells, but its derivative may exhibit a jump there. Moreover, the trace of the enriched basis $\phi_{c,i}^\theta$ on this boundary is that of the classical basis $\phi_{c,i}$, so that the basis functions associated with the same node in two neighboring cells take the same value at this node, as for the classical basis, which preserves the continuity of the basis. In the patchwise case, on the other hand, the trace is modified at the boundary of the cell by the multiplier m_θ.

As in the patchwise case, we want the training to start from the classical basis, and hence from the classical finite element solution. In patchwise, this is almost automatic, because the initial weights are small, so that $v_\theta \approx 0$ and the multiplier m_θ is almost constant, which modifies the classical space only very slightly. Here, this is no longer the case because the level-set φ_c transforms the constant level $\tanh(v_{c,\theta})$ of each cell into a non-constant multiplier, and the networks being drawn independently, these levels differ from one cell to another. To limit this effect, the last layer of each network (weights and bias) is drawn in an interval ten times smaller than the default one, which decreases the amplitude of $v_{c,\theta}$ on each cell and brings $m_{c,\theta}$ closer to 1, but the multiplier considered in cellwise is nevertheless not constant at initialization (Figure 5.22). For $\boldsymbol{\mu}^{(1)}$, $k = 3$ and $N_{\text{cells}} = 4$, the error of the finite element solution computed at initialization, before any training, is thus between 0.89 and 1.15 times that of the classical finite elements depending on the 20 seeds.

The weights of the networks are bounded with the same softclip operator as in the patchwise case (5.10), but it is applied here to each layer of each of the N_{cells} networks, each of them thus being bounded individually. For each layer, the bound equals four times the norm of the default draw of its weights, that is, the one before the reduction of the last layer described above. For the last layer, this bound is therefore about 40 times its effective initial norm, which makes it much less constrained than in the patchwise case. The training, the seeds (Remark 5.7) and the evaluation of the error with respect to the reference solution (Remark 5.4), compared with that of the classical finite elements at the same number of degrees of freedom (Remark 5.1), are those of the patchwise case (Section 5.2.1.5). The number of weights N_θ, which grows with N_{cells}, is not taken into account in this comparison.

Gains achieved with the learned basis. For the three values of $\boldsymbol{\mu}$ defined in (5.4), we compare, in Tables 5.11 and 5.12, the L^2 errors between the reference solution u and the standard finite element solution u_h, as well as the enriched solution $u_{h,\theta}$, for two polynomial degrees $k = 3$ and $k = 4$ and $N_{\text{cells}} \in \{2, 4, 8\}$.

Tables 5.11 and 5.12 show that the learned basis, associated with the best seed, reduces the error by several orders of magnitude wherever at least one seed goes below the threshold, the gain of the best seed ranging from 198 to about 78 000. This is of the same order as for the patchwise basis with the same multiplier (Tables 5.9 and 5.10), at the same number of degrees of freedom and thus at the same classical error. We also find the same trends as in the patchwise case. The gain tends to decrease as $\boldsymbol{\mu}$ increases, and the case $N_{\text{cells}} = 2$ is the most difficult, with three configurations without any seed below the threshold, whereas from $N_{\text{cells}} = 4$, at least 13 seeds out of 20 go below the threshold.

Concerning the difference between the two bases, neither of them seems to systematically prevail in terms of gain. The cellwise gain is sometimes markedly higher, for example for $k = 3$ and $N_{\text{cells}} = 8$ with $\boldsymbol{\mu}^{(3)}$, and sometimes lower, for example for $k = 4$ and $N_{\text{cells}} = 4$ with $\boldsymbol{\mu}^{(1)}$. In terms of robustness, the two bases are comparable from $N_{\text{cells}} = 4$, and the difference lies in the coarsest mesh. For $k = 4$ and $N_{\text{cells}} = 2$, only 13 seeds go below the threshold for $\boldsymbol{\mu}^{(2)}$ and none for $\boldsymbol{\mu}^{(3)}$, unlike the patchwise basis. For this test case, the cellwise basis thus brings no systematic gain in accuracy with respect to the patchwise basis, and it is slightly less robust on coarse meshes, for a number of

weights that grows with N_{cells} and a heavier construction.

A possible explanation is that the constraint $m_{c,\theta} = 1$ on the boundary of the cells limits the shapes of the multiplier available in the cell of the boundary layer. This remains to be verified.

				classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	N_{cells}	N_{dofs}	N_θ	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	7	122	$1.95 \cdot 10^{-1}$	$9.83 \cdot 10^{-4}$	198	$1.22 \cdot 10^{-4}$	17
	4	13	244	$5.11 \cdot 10^{-2}$	$2.87 \cdot 10^{-6}$	17 830	$1.53 \cdot 10^{-8}$	20
	8	25	488	$8.64 \cdot 10^{-3}$	$1.35 \cdot 10^{-7}$	63 999	$2.99 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(2)}$	2	7	122	$2.61 \cdot 10^{-1}$	–	–	–	0
	4	13	244	$7.15 \cdot 10^{-2}$	$1.96 \cdot 10^{-5}$	3 644	$2.07 \cdot 10^{-7}$	15
	8	25	488	$1.44 \cdot 10^{-2}$	$3.94 \cdot 10^{-7}$	36 475	$1.24 \cdot 10^{-8}$	17
$\boldsymbol{\mu}^{(3)}$	2	7	122	$3.42 \cdot 10^{-1}$	–	–	–	0
	4	13	244	$9.08 \cdot 10^{-2}$	$8.33 \cdot 10^{-5}$	1 090	$2.83 \cdot 10^{-6}$	13
	8	25	488	$2.09 \cdot 10^{-2}$	$2.8 \cdot 10^{-7}$	74 657	$4.1 \cdot 10^{-9}$	18

Table 5.11.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 3$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the cellwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). N_θ – total number of weights (61 per cell). / 20 – number of seeds below the loss threshold.

				classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	N_{cells}	N_{dofs}	N_θ	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	9	122	$9.16 \cdot 10^{-2}$	$3.2 \cdot 10^{-5}$	2 865	$2.3 \cdot 10^{-7}$	20
	4	17	244	$1.7 \cdot 10^{-2}$	$2.19 \cdot 10^{-7}$	77 598	$3.3 \cdot 10^{-9}$	20
	8	33	488	$1.63 \cdot 10^{-3}$	$5.75 \cdot 10^{-8}$	28 323	$1.8 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(2)}$	2	9	122	$1.24 \cdot 10^{-1}$	$1.59 \cdot 10^{-4}$	778	$8.46 \cdot 10^{-6}$	13
	4	17	244	$2.76 \cdot 10^{-2}$	$3.57 \cdot 10^{-7}$	77 483	$1.63 \cdot 10^{-9}$	19
	8	33	488	$3.29 \cdot 10^{-3}$	$9.76 \cdot 10^{-8}$	33 676	$3.85 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(3)}$	2	9	122	$1.54 \cdot 10^{-1}$	–	–	–	0
	4	17	244	$3.92 \cdot 10^{-2}$	$1.57 \cdot 10^{-6}$	24 990	$2.32 \cdot 10^{-8}$	19
	8	33	488	$5.57 \cdot 10^{-3}$	$2.45 \cdot 10^{-7}$	22 680	$1.04 \cdot 10^{-8}$	20

Table 5.12.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 4$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the cellwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical FEM. Learned basis (best seed) – L^2 relative error, gain with respect to the classical FEM and best loss of the best seed (see Remark 5.7). N_θ – total number of weights (61 per cell). / 20 – number of seeds below the loss threshold.

Comparison of the classical and learned bases. We now focus on $\boldsymbol{\mu}^{(1)}$ and $N_{\text{cells}} = 4$. We compare the solutions and the pointwise errors of the classical and learned bases (Figure 5.21), before looking at the learned multiplier $m_{c,\theta}$ and the enriched basis functions (Figure 5.22).

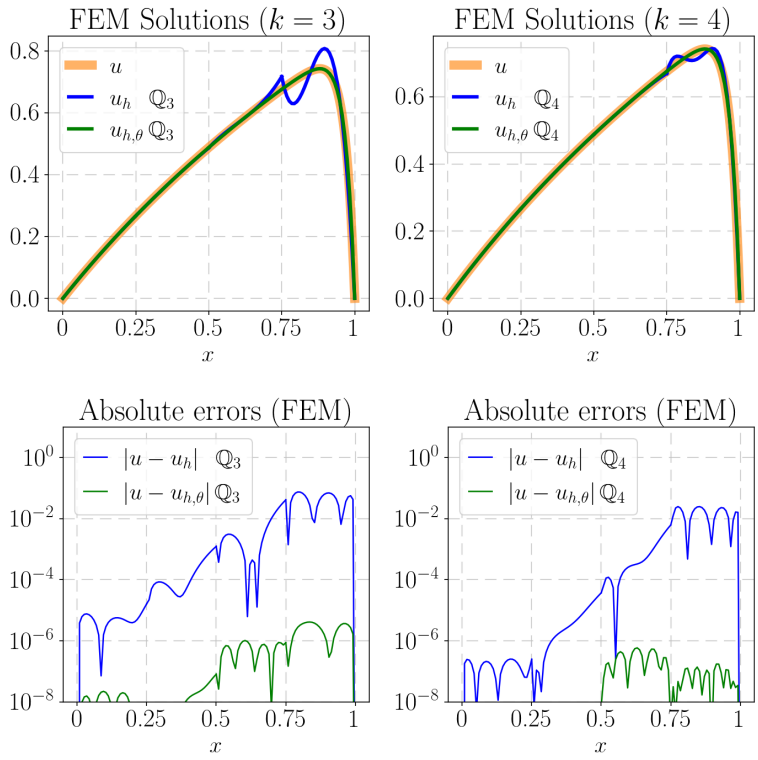


Figure 5.21.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k \in \{3, 4\}$, $N_{\text{cells}} = 4$ and the cellwise positive multiplier with bounded network weights. Top – Classical FEM solution u_h and learned-basis solution $u_{h,\theta}$ (best seed, see Remark 5.7) compared with the reference solution u. Bottom – Corresponding absolute errors. In each row, from left to right – $k = 3$ and $k = 4$.

Figure 5.21 shows that the enriched solution $u_{h,\theta}$ is indistinguishable from the reference solution u at the scale of the figure, which is consistent with the results of Tables 5.11 and 5.12. Its pointwise absolute errors are at most about $4 \cdot 10^{-6}$ ($k = 3$) and $3 \cdot 10^{-7}$ ($k = 4$), against about $7 \cdot 10^{-2}$ and $2 \cdot 10^{-2}$ for the classical solution.

Figure 5.22 is plotted for $\boldsymbol{\mu}^{(1)}$, $k = 3$ and $N_{\text{cells}} = 4$. At initialization, the multiplier is close to 1, but it is already not constant, with variations of about 10 % inside the cells. After training, it remains almost constant in the first three cells, but varies strongly in the last one, which is the one containing the boundary layer. In this last cell, the multiplier decreases down to the maximum of the solution ($x \approx 0.88$) before rising back up to 1 at $x = 1$. This dip shape may be surprising, since one would rather expect a multiplier that grows toward the boundary layer. It is related to the constraint $m_{c,\theta} = 1$ on the boundary of the cell K_c, which forces the multiplier to come back to 1 at both ends of the cell. The rise of $m_{c,\theta}$ on the right half of the cell accompanies the steep drop of the solution toward the boundary layer, whereas the decrease that precedes it

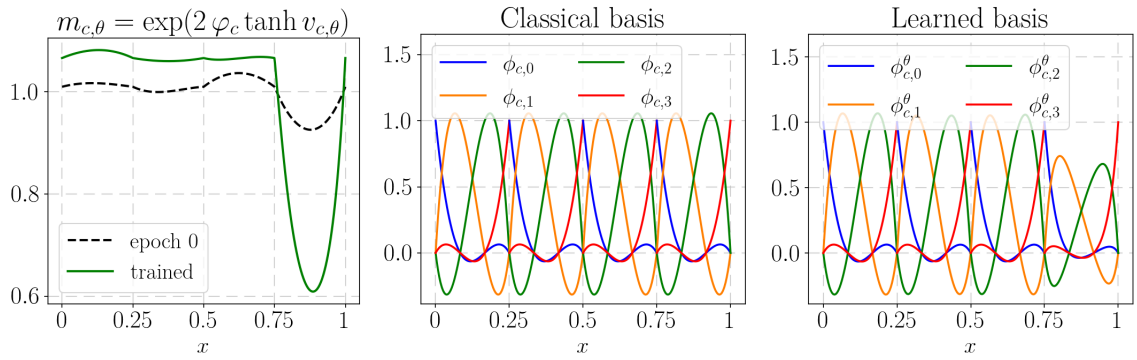


Figure 5.22.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N_{\text{cells}} = 4$ and the cellwise positive multiplier with bounded network weights. Left – Learned multiplier (L^2 normalized), at initialization and after training (best seed, see Remark 5.7). Middle – Classical Lagrange basis functions. Right – Resulting enriched basis functions.

appears to be a consequence of the value 1 imposed on the left boundary.

In the first three cells, the multiplier being almost constant, the enriched space is almost the classical finite element space, and the enriched basis functions roughly keep the shape of the Lagrange functions. Only the last cell is modified, with in particular a reduced amplitude of the functions associated with the interior nodes of the cell. The functions associated with the nodes located on the boundary of the cells, on the other hand, keep the value 1 at their node, which is ensured by φ_c. As in the patchwise case, the learned basis thus concentrates its approximation capacity in the cell of the boundary layer.

5.2.3. Discontinuous Galerkin – Patchwise basis

In this section, we consider the discontinuous Galerkin method (Section 2.2.2) with the SIPG flux, whose local Lagrange basis $\phi_{c,i}$ is enriched by a single neural network v_θ for the whole domain (patchwise basis), by means of a multiplier m_θ, as defined in (5.6). The objective is to see whether, as for continuous finite elements (Section 5.2.1), the error can be reduced by several orders of magnitude by the learned basis. The interfaces being treated weakly here by the SIPG flux, no continuity constraint weighs on the enriched basis, and the multiplier of the FEM patchwise case can be used as is, without any particular precaution on the boundary of the cells. The SIPG penalty is $\kappa = \eta_F / h_F$ with $\eta_F = 10\,k(k+1)$, a larger value than in Section 5.1, chosen empirically for the advection-diffusion problem.

We again place ourselves in the non-parametric setting, that is, the network is trained with the PINN loss (5.5) for a fixed value of $\boldsymbol{\mu}$, and we limit ourselves, as in the FEM cellwise case, to the study of the gain brought by the learned basis, analogous to the one carried out in the FEM patchwise case with the retained multiplier (Section 5.2.1.5), namely the positive multiplier with bounded weights resulting from the study of the choice of the parametrization (Section 5.2.1.3).

The multiplier is thus the one defined in (5.9), $m_\theta = \exp\big(2\tanh v_\theta\big)$, where v_θ is a single MLP shared by all the cells, with tanh activation and 2 hidden layers of width 16 (i.e. $N_\theta = 321$ weights), as in the FEM patchwise case. Its weights are bounded with the softclip operator (5.10), applied to each layer, the bound equaling four times the norm of its weights at initialization. At initialization, the multiplier is almost constant (Figure 5.24), which leaves the enriched space close to the classical DG space, a constant multiplier not modifying it. The training, the seeds (Remark 5.7) and the evaluation of the error with respect to the reference solution (Remark 5.4), compared with that of the classical DG method at the same number of degrees of freedom (Remark 5.1), are those of the FEM patchwise case (Section 5.2.1.5).

Gains achieved with the learned basis. For the three values of $\boldsymbol{\mu}$ defined in (5.4), we compare the L^2 errors between the reference solution u and the standard DG solution u_h, as well as the enriched solution $u_{h,\theta}$, for two polynomial degrees $k = 3$ and $k = 4$ and $N_{\text{cells}} \in \{2, 4, 8\}$ (Tables 5.13 and 5.14).

			classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	N_{cells}	N_{dofs}	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	8	$1.76 \cdot 10^{-1}$	$1.12 \cdot 10^{-3}$	157	$1.53 \cdot 10^{-4}$	18
	4	16	$4.72 \cdot 10^{-2}$	$7.14 \cdot 10^{-6}$	6 608	$1.33 \cdot 10^{-7}$	18
	8	32	$8.13 \cdot 10^{-3}$	$9.29 \cdot 10^{-8}$	87 486	$2.06 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(2)}$	2	8	$2.34 \cdot 10^{-1}$	–	–	–	0
	4	16	$6.55 \cdot 10^{-2}$	$3.43 \cdot 10^{-5}$	1 910	$4.64 \cdot 10^{-6}$	17
	8	32	$1.34 \cdot 10^{-2}$	$2.01 \cdot 10^{-7}$	66 981	$4.5 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(3)}$	2	8	$3.02 \cdot 10^{-1}$	–	–	–	0
	4	16	$8.25 \cdot 10^{-2}$	$1.44 \cdot 10^{-4}$	574	$1.42 \cdot 10^{-5}$	18
	8	32	$1.95 \cdot 10^{-2}$	$7.38 \cdot 10^{-7}$	26 435	$3.32 \cdot 10^{-8}$	16

Table 5.13.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 3$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the patchwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical DG SIPG method. Learned basis (best seed) – L^2 relative error, gain with respect to the classical DG SIPG method and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

Tables 5.13 and 5.14 show that the learned basis, associated with the best seed, reduces the error by several orders of magnitude wherever at least one seed goes below the threshold, the gain of the best seed ranging from 157 to about 87 000. These results are very close to those of the FEM patchwise case with the same multiplier and the same network (Tables 5.9 and 5.10). The number of degrees of freedom is not the same for equal k and N_{cells}, but the classical errors remain close. The gains (associated with the best seeds) are of the same order, without either of the two systematically prevailing. The configurations without any seed below the threshold are the same, the number of seeds below the threshold is of the same order, and the trends as a function of $\boldsymbol{\mu}$, k and N_{cells} are the same, including the decrease of the gain between $N_{\text{cells}} = 4$ and $N_{\text{cells}} = 8$

for $k = 4$. Treating the interfaces weakly, without continuity constraint on the basis, thus changes neither the attainable accuracy nor the robustness of the enrichment for this test case.

			classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	$N_{\mathbf{cells}}$	$N_{\mathbf{dofs}}$	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	10	$8.19 \cdot 10^{-2}$	$3.06 \cdot 10^{-5}$	2 675	$4.69 \cdot 10^{-7}$	19
	4	20	$1.59 \cdot 10^{-2}$	$3.79 \cdot 10^{-7}$	41 975	$2.55 \cdot 10^{-9}$	19
	8	40	$1.56 \cdot 10^{-3}$	$1.14 \cdot 10^{-7}$	13 599	$2.9 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(2)}$	2	10	$1.07 \cdot 10^{-1}$	$2.2 \cdot 10^{-4}$	487	$1.99 \cdot 10^{-5}$	19
	4	20	$2.56 \cdot 10^{-2}$	$6.67 \cdot 10^{-7}$	38 409	$2.55 \cdot 10^{-9}$	18
	8	40	$3.12 \cdot 10^{-3}$	$8.43 \cdot 10^{-8}$	37 026	$2.07 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(3)}$	2	10	$1.29 \cdot 10^{-1}$	$7.14 \cdot 10^{-4}$	180	$1.09 \cdot 10^{-4}$	14
	4	20	$3.59 \cdot 10^{-2}$	$3.58 \cdot 10^{-6}$	10 028	$5.2 \cdot 10^{-8}$	19
	8	40	$5.27 \cdot 10^{-3}$	$2.34 \cdot 10^{-7}$	22 553	$5.42 \cdot 10^{-9}$	19

Table 5.14.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 4$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the patchwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical DG SIPG method. Learned basis (best seed) – L^2 relative error, gain with respect to the classical DG SIPG method and best loss of the best seed (see Remark 5.7). / 20 – number of seeds below the loss threshold.

Comparison of the classical and learned bases. We now focus on $\boldsymbol{\mu}^{(1)}$ and $N_{\text{cells}} = 4$. We compare the solutions and the pointwise errors of the classical and learned bases (Figure 5.23), before looking at the learned multiplier m_θ and the enriched basis functions (Figure 5.24).

Figure 5.23 shows that the enriched solution $u_{h,\theta}$ is indistinguishable from the reference solution u at the scale of the figure, which is consistent with the results of Tables 5.13 and 5.14. Its pointwise absolute errors are at most about 10^{-5} ($k = 3$) and $4 \cdot 10^{-7}$ ($k = 4$), against about $7 \cdot 10^{-2}$ and $2.5 \cdot 10^{-2}$ for the classical solution. The DG solution not being continuous, one can also look at its jump at the interfaces, whereas the reference solution is continuous. At the interface $x = 0.75$, where it is the largest for the classical solution, it is computed as the absolute difference between the values taken from the two neighboring cells and is about $7 \cdot 10^{-3}$ ($k = 3$) and $2 \cdot 10^{-3}$ ($k = 4$) for the classical solution, against $4 \cdot 10^{-7}$ and $7 \cdot 10^{-9}$ for the enriched solution.

Figure 5.24 is plotted for $\boldsymbol{\mu}^{(1)}$, $k = 3$ and $N_{\text{cells}} = 4$. At initialization, the multiplier is almost constant, with variations of about 3 % over the whole domain. After training, it is almost constant on the first three cells, but grows strongly near $x = 1$, that is, where the boundary layer is located, by a factor of about 6 between the plateau and the right boundary. It has the same overall shape as the one obtained in the FEM patchwise case, even though the latter was obtained with the affine multiplier (Figure 5.4), and the same observations hold for the enriched basis functions.

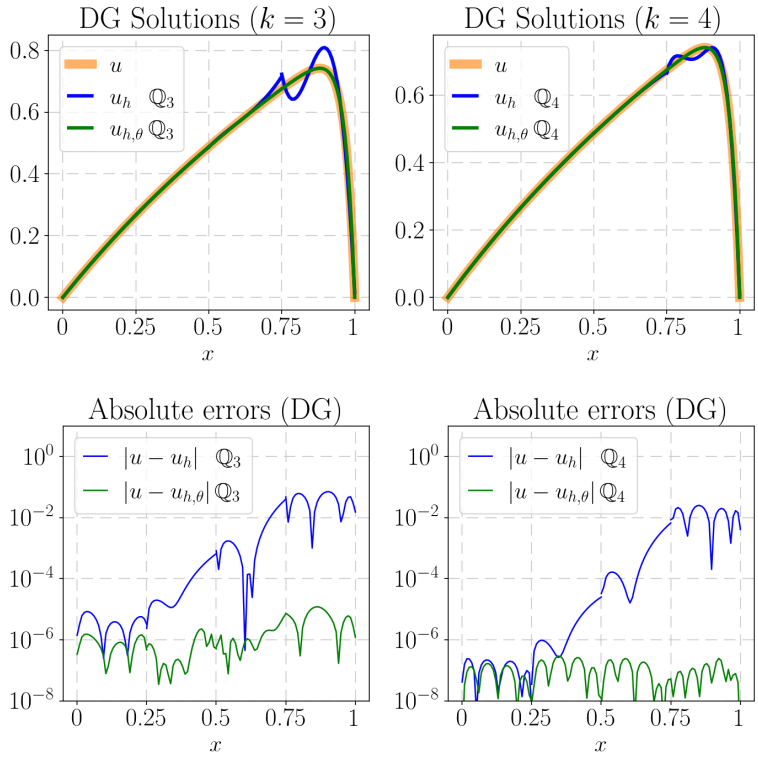


Figure 5.23.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k \in \{3, 4\}$, $N_{\text{cells}} = 4$ and the patchwise positive multiplier with bounded network weights. Top – Classical DG SIPG solution u_h and learned-basis solution $u_{h,\theta}$ (best seed, see Remark 5.7) compared with the reference solution u. Bottom – Corresponding absolute errors. In each row, from left to right – $k = 3$ and $k = 4$.

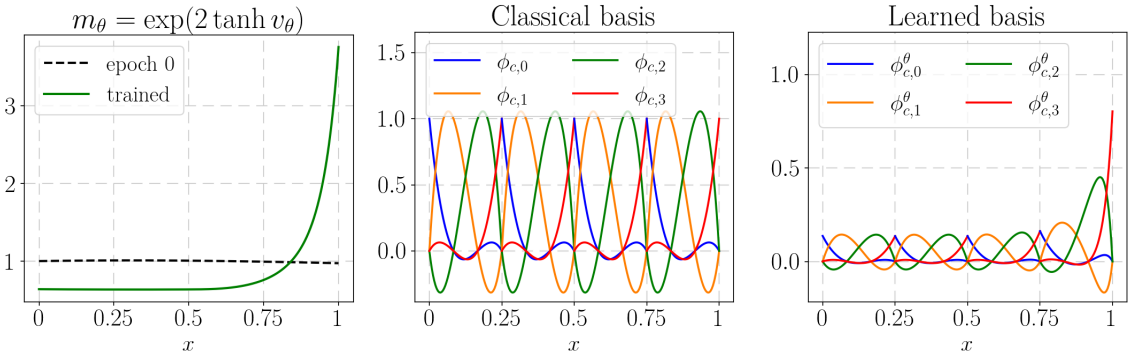


Figure 5.24.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N_{\text{cells}} = 4$ and the patchwise positive multiplier with bounded network weights. Left – Learned multiplier (L^2 normalized), at initialization and after training (best seed, see Remark 5.7). Middle – Classical Lagrange basis functions. Right – Resulting enriched basis functions.

