der Bundeswehr
Universität München

From Data-Driven Surrogate Models to PINNs: Machine Learning Techniques for Computational Engineering

Tarık Şahin

Vollständiger Abdruck der von der Fakultät für Bauingenieurwesen und Umweltwissenschaften der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

angenommenen Dissertation.

Gutachter:
1. Univ.-Prof. Dr.-Ing. Alexander Popp
2. Univ.-Prof. Dr.-Ing. habil. Fadi Aldakheel
3. Univ.-Prof. Dr.-Ing. Merten Stender

Die Dissertation wurde am 20. August 2025 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Bauingenieurwesen und Umweltwissenschaften am 10. Februar 2026 angenommen. Die mündliche Prüfung fand am 27. Februar 2026 statt.

Abstract

This thesis explores the development and application of data-driven machine learning and deep learning methods, together with physics-informed neural networks (PINNs), in the field of computational engineering. Motivated by the need for scalable, accurate, and interpretable models, especially in the presence of nonlinear and high-dimensional problems, the work addresses a broad range of applications including rough surface contact problems, beam mechanics, solid mechanics, and contact mechanics, covering both small and large deformation regimes.

In the first part, a data-driven surrogate modeling framework is developed to predict the effective contact area in rough surface contact problems involving a linear elastic half-space and a rigid rough surface. Relying solely on statistical descriptors and far-field displacement inputs, several regression models are trained on data generated by computationally expensive boundary element method (BEM) simulations. The study identifies the Kernel Ridge model as offering the best trade-off between predictive accuracy, generalization capability, and computational efficiency. Despite the high cost of offline database generation, the resulting surrogate model demonstrates excellent performance for multi-query scenarios such as uncertainty quantification and sensitivity analysis, while also highlighting the limitations of black-box methods in terms of physical interpretability and extrapolation.

To address the limitations of black-box models, the thesis transitions to physics-informed neural networks (PINNs), which incorporate governing equations and domain knowledge directly into the learning process. PINNs are applied to a wide spectrum of benchmark problems, ranging from linear elasticity and Euler-Bernoulli beam equations to contact mechanics with inequality constraints.

In beam mechanics, challenges associated with high-order differential operators are addressed by successfully applying PINNs to static and dynamic beam problems, including nonlinear loading and variable cross-sections. In inverse problems, the issue of non-uniqueness is analyzed by attempting to identify unknown load distributions from partial displacement data. It is shown that the incorporation of prior physical knowledge significantly improves solution fidelity.

For solid mechanics under small deformations, both displacement-based and mixed-variable PINN formulations are implemented and compared. Output transformations, particularly output scaling, are found to be essential for training stability and accuracy. These findings are validated on classical problems such as the Lamé test case and extended to high-dimensional setups including a 3D hollow sphere under time-dependent loading. For contact mechanics, which involve Karush-Kuhn-Tucker (KKT) constraints, various strategies are explored to enforce the KKT constraints, including sign-based, sigmoid-based, and Fischer-Burmeister formulations. These approaches are benchmarked against both analytical solutions and data-enhanced PINNs trained with FEM data, demonstrating the feasibility and limitations of PINNs in capturing complex contact behavior.

In the context of large deformation problems, a novel element-based energy PINN (EBE-PINN) formulation is developed based on the principle of minimum potential energy. This formulation,

inspired by finite element methodology, partitions the domain into subregions for more accurate and scalable integral evaluation. Hyperelastic Neo-Hookean materials are used to model nonlinear stress-strain relationships. Applications include 2D beam bending, 3D torsion of a square prism, and 3D contact problems such as the spherical contact problem and the torus instability example. EBE-PINNs are shown to be computationally competitive with FEM in training, and significantly faster during inference, making them a promising framework for nonlinear solid and contact mechanics simulations.