5.2.4. Discontinuous Galerkin – Cellwise basis

In this section, we consider the discontinuous Galerkin method (Section 2.2.2) with the SIPG flux, whose local Lagrange basis $\phi_{c,i}$ is enriched by a neural network $v_{c,\theta}$ specific to each cell K_c (cellwise basis), by means of a multiplier $m_{c,\theta}$, as defined in (5.7). The objective is to compare this basis with the patchwise basis of the previous section (Section 5.2.3), with the same multiplier, and to see whether the additional local freedom brings a gain, at the price of a number of weights N_θ that grows with the number of cells. The SIPG penalty is the same as in the patchwise case (Section 5.2.3).

The interfaces being treated weakly by the SIPG flux, the continuity of the basis does not have to be imposed, unlike in the FEM cellwise case (Section 5.2.2). The local level-set φ_c is thus no longer necessary, and the multiplier of the cell K_c is simply the one of the patchwise case applied to the network of the cell,

$$m_{c,\theta}(x) = \exp\Big(2\tanh\big(v_{c,\theta}(x)\big)\Big), \qquad x \in K_c,$$

where $v_{c,\theta}$ is the MLP with scalar output specific to the cell K_c, which takes the physical coordinate x as input. Nothing here forces $m_{c,\theta}$ to equal 1 on the boundary of the cell, nor to be continuous from one cell to another. The networks are built identically on each cell, as in the FEM cellwise case, with a tanh activation and 2 hidden layers of width 6 (i.e. 61 weights per cell, and thus $N_\theta = 61\,N_{\text{cells}}$ in total).

As in the FEM cellwise case, we want the training to start from the classical basis. The networks being drawn independently, the multipliers of two neighboring cells have different mean values at initialization (Figure 5.26). But multiplying the basis of a cell by a constant does not modify the DG space, whose degrees of freedom are not shared between cells. Only the variation of the multiplier inside each cell matters, and it is small at initialization. It is thus not necessary to reduce the amplitude of the last layer of the networks, as is done in FEM cellwise.

The weights of the networks are bounded with the same softclip operator as in the previous cases (5.10), applied to each layer of each of the N_{cells} networks, the bound equaling four times the norm of the initial draw of its weights. The training, the seeds (Remark 5.7) and the evaluation of the error with respect to the reference solution (Remark 5.4), compared with that of the classical DG method at the same number of degrees of freedom (Remark 5.1), are those of the FEM patchwise case (Section 5.2.1.5). The number of weights N_θ, which grows with N_{cells}, is not taken into account in this comparison.

Gains achieved with the learned basis. For the three values of $\boldsymbol{\mu}$ defined in (5.4), we compare the L^2 errors between the reference solution u and the standard DG solution u_h, as well as the enriched solution $u_{h,\theta}$, for two polynomial degrees $k = 3$ and $k = 4$ and $N_{\text{cells}} \in \{2, 4, 8\}$ (Tables 5.15 and 5.16).

Tables 5.15 and 5.16 show that the learned basis, associated with the best seed, reduces the error by several orders of magnitude wherever at least one seed goes below the threshold, the gain of the best seed ranging from 144 to about 88 000. These results are very close to those of the patchwise case (Tables 5.13 and 5.14), with the same multiplier, whereas the number of weights, which grows with N_{cells}, is, depending on

the case, lower or higher than that of the patchwise case. We also find the same trends as in the latter.

				classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	$N_{\mathbf{cells}}$	$N_{\mathbf{dofs}}$	N_θ	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	8	122	$1.76 \cdot 10^{-1}$	$1.23 \cdot 10^{-3}$	144	$1.36 \cdot 10^{-4}$	14
	4	16	244	$4.72 \cdot 10^{-2}$	$3.74 \cdot 10^{-6}$	12 626	$2.69 \cdot 10^{-8}$	16
	8	32	488	$8.13 \cdot 10^{-3}$	$1.15 \cdot 10^{-7}$	70 659	$2.27 \cdot 10^{-9}$	19
$\boldsymbol{\mu}^{(2)}$	2	8	122	$2.34 \cdot 10^{-1}$	–	–	–	0
	4	16	244	$6.55 \cdot 10^{-2}$	$2.03 \cdot 10^{-5}$	3 225	$3.76 \cdot 10^{-7}$	10
	8	32	488	$1.34 \cdot 10^{-2}$	$1.52 \cdot 10^{-7}$	88 299	$2.29 \cdot 10^{-9}$	15
$\boldsymbol{\mu}^{(3)}$	2	8	122	$3.02 \cdot 10^{-1}$	–	–	–	0
	4	16	244	$8.25 \cdot 10^{-2}$	$9.07 \cdot 10^{-5}$	910	$5.14 \cdot 10^{-6}$	6
	8	32	488	$1.95 \cdot 10^{-2}$	$3.21 \cdot 10^{-7}$	60 823	$9.85 \cdot 10^{-9}$	15

Table 5.15.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 3$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the cellwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical DG SIPG method. Learned basis (best seed) – L^2 relative error, gain with respect to the classical DG SIPG method and best loss of the best seed (see Remark 5.7). N_θ – total number of weights (61 per cell). / 20 – number of seeds below the loss threshold.

				classical	**learned basis (best seed)**			
$\boldsymbol{\mu}$	$N_{\mathbf{cells}}$	$N_{\mathbf{dofs}}$	N_θ	**error**	**error**	**gain**	**loss**	**/ 20**
$\boldsymbol{\mu}^{(1)}$	2	10	122	$8.19 \cdot 10^{-2}$	$3.91 \cdot 10^{-5}$	2 097	$4.03 \cdot 10^{-7}$	17
	4	20	244	$1.59 \cdot 10^{-2}$	$3.84 \cdot 10^{-7}$	41 453	$2.78 \cdot 10^{-9}$	18
	8	40	488	$1.56 \cdot 10^{-3}$	$4.22 \cdot 10^{-8}$	36 840	$1.44 \cdot 10^{-9}$	20
$\boldsymbol{\mu}^{(2)}$	2	10	122	$1.07 \cdot 10^{-1}$	$2.07 \cdot 10^{-4}$	516	$1.07 \cdot 10^{-5}$	10
	4	20	244	$2.56 \cdot 10^{-2}$	$4.28 \cdot 10^{-7}$	59 937	$5.26 \cdot 10^{-9}$	16
	8	40	488	$3.12 \cdot 10^{-3}$	$2.1 \cdot 10^{-7}$	14 888	$4.92 \cdot 10^{-9}$	18
$\boldsymbol{\mu}^{(3)}$	2	10	122	$1.29 \cdot 10^{-1}$	$7.5 \cdot 10^{-4}$	171	$9.73 \cdot 10^{-5}$	8
	4	20	244	$3.59 \cdot 10^{-2}$	$1.7 \cdot 10^{-6}$	21 074	$6.42 \cdot 10^{-8}$	13
	8	40	488	$5.27 \cdot 10^{-3}$	$5.77 \cdot 10^{-7}$	9 125	$1.91 \cdot 10^{-8}$	18

Table 5.16.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu} \in \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)}\}$, $k = 4$, $N_{\text{cells}} \in \{2, 4, 8\}$ and the cellwise positive multiplier with bounded network weights. Classical – L^2 relative error of the classical DG SIPG method. Learned basis (best seed) – L^2 relative error, gain with respect to the classical DG SIPG method and best loss of the best seed (see Remark 5.7). N_θ – total number of weights (61 per cell). / 20 – number of seeds below the loss threshold.

The main difference with the patchwise case is robustness. The number of seeds that go below the threshold is lower on coarse meshes and for large values of $\boldsymbol{\mu}$ (for example 6 seeds out of 20 for $k = 3$, $N_{\text{cells}} = 4$ and $\boldsymbol{\mu}^{(3)}$), whereas for $N_{\text{cells}} = 8$, at least 15 seeds out of 20 go below the threshold, which is comparable to the patchwise case. For the DG method, the cellwise basis thus brings no gain in accuracy with respect to the patchwise basis, and it is less robust, for a higher number of weights. This can be interpreted, without it having been tested, by the fact that the boundary layer only occupies the last cells, so that the additional freedom of the networks of the other cells is of little use, whereas the capacity available in the cell of the boundary layer remains that of a single network of width 6 with bounded weights. The latter might not suffice to produce a strong enough variation of the multiplier in a large cell, which would be consistent with the fact that the seeds below the threshold are less numerous for $N_{\text{cells}} \leq 4$ than for $N_{\text{cells}} = 8$.

Comparison of the classical and learned bases. We now focus on $\boldsymbol{\mu}^{(1)}$ and $N_{\text{cells}} = 4$. We compare the solutions and the pointwise errors of the classical and learned bases (Figure 5.25), before looking at the learned multiplier $m_{c,\theta}$ and the enriched basis functions (Figure 5.26).

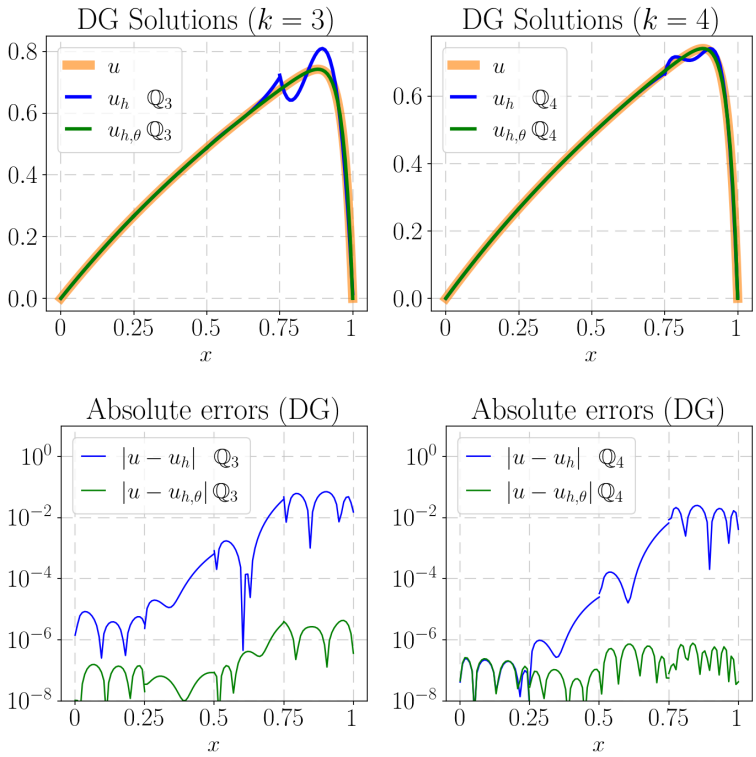


Figure 5.25.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k \in \{3, 4\}$, $N_{\text{cells}} = 4$ and the cellwise positive multiplier with bounded network weights. Top – Classical DG SIPG solution u_h and learned-basis solution $u_{h,\theta}$ (best seed, see Remark 5.7) compared with the reference solution u. Bottom – Corresponding absolute errors. In each row, from left to right – $k = 3$ and $k = 4$.

Figure 5.25 shows that the enriched solution $u_{h,\theta}$ is indistinguishable from the reference solution u at the scale of the figure, which is consistent with the results of Tables 5.15 and 5.16. Its pointwise absolute errors are at most about $5 \cdot 10^{-6}$ ($k = 3$) and $7 \cdot 10^{-7}$ ($k = 4$), against about $7 \cdot 10^{-2}$ and $2.5 \cdot 10^{-2}$ for the classical solution. At the interface $x = 0.75$, where it is the largest for the classical solution, the jump of the solution, computed as the absolute difference between the values taken from the two neighboring cells, is about $7 \cdot 10^{-3}$ ($k = 3$) and $2 \cdot 10^{-3}$ ($k = 4$) for the classical solution, against $1.4 \cdot 10^{-7}$ and $3 \cdot 10^{-9}$ for the enriched solution.

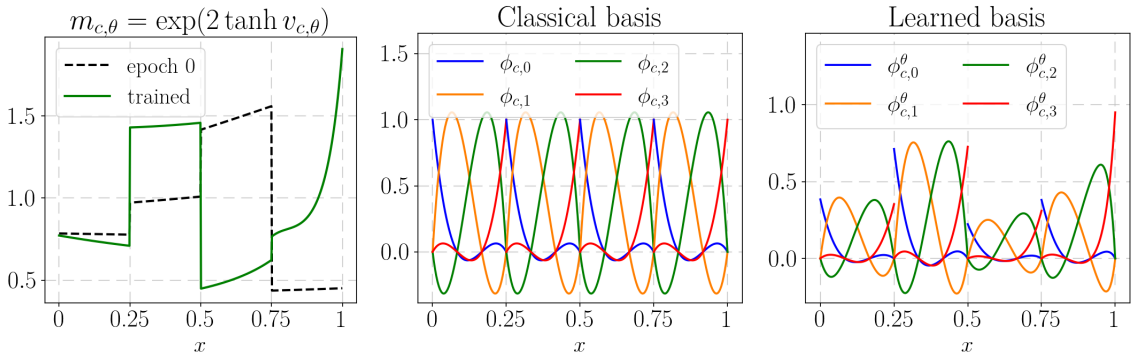


Figure 5.26.: Considering the *1D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 3$, $N_{\text{cells}} = 4$ and the cellwise positive multiplier with bounded network weights. Left – Learned multiplier (L^2 normalized), at initialization and after training (best seed, see Remark 5.7). Middle – Classical Lagrange basis functions. Right – Resulting enriched basis functions.

Figure 5.26 is plotted for $\boldsymbol{\mu}^{(1)}$, $k = 3$ and $N_{\text{cells}} = 4$. At initialization, the mean value of the multiplier differs from one cell to another, but the multiplier is almost constant inside each of them, with variations of at most about 10 %. After training, it remains discontinuous at the interfaces, its mean value still differing from one cell to another. The DG method allows this, and it has no effect on the enriched space, multiplying the basis of a cell by a constant not modifying it. Only the variation inside each cell matters. It is small in the first two cells (about 9 % and 2 %), more marked in the third (40 % or so), and strong in the last one, where the multiplier grows by a factor of about 2.5 as it approaches $x = 1$, that is, the boundary layer. Unlike the FEM cellwise case (Figure 5.22), where the multiplier decreases before rising back to 1 on the last cell, it grows here over the whole cell, since it is no longer constrained to equal 1 on the boundary of the cell.

In the first two cells, the multiplier being almost constant, the enriched basis functions roughly keep the shape of the Lagrange functions, up to a factor per cell, and the enriched space there is almost the classical DG space. The third cell is slightly deformed and the last one is strongly deformed, with functions amplified as they approach $x = 1$. As in the patchwise case, the learned basis thus concentrates its approximation capacity in the boundary layer.

5.2.5. Comparison of the schemes, conclusion and limitations

The four variants studied (FEM or DG, patchwise or cellwise basis) give results of the same order on this test case. In each of them, the learned basis reduces the L^2 error by several orders of magnitude at a fixed number of degrees of freedom, with gains, for the best seed, of the order of 10^2 to 10^5. The trends are also the same from one variant to the other. The gain tends to decrease as $\boldsymbol{\mu}$ increases, the boundary layer being steeper, and the case $N_{\text{cells}} = 2$ is the most difficult, several configurations then having no seed below the threshold.

Patchwise versus cellwise basis. The cellwise basis does not bring a systematic gain in accuracy compared with the patchwise basis. It is slightly less robust on coarse meshes for FEM (Section 5.2.2), and clearly less so for DG (Section 5.2.4), with a number of weights that grows with N_{cells} and a heavier construction. This result should nevertheless be put into perspective. In our studies, the same network architecture is used in each cell, so that the budget is equal from one cell to another, whereas the solution only has structure to capture in the cells of the boundary layer. Putting more freedom in the cells concerned and less in the others could give better results. There are probably cases where the cellwise basis is more interesting than the patchwise one, for example when the solution exhibits local structures in several places, but studies of this kind are not carried out here.

FEM versus DG. In the patchwise case, the two methods give very close results, with gains of the same order, the same configurations with no seed below the threshold and the same trends. Treating the interfaces weakly, without any continuity constraint on the basis, therefore does not seem to change either the attainable accuracy or the robustness of the enrichment for this test case. In the cellwise case, the gains are also of the same order, but the DG version has fewer seeds below the threshold than the FEM version in most configurations. This may be explained by the fact that the two cellwise constructions differ in several respects, in particular the use of local level-sets to impose continuity between cells and the reduction of the last layer of the networks in FEM, so that this gap cannot be attributed to the scheme alone. Moreover, three structural differences should be kept in mind. For equal k and N_{cells}, DG has more degrees of freedom than FEM ($N_{\text{cells}}(k+1)$ versus $kN_{\text{cells}}+1$), which leads us to compare the gains rather than N_{dofs}. The Dirichlet condition is imposed exactly in FEM and only weakly in DG, so that the boundary term $\mathcal{J}_b$ of the loss is not necessarily zero in DG. Finally, in cellwise FEM, the multiplier is constrained to equal 1 on the cell boundary to preserve continuity, whereas no such constraint applies in DG. This study, carried out in dimension 1 where the continuity constraint is not very restrictive, does not make it possible to say whether these differences would matter more in other cases.

Limitations. This study has several limitations. First, the learned basis is only valid for one value of $\boldsymbol{\mu}$, and its construction requires a training for each triplet $(\boldsymbol{\mu}, k, N_{\text{cells}})$, which motivates the parametric framework of the following sections. Next, the training depends on the initialization, since the result varies from one seed to another and not all the trainings reach the loss threshold. As each configuration is trained with

only 20 seeds, and as the loss is moreover only an imperfect indicator of the error, the reported numbers of seeds are only estimates. Moreover, the best seed is selected with the error with respect to the reference solution, which is not available in practice, so that the gains reported for the best seed are those of the best of the 20 trainings and not of a training chosen without reference. The median errors of the budget study, computed over the seeds below the loss threshold, a criterion that does not use the reference, are nevertheless also several orders of magnitude below the classical error (Figure 5.16). Finally, these conclusions come from a single problem in dimension 1, and the comparison with the classical methods is made at an equal number of degrees of freedom and not at an equal computational cost, the cost of training not being taken into account. Moreover, the comparison is made with uniform meshes, whereas, for a boundary layer, the natural classical competitor at equal N_{dofs} is a graded mesh, an adaptively refined mesh [3] or an *hp* refinement (see e.g. [17]). The gains reported here are therefore upper bounds of the benefit over the best classical method.

Computational cost. The comparisons above are made at an equal number of degrees of freedom and leave aside the cost of training, which cannot be ignored. Concretely, the learned basis replaces a single resolution of the classical scheme on a fine mesh by a training on a coarser mesh. Each epoch requires solving the scheme for the current weights, then computing the gradient by the adjoint method, which amounts to one more linear solve with the transposed Jacobian, followed by vector-Jacobian products on the loss and on the assembly (Algorithm 5.1). For the linear problems considered here, an epoch thus costs about two solves of the coarse scheme with an optimizer based on this gradient, so that the training costs about $2N_{\text{epochs}}$ of them. The ENG optimizer is more expensive (Remark 5.5). It replaces the adjoint solve by the Jacobian of the degrees of freedom with respect to θ, forms and inverts at each epoch a regularized Gram matrix of size $N_\theta \times N_\theta$, at a cost that depends on the size of the network and not on that of the mesh, and solves the scheme for several trial steps in its line search.

The enrichment is therefore advantageous in cost only if this training is cheaper than a single classical resolution reaching the same error, which can happen only for large problems, where the matrices to be handled are very large. This is not the case in dimension 1, where the classical resolution is almost instantaneous and the learned basis is always more expensive, so that this study cannot settle the question. This is why we turn to dimension 3, where the cost of resolving a boundary layer with classical methods is amplified, the number of degrees of freedom of a uniform mesh growing as h^{-3} (see also Section 5.1).

Having shown, in dimension 1, that the learned basis reduces the L^2 error by several orders of magnitude at a fixed number of degrees of freedom, but without being able to conclude on the computational cost, which can only become favorable on large problems, we now turn to a 3D advection-diffusion problem with a manufactured solution, where the cost of resolving a boundary layer with classical methods is amplified by the dimension (Section 5.1). We still consider a fixed value of the parameter $\boldsymbol{\mu}$, and consider continuous finite elements with a patchwise basis.

5.3. Non-parametric enrichment (PINN loss) – 3D test case

This section extends the non-parametric study of Section 5.2 to dimension $d = 3$. The objective is to check whether the multiplicative enrichment of a Lagrange basis by a network, learned through the solver, still reduces the error in dimension 3, where the cost of resolving a boundary layer with classical methods is amplified, the number of degrees of freedom of a uniform mesh growing as h^{-3} (Section 5.1). We restrict ourselves to continuous finite elements with a patchwise basis whose multiplier is positive and has bounded weights (Section 5.2.1), and we consider a single value of the parameter $\boldsymbol{\mu}$. Here, we are interested both in the error obtained for a fixed number of degrees of freedom and in the computational cost for a given error.

We consider problem (2.1) in its advection-diffusion form on the unit cube $\Omega = (0,1)^3$,

$$\begin{cases} -\dfrac{1}{\mathrm{Pe}}\,\Delta u + \boldsymbol{C}(\boldsymbol{x},\boldsymbol{\mu})\cdot\nabla u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases} \tag{5.13}$$

with $\mathrm{Pe} = 20$, which corresponds to the operator (2.2) for $R = 0$ and $D = \mathrm{Id}$. The transport field $\boldsymbol{C}$ is the affine field of the 1D study, carried by the main diagonal of the cube:

$$\boldsymbol{C}(\boldsymbol{x},\boldsymbol{\mu}) = \big(\mu_1 + \mu_2\,\boldsymbol{x}\cdot\boldsymbol{e}_{\mathrm{diag}}\big)\,\boldsymbol{e}_{\mathrm{diag}}, \qquad \boldsymbol{e}_{\mathrm{diag}} := \frac{1}{\sqrt{3}}(1,1,1)^\top.$$

The unit vector $\boldsymbol{e}_{\mathrm{diag}}$ gives the direction of the transport, and $\boldsymbol{x}\cdot\boldsymbol{e}_{\mathrm{diag}}$ is the coordinate of the point $\boldsymbol{x}$ along the diagonal, of length $\sqrt{3}$. The parameter is set to $\boldsymbol{\mu}^{(1)} = (0.8, 0.8)$. Since the transport is directed towards the corner $(1,1,1)$, the solution considered below exhibits a boundary layer along the three faces $x_i = 1$, which is steepest near this corner.

Unlike in the 1D case, we do not compute a reference solution obtained by finite elements on a very fine mesh, since this would be too costly in dimension 3. We therefore build a manufactured solution, defined by:

$$u(\boldsymbol{x},\boldsymbol{\mu}) = \prod_{i=1}^{3} x_i\Big(1 - \exp\Big(-\frac{\mathrm{Pe}\,(\mu_1+\mu_2)}{\sqrt{3}}\,(1-x_i)\Big)\Big),$$

vanishing on the 6 faces of the cube. The source term f is computed from the exact solution and the transport field $\boldsymbol{C}$, according to (5.13). Figure 5.27 shows the exact solution on three planes $x_i = 0.84$, close to the maximum of u along the diagonal.

We use a discretization by continuous $\mathbb{Q}_2$ finite elements on a Cartesian mesh of $\boldsymbol{N}_{\mathrm{cells}} = (N_1, N_2, N_3)$ cells, with $N_1 = N_2 = N_3 = 4$, that is $N_{\mathrm{cells}} = 64$ cells in total, which gives $N_{\mathrm{dofs}} = 729$ degrees of freedom. The local Lagrange basis $\phi_{c,i}$ is enriched by a single network v_θ defined on the whole domain (patchwise basis), according to (5.6). The multiplier defined by:

$$m_\theta(\boldsymbol{x}) = \exp\big(6\tanh(v_\theta(\boldsymbol{x}))\big), \tag{5.14}$$

is similar to the positive multiplier of (5.9) (with a factor 6 instead of 2), and $v_\theta : \mathbb{R}^3 \to \mathbb{R}$ is an MLP with tanh activation and 3 hidden layers of width 10 (that is, $N_\theta = 271$

weights). As in 1D, the weights of each layer are bounded by the soft clipping of (5.10), with a bound of 16 instead of 4.

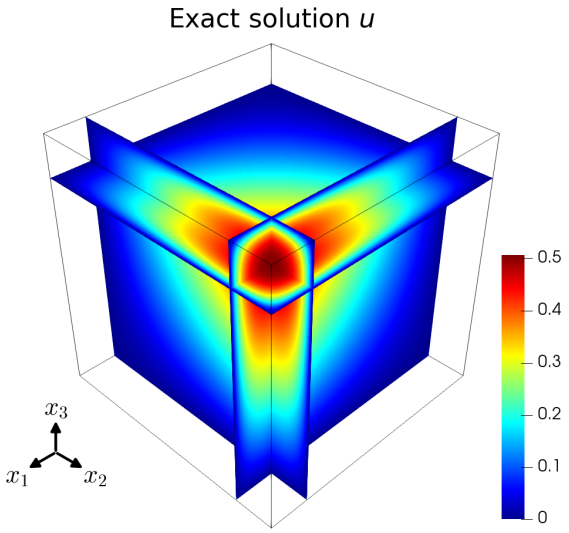


Figure 5.27.: Considering the *3D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$. Exact solution u on the three planes $x_1 = 0.84$, $x_2 = 0.84$ and $x_3 = 0.84$.

Remark 5.12. *The two settings that differ from the 1D multiplier and soft clipping were tuned during preliminary tests. By increasing the factor in front of the* tanh *in the multiplier (5.14), we increase the amplitude of the possible values of the multiplier (varying from e^{-6} to e^{6}, that is a ratio of about $e^{12} \approx 1.6 \cdot 10^5$ between its extreme values), which seems necessary for this 3D problem. Likewise, the bound on the weights was replaced by* 16 *instead of* 4*, which leaves the network the possibility of producing steeper variations, which also seems necessary to represent the thin boundary layers of this problem.*

The network is trained through the solver (Section 4.9) with the PINN loss of the 1D study (5.5), adapted to problem (5.13). Since the Dirichlet condition is imposed by the finite element scheme, the boundary term $\mathcal{J}_b$ vanishes and the loss reduces to the residual loss $\mathcal{J}_r$ given by:

$$\mathcal{J}(\theta) = \mathcal{J}_r(\theta) = \int_\Omega \left| -\frac{1}{\mathrm{Pe}}\,\Delta u_{h,\theta}(\boldsymbol{x}) + \boldsymbol{C}(\boldsymbol{x},\boldsymbol{\mu}) \cdot \nabla u_{h,\theta}(\boldsymbol{x}) - f(\boldsymbol{x}) \right|^2 \mathrm{d}\boldsymbol{x},$$

estimated by Monte Carlo on $N_{\mathrm{col}} = 6\,000$ collocation points drawn randomly in Ω and redrawn at each epoch, so that the 70 epochs sample the domain much more densely than a single draw, which limits the sampling error of the Monte Carlo estimate of the residual. The exact solution therefore never intervenes in the training.

The training follows the principle of Algorithm 5.1, with the ENG optimizer [98] of `ScimBa` (Remark 5.5), in its dense version as in 1D, that is, the Gram matrix, of size $N_\theta \times N_\theta = 271 \times 271$, is assembled explicitly. In the loss, the linear system of the scheme, of size $N_{\mathrm{dofs}} \times N_{\mathrm{dofs}} = 729 \times 729$, is solved by a direct method (LU factorization) because it is small enough for this. The regularization of the Gram matrix is initialized at 10^{-2}, then decreases geometrically over the first 30 epochs down to 10^{-6}, a value that it keeps until the end of the training. This warm-up is intended to stabilize the

first steps, when the Gram matrix is ill-conditioned, before letting the optimizer use steps closer to the natural gradient. The training lasts $N_{\text{epochs}} = 70$ epochs, the retained network being, as in 1D, the one with the lowest loss over the epochs. Outside the training, the final solve of the scheme with the learned basis, like all the classical FEM solves, uses the matrix-free BiCGStab method [128] (Section 4.6.2) with a tolerance of 10^{-10} and no preconditioner. For the classical FEM, an iterative method is necessary because the assembly of the matrix exceeds the available memory on fine meshes, and BiCGStab is suited to the non-symmetric system created by the advection term.

> **Remark 5.13.** *We perform* 3 *trainings, one per seed, each taking on the order of a minute, compilation included, on an AMD Instinct MI210 GPU (the 1D computations of Section 5.2.1 being run on an NVIDIA RTX 2000). We first report the three seeds, then focus on seed* 1*, which has both the lowest loss and the lowest error (see Figure 5.28 and Table 5.17).*

Training loss of the three seeds. Figure 5.28 represents the evolution of the loss of the three trainings (Remark 5.13) as a function of the number of epochs, which decreases by several orders of magnitude for each seed. Seed 1 is the one that reaches the best loss, of the order of $1.5 \cdot 10^{-7}$, which it reaches after about fifty epochs before stabilizing. For seeds 2 and 3, the loss on the contrary keeps decreasing up to the end of the 70 epochs, which suggests that a larger epoch budget would improve their results.

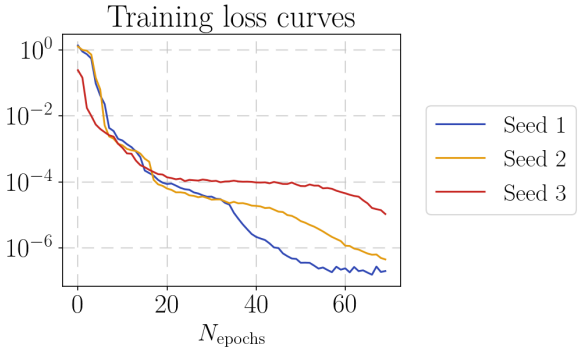


Figure 5.28.: Considering the *3D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 2$, $N_1 = N_2 = N_3 = 4$ and the positive multiplier with bounded weights. Training loss as a function of the epoch, for the three seeds.

Gains of the learned basis over the classical FEM. Figure 5.29 compares the L^2 relative error of the classical $\mathbb{Q}_2$ FEM with that of the three learned bases. The classical error is computed for meshes $N_1 = N_2 = N_3$ from 2 to 40, that is up to about $5 \cdot 10^5$ degrees of freedom, whereas the learned bases are all built on the mesh $N_1 = N_2 = N_3 = 4$, that is, for $N_{\text{dofs}} = 729$. This figure allows us to compare the two methods at a fixed number of degrees of freedom, then at equal error, by reading the number of degrees of freedom that the classical FEM needs to reach the error of each learned basis (horizontal dotted lines). When this error is lower than all the computed ones, the classical curve is extrapolated with its theoretical order of convergence (green

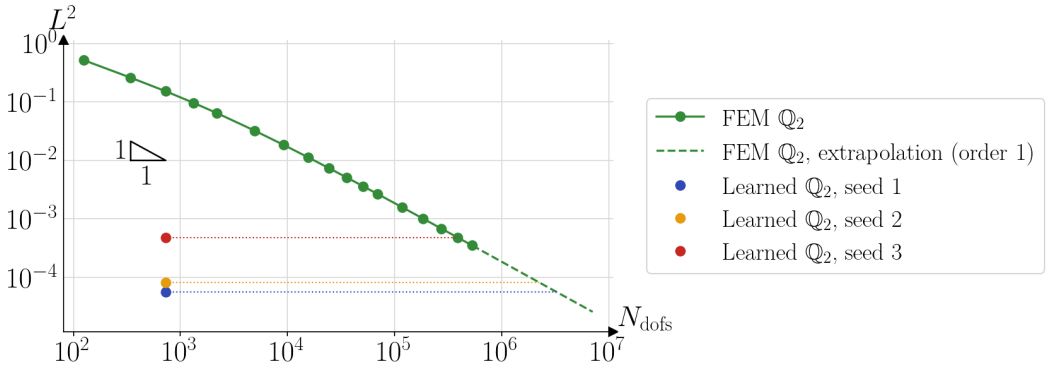


Figure 5.29.: Considering the *3D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 2$ and the positive multiplier with bounded weights. L^2 relative error as a function of N_{dofs} for the classical $\mathbb{Q}_2$ FEM on meshes with $N_1 = N_2 = N_3$ ranging from 2 to 40 (solid), its extrapolation with the theoretical order (dashed), and the three learned bases on the mesh with $N_1 = N_2 = N_3 = 4$. A dotted horizontal line joins each learned error to the number of degrees of freedom at which the classical error reaches it.

dashed line). These comparisons do not account for the training cost and the weights of the network (Remark 5.1), which we examine in a second step.

For a fixed $N_{\text{dofs}} = 729$, the error of the three seeds is below $5 \cdot 10^{-4}$, against about 0.15 for the classical FEM, that is a gain of two to three orders of magnitude. This high classical error is explained by a mesh that is too coarse to describe the corner layer, whose thickness is much smaller than the cell size. The error of the learned basis depends on the seed, and the order of the errors follows that of the losses (Figure 5.28).

On the finest meshes, the classical error approaches the theoretical slope 1 in N_{dofs} (order 3 in h for degree $k = 2$, the number of degrees of freedom varying as h^{-3}), which justifies this extrapolation. At equal error, the classical FEM would require of the order of $2 \cdot 10^6$ to $3 \cdot 10^6$ degrees of freedom for seeds 1 and 2, whose errors are lower than all the computed errors, this number being therefore an extrapolation. For seed 3, whose error is reached on a computed mesh, it is of the order of $4 \cdot 10^5$. The classical FEM thus requires from several hundred to several thousand times more degrees of freedom than the learned basis.

Computational cost at equal time. Table 5.17 compares, for each seed, the learned basis and the classical FEM at comparable total computation time (no extrapolation therefore intervenes in this table). For the learned basis, the retained time is the total duration of the construction of the coarse mesh, the training and the final solve. For the classical FEM, we retain, among the computed meshes, the one whose solve time is the closest, including the construction of the fine mesh and the final solve. The table gives the total times, which include the compilation, and the times without compilation. Both methods use the same solver (see above). The times of Table 5.17 depend on the solver and on the implementation, so that this comparison is meaningful within this setting (same library, same solvers, same GPU), other solvers, for instance preconditioned or direct, possibly giving different times for the classical FEM.

	time (s)				N_{dofs}			error		
	w/ comp.		w/o comp.							
seed	learn.	class.	learn.	class.	learn.	class.	ratio	learn.	class.	ratio
1	59	51	10.1	49.5	729	$5.31 \cdot 10^5$	729	$5.68 \cdot 10^{-5}$	$3.53 \cdot 10^{-4}$	6.2
2	41	37	6.8	26.6	729	$3.89 \cdot 10^5$	534	$8.13 \cdot 10^{-5}$	$4.82 \cdot 10^{-4}$	5.9
3	41	37	6.9	26.6	729	$3.89 \cdot 10^5$	534	$4.77 \cdot 10^{-4}$	$4.82 \cdot 10^{-4}$	1

Table 5.17.: Considering the *3D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 2$ and the positive multiplier with bounded weights. For each seed, the learned basis ($N_1 = N_2 = N_3 = 4$, $N_\theta = 271$ weights) and the computed classical $\mathbb{Q}_2$ FEM whose time is the closest to the total time of the learned basis, with their N_{dofs} and L^2 relative errors, and the ratio of the classical value to the learned one for N_{dofs} and for the error. Times are given with compilation (w/ comp.), that is the total times, and without compilation (w/o comp.).

At comparable time, seeds 1 and 2 have an error about six times lower than the classical FEM, whereas seed 3 has an equivalent error. For seed 1, the classical error in the table is the lowest that we could obtain, the mesh $N_1 = N_2 = N_3 = 40$ being the finest that the memory required to solve the classical scheme allows us to compute. The computation time of this classical solution is therefore slightly lower than that of the learned basis. The gain at comparable time is thus much more modest than the one obtained at fixed N_{dofs} (for the coarse mesh), and it depends strongly on the seed. On the other hand, the learned basis remains clearly advantaged in number of degrees of freedom, all seeds having several hundred times fewer than the classical FEM, even though this number does not count the $N_\theta = 271$ weights of the network. The time of the learned basis is moreover dominated by the compilation, since without it the learned basis takes 6.8 to 10.1 s depending on the seed, against 26.6 to 49.5 s for the classical FEM on the finest meshes, whose time is dominated by the solve. The learned basis is therefore much cheaper in execution than the finest classical meshes of the table, and the comparison at equal total time mostly compares compilation times with solve times.

Learned solution and multiplier for the best seed. We now turn to seed 1, the best seed. Figure 5.30 compares, on the left, the classical solution u_h, the enriched solution $u_{h,\theta}$ and the exact solution u along the diagonal, parametrized by the coordinate $\boldsymbol{x} \cdot \boldsymbol{e}_{\text{diag}} \in [0, \sqrt{3}]$, and in the middle their absolute errors. On the right, it represents the learned multiplier m_θ along the same diagonal, at initialization and after training, together with the bounds $e^{\pm 6}$ that it cannot exceed.

On the left and middle parts of Figure 5.30, the classical solution exhibits a marked overshoot around the maximum of u, whereas the enriched solution is indistinguishable from the exact solution at the scale of the figure. The absolute error of the enriched solution remains below 10^{-4} over the whole diagonal, against an error of the order of 10^{-1} near the maximum for the classical solution. The right part of Figure 5.30 shows that, at initialization, the multiplier varies little. Since a constant multiplier does not change the space spanned by the basis, the starting point is close to the classical basis.

After training, it increases monotonically along the diagonal, from about $6 \cdot 10^{-3}$ near the origin to about 30 near the corner $(1, 1, 1)$, that is a ratio of the order of $5 \cdot 10^3$, and it exceeds what the factor 2 of (5.9) would allow (Remark 5.12). The multiplier remains within its bounds, its minimum approaching the lower bound whereas its maximum remains far from the upper bound.

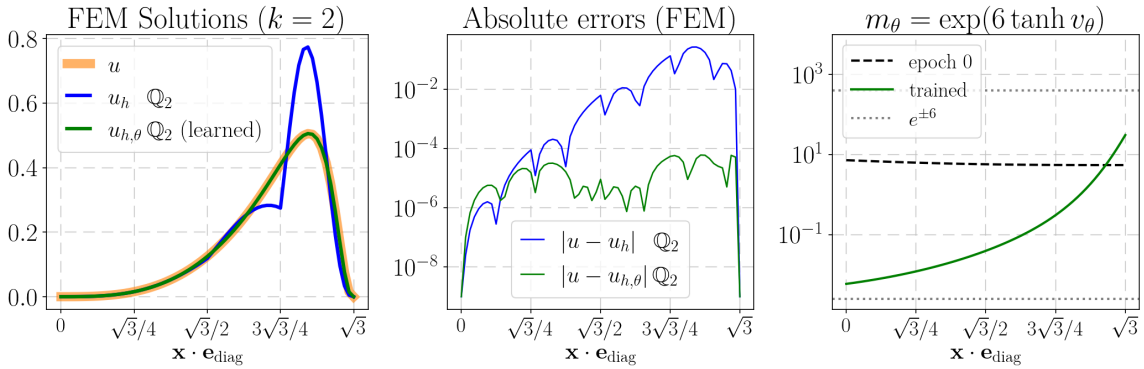


Figure 5.30.: Considering the *3D advection-diffusion problem* with $\boldsymbol{\mu}^{(1)}$, $k = 2$, $N_1 = N_2 = N_3 = 4$ and the positive multiplier with bounded weights (seed 1), along the diagonal, as a function of $\boldsymbol{x} \cdot \boldsymbol{e}_{\text{diag}}$. Left – Classical FEM solution u_h and learned-basis solution $u_{h,\theta}$ compared with the exact solution u. Middle – Corresponding absolute errors. Right – Learned multiplier m_θ at initialization and after training, with the bounds $e^{\pm 6}$ of the multiplier.

Conclusion and limitations. On this 3D advection-diffusion problem, the $\mathbb{Q}_2$ basis enriched by a single network of $N_\theta = 271$ weights reduces the L^2 error of the classical FEM by two to three orders of magnitude for a fixed $N_{\text{dofs}} = 729$. The learned multiplier varies over several orders of magnitude along the diagonal, which allows the basis to represent the steep variations of the corner layer. At equal error, the classical FEM would require from several hundred to several thousand times more degrees of freedom, according to the extrapolation of its convergence curve for seeds 1 and 2 and according to the computed meshes for seed 3. At comparable computation time, we therefore obtain errors of the same order of magnitude, or up to six times lower, but with a much lower number of degrees of freedom for the learned basis.

This study nevertheless has several limitations. In particular, it relies on a single value of $\boldsymbol{\mu}$, a single mesh and only three seeds, which allows us neither to quantify the robustness of the enrichment nor to identify the most influential parameters as in 1D. Moreover, we consider the single degree $k = 2$, whereas the 1D study suggests that the degree is one of the determining levers. Finally, as in 1D, the comparison is made with a uniform Cartesian mesh and not with a graded, adaptive or *hp* strategy, so that the gains reported above are upper bounds of the benefit over the best classical method.

Having shown, in dimensions 1 and 3, that the learned basis reduces the L^2 error for a fixed number of dofs, but only for a fixed value of $\boldsymbol{\mu}$, so that a new training is necessary for each new value, we now turn to the parametric setting, where a single network is trained once on a whole family of parameters and the learned basis is then used for values of $\boldsymbol{\mu}$ never seen during training, on 1D advection-diffusion problems.

5.4. Parametric enrichment (neural operator) – 1D test case

As in the previous sections, we seek to enrich the basis of a classical Galerkin scheme with a neural network, so that, for a fixed number of degrees of freedom N_{dofs}, the solution $u_{h,\theta}$ is more accurate than the classical solution u_h. We restrict ourselves here to the DG scheme with the SIPG flux and to patchwise bases, of the form of the multiplicative basis of (5.6), where a single network v_θ shared by all the cells provides a multiplier m_θ of the local Lagrange functions $\phi_{c,i}$.

Until now, a network was trained for a fixed value of the parameter $\boldsymbol{\mu}$, so that the learned basis was only valid for this value, and a new training was necessary for each new $\boldsymbol{\mu}$. In this section, we move to the parametric setting of problem (2.1). A single network of weights θ is trained once on a family of values of the parameter $\boldsymbol{\mu}$ drawn in the parametric set $\mathcal{M}$, and the resulting enriched basis is then used to solve the problem for any $\boldsymbol{\mu} \in \mathcal{M}$, including for values never seen during training.

We consider three ways of accounting for the parameter in the enriched basis. In the first case, the basis is shared by all the values of $\boldsymbol{\mu}$, so that the same enriched space is used whatever $\boldsymbol{\mu}$,

$$\phi^\theta_{c,i}(x) = m_\theta(x)\, \phi_{c,i}(x), \tag{5.15}$$

where the learned basis can be seen as an average basis for the whole family of problems. In the second case, the network also receives $\boldsymbol{\mu}$ as input, so that each value of $\boldsymbol{\mu}$ has its own enriched space:

$$\phi^\theta_{c,i}(x; \boldsymbol{\mu}) = m_\theta(x, \boldsymbol{\mu})\, \phi_{c,i}(x). \tag{5.16}$$

In the third case, the network does not see $\boldsymbol{\mu}$ directly. The parameter only intervenes through the transport field $C(\cdot, \boldsymbol{\mu})$ of the PDE, which is summarized by a second network, the encoder $\mathcal{E}_\theta$, into a vector $\boldsymbol{z}$ called the latent state. The basis network then receives $\boldsymbol{z}$ as input in place of $\boldsymbol{\mu}$:

$$\phi^\theta_{c,i}(x; \boldsymbol{z}) = m_\theta(x, \boldsymbol{z})\, \phi_{c,i}(x), \qquad \boldsymbol{z} := \mathcal{E}_\theta\big(C(\cdot, \boldsymbol{\mu})\big). \tag{5.17}$$

The weights of the encoder are included in θ, and the two networks are trained jointly, without a loss specific to the encoder. We call these three bases the shared basis, the $\boldsymbol{\mu}$-dependent basis and the field-dependent basis. The field-dependent basis is only studied for the first test case. Compared with the shared basis, the second one only adds weights to the first layer of the network, which also receives $\boldsymbol{\mu}$, whereas the third one also adds those of the encoder.

Notation 5.14. *The three bases are obtained with three separately trained networks. Their weights are all denoted θ and their outputs m_θ, by abuse of notation. The multipliers $m_\theta(x)$, $m_\theta(x, \boldsymbol{\mu})$ and $m_\theta(x, \boldsymbol{z})$ are therefore three different functions, and it is their arguments that indicate the basis concerned. Likewise, $u_{h,\theta}$ denotes the enriched solution of the basis under consideration, which is specified by the captions of the figures and tables.*

The last two bases fall under neural operator learning, discussed in Chapter 1. In both cases, the network defines a map that associates with the input of the problem (the parameter $\boldsymbol{\mu}$, or the transport field $C(\cdot, \boldsymbol{\mu})$ through the latent state $\boldsymbol{z}$) a function

of x, the multiplier $m_\theta(\cdot\,, \boldsymbol{\mu})$ or $m_\theta(\cdot\,, \boldsymbol{z})$, and hence the basis $\phi^\theta_{c,i}$. The shared basis, for its part, corresponds to a constant operator, which does not depend on this input. Unlike usual neural operators, whose output is the solution, the output of the operator is here the basis, and the solution $u_{h,\theta}(\cdot\,; \boldsymbol{\mu})$ is obtained by solving the DG scheme in this basis, as presented in Sections 4.5.4 and 4.6. The solver is a fixed step applied to the output of the operator, and it is the solver that carries the physics. Since there is no reference basis, the operator cannot be trained directly on its output, and it is the solution $u_{h,\theta}$ that is compared with reference solutions, through the solver.

Let $\mathcal{D}_{\text{train}} \subset \mathcal{M}$ be a set of training values of the parameter $\boldsymbol{\mu}$, drawn uniformly in $\mathcal{M}$, on which the network is trained. The quality of the learned basis is then evaluated on a set $\mathcal{D}_{\text{test}} \subset \mathcal{M}$ of test values, distinct from the training ones and also drawn uniformly in $\mathcal{M}$. This set is only used after the training, to evaluate the learned basis on values of $\boldsymbol{\mu}$ never seen during training. Unlike in the non-parametric sections, where the network was trained with the PINN loss, we use here a purely supervised loss, which avoids complicating the training of a whole family of problems. The weights θ are trained by minimizing the data loss (2.39), in which u_θ is replaced by the solution of the scheme,

$$\mathcal{J}(\theta) = \mathcal{J}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| u_{h,\theta}\big(x^{(i)}_{\text{data}}; \boldsymbol{\mu}^{(i)}_{\text{data}}\big) - u^{(i)}_{\text{data}} \right|^2, \tag{5.18}$$

where $\big(x^{(i)}_{\text{data}}, \boldsymbol{\mu}^{(i)}_{\text{data}}, u^{(i)}_{\text{data}}\big)_{i=1,\dots,N_{\text{data}}}$ are N_{data} known data. The parameters $\boldsymbol{\mu}^{(i)}_{\text{data}}$ are values of $\mathcal{D}_{\text{train}}$, $u^{(i)}_{\text{data}}$ being a reference solution at the point $x^{(i)}_{\text{data}}$ for the parameter $\boldsymbol{\mu}^{(i)}_{\text{data}}$. For each parameter, the reference solution, computed numerically on a fine mesh (see Remark 5.15), is evaluated at the same points $x^{(i)}_{\text{data}}$, which form a regular grid of 101 points of Ω.

Remark 5.15. *For the test cases of this section, no analytical solution is available. For each value of $\boldsymbol{\mu}$ of the sets $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, we therefore compute a reference solution $u(\cdot\,; \boldsymbol{\mu})$. It is obtained with the DG scheme and the SIPG flux, like the classical and enriched solutions that it allows us to evaluate, on a very fine uniform mesh of $N^{ref}_{cells} = 128$ cells, with Lagrange polynomials $\mathbb{Q}_{k_{ref}}$ of degree $k_{ref} = 4$. It is computed only once for each value of $\boldsymbol{\mu}$. Unlike in the non-parametric sections, it also intervenes in the training, through the loss (5.18).*

As in the non-parametric case, the optimization is carried out through the solver (Section 4.9). At each iteration, the scheme is assembled and solved for each of the values of $\boldsymbol{\mu}$ of the batch, and the gradient with respect to θ is obtained by the adjoint method, as summarized in Algorithm 5.2. The loops over the batch are written as such for readability, but in practice they are vectorized rather than executed sequentially.

We study two test cases of increasing difficulty. The first, presented in Section 5.4.1, is the linear transport problem of the previous sections, with field $C(x, \boldsymbol{\mu}) = \mu_1 + \mu_2 x$, for which we now vary both coefficients $\boldsymbol{\mu} = (\mu_1, \mu_2)$ in the square $\mathcal{M} = [0.8, 1.2]^2$. We compare here the three types of bases (shared, dependent on $\boldsymbol{\mu}$, and dependent on the field) to measure the contribution of the dependence on $\boldsymbol{\mu}$ with respect to the shared basis (5.15), and to check that the encoder recovers the useful information from the field alone. The second, presented in Section 5.4.2, is the turning point problem, with field $C(x, \mu) = \mu - x$, where the scalar parameter $\mu \in \mathcal{M} = [0.3, 0.7]$ sets the position

of the turning point, and hence of the layer, which moves with it. We only compare the shared basis and the $\boldsymbol{\mu}$-dependent basis there. Finally, Section 5.4.3 compares the two problems and discusses the limitations of this study.

Algorithm 5.2 Training of an enriched DG space on a family of parameters with the data loss. The loops over the batch are vectorized.

Input: mesh, degree k, training set $\mathcal{D}_{\text{train}}$, initial weights θ, N_{epochs}

for $n = 1, \dots, N_{\text{epochs}}$ **do**
  Draw a batch of 16 values of $\boldsymbol{\mu}$ in $\mathcal{D}_{\text{train}}$
  for each $\boldsymbol{\mu}$ of the batch **do**
    Build the basis $\phi^{\theta}_{c,i} = m_\theta \, \phi_{c,i}$ (shared, $\boldsymbol{\mu}$-dep. or field-dep.) ▷ solver
    Assemble $A_h(\theta;\boldsymbol{\mu})$ and $\boldsymbol{b}_h(\theta;\boldsymbol{\mu})$ ▷ solver
    Solve $A_h(\theta;\boldsymbol{\mu})\,\boldsymbol{\beta}_h(\theta;\boldsymbol{\mu}) = \boldsymbol{b}_h(\theta;\boldsymbol{\mu})$ for $\boldsymbol{\beta}_h(\theta;\boldsymbol{\mu})$ ▷ solver
    Evaluate $u_{h,\theta}(\cdot\,;\boldsymbol{\mu})$
  end for
  Evaluate the data loss $\mathcal{J}_{\text{data}}(\theta)$
  for each $\boldsymbol{\mu}$ of the batch **do**
    Solve the adjoint system $J_{\boldsymbol{R}_h}(\theta;\boldsymbol{\mu})^T \, \boldsymbol{\lambda}_h(\theta;\boldsymbol{\mu}) = \big(\partial\mathcal{J}/\partial\boldsymbol{\beta}_h(\theta;\boldsymbol{\mu})\big)^T$ ▷ backward
    Compute the contribution of $\boldsymbol{\mu}$ to $d\mathcal{J}/d\theta$ by VJP ▷ backward
  end for
  Sum the contributions to get $d\mathcal{J}/d\theta$
  Update θ with the optimizer (e.g. Adam)
end for

Output: weights θ

Remark 5.16. *As in Remark 5.1, Galerkin methods with a classical basis and with an enriched basis are compared at a fixed number of degrees of freedom N_{dofs}, through relative errors in the L^2 norm with respect to the reference solution (Remark 5.15). For a given parameter $\boldsymbol{\mu}$, the gain is the ratio between the error of the classical solution u_h and that of the enriched solution $u_{h,\theta}$, at the same parameter. Since it varies with the parameter $\boldsymbol{\mu}$, we are interested in the minimal, median and maximal values over the test set $\mathcal{D}_{test}$. The number of weights of the network is counted separately, and the training cost is not taken into account.*

The compared bases are trained with exactly the same hyperparameters, only the input of the basis network differing. We consider the DG scheme with the SIPG flux (Section 2.2.2.2), with penalty $\kappa = \eta_F/h$ where $\eta_F = 10\,k(k+1)$ and $h = 1/N_{\text{cells}}$, and all the pairs (N_{cells}, k) with $N_{\text{cells}} \in \{4, 8, 12\}$ and $k \in \{1, 2, 3\}$, so that $N_{\text{dofs}} = N_{\text{cells}}(k+1)$. The multiplier is positive and takes bounded values, of the form of (5.9), where the factor 2 in the exponential is specified for each test case and v_θ is an MLP with tanh activation and 2 hidden layers of width 16. Since the enriched space depends on the mesh and on the degree, as in the non-parametric case, a network is trained for each basis and each pair (N_{cells}, k).