Finally, the concept of hybrid digital twins is introduced to overcome the data limitations of purely deterministic or synthetic modeling approaches. A proof of concept is demonstrated using a steel-reinforced concrete beam, where real sensor data is integrated with PINN-based reduced-order models (ROMs). Both temporal and spatial surrogate modeling strategies are explored, and the models are benchmarked against analytical and purely data-driven alternatives. The results confirm that hybrid digital twins, when designed with appropriate physics and data coupling, provide flexible, interpretable, and accurate digital representations of physical systems.

Zusammenfassung

Diese Dissertation befasst sich mit der Entwicklung und Anwendung datengetriebener Methoden des maschinellen Lernens und Deep Learnings sowie von physikalisch informierten neuronalen Netzen (PINNs) im Bereich der computergestützte Ingenieurwissenschaft. Begründet in dem Bedarf an skalierbaren, genauen und interpretierbaren Modellen, insbesondere für nichtlineare und hochdimensionale Probleme, befasst sich die Arbeit mit einem breiten Spektrum von Anwendungen, darunter Probleme des Kontakts rauer Oberflächen, Balkenmechanik, Festkörpermechanik und Kontaktmechanik, die sowohl kleine als auch große Deformationsregime abdecken.

Im ersten Teil wird ein datengetriebenes Ersatzmodell-Framework entwickelt, um die effektive Kontaktfläche bei Kontaktproblemen mit rauen Oberflächen unter Einbeziehung eines linearen elastischen Halbraums und einer starren rauen Oberfläche vorherzusagen. Unter ausschließlicher Verwendung von statistischen Deskriptoren und Fernfeld-Verschiebungseingaben werden mehrere Regressionsmodelle auf Daten trainiert, die durch rechenintensive Simulationen mit der Randelementmethode (BEM) generiert wurden. Die Studie identifiziert das Kernel-Ridge-Modell als das Modell mit dem besten Kompromiss zwischen Vorhersagegenauigkeit, Generalisierungsfähigkeit und Recheneffizienz. Trotz der hohen Kosten für die Offline-Datenbankgenerierung zeigt das resultierende Ersatzmodell eine hervorragende Leistung für Multi-Query-Szenarien wie Unsicherheitsquantifizierung und Sensitivitätsanalyse, während es gleichzeitig die Grenzen von Black-Box-Methoden in Bezug auf die physikalische Interpretierbarkeit und Extrapolation aufzeigt.

Um diese Einschränkungen zu beheben, wendet sich die Arbeit physikalisch-informierten neuronalen Netzen zu, die die zugrundeliegenden Gleichungen direkt in den Lernprozess einbetten. PINNs werden auf ein breites Spektrum von Benchmark-Problemen angewendet, von linearer Elastizität und Euler-Bernoulli-Balkengleichungen bis hin zur Kontaktmechanik mit Ungleichungsbeschränkungen.

In der Balkenmechanik werden Herausforderungen im Zusammenhang mit Differentialoperatoren höherer Ordnung durch die erfolgreiche Anwendung von PINNs auf statische und dynamische Balkenprobleme, einschließlich nichtlinearer Belastungen und variabler Querschnitte, angegangen. Bei inversen Problemen wird das Problem der Nicht-Eindeutigkeit analysiert, indem versucht wird, unbekannte Lastverteilungen aus partiellen Verschiebungsdaten zu identifizieren. Es wird gezeigt, dass die Einbeziehung von physikalischem Vorwissen die Lösungsgenauigkeit erheblich verbessert.

Für die Festkörpermechanik unter kleinen Verformungen werden sowohl verschiebungsbasierte variable PINN-Formulierungen mit gemischten Variablen implementiert und verglichen. Die Transformationen der Ausgangsgrößen, insbesondere deren Skalierung, erweisen sich als wesentlich für die Trainingsstabilität und Genauigkeit. Diese Ergebnisse werden an klassischen Problemen wie dem Lamé-Testfall validiert und auf hochdimensionale Konfigurationen einschließlich einer 3D-Hohlkugel unter zeitabhängiger Belastung erweitert. Für die Kontaktmechanik, die Karush-Kuhn-Tucker (KKT)-Beschränkungen beinhaltet, werden verschiedene Strategien zur Durchsetzung dieser Beschränkungen untersucht, darunter zeichenbasierte, sigmoid-

basierte und Fischer-Burmeister-Formulierungen. Diese Ansätze werden sowohl mit analytischen Lösungen als auch mit PINNs verglichen, die mit FEM-Daten trainiert wurden, um die Umsetzbarkeit und Grenzen von PINNs für die Erfassung komplexer Kontaktverhalten aufzuzeigen.

Im Zusammenhang mit Problemen großer Verformungen wird eine neuartige elementbasierte Energie-PINN-Formulierung (EBE-PINN) entwickelt, die auf dem Prinzip der minimalen Potentialenergie basiert. Diese Formulierung, die von der Finite-Elemente-Methode inspiriert ist, unterteilt das Rechengebiet in Teilbereiche in Unterregionen, um eine genauere und skalierbare Integralauswertung zu ermöglichen. Hyperelastische Neo-Hooke-Materialien werden verwendet, um nichtlineare Spannungs-Dehnungs-Beziehungen zu modellieren. Zu den Anwendungsbereichen gehören eine 2D-Balkenbiegung, die 3D-Torsion eines quadratischen Prismas und 3D-Kontaktprobleme wie die kugelförmige Pressung und die Torusinstabilität. EBE-PINNs sind rechnerisch mit FEM im Training konkurrenzfähig und während der Inferenz deutlich schneller, was sie ideal für nichtlineare Festkörper- und Kontaktmechanik-Simulationen macht.

Zuletzt wird das Konzept der hybriden digitalen Zwillinge eingeführt, um die Datenbeschränkungen rein deterministischer oder synthetischer Modellierungsansätze zu überwinden. Anhand eines Stahlbetonträgers wird ein Proof-of-Concept entwickelt, bei dem reale Sensordaten mit PINN-basierten reduzierten Ordnungsmodellen (ROMs) integriert werden. Es werden sowohl zeitliche als auch räumliche Surrogatsmodellierungsstrategien untersucht, und die Modelle werden mit analytischen und rein datengesteuerten Alternativen verglichen. Die Ergebnisse bestätigen, dass hybride digitale Zwillinge, wenn sie mit geeigneter Physik und Datenkopplung entworfen werden, flexible, interpretierbare und genaue digitale Darstellungen physikalischer Systeme liefern.

Contents

Nomenclature **xxv**

1. Introduction **1**
1.1. Motivation . . . 1
1.2. Modeling approaches for computational engineering problems . . . 2
1.2.1. Physics-based modeling . . . 2
1.2.2. Purely data-driven modeling . . . 2
1.2.3. Hybrid modeling . . . 4
1.3. Scope and contribution . . . 6
1.4. Structure of the thesis . . . 7

2. Computational Mechanics: Initial and Boundary Value Problems **9**
2.1. Beam mechanics: Euler-Bernoulli beam theory . . . 9
2.1.1. Overview . . . 9
2.1.2. Governing equations . . . 10
2.1.2.1. Time-dependent beam equation . . . 10
2.1.2.2. Static beam equation . . . 10
2.1.3. Boundary conditions and beam supports . . . 11
2.1.3.1. Cantilever beam . . . 11
2.1.3.2. Simply supported beam . . . 12
2.1.3.3. Clamped beam . . . 12
2.2. Solid mechanics . . . 12
2.2.1. Overview . . . 12
2.2.2. Infinitesimal strain theory: Linear elasticity . . . 13
2.2.2.1. Governing equations . . . 13
2.2.2.2. Displacement-based formulation: Navier equations . . . 21
2.2.2.3. Mixed-variable formulation: the Hellinger-Reissner principle 22
2.2.2.4. Weak formulation . . . 22
2.2.3. Finite strain theory: Hyperelasticity . . . 24
2.2.3.1. Governing equations . . . 24
2.2.3.2. Stress measures . . . 26
2.2.3.3. Minimum potential energy principle . . . 27
2.3. Contact mechanics . . . 28
2.3.1. Overview . . . 28
2.3.2. Frictionless contact under small deformations . . . 28
2.3.2.1. Governing equations . . . 28
2.3.2.2. Mixed-variable contact formulation . . . 31
2.3.3. Frictionless contact under large deformations . . . 31