The training set $\mathcal{D}_{\text{train}}$ contains 32 values of $\boldsymbol{\mu}$. The optimization uses the self-scaled BFGS (SSBFGS) optimizer [126] of `ScimBa` for $N_{\text{epochs}} = 400$ epochs, each one over a batch of 16 values of $\boldsymbol{\mu}$ drawn in $\mathcal{D}_{\text{train}}$, that is, $N_{\text{data}} = 16 \times 101$ data in the loss (5.18). Since the results depend on the initialization of the network, each training is repeated

with 10 different seeds, and we only keep the one with the lowest mean L^2 relative error on $\mathcal{D}_{\text{train}}$, so that the test set intervenes neither in the training nor in the choice of the seed. Unlike in the non-parametric sections, we are not interested here in the robustness of the training over all the seeds, but in the best seed. The weights of the networks are therefore not bounded by the softclip operator. The size of the set of test parameters $\mathcal{D}_{\text{test}}$ is specified for each test case.

5.4.1. Linear transport problem

We consider problem (5.3) in its parametric form, where the parameter $\boldsymbol{\mu} = (\mu_1, \mu_2)$ is no longer fixed but belongs to the parametric domain $\mathcal{M} = [0.8, 1.2]^2$:

$$\begin{cases} -\dfrac{1}{\mathrm{Pe}}\,\partial_{xx}u + (\mu_1 + \mu_2 x)\,\partial_x u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases} \tag{5.19}$$

with $\Omega = (0,1)$, $D = 1$, $\mathrm{Pe} = 20$, a constant source $f = 1$ and the affine transport field $C(x, \boldsymbol{\mu}) = \mu_1 + \mu_2 x$. Since $C(x, \boldsymbol{\mu}) > 0$ on Ω, the transport in (5.19) is directed towards $x = 1$ and the solution exhibits a boundary layer at the outflow boundary. The parameter changes the intensity of the transport and hence the shape of the solution, but it does not move the layer. The characteristic thickness of this layer is of the order of $1/\big(\mathrm{Pe}\,C(1,\boldsymbol{\mu})\big) = 1/\big(\mathrm{Pe}\,(\mu_1+\mu_2)\big)$, that is, between 0.02 and 0.03. This thickness is smaller than the cell size $h = 1/N_{\text{cells}}$ for all the meshes considered, so that the classical basis cannot represent the layer well, which motivates the enrichment of the basis.

In this section, we compare the three bases (5.15) to (5.17) with the settings described above. We consider the multiplier $m_\theta = \exp\big(2\tanh v_\theta\big)$ of (5.9). For the shared basis, the network only receives the point x, for a total of $N_\theta = 321$ weights, so that the basis functions are the same for all the parameters $\boldsymbol{\mu}$. For the $\boldsymbol{\mu}$-dependent basis, the network receives both the point x and the parameter $\boldsymbol{\mu}$, for a total of $N_\theta = 353$ weights (the first layer having two more inputs), so that the basis functions depend on each parameter $\boldsymbol{\mu}$. For the field-dependent basis, the encoder $\mathcal{E}_\theta$ receives the transport field $C(\cdot, \boldsymbol{\mu}) = \mu_1 + \mu_2 x$ sampled at 64 points of Ω, keeps its first 20 Fourier modes, and feeds them to an MLP with tanh activation and 2 hidden layers of width 16, whose output is the latent state $\boldsymbol{z}$, of size 4. The basis network, identical to that of the other bases, receives x and the latent state $\boldsymbol{z}$ built by the encoder. This basis has a total of $N_\theta = 1\,381$ weights, of which 996 for the encoder.

Once each basis is learned, it is evaluated on the test set $\mathcal{D}_{\text{test}}$ of 500 values of $\boldsymbol{\mu}$, for which the reference solution is computed as for the training values (Remark 5.15). The errors and the gains are defined as in Remark 5.16. We first study the variability of the solutions and of the classical DG error over $\mathcal{M}$, then compare the gains of the three bases over $\mathcal{D}_{\text{test}}$, and finally compare their multipliers and their solutions at fixed parameters to understand where the differences come from.

Variability of the solutions and classical error. The left panel of Figure 5.31 shows the reference solution for nine parameters of $\mathcal{M}$ (at the extremities and at the center of the domain), and the right panel the L^2 relative error of the classical DG SIPG method at each parameter of $\mathcal{D}_{\text{test}}$, for $k = 2$ and $N_{\text{cells}} = 8$.

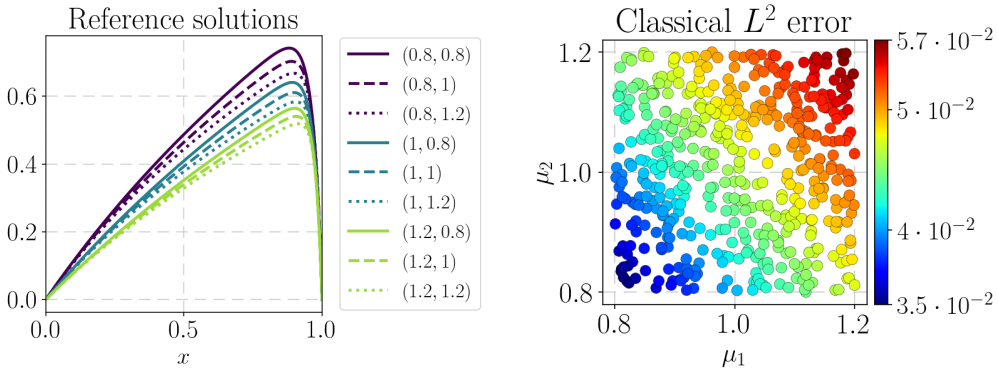


Figure 5.31.: Considering the *1D linear transport problem* with $k = 2$ and $N_{\text{cells}} = 8$. Left – Reference solutions at nine parameters $\boldsymbol{\mu} = (\mu_1, \mu_2)$. Right – L^2 relative error of the classical DG SIPG method at each parameter of $\mathcal{D}_{\text{test}}$.

The reference solutions of Figure 5.31 (left) all have the same shape, a regular growth up to a maximum around $x \approx 0.9$, then a rapid drop down to the boundary condition at $x = 1$, which constitutes the boundary layer and whose position does not depend on the parameter. Only the amplitude varies, from about 0.75 for $\boldsymbol{\mu} = (0.8, 0.8)$ to about 0.5 for $\boldsymbol{\mu} = (1.2, 1.2)$. The classical DG error (right) varies little over $\mathcal{M}$, from $3.5 \cdot 10^{-2}$ to $5.7 \cdot 10^{-2}$. It is the lowest near $(0.8, 0.8)$ and the highest near $(1.2, 1.2)$, that is, the classical DG represents the layer less well when the transport is stronger, where the layer is thinner. The family of problems thus varies little, for both the solutions and the classical error. The variations of the gain studied below, much larger than those of the classical error, therefore mainly come from the error of the learned bases.

Gains of the shared, μ-dependent and field-dependent bases. We now compare the gains of the three bases (5.15) to (5.17), trained with the same hyperparameters and the same sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, only their input differing. For each basis and each configuration (N_{cells}, k), the minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ are given in Tables 5.18 to 5.20, with the L^2 relative error of the learned solution at the parameter that achieves each of them. These parameters are specific to each basis and to each configuration, so that they differ from one table to another, and from one row to another within a single table.

In all the configurations, the three bases do better than the classical basis at every parameter of $\mathcal{D}_{\text{test}}$, since the minimal gains are greater than 1, being about 4 at least for the shared basis (Table 5.18), 48 for the $\boldsymbol{\mu}$-dependent basis (Table 5.19) and 38 for the field-dependent basis (Table 5.20). For the shared basis, the median gain is of the order of ten to a hundred (14 to 167), and it increases with k for $N_{\text{cells}} \geq 8$.

The gains of the $\boldsymbol{\mu}$-dependent basis are of a different order of magnitude as soon as $k \geq 2$ for $N_{\text{cells}} \geq 8$, and for $k = 3$ when $N_{\text{cells}} = 4$. For $k = 1$, the median gain remains of the order of 10^2, whereas it reaches several thousands for $k \geq 2$ when $N_{\text{cells}} \geq 8$. For all the degrees $k \geq 2$, the minimal gain of the $\boldsymbol{\mu}$-dependent basis even exceeds the maximal gain of the shared basis. This improvement is obtained with 353 weights instead of 321, so that it cannot be explained by the size of the network. The $\boldsymbol{\mu}$-dependent basis therefore does much better than the shared basis, which is notable

given the low variability of the solutions observed above.

The field-dependent basis is of the same order of magnitude as the $\boldsymbol{\mu}$-dependent one. Its median gains are close to those of the latter for $k = 2$ and slightly lower for $k = 3$. For the configuration $N_{\text{cells}} = 12$ and $k = 1$, the median gain is only 47 (against 116 with the $\boldsymbol{\mu}$-dependent basis) and the gains there are almost uniform over $\mathcal{D}_{\text{test}}$ (from 38 to 48). As for the $\boldsymbol{\mu}$-dependent basis, the minimal gain exceeds the maximal gain of the shared basis for $k \geq 2$. The encoder thus recovers, from the field alone, most of the useful information of the parameter, but, for this affine field, it does no better than the parameter itself, with 1 381 weights instead of 353.

		minimal gain		**median gain**		**maximal gain**	
N_{cells}	k	**learned**	**gain**	**learned**	**gain**	**learned**	**gain**
4	1	$1.1 \cdot 10^{-2}$	36	$7.8 \cdot 10^{-3}$	69	$4.4 \cdot 10^{-3}$	110
4	2	$2.3 \cdot 10^{-2}$	5	$6.7 \cdot 10^{-3}$	20	$1.3 \cdot 10^{-3}$	106
4	3	$2.6 \cdot 10^{-3}$	31	$9.1 \cdot 10^{-4}$	82	$2.5 \cdot 10^{-4}$	288
8	1	$1.1 \cdot 10^{-2}$	13	$4.7 \cdot 10^{-3}$	37	$1.7 \cdot 10^{-3}$	99
8	2	$2.9 \cdot 10^{-3}$	20	$1.3 \cdot 10^{-3}$	42	$7.4 \cdot 10^{-4}$	69
8	3	$4.7 \cdot 10^{-4}$	18	$1.1 \cdot 10^{-4}$	120	$6.8 \cdot 10^{-5}$	202
12	1	$2 \cdot 10^{-2}$	4	$7.1 \cdot 10^{-3}$	14	$2.9 \cdot 10^{-3}$	32
12	2	$6 \cdot 10^{-4}$	24	$3.6 \cdot 10^{-4}$	59	$2.4 \cdot 10^{-4}$	98
12	3	$1.4 \cdot 10^{-4}$	18	$3.3 \cdot 10^{-5}$	167	$1.5 \cdot 10^{-5}$	315

Table 5.18.: Considering the *1D linear transport problem* with the shared basis. Minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ of the best seed, with the L^2 relative error of the enriched solution at the same parameter (learned), see Remark 5.16.

		minimal gain		**median gain**		**maximal gain**	
N_{cells}	k	**learned**	**gain**	**learned**	**gain**	**learned**	**gain**
4	1	$7.9 \cdot 10^{-3}$	48	$4.7 \cdot 10^{-3}$	118	$4 \cdot 10^{-3}$	130
4	2	$1.3 \cdot 10^{-3}$	121	$5.9 \cdot 10^{-4}$	235	$3.2 \cdot 10^{-4}$	390
4	3	$8.7 \cdot 10^{-5}$	564	$2.5 \cdot 10^{-5}$	2 538	$1.8 \cdot 10^{-5}$	3 521
8	1	$1.9 \cdot 10^{-3}$	70	$1.2 \cdot 10^{-3}$	125	$9.3 \cdot 10^{-4}$	171
8	2	$4.3 \cdot 10^{-5}$	811	$2 \cdot 10^{-5}$	2 381	$9 \cdot 10^{-6}$	4 590
8	3	$5.4 \cdot 10^{-6}$	1 579	$1.7 \cdot 10^{-6}$	7 143	$1.6 \cdot 10^{-6}$	9 123
12	1	$1.1 \cdot 10^{-3}$	66	$6.9 \cdot 10^{-4}$	116	$4.5 \cdot 10^{-4}$	181
12	2	$2 \cdot 10^{-5}$	704	$7.4 \cdot 10^{-6}$	2 653	$3.9 \cdot 10^{-6}$	4 833
12	3	$2.7 \cdot 10^{-6}$	897	$6.5 \cdot 10^{-7}$	6 041	$6.8 \cdot 10^{-7}$	7 275

Table 5.19.: Considering the *1D linear transport problem* with the $\boldsymbol{\mu}$-dependent basis. Minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ of the best seed, with the L^2 relative error of the enriched solution at the same parameter (learned), see Remark 5.16.

		minimal gain		median gain		maximal gain	
N_{cells}	k	learned	gain	learned	gain	learned	gain
4	1	$5.9 \cdot 10^{-3}$	65	$3.3 \cdot 10^{-3}$	131	$2.9 \cdot 10^{-3}$	173
4	2	$1.4 \cdot 10^{-3}$	108	$6.6 \cdot 10^{-4}$	202	$4 \cdot 10^{-4}$	310
4	3	$9.2 \cdot 10^{-5}$	891	$2.6 \cdot 10^{-5}$	2 541	$1.4 \cdot 10^{-5}$	4 157
8	1	$2.3 \cdot 10^{-3}$	59	$1.4 \cdot 10^{-3}$	128	$9.4 \cdot 10^{-4}$	170
8	2	$6.2 \cdot 10^{-5}$	917	$2.2 \cdot 10^{-5}$	2 369	$8.3 \cdot 10^{-6}$	4 919
8	3	$2.7 \cdot 10^{-6}$	3 245	$2.3 \cdot 10^{-6}$	6 379	$2.1 \cdot 10^{-6}$	7 234
12	1	$1.9 \cdot 10^{-3}$	38	$1.8 \cdot 10^{-3}$	47	$2 \cdot 10^{-3}$	48
12	2	$1.2 \cdot 10^{-5}$	1 217	$6.2 \cdot 10^{-6}$	3 654	$1.8 \cdot 10^{-6}$	10 263
12	3	$2.6 \cdot 10^{-6}$	939	$1.2 \cdot 10^{-6}$	4 686	$7 \cdot 10^{-7}$	6 745

Table 5.20.: Considering the *1D linear transport problem* with the field-dependent basis. Minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ of the best seed, with the L^2 relative error of the enriched solution at the same parameter (learned), see Remark 5.16.

The tables give, for each configuration, the minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$, but not how they are distributed over the parametric domain. We now concentrate on the configuration $N_{\text{cells}} = 8$ and $k = 2$, and we seek to represent where the gains are the highest and the lowest in $\mathcal{M}$. For this, Figure 5.32 shows, for the three bases, the gain at each parameter of $\mathcal{D}_{\text{test}}$, with the parameters of maximal, median and minimal gain of each.

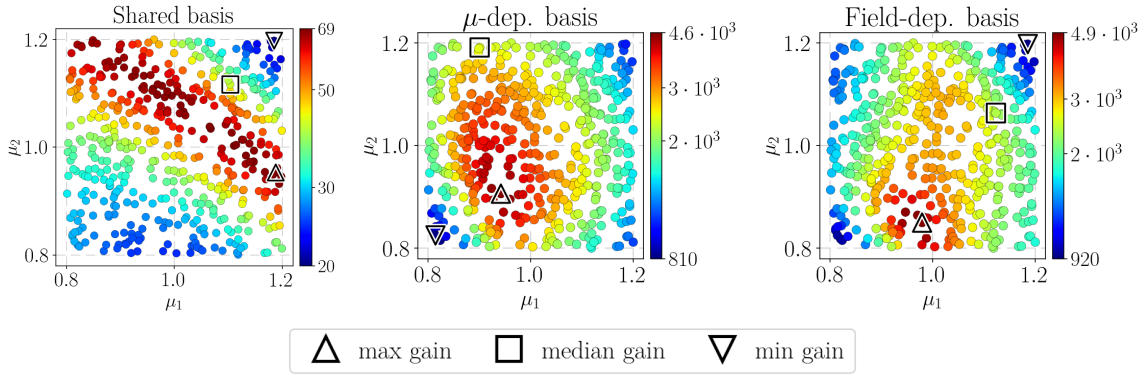


Figure 5.32.: Considering the *1D linear transport problem* with $k = 2$ and $N_{\text{cells}} = 8$. Gain, with respect to the classical DG SIPG method, at each parameter of $\mathcal{D}_{\text{test}}$. Left – Shared basis. Middle – $\boldsymbol{\mu}$-dependent basis. Right – Field-dependent basis. The markers give the parameters of maximal, median and minimal gain of each basis.

In Figure 5.32, we see that, for the shared basis (left), the gains range from 20 to 69 and that, along the diagonal going from the lower left corner to the upper right corner of $\mathcal{M}$, they increase then decrease. The maximum is located a little beyond the center, and the values are low at both ends of this same diagonal. This distribution recalls that of the classical DG error of Figure 5.31 (right), which varies along the

same diagonal but grows continuously, without decreasing. For the $\boldsymbol{\mu}$-dependent basis (middle), the structure is different. The gains range from 811 to $4.6 \cdot 10^3$, and they no longer follow the diagonal of the classical error. They are the highest in the central part of $\mathcal{M}$ (slightly off-center) and decrease towards the edges. For the field-dependent basis (right), the gains range from 917 to $4.9 \cdot 10^3$, almost the same range as for the previous basis, and the structure seems comparable, with a slight shift of the maximum towards the bottom. On the other hand, they are also lower near the edges (except the bottom edge) and the corners of $\mathcal{M}$ than in its central part. These lower gains near the edges could be correlated with the fact that only 32 training parameters are available, few of which lie near the edges of $\mathcal{M}$.

Comparison of the solutions and multipliers. To understand where the difference between the bases comes from, we compare them at three fixed parameters. We again place ourselves in the configuration $N_{\text{cells}} = 8$, $k = 2$ of Figure 5.32 and denote:

$$\boldsymbol{\mu}^{\triangledown} := (0.81, 0.83), \qquad \boldsymbol{\mu}^{\square} := (0.90, 1.19), \qquad \boldsymbol{\mu}^{\triangle} := (0.94, 0.91),$$

the parameters of minimal, median and maximal gain of the $\boldsymbol{\mu}$-dependent basis. The left panel of Figure 5.33 shows the reference solution at these three parameters, then the multipliers learned by the $\boldsymbol{\mu}$-dependent basis (middle) and by the field-dependent basis (right), the multiplier of the shared basis being shown in gray in these two panels. Unlike in the previous figures, the parameters are fixed here, and they are the same for the three bases. We are therefore not interested in the parameters of best or worst gain of each basis, but in the comparison of the bases at common parameters. For the field-dependent basis, the multipliers are evaluated at the latent states:

$$\boldsymbol{z}^{\triangledown} := \mathcal{E}_\theta\big(C(\cdot, \boldsymbol{\mu}^{\triangledown})\big), \qquad \boldsymbol{z}^{\square} := \mathcal{E}_\theta\big(C(\cdot, \boldsymbol{\mu}^{\square})\big), \qquad \boldsymbol{z}^{\triangle} := \mathcal{E}_\theta\big(C(\cdot, \boldsymbol{\mu}^{\triangle})\big).$$

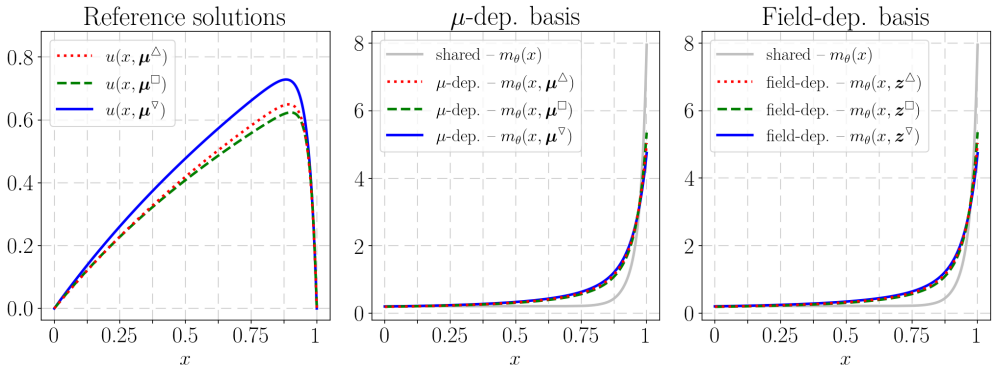


Figure 5.33.: Considering the *1D linear transport problem* with $k = 2$ and $N_{\text{cells}} = 8$. Left – Reference solutions at the parameters $\boldsymbol{\mu}^{\triangledown}$ (solid), $\boldsymbol{\mu}^{\square}$ (dashed) and $\boldsymbol{\mu}^{\triangle}$ (dotted). Middle – Multipliers $m_\theta(x, \boldsymbol{\mu})$ of the $\boldsymbol{\mu}$-dependent basis at the same parameters. Right – Multipliers $m_\theta(x, \boldsymbol{z})$ of the field-dependent basis at the corresponding latent states. In the two last panels, the multiplier $m_\theta(x)$ of the shared basis is shown in gray. All multipliers are those of the best seed, divided by their L^2 norm.

The three reference solutions of Figure 5.33 (left) have the same shape and differ mainly in amplitude, the solution at $\boldsymbol{\mu}^{\triangledown}$ being the largest. The multiplier of the shared basis (in gray), which does not depend on $\boldsymbol{\mu}$, is almost constant, around 0.2, up to $x \approx 0.8$, then grows abruptly towards $x = 1$ up to about 8. We thus recover the shape of the multiplier obtained in the non-parametric case (Figure 5.24), almost constant then strongly increasing near $x = 1$, where the boundary layer is located. Since a constant multiplier does not modify the DG space, the shared basis remains close to the classical basis in the first cells and is only enriched in the last ones.

The multipliers of the $\boldsymbol{\mu}$-dependent basis (middle) start from the same value as the shared one (about 0.2), but grow earlier and more progressively, and only reach about 5 at $x = 1$, so that they modify the basis on a larger number of cells. Those of the field-dependent basis (right) have almost the same shape and the same values, also around 5 at $x = 1$, which is consistent with the closeness of their gains. For each of the two bases, the three curves obtained for $\boldsymbol{\mu}^{\triangledown}$, $\boldsymbol{\mu}^{\square}$ and $\boldsymbol{\mu}^{\triangle}$ are close and only slightly differ from one another, near $x = 1$. At $x = 1$, they are ordered the same way for both bases (with $\boldsymbol{\mu}^{\square}$ on top, then $\boldsymbol{\mu}^{\triangle}$, then $\boldsymbol{\mu}^{\triangledown}$). The multipliers thus adapt little to the parameter, which does not prevent the dependent bases from obtaining much better gains than the shared basis (Tables 5.18 to 5.20). However, these two families clearly differ from the multiplier of the shared basis, and the figures do not allow us to say whether the gap in gain with the latter comes from the low adaptation to the parameter, or from the different shape of the multiplier.

We now consider only a single parameter, $\boldsymbol{\mu}^{\square}$, in order to really compare the solutions obtained and to observe their absolute errors. This parameter is the one of median gain of the $\boldsymbol{\mu}$-dependent basis. At this parameter, the solutions (left) and the absolute errors (right) of the classical DG and of the three bases are shown in Figure 5.34.