2.4. Rough surface contact problems . . . 35
2.4.1. Overview . . . 35
2.4.2. Problem formulation . . . 37
2.4.3. Random Midpoint Displacement algorithm . . . 39
2.4.4. Statistical characterization of topographies . . . 40
2.4.5. State of the art: BEM-based numerical simulations . . . 41

3. Fundamentals of Machine Learning 47
3.1. Machine learning terminology . . . 47
3.2. Types of learning systems . . . 48
3.2.1. Learning type . . . 49
3.2.1.1. Supervised learning . . . 49
3.2.1.2. Unsupervised learning . . . 50
3.2.1.3. Semi-supervised learning . . . 51
3.2.1.4. Reinforcement learning . . . 52
3.2.2. Output type . . . 52
3.2.2.1. Classification . . . 52
3.2.2.2. Regression . . . 53
3.3. Regression in machine learning . . . 53
3.3.1. k-Nearest Neighbors . . . 55
3.3.2. Linear Regression . . . 57
3.3.2.1. Polynomial Regression . . . 59
3.3.2.2. Shrinkage methods . . . 60
3.3.3. Kernel-based methods . . . 63
3.3.3.1. Support Vector Machines: SVM Regressor . . . 63
3.3.3.2. Kernel Ridge Regressor . . . 65
3.3.3.3. Gaussian Process Regressor . . . 67
3.3.4. Tree-based models . . . 69
3.3.4.1. Decision Tree . . . 70
3.3.4.2. Random Forest . . . 71
3.3.4.3. AdaBoost . . . 73
3.3.4.4. Gradient Boost . . . 75
3.3.4.5. XGBoost . . . 77
3.4. Neural networks . . . 79
3.4.1. Forward propagation and backpropagation . . . 81
3.4.2. Learning algorithm and optimizers . . . 86
3.4.2.1. Common optimizers . . . 86
3.4.3. Hyperparameters of neural networks . . . 90
3.4.3.1. Layers and neurons . . . 90
3.4.3.2. Activation function . . . 91
3.4.3.3. Network initializer . . . 97
3.4.3.4. Cost (loss) function . . . 98
3.4.3.5. Optimizer . . . 99
3.4.4. Regularization in neural networks . . . 101
3.4.4.1. Weight decay . . . 101

3.4.4.2. Dropout . . . 102
3.4.4.3. Data augmentation . . . 102
3.4.4.4. Early stopping . . . 103
3.4.5. Architecture types . . . 103
3.4.5.1. Fully connected neural networks . . . 103
3.4.5.2. Convolutional neural networks . . . 103
3.5. Main challenges of machine learning . . . 107
3.5.1. Quantity and quality of data . . . 108
3.5.2. Data preprocessing . . . 108
3.5.3. Underfitting vs. overfitting . . . 112
3.6. Hyperparameter optimization . . . 114
3.7. Illustrative example: Using CNNs to predict load in beam models . . . 115

4. Application of Machine Learning Techniques to Rough Surface Contact Problems 119
4.1. Data-driven problem: learning a surrogate model . . . 119
4.2. Surrogate model for prediction of effective contact area . . . 121
4.3. Methodolody . . . 123
4.3.1. Surrogate modeling workflow . . . 123
4.3.2. Generation of database . . . 125
4.3.3. Fast-to-evaluate modeling with hyperparameter optimization . . . 126
4.3.4. Accuracy metrics . . . 129
4.4. Results and discussion . . . 131
4.4.1. Evaluation of baseline models . . . 131
4.4.2. Evaluation of tuned models . . . 135
4.4.3. Comparative cost analysis of reference and surrogate models . . . 137
4.4.4. Discussion . . . 140
4.5. Conclusion . . . 141

5. Combining Neural Networks with Physics: Physics-Informed Neural Networks 143
5.1. Generic PINNs with output transformation . . . 145
5.1.1. PINNs for inverse problems . . . 147
5.1.1.1. Network Inversion . . . 147
5.1.1.2. Data-driven discovery . . . 147
5.1.1.3. Learnable parameter via optimization . . . 149
5.2. Element-based PINN formulations . . . 149
5.2.1. Variational PINNs . . . 150
5.2.2. Element-based energy PINNs . . . 153
5.2.3. Numerical integration . . . 155
5.3. Algorithmic challenges . . . 161
5.3.1. Vanishing gradients: Choosing a proper activation function . . . 162
5.3.2. Stiffness of PDEs . . . 164
5.3.3. Balancing of loss terms . . . 164
5.3.4. Non-uniqueness in inverse problems . . . 164

5.3.5. Gradient calculation . . . 165
5.3.6. Selection of an appropriate optimizer . . . 165

6. Application of Physics-Informed Neural Networks to Computational Mechanics 167
6.1. Use Case 1: Beam mechanics . . . 167
6.1.1. PINN formulation for beam mechanics . . . 167
6.1.1.1. Dynamic beam equation . . . 167
6.1.1.2. Static beam equation . . . 168
6.1.2. Numerical examples . . . 170
6.1.2.1. Static models . . . 171
6.1.2.2. Inverse problem: Load discovery . . . 175
6.1.2.3. Dynamic models . . . 176
6.2. Use case 2: Linear elasticity . . . 179
6.2.1. PINNs with displacement-based formulation . . . 179
6.2.1.1. Elastodynamics . . . 179
6.2.1.2. Elastostatics . . . 185
6.2.2. PINNs with mixed-variable formulation . . . 187
6.2.2.1. Elastodynamics . . . 187
6.2.2.2. Elastostatics . . . 192
6.2.2.3. Importance of output scaling . . . 194
6.2.3. Comparison of displacement-based and mixed-variable approaches . . . 195
6.2.4. Numerical examples . . . 197
6.2.4.1. Lamé problem of elasticity . . . 198
6.2.4.2. Inverse problem: Identifying material parameters . . . 201
6.2.4.3. 3D hollow sphere subjected to internal pressure . . . 203
6.2.4.4. 2D block under time-dependent loading . . . 205
6.2.4.5. 4D problem: 3D hollow sphere under time-dependent loading 211
6.3. Use Case 3: Contact mechanics under linear elasticity . . . 212
6.3.1. PINNs with mixed-variable formulation . . . 212
6.3.2. Enforcing Karush-Kuhn-Tucker constraints . . . 215
6.3.2.1. Sign-based method . . . 217
6.3.2.2. Sigmoid-based method . . . 217
6.3.2.3. Nonlinear complementarity function: Fischer-Burmeister . . . 218
6.3.3. Numerical examples . . . 221
6.3.3.1. 2D contact between an elastic block and a rigid surface . . . 222
6.3.3.2. 2D Hertzian contact problem . . . 225
6.3.3.3. 3D single contact patch test . . . 232
6.3.3.4. 3D cylindrical contact problem . . . 233
6.4. Use Case 4: Large deformations . . . 236
6.4.1. EBE-PINNs for solid mechanics . . . 238
6.4.2. EBE-PINNs for contact mechanics . . . 242
6.4.3. Numerical examples . . . 244
6.4.3.1. 2D bending beam . . . 244
6.4.3.2. 3D torsion of a square prism . . . 247

6.4.3.3. 3D spherical contact problem 251
6.4.3.4. 3D torus contact instability problem 257