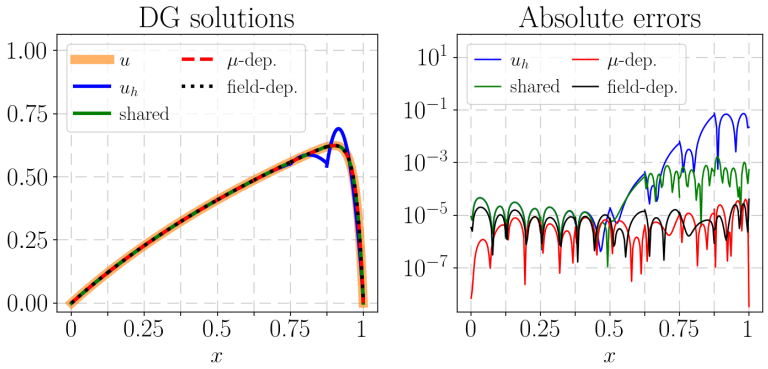


Figure 5.34.: Considering the *1D linear transport problem* with $\boldsymbol{\mu}^{\square} = (0.90, 1.19)$, $k = 2$ and $N_{\text{cells}} = 8$. Left – Reference solution u, classical DG SIPG solution u_h and enriched solutions $u_{h,\theta}$ of the shared, $\boldsymbol{\mu}$-dependent and field-dependent bases, labeled by their basis. Right – Corresponding absolute errors.

In Figure 5.34, the classical solution u_h exhibits a jump at the interfaces of the last cells and exceeds the maximum of the reference solution, whereas the enriched solutions $u_{h,\theta}$ of the three bases (shared, $\boldsymbol{\mu}$-dependent and field-dependent) are indistinguishable from the reference solution at this scale. Regarding the errors, the classical error reaches about $7 \cdot 10^{-2}$ in the vicinity of the layer. The error of the shared basis is of the

order of 10^{-3} in the last cells, but it coincides with the classical error in the first ones, which is consistent with the almost constant multiplier of Figure 5.33. That of the $\boldsymbol{\mu}$-dependent basis remains at most of the order of 10^{-5} over the whole Ω, and that of the field-dependent basis is of the same order. The superiority of the $\boldsymbol{\mu}$-dependent and field-dependent bases over the shared basis thus mainly comes from a better treatment of the layer, where the error of the shared basis remains of the order of 10^{-3} against 10^{-5} for the other two. In the first cells, they are also slightly more precise than the shared basis, but the gap there is much smaller.

5.4.2. Turning point problem

We consider again problem (5.3) in its parametric form, but with a transport field that changes sign in the domain. The parameter μ is here a scalar and belongs to $\mathcal{M} = [0.3, 0.7]$:

$$\begin{cases} -\dfrac{1}{\mathrm{Pe}}\,\partial_{xx}u + (\mu - x)\,\partial_x u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma, \end{cases} \tag{5.20}$$

with $\Omega = (0,1)$, $D = 1$, $\mathrm{Pe} = 1\,000$, a constant source term $f = 1$ and the transport field $C(x,\mu) = \mu - x$. This field is positive to the left of μ and negative to its right, so that the transport is directed towards $x = \mu$ on both sides of this point. Both boundaries are therefore inflow boundaries and the solution of (5.20) does not present a boundary layer, but an interior layer located around $x = \mu$. The essential difference with the linear transport problem presented in Section 5.4.1 is that the parameter here moves the layer. For the shared basis, which is the same for all the values of μ, the enrichment must therefore serve all the positions of the layer, whereas for the μ-dependent basis it can follow its location.

In this section, we compare the shared basis (5.15) and the μ-dependent basis (5.16) with the same settings as for the linear transport. The multiplier considered in this section, $m_\theta = \exp\big(4\tanh v_\theta\big)$, differs slightly from the one chosen for the linear transport case. Indeed, replacing the factor 2 by a factor 4 increases the amplitude of variation of the multiplier, that is, the ratio between its maximal and minimal possible values, which allows a more pronounced enrichment of the basis. This turned out to be necessary for this problem, since the layer is thinner than a cell there, so that the basis must vary strongly inside a single cell, which was limited by the narrower range in preliminary tests. For the shared basis, the network only receives the point x and has $N_\theta = 321$ weights. For the μ-dependent basis, it also receives the parameter μ and has $N_\theta = 337$ weights, the first layer having one additional input. Since the transport field is entirely determined by the scalar μ, the field-dependent basis is not studied here. As for the linear transport, each training is repeated with 10 seeds and, for each basis, only the one with the lowest mean L^2 relative error on $\mathcal{D}_{\text{train}}$ is kept.

Once each basis has been learned, it is evaluated on a test set $\mathcal{D}_{\text{test}}$ of 32 values of μ, drawn uniformly in $\mathcal{M}$, for which the reference solution is computed as for the training values (Remark 5.15). Since the parameter is here of dimension 1, the 32 training values cover $\mathcal{M}$ much more densely than in the linear transport case (Section 5.4.1), where they covered a domain of dimension 2. The errors and the gains are defined as in Remark 5.16. We first study the variability of the solutions and of the classical

DG error over $\mathcal{M}$, then compare the gains of the two bases over $\mathcal{D}_{\text{test}}$, and finally compare their multipliers and their solutions at fixed parameters to understand where the differences come from.

Variability of the solutions and classical error. We begin by studying the variability of the solutions and of the error of the classical DG method over $\mathcal{M}$. The left panel of Figure 5.35 shows the reference solution for five values of μ spread over $\mathcal{M}$, and the right panel the L^2 relative error of the classical DG SIPG method at each parameter of $\mathcal{D}_{\text{test}}$, for $k = 2$ and $N_{\text{cells}} = 8$.

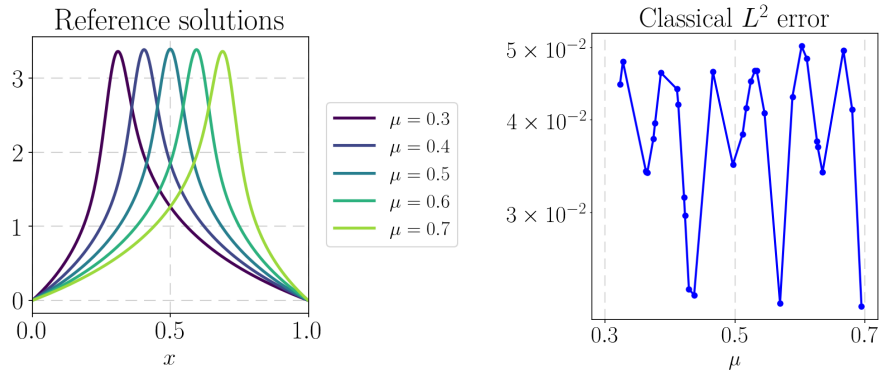


Figure 5.35.: Considering the *1D turning point problem* with $k = 2$ and $N_{\text{cells}} = 8$. Left – Reference solutions at five parameters $\mu \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$. Right – L^2 relative error of the classical DG SIPG method at each parameter of $\mathcal{D}_{\text{test}}$.

The reference solutions of Figure 5.35 (left) all have a sharp peak of height about 3.4, located in the neighborhood of $x = \mu$. Unlike in the linear transport, where only the amplitude varied, it is here mainly the position of the layer that varies with μ. The error of the classical DG method (right) varies between about $2 \cdot 10^{-2}$ and $5 \cdot 10^{-2}$, which is of the same order as for the linear transport. It does not vary monotonically with μ and oscillates. This is consistent with the fact that the layer, whose thickness is of order $1/\sqrt{\text{Pe}} \approx 0.03$ and thus thinner than a cell ($h \in [1/12, 1/4]$), is placed differently with respect to the interfaces of the mesh depending on the value of μ, which changes the way the classical method represents it. Here again, the classical error only varies by a factor of about 2 over $\mathcal{M}$.

Gains of the shared and μ-dependent bases. We now compare the gains of the two bases (5.15) and (5.16), trained with the same hyperparameters and the same sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, only their input differing. For each basis and each configuration (N_{cells}, k), the minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ are given in Tables 5.21 and 5.22, together with the L^2 relative error of the learned solution at the parameter that attains each of them. These parameters are specific to each basis and to each configuration.

The shared basis (Table 5.21) brings little with respect to the classical basis. Its minimal gain is greater than 1 only for $N_{\text{cells}} = 4$ and $k = 1$ (it is 0.99 for $N_{\text{cells}} = 8$ and $k = 1$), and it goes down to 0.4, that is, the learned solution is in some cases more than twice less accurate than the classical solution. Its median gain lies between 1.0

and 1.5 for all the configurations, except for $N_{\text{cells}} = 4$ with $k \in \{1, 2\}$, where it equals 4.4 and 2.7. The maximal gain exceeds 4 only for these same two configurations, with a maximum of 18 for the first one. These two configurations are those for which the classical basis is the least accurate, the layer being then much thinner than a cell. The shared basis is therefore well below what it brought for the linear transport, which is consistent with the fact that the parameter moves the layer and that the shared basis cannot follow this displacement.

		minimal gain		**median gain**		**maximal gain**	
N_{cells}	k	**learned**	**gain**	**learned**	**gain**	**learned**	**gain**
4	1	$3.9 \cdot 10^{-1}$	1.6	$1.5 \cdot 10^{-1}$	4.4	$4.1 \cdot 10^{-1}$	18.1
4	2	$1.9 \cdot 10^{-1}$	0.5	$7.6 \cdot 10^{-2}$	2.7	$3.2 \cdot 10^{-2}$	16.7
4	3	$6.1 \cdot 10^{-2}$	0.8	$3.6 \cdot 10^{-2}$	1	$2.1 \cdot 10^{-2}$	3.7
8	1	$8.7 \cdot 10^{-2}$	1	$1.8 \cdot 10^{-1}$	1.4	$7.7 \cdot 10^{-2}$	3.5
8	2	$9.3 \cdot 10^{-2}$	0.5	$3.3 \cdot 10^{-2}$	1.2	$2.2 \cdot 10^{-2}$	2.1
8	3	$1.4 \cdot 10^{-2}$	0.4	$1 \cdot 10^{-2}$	1.4	$3.8 \cdot 10^{-3}$	3.8
12	1	$4.9 \cdot 10^{-2}$	0.8	$7.4 \cdot 10^{-2}$	1.2	$2.4 \cdot 10^{-2}$	1.9
12	2	$1.6 \cdot 10^{-2}$	0.4	$7 \cdot 10^{-3}$	1.1	$5.4 \cdot 10^{-3}$	2.9
12	3	$3.9 \cdot 10^{-3}$	0.7	$2.1 \cdot 10^{-3}$	1.5	$7.9 \cdot 10^{-4}$	3.3

Table 5.21.: Considering the *1D turning point problem* with the shared basis. Minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ of the best seed, with the L^2 relative error of the enriched solution at the same parameter (learned), see Remark 5.16.

		minimal gain		**median gain**		**maximal gain**	
N_{cells}	k	**learned**	**gain**	**learned**	**gain**	**learned**	**gain**
4	1	$7.3 \cdot 10^{-2}$	8	$6.3 \cdot 10^{-2}$	15	$4.9 \cdot 10^{-2}$	81
4	2	$2 \cdot 10^{-2}$	5	$3.1 \cdot 10^{-2}$	14	$1.1 \cdot 10^{-2}$	46
4	3	$9.2 \cdot 10^{-3}$	4	$1.1 \cdot 10^{-2}$	9	$3.9 \cdot 10^{-3}$	25
8	1	$8.1 \cdot 10^{-3}$	11	$5.2 \cdot 10^{-3}$	34	$3.1 \cdot 10^{-3}$	76
8	2	$2.7 \cdot 10^{-4}$	85	$2.1 \cdot 10^{-4}$	211	$1.4 \cdot 10^{-4}$	366
8	3	$1.2 \cdot 10^{-3}$	7	$1.1 \cdot 10^{-3}$	15	$4.6 \cdot 10^{-4}$	29
12	1	$6.2 \cdot 10^{-3}$	6	$8.1 \cdot 10^{-3}$	11	$4.7 \cdot 10^{-3}$	18
12	2	$1.4 \cdot 10^{-4}$	50	$7.1 \cdot 10^{-5}$	161	$2.7 \cdot 10^{-5}$	572
12	3	$3.1 \cdot 10^{-4}$	8	$2.1 \cdot 10^{-4}$	19	$8.6 \cdot 10^{-5}$	49

Table 5.22.: Considering the *1D turning point problem* with the μ-dependent basis. Minimal, median and maximal gains over $\mathcal{D}_{\text{test}}$ of the best seed, with the L^2 relative error of the enriched solution at the same parameter (learned), see Remark 5.16.

The μ-dependent basis (Table 5.22) is of a completely different order of magnitude. It does better than the classical basis at every parameter of $\mathcal{D}_{\text{test}}$ and for all the configurations, with a minimal gain of at least 4 and a median gain of at least 9. For most configurations, its minimal gain even exceeds the maximal gain of the shared basis. This improvement is obtained with 337 weights instead of 321, that is, only 16 additional weights, which is unlikely to explain it by itself. Whereas the shared basis already brought a substantial gain in the linear transport, the dependence on μ is here indispensable, which is consistent with the fact that the parameter moves the layer and that the multiplier must follow this displacement in order to enrich the basis at the right place. The highest gains are obtained for $k = 2$ with $N_{\text{cells}} \in \{8, 12\}$, where the median gain reaches 211 and 161. Unlike in the linear transport, the gains do not seem to increase with k. This may come from a less successful training for $k = 3$, with the same budget of 400 epochs, rather than from a limit of the basis: for $N_{\text{cells}} = 8$ and 12, the loss reached by the best seed is $1.0 \cdot 10^{-6}$ and $3.7 \cdot 10^{-8}$ for $k = 3$, against $5.6 \cdot 10^{-8}$ and $3.0 \cdot 10^{-9}$ for $k = 2$.

The previous tables give the minimal, median and maximal gains for each configuration, but not their distribution over the parametric domain. In Figure 5.36, we show, for $N_{\text{cells}} = 8$ and $k = 2$, the gain at each parameter of $\mathcal{D}_{\text{test}}$ for the two types of bases, with the parameters of maximal, median and minimal gain of each of them.

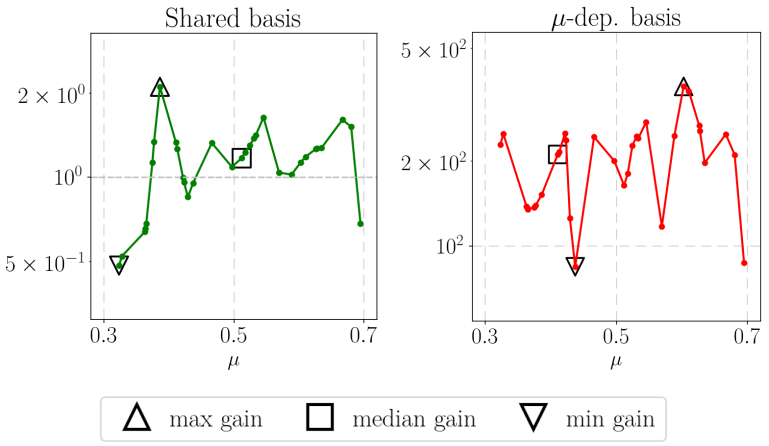


Figure 5.36.: Considering the *1D turning point problem* with $k = 2$ and $N_{\text{cells}} = 8$. Gain, with respect to the classical DG SIPG method, at each parameter of $\mathcal{D}_{\text{test}}$. Left – Shared basis. Right – μ-dependent basis. The markers give the parameters of maximal, median and minimal gain of each basis.

In Figure 5.36, we see that the gain of the shared basis (left) stays around 1 over the whole of $\mathcal{M}$, more precisely between 0.5 and 2.1. In the worst cases, the shared basis therefore degrades the classical solution by a factor of about 2, and in the best cases, it only improves it by a factor of about 2. The gain seems lower at the edges of $\mathcal{M}$, but, with only 32 test parameters, we cannot really identify a range of parameters that is clearly more favorable. On the other hand, for the μ-dependent basis (right), the gain ranges from 85 to 366, and it remains of the same order of magnitude over the whole of $\mathcal{M}$, without visible degradation at the edges. There is no clear structure as a function of μ either.

Comparison of the multipliers and solutions. To understand the origin of this gap between the two bases, we compare their multipliers at three fixed parameters. We place ourselves again in the configuration $N_{\text{cells}} = 8$, $k = 2$ of Figure 5.36 and we denote:

$$\mu^{\triangledown} := 0.44, \qquad \mu^{\square} := 0.41, \qquad \mu^{\triangle} := 0.60,$$

the parameters of minimal, median and maximal gain of the μ-dependent basis. The left panel of Figure 5.37 shows the reference solution at these three parameters, and the right panel the multipliers learned by the μ-dependent basis, with that of the shared basis in gray. As for the linear transport, the three parameters are chosen according to the μ-dependent basis and used as such for both bases, whereas in Figure 5.36 the marked parameters are specific to each basis.

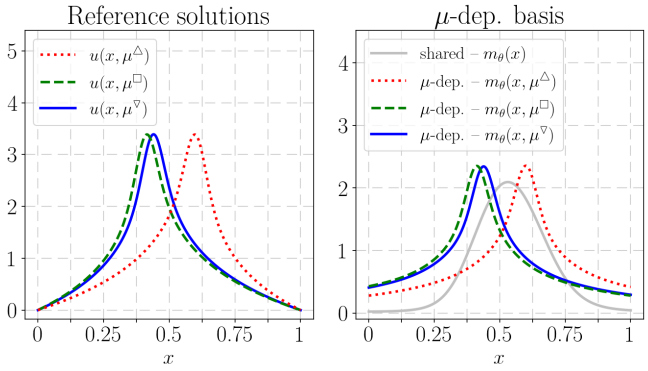


Figure 5.37.: Considering the *1D turning point problem* with $k = 2$ and $N_{\text{cells}} = 8$. Left – Reference solutions at the parameters $\mu^{\triangledown}$ (solid), $\mu^{\square}$ (dashed) and $\mu^{\triangle}$ (dotted). Right – Multipliers $m_\theta(x, \mu)$ of the μ-dependent basis at the same parameters, with the multiplier $m_\theta(x)$ of the shared basis in gray. All multipliers are those of the best seed, divided by their L^2 norm.

The three reference solutions of Figure 5.37 (left) seem to have the same amplitude, but differ by the position of their peak, located in the neighborhood of the parameters themselves. The multiplier of the shared basis (in gray), which does not depend on μ, presents a single bump towards the middle of the domain, over a width much greater than that of the layer. The shared basis is therefore only enriched in this central zone, which is a compromise between all the possible positions of the layer. Even for the parameters whose layer lies at the level of this bump, the enrichment is not concentrated on the layer and therefore does not bring much, which is consistent with gains that remain close to 1 for these parameters (Figure 5.36). For the other parameters, the layer lies outside the bump, and the enrichment is no longer located at the right place, which can explain gains lower than 1, that is, a degradation of the basis and hence of the solution. The multipliers of the μ-dependent basis have, on the contrary, a peak of about 2.4 located at the level of the layer of each of the three parameters, and their shape, a sharp peak followed by a slower decrease, is comparable to that of the reference solution. This multiplier therefore follows the displacement of the layer with μ, which allows the basis to be enriched where the solution varies rapidly, whatever the value of the parameter.

We now consider the single parameter $\mu^{\square}$ (associated with the median gain of the μ-dependent basis), in order to really compare the solutions obtained and to observe their absolute errors. In Figure 5.38, the solutions (left and middle) and the absolute errors (right) of the classical DG method and of the two bases are shown at this parameter.

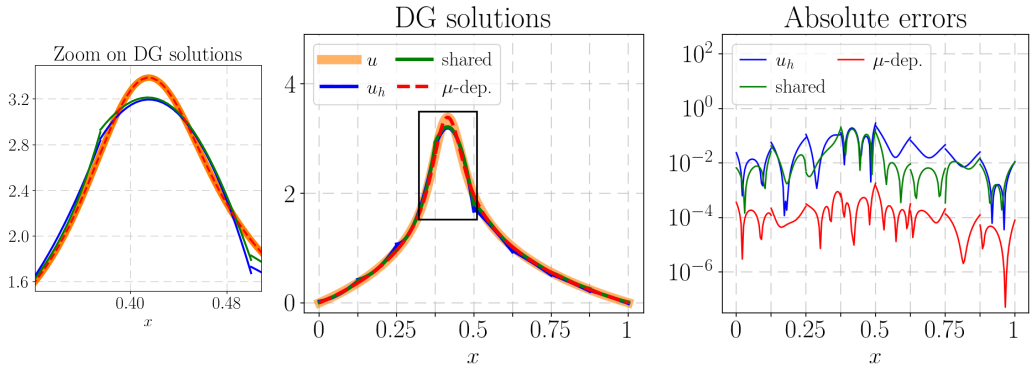


Figure 5.38.: Considering the *1D turning point problem* with $\mu = \mu^{\square} = 0.41$, $k = 2$ and $N_{\text{cells}} = 8$. Left – Zoom on the solutions around their maximum, in the rectangle of the middle panel. Middle – Reference solution u, classical DG SIPG solution u_h and enriched solutions $u_{h,\theta}$ of the shared and μ-dependent bases, labeled by their basis. Right – Corresponding absolute errors.

In Figure 5.38, the zoom shows that the classical solution u_h underestimates the maximum of the reference solution and presents jumps at the interfaces neighboring the peak. The solution of the shared basis has almost the same maximum and the same jumps, whereas that of the μ-dependent basis is indistinguishable from the reference solution at this scale. As for the errors, the classical absolute error reaches the order of 10^{-1} in the neighborhood of the layer. The error of the shared basis is of the same order in the layer, but lower than the classical error to the right of it, where it is of the order of 10^{-3} to 10^{-2}. This is consistent with the shared multiplier of Figure 5.37, whose peak lies to the right of the layer of $\mu^{\square}$ and which therefore does not enrich the basis where the solution varies the most. The error of the μ-dependent basis remains, for its part, of the order of 10^{-4} over most of Ω and only reaches the order of 10^{-3} very locally in the layer.

5.4.3. Comparison, conclusion and limitations

On the two problems, a basis enriched by a network trained once on a family of parameters can improve the DG solution for values of $\boldsymbol{\mu}$ never seen during training, but the benefit depends strongly on how the basis accounts for the parameter. For the linear transport problem, where the parameter changes the amplitude of the solution but not the position of the layer, the shared basis already does better than the classical basis at every test parameter, with median gains of the order of ten to a hundred. The $\boldsymbol{\mu}$-dependent basis is nevertheless much more accurate, with median gains that reach several thousand for $k = 3$ and for $k = 2$ on the finer meshes, with only slightly more

weights. The field-dependent basis gives results of the same order as the $\boldsymbol{\mu}$-dependent one: for this affine field, the encoder recovers the information of the parameter from the field alone, but it does not do better than the parameter itself, although it has about four times more weights. For the turning point problem, where the parameter moves the layer, the shared basis brings almost nothing (median gains between 1.0 and 1.5 for most of the configurations), whereas the $\boldsymbol{\mu}$-dependent basis does better than the classical basis at every test parameter, with median gains of at least 9. The dependence of the basis on the parameter thus seems to become indispensable when the structure of the solution moves with it.

This study has several limitations. First, it is restricted to 1D problems, to the DG scheme with a patchwise basis, and to a parameter space of dimension at most 2, so that the training sets, of 32 values of $\boldsymbol{\mu}$, are small. The test parameters are drawn in the same set $\mathcal{M}$ as the training ones, so that we do not assess any extrapolation outside of $\mathcal{M}$. The size of the test set differs between the two problems (500 and 32 values, the parameter being scalar for the turning point problem). The minimal and maximal gains, which are extreme values, are therefore not directly comparable between them, since over 32 values they are in general less extreme and more sensitive to the draw than over 500. The field-dependent basis is only tested on a single problem, where the field is entirely determined by $\boldsymbol{\mu}$, so that the encoder has nothing to compress. Moreover, the trainings depend on the initialization and on a budget of 400 epochs, and only the best seed is reported, so that the spread of the results over the seeds is not measured. A few configurations, such as the turning point problem with $k = 3$ and $N_{\text{cells}} \in \{8, 12\}$, seem to be limited by the optimization rather than by the basis, the loss reached by the best seed being higher than for $k = 2$. The multiplier also differs between the two problems (a factor 2 for the linear transport and 4 for the turning point), so that they are not compared with exactly the same settings. Finally, as in the non-parametric case, the comparison is made at an equal number of degrees of freedom and not at an equal computational cost, the cost of training, which now involves the solution of the scheme for a whole batch of parameters at each epoch, not being taken into account.

The results of this section cannot be compared directly with those of the non-parametric sections. Moving to the parametric setting changes several things at the same time: the network is trained with a supervised loss instead of the PINN loss, with another optimizer (SSBFGS instead of ENG), the number of seeds differs, and the reference solution intervenes in the training. A parametric version of the PINN loss, which would make it possible to isolate the effect of the parameter, has not been studied. A finer study of the turning point problem, relating the position of the layer with respect to the cells to the gains, would also be of interest, since the classical error and the gain of the shared basis seem to vary with this position.

Having shown, on a linear transport problem and on a turning point problem, that a basis enriched by a network trained once on a whole family of parameters can improve the Galerkin solution for values of $\boldsymbol{\mu}$ never seen during training, and that the dependence of the basis on the parameter becomes essential when the structure of the solution moves with it, we now conclude. In Chapter 6, we come back to the contributions of this manuscript and to the limits and perspectives of these results.

6 Conclusion

Contents

6.1. Summary and contributions **235**
6.2. Limitations and perspectives **236**

This thesis was set in the context of solving parametric PDEs, which must be solved a large number of times for different values of the parameters, with a particular focus on elliptic partial differential equations. The introduction identified three limitations of classical high-fidelity methods. The first is the difficulty of building a mesh that conforms to a complex geometry. The second is the prohibitive cost of repeating full resolutions as soon as the problem depends on parameters. The third is the limited expressivity of polynomial approximation spaces, which requires a very fine mesh to represent structures such as boundary layers. Neural methods partly address these limitations, but they offer neither theoretical guarantees nor reliable error estimators, and their training can be very sensitive to hyperparameters. Rather than opposing these two families of methods, we chose to hybridize them while keeping the structure of Galerkin methods, in two complementary ways. This conclusion goes back over these two contributions, compares them, and then discusses their limitations and perspectives.