7. Towards a Hybrid Digital Twin **263**
7.1. The concept of hybrid digital twins 263
7.1.1. Proof of concept for a steel-reinforced concrete beam 265
7.1.2. Physical asset and sensor information 266
7.2. Surrogate modeling of a reinforced concrete beam 268
7.2.1. Capturing the temporal response 269
7.2.1.1. Motion of a simple harmonic oscillator 270
7.2.1.2. A PINN-based temporal ROM 270
7.2.1.3. Comparison of temporal models 271
7.2.2. Capturing the spatial response 273
7.2.2.1. A PINN-based spatial ROM 274
7.2.2.2. Comparison of spatial ROM scenarios 276
7.2.3. Conclusion 276

8. Summary and Outlook **281**

A. Comparison of GP Regressor and Kernel Ridge **287**

List of Figures

1.1. An overview of hybrid modeling, illustrating the integration of physics-assisted machine learning and machine learning-assisted simulation, along with key subfields such as physics-informed neural networks, surrogate modeling, and model order reduction. 5

2.1. Comparison between Euler-Bernoulli and Timoshenko beam theories. 9
2.2. A fixed-fixed supported beam under a space-time dependent distributed load. . 10
2.3. An illustration of a fixed supported beam. 11
2.4. An illustration of a simply supported beam. 12
2.5. An illustration of a clamped beam. 12
2.6. Illustration of reference configuration and current configuration of deformed object. In the reference configuration, the elastic body is denoted by Ω_0, and in the deformed (current) configuration, it is represented by Ω_t. 14
2.7. Plane strain configuration. 16
2.8. Plane stress configuration. 17
2.9. Tonti's diagram for the displacement-based formulation of linear elasticity problems. 22
2.10. Tonti's diagram based on Hellinger-Reissner principle for linear elasticity problems with small strain theory. 23
2.11. Illustration of a contact problem between an elastic body and a rigid obstacle. (a) Reference configuration, (b) deformed (current) configuration, (c) boundary conditions, showing the gap g_n, tangential traction $t_{\boldsymbol{\tau}}$, and contact pressure p_n. . 29
2.12. Tonti's diagram of Hellinger-Reissner (HR) principle applied to linear elastic contact problems with frictionless sliding conditions. 32
2.13. Graphical representation of the problem under consideration. At a macroscopic scale, two deformable bodies come into contact under the action of an external imposed displacement Δ. At the microscopic scale, roughness features can be revealed, resulting in an effective contact area A_e much smaller than the nominal contact area that is assumed at the macroscopic scale. 38
2.14. Schematic depicting the standard rough surface contact problem. Since a composite formulation is applied, the rough surface is assumed to be rigid while the flat surface is linear elastic with composite material properties. 39
2.15. An illustration of the *Random Midpoint Displacement* algorithm. A higher interaction count k results in a finer discretization of the generated self-affine fractal surface. 40
2.16. (a) 3D representation of a rough topography. (b) A corresponding 2D profile characterized by specific statistical parameters, such as peak heights $z_{\mathrm{p},i}$ with accompanying maximum and mean values, $z_{\mathrm{p}}^{\max}$, $\bar{z}_{\mathrm{p}}$, respectively. 41

2.17. Under the action of the vertical far-field displacement Δ, the deformable bodies are pushed into contact. To properly formulate the LCP problem, an interference $\overline{\boldsymbol{u}}$ can be defined, equal, at each node, to $\boldsymbol{z} - (z_{\mathrm{p}}^{\max} - \Delta)$. 42
2.18. The top row of images (Fig. a-c) shows two surface topographies generated by the RMD. Dark blue indicates the deepest valleys, while dark red represents the highest summits. The bottom row of images (Fig. d-f) shows the resulting contact force per grid point f, for two far-field displacements $\Delta_0 = 25\,\mu\mathrm{m}$ and $\Delta_1 = 35\,\mu\mathrm{m}$. The corresponding effective contact area A_e is also shown. 43
2.19. Comparison between analytical and BEM solutions for the paraboloid rigid indenter example. BEM models have surfaces discretized on a grid with 128×128 and 256×256 points per side, respectively. 44
2.20. Comparison of simulation time for rough surface contact simulations with a grid size of 128×128. (a) shows the effect of different far-field displacements and surface realisations on the overall simulation time. Each point represents a randomly generated surface and a specific far-field displacement. (b) The correlation between effective contact area and simulation time. 45