6.1. Summary and contributions

The first contribution (Chapter 3) enriches the continuous Lagrange finite element space with a fixed neural prior, namely a parametric PINN trained once offline for the whole family of parameters and then injected online, either additively or multiplicatively. We proved that, in both cases, the classical convergence orders are preserved, and that the gain explicitly depends on the quality of the prior, in particular on the accuracy of its derivatives. The experiments in dimensions 1, 2 and 3 validate these results. For a given accuracy, and on average over a set of parameters, the additive approach with $\mathbb{P}_1$ elements makes it possible to use meshes 5 to 28 times coarser than the standard method, depending on the test cases. In dimension 3, to reach a relative L^2 error of 10^{-3}, it becomes more advantageous (training of the network included) as soon as 19 sets of parameters have to be solved. The comparison of the two types of enrichment shows, from a theoretical point of view, that the gain of the multiplicative approach converges to that of the additive one when its lifting parameter tends to infinity and, from a numerical point of view, that it only does better than the additive approach in very particular cases, for a good choice of this parameter.

The second contribution (Chapters 4 and 5) conversely makes the solver itself differentiable, and therefore learnable. We developed, in the `JAX` version of the `ScimBa`

library, the modules of a differentiable discontinuous Galerkin scheme, extended to continuous finite elements and to φ-FEM, as well as the learning formalism, which relies on implicit differentiation and the adjoint method and makes it possible to train a network directly through the solver. We first focused on the problem of the limited expressivity of polynomial approximation spaces, by enriching Galerkin bases with a neural network. For a network trained for a given value of the parameter, the numerical results showed, in dimension 1, that for a fixed number of degrees of freedom, the L^2 error is reduced by several orders of magnitude compared with the classical basis. In dimension 3, where the cost of solving with classical methods explodes with the number of degrees of freedom, we obtained errors of the same order in comparable computation times, but with far fewer degrees of freedom. We then turned to the second limitation, the prohibitive cost of repeating full resolutions as soon as the problem depends on parameters. A single network is then trained, from reference solutions and in the manner of a neural operator, to build the enriched basis for a whole family of parameters, and then used without any new training for values of the parameter never seen. This study, carried out in dimension 1 on a linear transport problem and on a turning point problem, showed that a parameter-dependent basis improves the classical solution in terms of the L^2 error at every test parameter and, at an equal number of degrees of freedom, with median gains ranging from about 9 (turning point problem) to about 7000 (linear transport problem), depending on the configuration (N_{cells}, k).

These two contributions provide different answers to the initial limitations. Both enrich the approximation space to overcome its lack of expressivity, and both aim at reducing the cost of a resolution repeated for many parameters, which is demonstrated for the first one, and still preliminary but promising for the second one. The question of the geometry is, however, only partially addressed, with an annular domain for the first one and, for the second one, a 3-lobed domain treated with a learned mapping and with the implementation of φ-FEM in `ScimBa`. They differ above all in the compromise they offer between guarantees and expressivity. In the first one, the prior is trained then frozen, and the error of the enriched scheme is controlled a priori, but the gain remains bounded by the quality of the prior. In the second one, the basis is learned through the solver and adapts to the solution of the scheme instead of depending on a frozen prior, but it comes with no error estimate, and each training iteration requires solving the scheme.

6.2. Limitations and perspectives

For the first contribution, the gain degrades when the accuracy of the prior decreases, as shown by the Navier–Stokes case at a high Rayleigh number, and the theory remains limited to scalar linear elliptic problems. Several perspectives follow, such as improving the prior (neural operators, penalizing the derivatives in the training loss), then extending the method to time-dependent problems, non-linear problems and complex geometries, for example by combining the neural prior with φ-FEM.

For the second contribution, the results remain exploratory and constitute a feasibility study rather than a competitive solver. They rely on few test cases, the training is sensitive to the initialization, the parametrization of the multiplier is fixed by hand, and the approach fails for $k = 1$ in the non-parametric 1D study with continuous

elements. The comparison with classical methods is mainly made for an equal number of degrees of freedom and with uniform meshes only. The training cost is discussed only qualitatively in dimension 1 and ignored in the parametric case, and a comparison at comparable computation time is carried out only in dimension 3, for a single value of the parameter. It therefore remains to compare the learned bases with classical methods more systematically at equal computational cost, training included, and against adapted or graded meshes, and to study more test cases, in particular in dimension 2 in the parametric setting. The most direct way to recover, at least in part, guarantees on the error of the learned basis is to freeze the multiplier once it has been learned. The enriched basis then coincides with that of the multiplicative approach of the first contribution, so that its a priori analysis (Theorem 3.4) would apply and give a Céa-type estimate, whose gain depends on the derivatives of the solution and of the multiplier. This is the perspective that most directly links the two contributions. On the library side, extensions to time-dependent problems and to structured meshes are under way within the MACARON team, and shape optimization with φ-FEM, where the level-set itself becomes a learned object, is a natural use case of this differentiable framework.

More broadly, this work shows that it is not necessary to oppose classical numerical methods and neural networks. By keeping the structure of Galerkin methods while injecting neural information into it, it is possible to take advantage of the expressivity of networks while keeping, for the first contribution, the guarantees that make the strength of classical methods, and while seeking to recover part of them for the second one.

A Notations and acronyms

Contents

A.1. Notations and definitions **239**
A.1.1. Numerical methods for PDEs 240
A.1.2. Enriched finite element methods 244
A.1.3. SciML library and learnable Galerkin methods 246
A.2. Acronyms **250**

This appendix gathers the reference tables of the manuscript. The first section lists the notations introduced in the chapters, and the second one the acronyms and abbreviations.

A.1. Notations and definitions

The aim of this section is to introduce the notations used throughout the manuscript.

Table A.1.: *Some common notations (function spaces, norms, etc.).*

Notation	Description
$L^2(\Omega)$	Space of square-integrable functions on Ω
$H^s(\Omega)$	Sobolev space of functions with square-integrable derivatives up to order s on Ω
$H^1(\Omega)$	Sobolev space of functions with square-integrable derivatives on Ω
$H^1_0(\Omega)$	Sobolev space of functions in $H^1(\Omega)$ that vanish on $\partial\Omega$
$L^\infty(\Omega)$	Space of functions bounded almost everywhere on Ω
$W^{1,\infty}(\Omega)$	Sobolev space of functions in $L^\infty(\Omega)$ whose 1st derivatives are also in $L^\infty(\Omega)$
$\lvert\cdot\rvert_{H^s(\Omega)}$	Semi-norm in the Sobolev space $H^s(\Omega)$
$\lVert\cdot\rVert_{H^s(\Omega)}$	Norm in the Sobolev space $H^s(\Omega)$
$\lVert\cdot\rVert_{L^2(\Omega)}$	Norm in the space $L^2(\Omega)$
$\lVert\cdot\rVert_2$	Euclidean norm
k	Polynomial degree of the approximation spaces
$\mathbb{P}_k$	Space of polynomials of total degree at most k (natural on a simplex)
$\mathbb{Q}_k$	Space of polynomials of degree at most k in each variable separately (natural on a hypercube)

Table A.2.: *Section 2.1 – Parametric PDE.*

Notation	Description
Ω	Physical spatial domain
$\Gamma := \partial\Omega$	Boundary of the physical spatial domain
d	Physical spatial dimension
$\boldsymbol{x} = (x_1, \dots, x_d)$	Spatial coordinates
$\mathcal{M}$	Parameter space
p	Number of parameters
$\boldsymbol{\mu} = (\mu_1, \dots, \mu_p)$	Parameter vector
u	Solution of the problem
M	Lifting constant
u_M	Solution of the lifted problem (by M)
$\mathcal{L}$	Parametric differential operator of the PDE
$\mathcal{L}^*$	Adjoint operator of $\mathcal{L}$
R	Reaction coefficient
$\boldsymbol{C}$	Convection coefficient
D	Diffusion matrix
Pe	Péclet number
f	Right-hand side (source term) of the PDE
g	Prescribed boundary data on Γ (Dirichlet, Neumann or Robin)

A.1.1. Numerical methods for PDEs

Chapter 2 is dedicated to the classical numerical methods used throughout the manuscript. The following tables gather their notations: the standard finite element method, the φ-FEM method, the discontinuous Galerkin method, and physics-informed neural networks.

Table A.3.: *Section 2.2.1 – Standard Finite element method.*

Notation	Description
Ω	Physical spatial domain *(reminder, see Table A.2)*
$\Gamma := \partial\Omega$	Boundary of the physical spatial domain *(reminder, see Table A.2)*
h	Characteristic size of the mesh
$\mathcal{T}_h$	Computational mesh on Ω (simplexes in Chapter 3, deformed hypercubes in Chapters 4 and 5)
$V^0 \subset V$	Variational spaces $H_0^1(\Omega) \subset H^1(\Omega)$ of the continuous problem (with/without homogeneous Dirichlet BCs)

$V_h^0 \subset V_h$	FE approximation spaces of degree k (with/without homogeneous Dirichlet BCs)
u_h	FE approximation of u in V_h
v_h	Test function in the FE approximation space V_h^0
k	Polynomial degree of the approximation spaces *(reminder, see Table A.1)*
$\mathcal{I}_h$	Lagrange interpolation operator on $\mathcal{T}_h$, of degree k
ϕ_i	Lagrange shape function associated with $\boldsymbol{x}^{(i)}$
$\boldsymbol{x}^{(i)}$	Coordinates of the i-th dof in the FE space
N_{dofs}	Number of dofs in the FE space
a	Bilinear form associated with $\mathcal{L}$ (continuous FE)
l	Linear form associated with $\mathcal{L}$ (continuous FE)
γ	Continuity constant of the bilinear form a
α	Coercivity constant of the bilinear form a
C_s, C_e, C_1	Constants in error estimates of FE approximation

Table A.4.: *Section 2.2.2 – Discontinuous Galerkin method.*

Notation	**Description**
Ω	Physical spatial domain *(reminder, see Table A.2)*
$\Gamma := \partial\Omega$	Boundary of the physical spatial domain *(reminder, see Table A.2)*
h	Characteristic size of the mesh *(reminder, see Table A.3)*
$\mathcal{T}_h$	Computational mesh on Ω *(reminder, see Table A.3)* (deformed hypercubes in Chapters 4 and 5)
$\mathcal{F}_h = \mathcal{F}_h^I \cup \mathcal{F}_h^B$	Set of all faces of the mesh $\mathcal{T}_h$
$\mathcal{F}_h^I$	Set of interior interfaces (faces shared by exactly two elements)
$\mathcal{F}_h^B$	Set of faces lying on Γ
$K \in \mathcal{T}_h$	Arbitrary element of the mesh
$K_L, K_R \in \mathcal{T}_h$	Two elements (left and right) of the mesh sharing an interior face
$F \in \mathcal{F}_h$	Arbitrary face of the mesh
h_F	Diameter of the face F
$\boldsymbol{n}_K$	Outward physical unit normal vector to an arbitrary element $K \in \mathcal{T}_h$
$\{\cdot\}, [\![\cdot]\!]$	Average and jump trace operators on $F \in \mathcal{F}_h^I$ (on $\mathcal{F}_h^B$: only the jump of a scalar and the average of a vector are defined)
∇_h	Broken gradient (ignoring jumps across interfaces)
k	Polynomial degree of the approximation spaces *(reminder, see Table A.3)*
V_h^{disc}	Discontinuous Lagrange FE spaces of degree k *(analogue, see Table A.3)*
u_h	DG approximation of u in V_h^{disc} *(analogue, see Table A.3)*

v_h, v_K	Test function in V_h^{disc} (resp. restricted to $K \in \mathcal{T}_h$) *(analogue, see Table A.3)*
Σ_h^{disc}	Discontinuous vector-valued Lagrange FE space of degree k
$\boldsymbol{\varsigma}$	Auxiliary flux variable (first-order reformulation of diffusion)
$\boldsymbol{\varsigma}_h$	DG approximation of $\boldsymbol{\varsigma}$ in Σ_h^{disc}
$\boldsymbol{\tau}_h$, $\boldsymbol{\tau}_K$	Generic vector test function in Σ_h^{disc} (resp. restricted to $K \in \mathcal{T}_h$)
$\hat{\boldsymbol{\varsigma}}_h^D$, $\hat{u}_h$	Vector and scalar numerical fluxes of the diffusive term, on $\mathcal{F}_h$
a_h^{disc}, l_h^{disc}	Bilinear and linear forms of the unified DG primal formulation
a_h^{sipg}	Special case of a_h^{disc} for the SIPG flux choice
a_h^{nipg}	Special case of a_h^{disc} for the NIPG flux choice
$\mathcal{S}_h^{\text{disc}}$	Stabilization (interior penalty) bilinear form (SIPG/NIPG)
η_F	Penalty parameter on face $F \in \mathcal{F}_h$ (SIPG/NIPG)
κ	Penalty function on $\mathcal{F}_h$ (SIPG/NIPG)
$\hat{u}_h^C$	Upwind scalar numerical flux for the convective term, on $\mathcal{F}_h$
c_h^{up}	Bilinear form of the upwind convective flux (added to a_h^{sipg} or a_h^{nipg})

Table A.5.: *Section 2.2.3 – φ-FEM method.*

Notation	**Description**
$\mathcal{O}$	Simply-shaped background domain (e.g. a box) containing Ω
Ω	Physical spatial domain *(reminder, see Table A.2)*
Ω_h	Computational domain covered by $\mathcal{T}_h$ (slightly larger than Ω in general)
Ω_h^Γ	Narrow strip covered by $\mathcal{T}_h^\Gamma$
$\Gamma := \partial\Omega$	Boundary of the physical spatial domain *(reminder, see Table A.2)*
Γ_h	Approximate (piecewise polynomial) boundary, $\Gamma_h := \{\varphi_h = 0\}$
$\partial\Omega_h$	Boundary of the computational domain Ω_h
h	Characteristic size of the mesh *(reminder, see Table A.3)*
$\mathcal{T}_h^{\mathcal{O}}$	Quasi-uniform background mesh on $\mathcal{O}$ (deformed hypercubes in Chapter 4)
$\mathcal{T}_h \subset \mathcal{T}_h^{\mathcal{O}}$	Computational mesh on Ω_h, unfitted to Γ *(analogue, see Table A.3)*
$\mathcal{T}_h^\Gamma \subset \mathcal{T}_h$	Elements of the computational mesh $\mathcal{T}_h$ cut by Γ_h
$\mathcal{F}_h^\Gamma$	Interior facets of $\mathcal{T}_h$ belonging to a cut element of $\mathcal{T}_h^\Gamma$
k	Polynomial degree of the approximation spaces *(reminder, see Table A.1)*
l	Polynomial degree used to interpolate φ (with $l \geqslant k$)
$\mathcal{I}_h^{\mathcal{O}}$	Lagrange interpolation operator on $\mathcal{T}_h^{\mathcal{O}}$, of degree l
φ	Level-set function of the domain Ω, such that $\Gamma = \{\varphi = 0\}$
φ_h	Lagrange interpolation of φ on $\mathcal{T}_h^{\mathcal{O}}$, of degree l
V_h	FE approximation space of degree k *(analogue, see Table A.3)*
u_h	φ-FEM approximation of u in V_h, living on Ω_h *(analogue, see Table A.3)*

w_h	Auxiliary φ-FEM unknown in V_h, such that $u_h = \varphi_h w_h$
v_h	Test function in V_h, multiplied by φ_h *(analogue, see Table A.3)*
$\boldsymbol{n}$	Outward unit normal to Ω_h
$[\cdot]$	Jump of a scalar quantity across a facet $F \in \mathcal{F}_h^\Gamma$ of a cut element
a_h^{phi}	Bilinear form of the discrete φ-FEM problem *(analogue, see Table A.3)*
l_h^{phi}	Linear form of the discrete φ-FEM problem *(analogue, see Table A.3)*
$\mathcal{S}_h$, $\mathcal{S}_h^{\text{rhs}}$	Stabilization applied to bilinear and linear forms
τ	Stabilization parameter (independent of h)
C	Constant in the φ-FEM error estimate (independent of h, f and u)

Table A.6.: *Section 2.3 – Physics-informed neural networks.*

Notation	Description
u_θ	Neural network prediction of u
$u_{\theta,M}$	Neural network prediction of u_M *(reminder, see Table A.7)*
θ	Trainable weights of the neural network
$\theta^\star$	Optimal parameters of θ
φ	Level-set function used to strongly impose boundary conditions *(analogue, see Table A.5)*
w_θ	Raw neural network, before strong imposition of boundary conditions, $u_\theta = \varphi w_\theta + g$ *(analogue, see Table A.5)*
$\mathcal{J}_r$	Residual loss
$\mathcal{J}_b$	Boundary loss
$\mathcal{J}_{\text{data}}$	Data loss
$\mathcal{J}_{\text{sob}}$	Sobolev loss
$\omega_r, \omega_b, \omega_{\text{data}}, \omega_{\text{sob}}$	Weights balancing the terms of the PINN loss function
N_{col}, N_{bc}, N_{data}	Numbers of collocation, boundary and data points
$\boldsymbol{x}_{\text{col}}^{(i)}$, $\boldsymbol{x}_{\text{bc}}^{(i)}$, $\boldsymbol{x}_{\text{data}}^{(i)}$	Collocation, boundary and data sample points
$\boldsymbol{\mu}_{\text{col}}^{(i)}$, $\boldsymbol{\mu}_{\text{bc}}^{(i)}$, $\boldsymbol{\mu}_{\text{data}}^{(i)}$	Parameter values associated with the collocation, boundary and data points
$u_{\text{data}}^{(i)}$	Reference solution value at a data point
σ	MLP activation function
N_{epochs}	Number of training epochs
N_{switch}	Epoch at which training switches from Adam to LBFGS
`layers`	Sequence of integers giving the number of neurons of each MLP layer
`lr`	Learning rate
`decay`	Multiplicative factor of the learning-rate decay schedule
N_{FF}	Number of Fourier features
a_i, b_i	Trainable Fourier-feature frequencies

A.1.2. Enriched finite element methods

Chapter 3 is entirely dedicated to enriching the continuous FE approximation with a neural-network prior, in either an additive or a multiplicative way. The corresponding notations are gathered in the table below.

Table A.7.: *Chapter 3 – Enriched Finite element methods.*

	Notation	Description
Additive	V_h^+	FE space enriched with the additive prior
	u_h^+	FE approximation of u in V_h^+
	p_h^+	FE approximation of $u - u_\theta$ in V_h^0
	C_{gain}^+	Additive gain constant in both H^1 semi-norm and L^2 norm
Multiplicative	$V_h^\times$	FE space enriched with the multiplicative prior
	$u_{h,M}^\times$	FE approximation of u_M in $V_h^\times$
	$u_h^\times$	Solution of the original problem recovered from $u_{h,M}^\times$
	$p_h^\times$	FE approximation of $u_M/u_{\theta,M}$ in $1 + V_h^0$
	$\phi_i^\times$	Lagrange shape functions of $V_h^\times$ associated with $\boldsymbol{x}^{(i)}$
	$\mathcal{I}_h^\times$	Modified Lagrange interpolation operator on $V_h^\times$
	$C_{\text{gain},H^1}^{\times,M}$	Multiplicative gain constant in H^1 semi-norm
	$C_{\text{gain},L^2}^{\times,M}$	Multiplicative gain constant in L^2 norm
	$C_{\theta,M}$	Constant entering $C_{\text{gain},L^2}^{\times,M}$

Table A.8.: *Section 3.5.1 – Setup of the numerical experiments.*

Notation	Description
$\mathcal{D} = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N_\mu)}\}$	Set of N_μ parameter instances sampled from $\mathcal{M}$
N_μ	Number of parameter instances in $\mathcal{D}$
$\boldsymbol{\mu}^{(j)}$	j-th parameter instance of $\mathcal{D}$, $j = 1, \dots, N_\mu$
N	Number of nodes per direction of the Cartesian mesh (i.e. N^d total nodes in dimension d)
$u^{(j)}$	Reference solution for parameter $\boldsymbol{\mu}^{(j)}$
$u_h^{(j)}$	Standard FE approximation of $u^{(j)}$
$u_\theta^{(j)}$	PINN approximation of $u^{(j)}$
$u_{h,+}^{(j)}$	Additive-approach FE approximation of $u^{(j)}$ in V_h^+
$u_{h,M}^{(j)}$	Multiplicative-approach FE approximation of $u_M^{(j)}$ in $V_h^\times$

h_{ref}, k_{ref}	Mesh and degree of the polynomial for the reference solution calculated by FE (if no analytical solution is provided)
$e_h^{(j)}$	L^2 relative errors of the standard FEM for parameter $\boldsymbol{\mu}^{(j)}$
$e_\theta^{(j)}$	L^2 relative errors of the PINN for parameter $\boldsymbol{\mu}^{(j)}$
$e_{h,+}^{(j)}$, $e_{h,M}^{(j)}$	L^2 relative errors of the additive and multiplicative approaches, for parameter $\boldsymbol{\mu}^{(j)}$
$G_{+,\theta}^{(j)}$, $G_+^{(j)}$	Gains of the additive approach over the PINN and the standard FEM, for parameter $\boldsymbol{\mu}^{(j)}$
$G_{M,\theta}^{(j)}$, $G_M^{(j)}$	Gains of the multiplicative approach over the PINN and the standard FEM, for parameter $\boldsymbol{\mu}^{(j)}$
$G_{+,\theta}$, G_+, $G_{M,\theta}$, G_M	Samples of the above gains over $\mathcal{D}$

Table A.9.: *Section 3.6 – 2D Navier–Stokes test case.*

Notation	**Description**
$\boldsymbol{U} = (\boldsymbol{u}, P, T)$	Unknowns: velocity $\boldsymbol{u}$, pressure P and temperature T
$\boldsymbol{u} = (u_1, u_2)$	Velocity and its two components
$g_0 = 9.81$	Gravitational constant
$\boldsymbol{\mu} = (\nu, k_f)$	Test case parameters, namely viscosity thermal conductivity
Ra	Rayleigh number, which increases as $\boldsymbol{\mu}$ decreases
Γ_{ad}	Adiabatic (top and bottom) walls of the cavity
$\boldsymbol{U}_\theta = (\boldsymbol{u}_\theta, P_\theta, T_\theta)$	Parametric PINN prediction (velocity, pressure and temperature)
$\boldsymbol{u}_\theta = (u_{1,\theta}, u_{2,\theta})$	Velocity component of the PINN prediction and its two components
φ_1, φ_2	Level-set functions imposing BC exactly *(analogue, see Table A.6)*
$\mathcal{J}_{\text{mom}}$, $\mathcal{J}_{\text{inc}}$, $\mathcal{J}_{\text{ener}}$, $\mathcal{J}_{\text{ad}}$	Residual (momentum, incompressibility, energy) and adiabatic losses
ω_{mom}, ω_{inc}, ω_{ener}, ω_{ad}	Weights balancing the terms of the PINN loss
Q_h, W_h	Continuous Lagrange spaces for the pressure and the temperature
W_h^0	Continuous Lagrange space for the temperature (vanish on $\{x_1 = \pm 1\}$)
$X_h = [V_h^0]^2 \times Q_h \times W_h$	Taylor–Hood mixed FE space for the velocity, pressure and temperature
$X_h^0 = [V_h^0]^2 \times Q_h \times W_h^0$	Test space for the velocity, pressure and temperature
$\boldsymbol{U}_h = (\boldsymbol{u}_h, P_h, T_h)$	Standard FE approximation in X_h
$(\boldsymbol{v}_h, q_h, w_h)$	Test functions in X_h^0
X_h^+	Mixed space additively enriched by the PINN prior
$\boldsymbol{p}_h^+ = (\boldsymbol{p}_{h,\boldsymbol{u}}^+, p_{h,P}^+, p_{h,T}^+)$	Correction in X_h^0, unknown of the additive problem
$\boldsymbol{\beta}_h$, $\boldsymbol{\beta}_h^+$	Degrees of freedom of $\boldsymbol{U}_h$ and of the correction $\boldsymbol{p}_h^+$, unknowns of the Newton solver
F, F_θ	Non-linear operators of the standard and additive discrete problems
$\boldsymbol{\beta}_h^{(0)}$	Initialization of the Newton solver (standard FE problem)
$\boldsymbol{U}_{\theta,h}$	Degrees of freedom of the interpolation of $\boldsymbol{U}_\theta$ in X_h

$\boldsymbol{U}^{(j)}$	Reference solution for parameter $\boldsymbol{\mu}^{(j)}$
$\boldsymbol{U}_\theta^{(j)}$	Parametric PINN prediction for parameter $\boldsymbol{\mu}^{(j)}$
$\boldsymbol{U}_h^{(j)}$	Standard FE approximation of $\boldsymbol{U}^{(j)}$
$\boldsymbol{U}_{h,+}^{(j)}$	Additive-approach FE approximation of $\boldsymbol{U}^{(j)}$

A.1.3. SciML library and learnable Galerkin methods

The last two chapters of the manuscript, Chapter 4 and Chapter 5, introduce the `ScimBa` differentiable library and the learnable DG solver (with its extensions) built on top of it. The following tables gather their notations, module by module.

Table A.10.: *Section 4.2 – Geometry and integration modules of* `ScimBa`.

	Notation	Description
Reference domain	$\hat{K} = [0,1]^d$	Reference hypercube element
	ℓ	Direction in the reference/logical domain, $\ell \in \{1, \dots, d\}$
	$\hat{F}^{(\ell)}$	Reference face of $\hat{K}$ in the direction ℓ
	$\hat{\boldsymbol{\xi}} = (\hat{\xi}_1, \dots, \hat{\xi}_d)$	Arbitrary point in the reference domain $\hat{K}$
	$\hat{\boldsymbol{\xi}}_i$, $\hat{w}_i$	Volumetric quadrature points and weights on $\hat{K}$
	$\hat{\boldsymbol{\xi}}_{\ell,i}^{\text{surf}}$, $\hat{w}_i^{\text{surf}}$	Surface quadrature points and weights on $\hat{F}^{(\ell)}$
	q	Number of 1D quadrature points per direction
Logical domain	$\widetilde{\Omega} = [0,1]^d$	Logical domain (unit hypercube)
	$\tilde{\boldsymbol{\xi}} = (\tilde{\xi}_1, \dots, \tilde{\xi}_d)$	Arbitrary point in the logical domain $\widetilde{\Omega}$
	$\widetilde{\mathcal{T}}_h$	Uniform logical mesh on the logical domain $\widetilde{\Omega}$
	$\boldsymbol{N}_{\text{cells}} = (N_1, \dots, N_d)$	Tuple of the number of cells per direction in the logical mesh $\widetilde{\mathcal{T}}_h$
	N_{cells}	Total number of cells in the logical/physical mesh (number of active cells for φ-FEM)
	$\tilde{K}_c \in \widetilde{\mathcal{T}}_h$	Logical cell of flat index c
	$c, \bar{\boldsymbol{c}}(c) = (\bar{c}_1, \dots, \bar{c}_d)$	Flat and multi-dimensional index of cell $\tilde{K}_c$
	$\widetilde{\mathcal{F}}_h = \widetilde{\mathcal{F}}_h^I \cup \widetilde{\mathcal{F}}_h^B$	Set of all faces of the logical mesh $\widetilde{\mathcal{T}}_h$ (logical counterpart of $\mathcal{F}_h$)
	$\widetilde{\mathcal{F}}_h^I$	Set of internal faces of the logical mesh
	$\widetilde{\mathcal{F}}_h^B$	Set of external (boundary) faces of the logical mesh
	N_{faces}	Total number of faces in the logical/physical mesh
	$\tilde{F}_k \in \widetilde{\mathcal{F}}_h$	Logical face of flat index k
	$k, \bar{\boldsymbol{k}}(k) = (\ell; \bar{k}_1, \dots, \bar{k}_d)$	Flat and multi-dimensional index of face $\tilde{F}_k$ of group ℓ
	$\boldsymbol{e}_\ell$	Canonical basis vector, normal to a logical face of group ℓ
	$T_{\tilde{K}_c} : \hat{K} \to \tilde{K}_c$	Local affine transformation from $\hat{K}$ (reference) to $\tilde{K}_c$ (logical)
	$T_{\tilde{F}_k} : \hat{F}^{(\ell)} \to \tilde{F}_k$	Local affine transformation from $\hat{F}^{(\ell)}$ (reference) to $\tilde{F}_k$ (logical)

Mapping and physical domain	θ	Learnable parameters – mapping
	$\mathcal{G}_\theta$	Global mapping from $\widetilde{\Omega}$ (logical domain) to Ω (physical domain)
	$\mathcal{G}_{\theta,j}$	j-th invertible mapping of $\mathcal{G}_\theta$, $j \in \{1, \dots, N_{\text{maps}}\}$, $\mathcal{G}_{\theta,j} : \Omega_{j-1} \to \Omega_j$
	N_{maps}	Number of elementary mappings composed into $\mathcal{G}_\theta$
	Ω	Physical domain *(reminder, see Table A.2)*
	$\boldsymbol{x} = (x_1, \dots, x_d)$	Arbitrary point in the physical domain Ω *(reminder, see Table A.2)*
	$\Omega_0,\ \Omega_1,\ \dots,\ \Omega_{N_{\text{maps}}}$	Chain of intermediate domains successively produced by $\mathcal{G}_\theta$, $\Omega_0 = \widetilde{\Omega}$, $\Omega_{N_{\text{maps}}} = \Omega$
	$\boldsymbol{\xi}_j \in \Omega_j$	Point after applying the first j mappings of $\mathcal{G}_\theta$ to $\tilde{\boldsymbol{\xi}}$, $j \in \{0, \dots, N_{\text{maps}}\}$, $\boldsymbol{\xi}_0 = \tilde{\boldsymbol{\xi}}$, $\boldsymbol{\xi}_{N_{\text{maps}}} = \boldsymbol{x}$
	$\mathcal{T}_h$	Physical mesh, $\mathcal{T}_h = \mathcal{G}_\theta(\widetilde{\mathcal{T}}_h)$ *(reminder, see Table A.4)*
	$K_c \in \mathcal{T}_h$	Physical cell of flat index c, $K_c = \mathcal{G}_\theta(\tilde{K}_c)$
	$\boldsymbol{n}_K$	Physical outward unit normal vector to an arbitrary element $K \in \mathcal{T}_h$ *(reminder, see Table A.4)*
	$\mathcal{F}_h = \mathcal{F}_h^I \cup \mathcal{F}_h^B$	Set of all faces of the physical mesh $\mathcal{T}_h$ *(reminder, see Table A.4)*
	$\mathcal{F}_h^I$	Set of interior interfaces of $\mathcal{T}_h$ *(reminder, see Table A.4)*
	$\mathcal{F}_h^B$	Set of faces lying on Γ *(reminder, see Table A.4)*
	$F_k \in \mathcal{F}_h$	Physical face of flat index k, $F_k = \mathcal{G}_\theta(\tilde{F}_k)$
	$T_{K_c} = \mathcal{G}_\theta \circ T_{\tilde{K}_c}$	Full volumetric mapping chain, from $\hat{K}$ to K_c via $\tilde{K}_c$ (Figure 4.4)
	$T_{F_k} = \mathcal{G}_\theta \circ T_{\tilde{F}_k}$	Full surface mapping chain, from $\hat{F}^{(\ell)}$ to F_k via $\tilde{F}_k$ (Figure 4.5)
	$w_{c,i}$	Corrected (physical) volumetric quadrature weight on cell K_c
	$w_{k,i}^{\text{surf}}$	Corrected (physical) surface quadrature weight on face F_k

Table A.11.: *Section 4.3 – Basis functions modules of* `ScimBa`.

Notation	Description
k	Polynomial degree of the approximation spaces *(reminder, see Table A.3)*
$\mathbb{P}_k$	Space of polynomials of total degree at most k *(reminder, see Table A.3)*
$\mathbb{Q}_k$	Space of polynomials of degree at most k in each variable separately *(reminder, see Tables A.3 and A.4)*
N_b	Number of basis functions per cell (depend on k for polynomial bases)
$\hat{\phi}_i$	Reference polynomial basis function on $\hat{K}$, $i \in \{1, \dots, N_b\}$
$\phi_{c,i}$	i-th analytical local (trial) basis function on the physical cell K_c
θ	Learnable parameters – mapping, basis *(reminder, see Table A.10)*
v_θ	Patchwise neural networks (shared across all cells)
$v_{c,\theta}$	Cellwise neural networks on each cell K_c
m_θ	Patchwise multiplier built from the network v_θ
$m_{c,\theta}$	Cellwise multiplier built from the network $v_{c,\theta}$ on the cell K_c

φ_c	Level-set function local to the cell K_c *(analogue, see Table A.5)*
$\phi^\theta_{c,i}$	i-th trial basis function on K_c (analytical, patchwise or cellwise)
$\psi^\theta_{c,j}$	j-th test basis function on K_c (analytical, patchwise or cellwise)

Table A.12.: *Section 4.4 – DG variable and post-processing modules of* `ScimBa`.

Notation	Description
$\beta_{c,i}$	Linear DOF (scalar) associated with basis function $\phi^\theta_{c,i}$ on cell K_c
$\boldsymbol{\beta}_c = (\beta_{c,i})_{i=1}^{N_b}$	Local DOF vector on cell K_c
θ	Learnable parameters – mapping, basis, post-processing *(reminder, see Tables A.10 and A.11)*
$w_{h,\theta}$	Discrete solution reconstructed from the DOFs, before post-processing
$u_{h,\theta}$	Post-processed discrete solution, $u_{h,\theta}(\boldsymbol{x}) = \mathcal{P}_\theta(w_{h,\theta}(\boldsymbol{x}), \boldsymbol{x})$
$\mathcal{P}_\theta$	Local post-processing operator

Table A.13.: *Sections 4.5, 4.7 and 4.8 – Scheme assembly modules of* `ScimBa`.

	Notation	Description
Common	$\boldsymbol{R}_h$	Discrete residual of the scheme (DG, FEM or φ-FEM), of the same shape as $\boldsymbol{\beta}_h$
	A_h, $\boldsymbol{b}_h$	Matrix and right-hand side in the affine (linear-PDE-field) case
DG	$\boldsymbol{\beta}_h = (\boldsymbol{\beta}_c)_{c=0}^{N_{\text{cells}}-1}$	Global DOF vector, obtained by concatenating the local DOF vectors $\boldsymbol{\beta}_c$ over all cells of the mesh
	$N_{\text{dofs}} = N_{\text{cells}} \cdot N_b$	Total number of dofs of the DG scheme (i.e. the size of $\boldsymbol{\beta}_h$)
	r^Ω	Volume residual (integrand of the DG volume terms)
	$r^{\mathcal{F}_h^I}(u, v)$	Interior residual on an interior face $F \in \mathcal{F}_h^I$
	$r^{\mathcal{F}_h^B}(u, v)$	Boundary residual on a boundary face $F \in \mathcal{F}_h^B$
FEM	**conn**	Connectivity table mapping each pair (cell, local shape function) to the global index of the mesh node carrying it
	I	Global index of a mesh node, $I \in \{1, \dots, N_{\text{dofs}}\}$
	β_I	Dof attached to the mesh node of global index I
	$\boldsymbol{\beta}_h = (\beta_I)_{I=1}^{N_{\text{dofs}}}$	Global vector of the shared dofs of the continuous FE scheme
	$N_{\text{dofs}} = \prod_{\ell=1}^{d}(N_\ell k + 1)$	Total number of dofs of the continuous FE scheme
	$\boldsymbol{x}^{(I)}$	Physical position of the node of global index I
φ-FEM	$N^{\mathcal{O}}_{\text{dofs}}$	Total number of dofs of the background mesh $\mathcal{T}_h^{\mathcal{O}}$ (active and inactive nodes)

Table A.14.: *Section 4.9 – Learnable formalization of the Galerkin schemes.*

Notation	Description
θ	Learnable parameters – mapping, basis, post-processing, PDE fields *(reminder, see Tables A.10 to A.12)*
N_θ	Number of learnable weights (size of θ)
$\boldsymbol{\beta}_c(\theta)$, $\boldsymbol{\beta}_h(\theta)$	Local/global DOF vectors, with their explicit dependence on θ *(reminder, see Tables A.12 and A.13)*
$\boldsymbol{R}_h(\boldsymbol{\beta}_h(\theta);\theta)$	Residual of the Galerkin scheme, with its explicit dependence on θ *(reminder, see Table A.13)*
$A_h(\theta)$, $\boldsymbol{b}_h(\theta)$	Matrix and right-hand side in the affine case, with their explicit dependence on θ *(reminder, see Table A.13)*
$\mathcal{J}(\boldsymbol{\beta}_h(\theta);\theta)$	Training loss functional
$\boldsymbol{\lambda}_h$	Adjoint variable, $J_{\boldsymbol{R}_h}(\boldsymbol{\beta}_h;\theta)^T\boldsymbol{\lambda}_h = (\partial\mathcal{J}/\partial\boldsymbol{\beta}_h)^T$
R_θ	Learnable reaction coefficient (PDE field parameterized by θ, e.g. a neural network), in the inverse-problem setting

Table A.15.: *Chapter 5 – Learnable Galerkin methods.*

	Notation	Description
PINN	m_θ, $m_{c,\theta}$	Patchwise and cellwise multipliers built from v_θ and $v_{c,\theta}$ *(reminder, see Table A.11)*
	softclip(W)	Soft clipping of the weight matrix W of a layer
NO	$m_\theta(x)$	Multiplier of the shared basis, the same enriched space for all $\boldsymbol{\mu}$
	$m_\theta(x,\boldsymbol{\mu})$	Multiplier of the $\boldsymbol{\mu}$-dependent basis
	$m_\theta(x,\boldsymbol{z})$	Multiplier of the field-dependent basis
	$\boldsymbol{z} = \mathcal{E}_\theta\big(C(\cdot,\boldsymbol{\mu})\big)$	Latent state, obtained by encoding the transport field $C(\cdot,\boldsymbol{\mu})$
	$\mathcal{E}_\theta$	Encoder network mapping the transport field to the latent state
	$\boldsymbol{\mu}^\triangledown$, $\boldsymbol{\mu}^\square$, $\boldsymbol{\mu}^\triangle$	Parameters of minimal, median and maximal gain of the $\boldsymbol{\mu}$-dependent basis (scalars $\mu^\triangledown$, $\mu^\square$, $\mu^\triangle$ for the turning point problem)
	$\boldsymbol{z}^\triangledown$, $\boldsymbol{z}^\square$, $\boldsymbol{z}^\triangle$	Latent states associated with these three parameters
	$\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{test}}$	Sets of training and test values of the parameter $\boldsymbol{\mu}$, drawn in $\mathcal{M}$

A.2. Acronyms

The aim of this section is to list the acronyms and abbreviations used throughout the manuscript.

AD	Automatic Differentiation
Adam	Adaptive Moment Estimation
AMG	Algebraic Multigrid
BC	Boundary Condition
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CPU	Central Processing Unit
CutFEM	Cut Finite Element Method
DAG	Directed Acyclic Graph
DG	Discontinuous Galerkin
DOF	Degree Of Freedom
ENG	Energy Natural Gradient
FE(M)	Finite Element (Method)
FF	Fourier Features
FNO	Fourier Neural Operator
GFEM	Generalized Finite Element Method
GMRES	Generalized Minimal Residual
GPU	Graphics Processing Unit
HDG	Hybridizable Discontinuous Galerkin
HHO	Hybrid High-Order
IBM	Immersed Boundary Method
IGA	Isogeometric Analysis
IP	Interior Penalty
JVP	Jacobian-Vector Product
LBFGS	Limited-memory BFGS
MLP	Multi-Layer Perceptron
NIPG	Non-symmetric Interior Penalty Galerkin
NO	Neural Operator
PDE	Partial Differential Equation
PGD	Proper Generalized Decomposition
PINN	Physics-Informed Neural Network
PINO	Physics-Informed Neural Operator
POD	Proper Orthogonal Decomposition
ROM	Reduced Order Model
SBM	Shifted Boundary Method
SciML	Scientific Machine Learning
SIPG	Symmetric Interior Penalty Galerkin
SSBFGS	Self-Scaled BFGS
TPU	Tensor Processing Unit
VJP	Vector-Jacobian Product
VPINN	Variational Physics-Informed Neural Network
VRAM	Video RAM
XFEM	eXtended Finite Element Method
XLA	Accelerated Linear Algebra

B Code examples

Contents

B.1. Minimal example: classical and learned FE solvers 251

B.2. Solving a batch of parametric problems 255

This appendix illustrates, with excerpts of code, how the `ScimBa`[1] library is used in the manuscript. The first section (Appendix B.1) shows how a problem is defined and solved with a classical and a learned finite element solver. The second one (Appendix B.2) details how a batch of parametric problems is solved.

B.1. Minimal example: classical and learned FE solvers

The aim of this section is to illustrate, on a minimal example, how a PDE is defined and solved with the `ScimBa` library, first with a classical continuous finite element solver, then with a learned basis. We consider the 1D advection-diffusion problem of Section 5.2, given by (5.3), with $\mathrm{Pe} = 20$, $f = 1$ and the parameter $\boldsymbol{\mu}^{(1)} = (0.8, 0.8)$ of (5.4). The classical solver uses $\mathbb{P}_3$ Lagrange elements on $N_{\text{cells}} = 4$ cells. The learned basis is the patchwise basis of Section 5.2.1.1, with the multiplier (5.8). The complete example is available on GitHub[2]. The imports are omitted below, with the usual convention that `jnp` stands for `jax.numpy`, and each block is the part of the script described by the text that precedes it.

Parameters. The parameters of the problem and of the discretization are gathered first. `DIM` is the space dimension and `OUT_DIM` the number of components of the solution, `EPS` is 1/Pe and `MU` is the parameter $\boldsymbol{\mu}$. `DEGREE` and `N_CELLS` define the coarse discretization, `DEGREE_REF` and `N_CELLS_REF` the fine one that provides the reference solution (Remark 5.4). The last two constants are the number of epochs and of collocation points of the training.

```
DIM, OUT_DIM = 1, 1
EPS = 0.05
MU = jnp.array([0.8, 0.8])
DEGREE, N_CELLS = 3, 4
DEGREE_REF, N_CELLS_REF = 4, 128
N_EPOCHS, N_COLLOC = 300, 6000
```

[1] https://www.scimba.org/

[2] https://github.com/flecourtier/thesis-learnable-methods

Problem. The coefficients and the source term are plain functions of the point $\boldsymbol{x}$. The class `EllipticWeakForm` (Section 4.5.1) builds the weak form of the elliptic problem from the diffusion matrix (here 1/Pe times the identity), the advection field $\mu_1 + \mu_2 x$ and the source term f. The reaction coefficient, left unspecified, is absent. The class `EllipticWeakModel` (Section 4.5.3) then gathers this interior weak form with one boundary condition per boundary group: here a homogeneous `Dirichlet` condition on the whole boundary, designated by the label `"boundary"`.

```
def diffusion(_x):
    return EPS * jnp.eye(DIM)

def advection(x):
    return jnp.array([MU[0] + MU[1] * x[0]])

def source(_x):
    return jnp.ones(())

pde = EllipticWeakForm(dim=DIM, A=diffusion, b=advection, f=source)
boundary = {"boundary": Dirichlet(lambda _x: jnp.zeros(OUT_DIM))}
model = EllipticWeakModel(pde, boundary_conditions=boundary)
```

Classical solver. The mesh (Section 4.2) is a uniform mesh of $(0, 1)$ with `n_cells` cells, equipped with a quadrature rule on the reference cell and an identity mapping. The function `lagrange(degree)` is the local Lagrange basis of degree `degree`. From it, `AnalyticBasis` builds the basis of the space, and `VariablesFE` the continuous finite element variable (Sections 4.3 and 4.7.1). The function `solve` assembles the scheme `EllipticFEscheme` from the model and the variable (Section 4.7.2), and solves it with Newton's method (Section 4.6.1). The problem being linear, a single iteration is enough.

```
def build_mesh(n_cells, degree):
    return Mesh(
        dim=DIM,
        n_cells=[n_cells],
        ref_quad=UnitSquareTensorized(dim=DIM, order=2 * degree + 2),
        mapping=Mapping(mappings=[InvertibleFunction(lambda x: x, lambda y: y)]),
    )

def lagrange(degree):
    return lambda y, i, mesh: local_lagrange_basis(
        y, i, mesh, order=degree, out_dim=OUT_DIM
    )

def classical_variables(n_cells, degree):
    basis = AnalyticBasis(
        nb_basis=degree + 1,
        out_dim=OUT_DIM,
        mesh=build_mesh(n_cells, degree),
        local_basis=lagrange(degree),
        basis_type="scalar",
    )
    return VariablesFE(basis=basis, nb_variables=OUT_DIM)
```

```
def solve(variables):
    return EllipticFEscheme.solve(
        EllipticFEscheme(model, variables),
        matrix_free=False,
    )

classical = solve(classical_variables(N_CELLS, DEGREE))
```

Learned basis and training. The learned basis is obtained by multiplying every Lagrange shape function by an affine multiplier built from a network. The function `multiplier` computes $m_\theta = 2 + 0.1\, v_\theta$ of (5.8) from the output `v` of the network, and `enriched_lagrange` the enriched function $\phi^\theta_{c,i} = m_\theta\, \phi_{c,i}$ defined in (5.6). The function `learned_variables` is the counterpart of `classical_variables`, and its basis `PatchwiseParametricBasis` carries the network `net`, shared by all the cells. This network v_θ is an MLP with two hidden layers of width 16 and a scalar output, since the same multiplier applies to all the basis functions of a cell. The key `key_net` sets the initial weights of the network. The scheme `assembler` is the same as the classical one, with the learned variable in place of the classical one.

```
def multiplier(v):
    return 2.0 + 0.1 * v[0]

def enriched_lagrange(v, y, i, mesh):
    return lagrange(DEGREE)(y, i, mesh) * multiplier(v)

def learned_variables(net, n_cells, degree):
    basis = PatchwiseParametricBasis(
        nb_basis=degree + 1,
        out_dim=OUT_DIM,
        mesh=build_mesh(n_cells, degree),
        patchwise_parametric_function=net,
        local_basis=enriched_lagrange,
        basis_type="scalar",
    )
    return VariablesFE(basis=basis, nb_variables=OUT_DIM)

key_net = jax.random.PRNGKey(0)
net = MLP(in_size=DIM, out_size=1, hidden_sizes=[16, 16], key=key_net)
assembler = EllipticFEscheme(model, learned_variables(net, N_CELLS, DEGREE))
```

The training minimizes the PINN loss (5.5), which requires the strong residual of the same PDE. It is described by `GeneralElliptic`, with the same diffusion and advection functions. The argument `model_type="x_dofsl"` indicates that the functions of the model depend on the point $\boldsymbol{x}$ and on the degrees of freedom `dofsl`, and `bc="weak"` imposes the boundary condition in the loss only, since the scheme imposes it exactly. The approximation space `GalerkinEllipticApproximationSpace` (Section 4.6.3) wraps the scheme so that each evaluation of the solution solves it for the current weights of the network. The sampler draws the collocation points inside the domain and on its boundary, and the `Projector` associates the loss, the space and the sampler. Its

method `project` performs the training through the solver (Section 4.9), with the default ENG optimizer, $N_{\text{epochs}} = 300$ epochs and $N_{\text{col}} = 6\,000$ collocation points, as in the first study, and the key `key_train` for the random draws of the training. Finally, the scheme is solved with the trained enriched basis, with a single Newton iteration since the problem is linear.

```
domain = Segment1D((0.0, 1.0), is_main_domain=True)
pinn = GeneralElliptic(
    main_domain=domain,
    model_type="x_dofsl",
    f_rhs=lambda x: source(x)[None],
    A=lambda x, dofsl: diffusion(x),
    b=advection,
    bc="weak",
)
space = GalerkinEllipticApproximationSpace(
    dims={"x": DIM, "dofsl": OUT_DIM},
    list_assemblers=[assembler],
    model_type="x_dofsl",
    newton_kwargs={"max_iter": 1, "tol": 1e-6},
)
sampler = TensorizedSampler([DomainSampler(domain)], bc=True)
projector = Projector(pinn, space, sampler)
key_train = jax.random.PRNGKey(1)
_, trained = projector.project(key_train, space, N_EPOCHS, N_COLLOC)

learned = EllipticFEscheme.solve(
    trained.space.assemblers[0], max_iter=1, matrix_free=False
)
```

Errors. The reference solution is obtained with the same function `solve`, on a mesh of `N_CELLS_REF` cells with the degree `DEGREE_REF`, and the function `u_ref` evaluates it at a point. The relative L^2 errors of the classical and learned solutions are computed with respect to it. For this seed, the classical error is about $5.1 \cdot 10^{-2}$ and the learned error about $9 \cdot 10^{-6}$, i.e. a gain of about $6 \cdot 10^{3}$, in agreement with the first study (Section 5.2.1.1).

```
reference = solve(classical_variables(N_CELLS_REF, DEGREE_REF))
def u_ref(x):
    return reference.variables.evaluate(x[None, :])[0]

err_classical = l2_error(assembler=classical, u_exact_fn=u_ref, relative=True)
err_learned = l2_error(assembler=learned, u_exact_fn=u_ref, relative=True)

print(f"classical FEM, relative L2 error: {err_classical:.3e}")
print(f"learned basis, relative L2 error: {err_learned:.3e}")
print(f"gain: {err_classical / err_learned:.1f}")
```

B.2. Solving a batch of parametric problems

The aim of this section is to detail the two ways of solving a batch of N_μ parametric problems compared in Table 4.2, namely the 2D Poisson problem of Section 4.10.5 discretized by DG with SIPG flux. The complete example is available on GitHub[3]. As in Appendix B.1, the imports are omitted.

We assume given an array `mus` of N_μ instances of $\boldsymbol{\mu}$, a scheme `assembler` built once and shared by all the problems, and a function `make_model(mu)` returning the weak model for the value `mu`. Similarly to Appendix B.1, the DG scheme is built from a weak model and variables, with in addition a numerical flux. Only the source term, hence the weak model, depends on $\boldsymbol{\mu}$. The source term is defined as a `ScimbaPytree` in which $\boldsymbol{\mu}$ is stored as a `jnp` array, hence a leaf of the pytree, i.e. a traced value rather than a constant, which is what allows `jax.jit` and `jax.vmap`.

Both versions rely on the function `solve_one`, which copies `assembler`, replaces its model by `make_model(mu)` and solves the problem with `EllipticDGscheme.solve_pure`. Contrary to `EllipticDGscheme.solve`, which stores the solution in the scheme, this solve is pure, i.e. it returns the degrees of freedom without modifying the scheme, which is required by `jax.jit` and `jax.vmap`. The first version compiles `solve_one` once with `jax.jit` and loops over the values of $\boldsymbol{\mu}$. The second applies `jax.vmap` to `solve_one`, so that the N_μ problems are solved in parallel in a single compiled program.

```
def solve_one(mu):
    assembler_mu = copy.copy(assembler)
    assembler_mu.pde = make_model(mu)
    dofs, _ = EllipticDGscheme.solve_pure(assembler_mu)
    return dofs

# Compiled loop
solve_one_jit = jax.jit(solve_one)
dofs_loop = jnp.stack([solve_one_jit(mu) for mu in mus])

# Vectorized solve
solve_vmap = jax.jit(jax.vmap(solve_one))
dofs_vmap = solve_vmap(mus)
```

[3] https://github.com/flecourtier/thesis-learnable-methods

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[2] J. Aghili, E. Franck, R. Hild, V. Michel-Dansac, and V. Vigon. "Accelerating the convergence of Newton's method for nonlinear elliptic PDEs using Fourier neural operators". In: *Communications in Nonlinear Science and Numerical Simulation* 140 (2025), p. 108434.

[3] M. Ainsworth and J. T. Oden. "A posteriori error estimation in finite element analysis". In: *Computer Methods in Applied Mechanics and Engineering* 142.1–2 (1997), pp. 1–88.

[4] E. L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods*. Society for Industrial and Applied Mathematics, 2003.

[5] R. C. Almeida and R. S. Silva. "A stable Petrov-Galerkin method for convection-dominated problems". In: *Computer Methods in Applied Mechanics and Engineering* 140.3–4 (1997), pp. 291–304.

[6] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. "Unified form language: A domain-specific language for weak formulations of partial differential equations". In: *ACM Transactions on Mathematical Software* 40.2 (2014), pp. 1–37.

[7] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. Rognes, and G. Wells. "The FEniCS Project Version 1.5". In: *Archive of Numerical Software* 3.100 (2015).

[8] D. N. Arnold. "An Interior Penalty Finite Element Method with Discontinuous Elements". In: *SIAM Journal on Numerical Analysis* 19.4 (1982), pp. 742–760.

[9] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. "Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems". In: *SIAM Journal on Numerical Analysis* 39.5 (2002), pp. 1749–1779.

[10] B. Ayuso and L. D. Marini. "Discontinuous Galerkin Methods for Advection-Diffusion-Reaction Problems". In: *SIAM Journal on Numerical Analysis* 47.2 (2009), pp. 1391–1420.

[11] I. Babuška and J. M. Melenk. "The partition of unity method". In: *International Journal for Numerical Methods in Engineering* 40.4 (1997), pp. 727–758.

[12] I. Babuška, U. Banerjee, and J. E. Osborn. "Generalized finite element methods – main ideas, results and perspective". In: *International Journal of Computational Methods* 1.1 (2004), pp. 67–103.

[13] S. Badia, W. Li, and A. F. Martín. "Finite element interpolated neural networks for solving forward and inverse problems". In: *Computer Methods in Applied Mechanics and Engineering* 418 (2024), p. 116505.

[14] I. Baratta, J. Dean, J. S. Dokken, M. Habera, J. Hale, C. Richardson, M. Rognes, M. Scroggs, N. Sime, and G. Wells. *DOLFINx: The next generation FEniCS problem solving environment*. Preprint, Zenodo. 2023.

[15] A. R. Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information Theory* 39.3 (1993), pp. 930–945.

[16] H. Barucq, M. Duprez, F. Faucher, E. Franck, F. Lecourtier, V. Lleras, V. Michel-Dansac, and N. Victorion. "Enriching continuous Lagrange finite element approximation spaces using neural networks". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 60.2 (2026), pp. 541–603.

[17] C. E. Baumann and J. T. Oden. "A discontinuous hp finite element method for convection-diffusion problems". In: *Computer Methods in Applied Mechanics and Engineering* 175.3–4 (1999), pp. 311–341.

[18] W. Baur and V. Strassen. "The complexity of partial derivatives". In: *Theoretical Computer Science* 22.3 (1983), pp. 317–330.

[19] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43.

[20] Y. Bazilevs, L. Beirão Da Veiga, J. A. Cottrell, T. J. R. Hughes, and G. Sangalli. "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.

[21] T. Belytschko, R. Gracie, and G. Ventura. "A review of extended/generalized finite element methods for material modeling". In: *Modelling and Simulation in Materials Science and Engineering* 17.4 (2009), p. 043001.

[22] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. "Meshless methods: an overview and recent developments". In: *Computer Methods in Applied Mechanics and Engineering* 139.1–4 (1996), pp. 3–47.

[23] G. Berkooz, P. Holmes, and J. L. Lumley. "The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows". In: *Annual Review of Fluid Mechanics* 25.1 (1993), pp. 539–575.

[24] N. Berman, A. Hallak, and A. Shocher. *Who Said Neural Networks Aren't Linear?* 2025. arXiv: 2510.08570.

[25] M. Blondel and V. Roulet. *The Elements of Differentiable Programming*. 2024. arXiv: 2403.14606.

[26] J. Bradbury, R. Frostig, P. Hawkins, et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/jax-ml/jax.

[27] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. 3rd ed. Vol. 15. Texts in Applied Mathematics. Springer New York, 2008.

[28] H.-J. Bungartz and M. Griebel. “Sparse grids”. In: *Acta Numerica* 13 (2004), pp. 147–269.

[29] E. Burman. “Ghost penalty”. In: *Comptes Rendus. Mathématique* 348.21–22 (2010), pp. 1217–1220.

[30] E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing. “CutFEM: Discretizing geometry and partial differential equations”. In: *International Journal for Numerical Methods in Engineering* 104.7 (2015), pp. 472–501.

[31] E. Burman and P. Hansbo. “Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method”. In: *Computer Methods in Applied Mechanics and Engineering* 199.41–44 (2010), pp. 2680–2686.

[32] R. E. Caflisch. “Monte Carlo and quasi-Monte Carlo methods”. In: *Acta Numerica* 7 (1998), pp. 1–49.

[33] D. Canales, A. Leygue, F. Chinesta, D. González, E. Cueto, E. Feulvarch, J.-M. Bergheau, and A. Huerta. “Vademecum-based GFEM (V-GFEM): optimal enrichment for transient problems”. In: *International Journal for Numerical Methods in Engineering* 108.9 (2016), pp. 971–989.

[34] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang. *Spectral Methods: Fundamentals in Single Domains*. Springer Berlin Heidelberg, 2006.

[35] P. Castillo. “Performance of Discontinuous Galerkin Methods for Elliptic PDEs”. In: *SIAM Journal on Scientific Computing* 24.2 (2002), pp. 524–547.

[36] S. Chandrasekhar. *Hydrodynamic and Hydromagnetic Stability*. Dover Books on Physics Series. Dover Publications, 1981.

[37] F. Chinesta, P. Ladeveze, and E. Cueto. “A Short Review on Model Order Reduction Based on Proper Generalized Decomposition”. In: *Archives of Computational Methods in Engineering* 18.4 (2011), pp. 395–404.

[38] P. G. Ciarlet. *Mathematical Elasticity: Three-Dimensional Elasticity*. Society for Industrial and Applied Mathematics, 2021.

[39] B. Cockburn, B. Dong, and J. Guzmán. “A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems”. In: *Mathematics of Computation* 77.264 (2008), pp. 1887–1916.

[40] B. Cockburn and C.-W. Shu. “The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems”. In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2440–2463.

[41] S. Cotin, M. Duprez, V. Lleras, A. Lozinski, and K. Vuillemot. “ϕ-FEM: An Efficient Simulation Tool Using Simple Meshes for Problems in Structure Mechanics and Heat Transfer”. In: *Partition of Unity Methods*. Ed. by S. Bordas, A. Menk, and S. Natarajan. Wiley, 2023, pp. 191–216.

[42] G. Coulaud, M. Le, and R. Duvigneau. *Investigations on Physics-Informed Neural Networks for Aerodynamics*. 2024. arXiv: 2403.17470.

[43] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.

[44] G. De Vahl Davis. "Natural convection of air in a square cavity: A bench mark numerical solution". In: *International Journal for Numerical Methods in Fluids* 3.3 (1983), pp. 249–264.

[45] L. F. Demkowicz. *Mathematical Theory of Finite Elements*. Society for Industrial and Applied Mathematics, 2023.

[46] L. Dinh, D. Krueger, and Y. Bengio. "NICE: Non-linear Independent Components Estimation". In: *International Conference on Learning Representations (ICLR), Workshop Track*. 2015.

[47] L. Dinh, J. Sohl-Dickstein, and S. Bengio. "Density estimation using Real NVP". In: *International Conference on Learning Representations (ICLR)*. 2017.

[48] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. "Multilevel domain decomposition-based architectures for physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 429 (2024), p. 117116.

[49] J. Douglas and T. Dupont. "Interior Penalty Procedures for Elliptic and Parabolic Galerkin Methods". In: *Computing Methods in Applied Sciences (Second International Symposium, Versailles, 1975)*. Ed. by R. Glowinski and J. L. Lions. Vol. 58. Lecture Notes in Physics. 1976, pp. 207–216.

[50] M. Duprez, V. Lleras, and A. Lozinski. "ϕ-FEM: an optimally convergent and easily implementable immersed boundary method for particulate flows and Stokes equations". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 57.3 (2023), pp. 1111–1142.

[51] M. Duprez, V. Lleras, and A. Lozinski. "A new ϕ-FEM approach for problems with natural boundary conditions". In: *Numerical Methods for Partial Differential Equations* 39.1 (2023), pp. 281–303.

[52] M. Duprez, V. Lleras, A. Lozinski, and K. Vuillemot. "ϕ-FEM for the heat equation: optimal convergence on unfitted meshes in space". In: *Comptes Rendus. Mathématique* 361.G11 (2023), pp. 1699–1710.

[53] M. Duprez and A. Lozinski. "ϕ-FEM: A Finite Element Method on Domains Defined by Level-Sets". In: *SIAM Journal on Numerical Analysis* 58.2 (2020), pp. 1008–1028.

[54] W. E and B. Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12.

[55] A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Springer New York, 2004.

[56] A. Ern and M. Steins. "Convergence Analysis for the Wave Equation Discretized with Hybrid Methods in Space (HHO, HDG and WG) and the Leapfrog Scheme in Time". In: *Journal of Scientific Computing* 101.1 (2024), p. 7.

[57] L. C. Evans. *Partial Differential Equations*. 2nd ed. Graduate Studies in Mathematics 19. American Mathematical Society, 2010.

[58] R. Eymard, T. Gallouët, and R. Herbin. “Finite volume methods”. In: *Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. 7. Handbook of Numerical Analysis. Elsevier, 2000, pp. 713–1018.

[59] S. A. Faroughi, N. M. Pawar, C. Fernandes, M. Raissi, S. Das, N. K. Kalantari, and S. K. Mahjour. “Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks and Operators in Scientific Computing: Fluid and Solid Mechanics”. In: *Journal of Computing and Information Science in Engineering* 24.4 (2024), p. 040802.

[60] X. Feng, H. Shangguan, T. Tang, X. Wan, and T. Zhou. *A hybrid FEM-PINN method for time-dependent partial differential equations*. 2024. arXiv: 2409.02810.

[61] C. Fong. *Analytical Methods for Squaring the Disc*. 2015. arXiv: 1509.06344.

[62] S. Frambati, H. Barucq, H. Calandra, and J. Diaz. “Practical unstructured splines: Algorithms, multi-patch spline spaces, and some applications to numerical analysis”. In: *Journal of Computational Physics* 471 (2022), p. 111625.

[63] E. Franck, V. Michel-Dansac, and L. Navoret. “Approximately well-balanced Discontinuous Galerkin methods using bases enriched with Physics-Informed Neural Networks”. In: *Journal of Computational Physics* 512 (2024), p. 113144.

[64] Z. Frangella, J. A. Tropp, and M. Udell. “Randomized Nyström Preconditioning”. In: *SIAM Journal on Matrix Analysis and Applications* 44.2 (2023), pp. 718–752.

[65] T.-P. Fries and T. Belytschko. “The extended/generalized finite element method: An overview of the method and its applications”. In: *International Journal for Numerical Methods in Engineering* 84.3 (2010), pp. 253–304.

[66] R. Glowinski, T.-W. Pan, and J. Periaux. “A fictitious domain method for Dirichlet problem and applications”. In: *Computer Methods in Applied Mechanics and Engineering* 111.3–4 (1994), pp. 283–303.

[67] A. Griewank and A. Walther. *Evaluating Derivatives*. 2nd ed. Society for Industrial and Applied Mathematics, 2008.

[68] P. Grisvard. *Elliptic Problems in Nonsmooth Domains*. Vol. 69. Classics in Applied Mathematics. SIAM, 2011.

[69] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. “Can physics-informed neural networks beat the finite element method?” In: *IMA Journal of Applied Mathematics* 89.1 (2024), pp. 143–174.

[70] J. Haslinger and Y. Renard. “A New Fictitious Domain Approach Inspired by the Extended Finite Element Method”. In: *SIAM Journal on Numerical Analysis* 47.2 (2009), pp. 1474–1499.

[71] M. R. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems”. In: *Journal of Research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[72] R. Hiptmair, A. Moiola, and I. Perugia. “Error analysis of Trefftz-discontinuous Galerkin methods for the time-harmonic Maxwell equations”. In: *Mathematics of Computation* 82.281 (2013), pp. 247–268.

[73] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[74] Z. Hu, K. Shukla, G. E. Karniadakis, and K. Kawaguchi. "Tackling the curse of dimensionality with physics-informed neural networks". In: *Neural Networks* 176 (2024), p. 106369.

[75] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement". In: *Computer Methods in Applied Mechanics and Engineering* 194.39–41 (2005), pp. 4135–4195.

[76] A. Hungria, D. Prada, and F.-J. Sayas. "HDG methods for elastodynamics". In: *Computers & Mathematics with Applications* 74.11 (2017), pp. 2671–2690.

[77] L.-M. Imbert-Gérard, A. Moiola, and P. Stocker. "A space–time quasi-Trefftz DG method for the wave equation with piecewise-smooth coefficients". In: *Mathematics of Computation* 92.341 (2023), pp. 1211–1249.

[78] V. John, P. Knobloch, and J. Novo. "Finite elements for scalar convection-dominated equations and incompressible flow problems: a never ending story?" In: *Computing and Visualization in Science* 19.5–6 (2018), pp. 47–63.

[79] W. Johnson. "The Curious History of Faa di Bruno's Formula". In: *American Mathematical Monthly* 109.3 (2002), pp. 217–234.

[80] J. Jung and E. Constantinescu. *Learning differentiable weak-form corrections to accelerate finite element simulations*. 2026. arXiv: 2601.20019.

[81] S. Kang and E. M. Constantinescu. *Differentiable DG with neural operator source term correction*. 2023. arXiv: 2310.18897.

[82] E. Kharazmi, Z. Zhang, and G. E. M. Karniadakis. "*hp*-VPINNs: Variational physics-informed neural networks with domain decomposition". In: *Computer Methods in Applied Mechanics and Engineering* 374 (2021), p. 113547.

[83] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.

[84] I. E. Lagaris, A. Likas, and D. I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000.

[85] M. G. Larson and A. J. Niklasson. "Analysis of a family of discontinuous Galerkin methods for elliptic problems: the one dimensional case". In: *Numerische Mathematik* 99.1 (2004), pp. 113–130.

[86] H. Lee and I. S. Kang. "Neural algorithm for solving differential equations". In: *Journal of Computational Physics* 91.1 (1990), pp. 110–131.

[87] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

[88] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations (ICLR)*. 2021.

[89] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. “Physics-informed neural operator for learning partial differential equations”. In: *ACM/IMS Journal of Data Science* 1.3 (2024), pp. 1–27.

[90] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators”. In: *Nature Machine Intelligence* 3.3 (2021), pp. 218–229.

[91] A. Main and G. Scovazzi. “The shifted boundary method for embedded domain computations. Part I: Poisson and Stokes problems”. In: *Journal of Computational Physics* 372 (2018), pp. 972–995.

[92] N. Margenberg, R. Jendersie, C. Lessig, and T. Richter. “DNN-MG: A hybrid neural network/finite element method with applications to 3D simulations of the Navier-Stokes equations”. In: *Computer Methods in Applied Mechanics and Engineering* 420 (2024), p. 116692.

[93] A. J. Meade and A. A. Fernandez. “The numerical solution of linear ordinary differential equations by feedforward neural networks”. In: *Mathematical and Computer Modelling* 19.12 (1994), pp. 1–25.

[94] A. Meurer, C. P. Smith, M. Paprocki, et al. “SymPy: symbolic computing in Python”. In: *PeerJ Computer Science* 3 (2017), e103.

[95] R. Mittal and G. Iaccarino. “Immersed Boundary Methods”. In: *Annual Review of Fluid Mechanics* 37.1 (2005), pp. 239–261.

[96] N. Moës, J. Dolbow, and T. Belytschko. “A finite element method for crack growth without remeshing”. In: *International Journal for Numerical Methods in Engineering* 46.1 (1999), pp. 131–150.

[97] A. Moiola and I. Perugia. “A space–time Trefftz discontinuous Galerkin method for the acoustic wave equation in first-order formulation”. In: *Numerische Mathematik* 138.2 (2018), pp. 389–435.

[98] J. Müller and M. Zeinhofer. “Achieving High Accuracy with PINNs via Energy Natural Gradient Descent”. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 25471–25485.

[99] J. C. C. Nitsche. “Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind”. In: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 36.1 (1971), pp. 9–15.

[100] J. Nocedal and S. J. Wright. “Large-Scale Unconstrained Optimization”. In: *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. Springer, 2006. Chap. 7, pp. 164–192.

[101] A. Odot, R. Haferssas, and S. Cotin. “DeepPhysics: A physics aware deep learning framework for real-time simulation”. In: *International Journal for Numerical Methods in Engineering* 123.10 (2022), pp. 2381–2398.

[102] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 165–174.

[103] A. Paszke, S. Gross, F. Massa, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. Curran Associates, Inc., 2019.

[104] A. T. Patera and E. M. Rønquist. "Reduced basis approximation and *a posteriori* error estimation for a Boltzmann model". In: *Computer Methods in Applied Mechanics and Engineering* 196.29–30 (2007), pp. 2925–2942.

[105] C. S. Peskin. "Flow patterns around heart valves: a numerical method". In: *Journal of Computational Physics* 10.2 (1972), pp. 252–271.

[106] H. Pham, F. Faucher, and H. Barucq. "Numerical investigation of stabilization in the Hybridizable Discontinuous Galerkin method for linear anisotropic elastic equation". In: *Computer Methods in Applied Mechanics and Engineering* 428 (2024), p. 117080.

[107] C. Prud'homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, and G. Turinici. "Reliable Real-Time Solution of Parametrized Partial Differential Equations: Reduced-Basis Output Bound Methods". In: *Journal of Fluids Engineering* 124.1 (2002), pp. 70–80.

[108] 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.

[109] S. S. Ravindran. "A reduced-order approach for optimal control of fluids using proper orthogonal decomposition". In: *International Journal for Numerical Methods in Fluids* 34.5 (2000), pp. 425–448.

[110] J. N. Reddy. *Introduction to the Finite Element Method*. 4th ed. McGraw-Hill Education, 2019.

[111] B. Rivière, M. F. Wheeler, and V. Girault. "Improved energy estimates for interior penalty, constrained and discontinuous Galerkin methods for elliptic problems. Part I". In: *Computational Geosciences* 3.3–4 (1999), pp. 337–360.

[112] Y. Saad and M. H. Schultz. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869.

[113] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Society for Industrial and Applied Mathematics, 2003.

[114] M. W. Scroggs, I. A. Baratta, C. N. Richardson, and G. N. Wells. "Basix: a runtime finite element basis evaluation library". In: *Journal of Open Source Software* 7.73 (2022), p. 3982.

[115] M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells. “Construction of Arbitrary Order Finite Element Degree-of-Freedom Maps on Polygonal and Polyhedral Cell Meshes”. In: *ACM Transactions on Mathematical Software* 48.2 (2022), pp. 1–23.

[116] K. Shahbazi. “An explicit expression for the penalty parameter of the interior penalty method”. In: *Journal of Computational Physics* 205.2 (2005), pp. 401–407.

[117] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. “Implicit Neural Representations with Periodic Activation Functions”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., 2020, pp. 7462–7473.

[118] K. Škardová, A. Daby-Seesaram, and M. Genet. “Finite element neural network interpolation: Part I—interpretable and adaptive discretization for solving PDEs”. In: *Computational Mechanics* 77.2 (2026), pp. 547–567.

[119] H. Son, J. W. Jang, W. J. Han, and H. J. Hwang. “Sobolev Training for Physics Informed Neural Networks”. In: *Communications in Mathematical Sciences* 21.6 (2023), pp. 1679–1705.

[120] T. Strouboulis, K. Copps, and I. Babuška. “The generalized finite element method”. In: *Computer Methods in Applied Mechanics and Engineering* 190.32–33 (2001), pp. 4081–4193.

[121] N. Sukumar, N. Moës, B. Moran, and T. Belytschko. “Extended finite element method for three-dimensional crack modelling”. In: *International Journal for Numerical Methods in Engineering* 48.11 (2000), pp. 1549–1570.

[122] N. Sukumar and A. Srivastava. “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333.

[123] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., 2020, pp. 7537–7547.

[124] C. Taylor and P. Hood. “A numerical solution of the Navier-Stokes equations using the finite element technique”. In: *Computers & Fluids* 1.1 (1973), pp. 73–100.

[125] J. M. Taylor, D. Pardo, and J. Muñoz-Matute. “Regularity-conforming neural networks (ReCoNNs) for solving partial differential equations”. In: *Journal of Computational Physics* 532 (2025), p. 113954.

[126] J. F. Urbán, P. Stefanou, and J. A. Pons. “Unveiling the optimization process of physics informed neural networks: How accurate and competitive can PINNs be?” In: *Journal of Computational Physics* 523 (2025), p. 113656.

[127] A. Virmaux and K. Scaman. “Lipschitz regularity of deep neural networks: analysis and efficient estimation”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[128] H. A. van der Vorst. "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644.

[129] D. Wang, H. Li, and Q. Zhang. "General enrichments of stable GFEM for interface problems: Theory and extreme learning machine construction". In: *Applied Numerical Mathematics* 214 (2025), pp. 143–159.

[130] W. Xiong, X. Long, S. P. A. Bordas, and C. Jiang. "The deep finite element method: A deep learning framework integrating the physics-informed neural networks with the finite element method". In: *Computer Methods in Applied Mechanics and Engineering* 436 (2025), p. 117681.

[131] T. Xue, S. Liao, Z. Gan, C. Park, X. Xie, W. K. Liu, and J. Cao. "JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science". In: *Computer Physics Communications* 291 (2023), p. 108802.

[132] D. Yarotsky. "Error bounds for approximations with deep ReLU networks". In: *Neural Networks* 94 (2017), pp. 103–114.

[133] W. Zhang and M. Al Kobaisi. "On the Monotonicity and Positivity of Physics-Informed Neural Networks for Highly Anisotropic Diffusion Equations". In: *Energies* 15.18 (2022), p. 6823.

Université de Strasbourg

Frédérique Lecourtier

Inria

Hybrid finite element and neural network methods for parametric PDEs

Résumé

Cette thèse traite de la résolution d'équations aux dérivées partielles (EDP) elliptiques paramétriques pour de nombreuses valeurs de paramètres. Si les méthodes d'éléments finis offrent des garanties théoriques solides, leur coût de calcul devient prohibitif lors du traitement d'un grand nombre de valeurs de paramètres. À l'inverse, les réseaux de neurones, bien qu'efficaces dans ce cadre, manquent de fiabilité lorsqu'ils sont utilisés seuls. Plutôt que d'opposer ces deux approches, nous proposons de les combiner selon deux stratégies. Premièrement, nous enrichissons l'espace éléments finis par un a priori neuronal, entraîné une seule fois pour l'ensemble de la famille de paramètres. Nous démontrons que les ordres de convergence sont conservés et que le gain en efficacité dépend de la qualité de cet a priori, permettant ainsi d'utiliser des maillages 5 à 28 fois plus grossiers pour une précision identique. Deuxièmement, nous développons un solveur numérique différentiable au sein de la bibliothèque ScimBa. Un réseau de neurones peut alors être entraîné à travers la résolution pour enrichir directement les fonctions de base du schéma, au lieu de servir d'a priori fixe. Ce principe s'applique aussi bien à un paramètre donné qu'à toute une famille de paramètres. Les premiers résultats, bien qu'exploratoires, montrent que les bases apprises capturent des structures fines de la solution que les éléments finis classiques ne peuvent représenter qu'au prix d'un maillage très fin.

Mots clés : Équations aux dérivées partielles paramétriques, méthodes éléments finis, réseau de neurones, différentiation automatique.

Abstract

This thesis addresses the solution of parametric elliptic partial differential equations (PDEs) for a large number of parameter values. While finite element methods offer strong theoretical guarantees, their computational cost becomes prohibitive when dealing with a large number of parameter values. Conversely, neural networks, although effective in this context, lack reliability when used alone. Rather than opposing these two approaches, we propose combining them using two strategies. First, we enrich the finite element space with a neural prior, trained once for the entire family of parameters. We demonstrate that the convergence orders are preserved and that the efficiency gain depends on the quality of this prior, thereby allowing the use of meshes that are 5 to 28 times coarser while maintaining the same accuracy. Second, we develop a differentiable numerical solver within the ScimBa library. A neural network can then be trained during the solution process to directly enrich the scheme's basis functions, rather than serving as a fixed prior. This principle applies equally well to a given parameter as it does to an entire family of parameters. Initial results, although exploratory, show that the learned basis captures fine structures of the solution that classical finite elements can represent only at the cost of a very fine mesh.

Keywords : Parametric partial differential equations, finite element methods, neural networks, automatic differentiation.