3.1. Comparison of classical programming and machine learning. 47
3.2. An illustrative example for supervised learning (classification). 50
3.3. An illustrative example for unsupervised learning (clustering). 51
3.4. An illustrative example of reinforcement learning. 53
3.5. List and categorization of regression models in machine learning. 54
3.6. Comparison of results of k-NN Regressor for different values of k. Subplots demonstrate the k-NN regression fit for $k = 1$ (left), $k = 3$ (center), and $k = 10$ (right). 56
3.7. Comparison of Linear Regression and Polynomial Regression training data. . . 59
3.8. The left figure shows Ridge Regression with varying regularization parameters and Polynomial Regression with Ridge regularization ($p = 5$) is given in the right figure. 61
3.9. The left figure shows Lasso Regression with varying regularization parameters and Polynomial Regression with Lasso regularization ($p = 5$) is given in the right figure. 62
3.10. An illustration for Support Vector Machine (SVM) Regressor. 64
3.11. Comparison of training data and predictions of SVM Regressor models based on different kernels. 65
3.12. Comparison of training data and predictions of Kernel Ridge models based on different kernels. 66
3.13. Comparison of training data and predictions of GP Regressor models based on different noise variance values σ_n^2. 69
3.14. A simple decision tree illustrating the components of tree-based models. 69
3.15. Comparison of the predictions of the Decision Tree models trained with different maximum depths. 71
3.16. Workflow of the Random Forest algorithm. 72
3.17. Comparison of the predictions of the Random Forest models trained with different maximum depths and number of Decision Trees (# of estimators). 73

3.18. Comparison of the predictions of the AdaBoost models trained with different number of weak learners (Decision Trees). 75
3.19. Comparison of the predictions of the Gradient Boost models trained using different numbers of weak learners. 77
3.20. Comparison of the predictions of the XGBoost models trained using different numbers of boosting trees. 79
3.21. An illustration to show the components of a neural network. 80
3.22. Computation at a hidden neuron shown in two approaches. (a) The cumulative method computes the weighted sum of inputs plus a bias and applies an activation function. (b) The vectorized method performs the same calculation using augmented input vectors. 81
3.23. Schematic of a simple neural network architecture, including weight and bias terms. 82
3.24. The linear activation function and its derivatives. 91
3.25. The sigmoid activation function and its derivatives. 92
3.26. The hyperbolic tangent (tanh) activation function and its derivatives. 92
3.27. Rectified linear unit (ReLU) activation function and its derivatives. 93
3.28. The Leaky ReLU activation function and its derivatives. 93
3.29. The parametric RELU (PReLU) activation function and its derivatives for different α values. 94
3.30. The exponential linear unit (ELU) activation function and its derivatives for $\alpha = 1$. 94
3.31. The Swish activation function and its derivatives for $\beta = 0.1$. 95
3.32. Effect of different learning rates on the convergence of the loss function. 100
3.33. Illustration of dropout in a fully connected neural network. (a) Without dropout, where all neurons are active. (b) With dropout, where a subset of neurons is randomly deactivated during training. 102
3.34. Illustration of early stopping. 103
3.35. (a) An illustration of a deep neural network for a grid-like topology and (b) adding convolutional layers to convert the network into a CNN. 104
3.36. Visualization of a grayscale image along with the pixel values of a selected region. 105
3.37. An illustration of convolution operation on a 4×4 pixel matrix using a 2×2 filter. 106
3.38. Types of pooling. 107
3.39. Handling with missing values: (a) row data with NaN, (b) removing the rows containing NaNs, (c) removing a row, and imputation by mean and backward fill. 110
3.40. Bias-variance tradeoff. 113
3.41. An illustration of 5-fold cross-validation. 114
3.42. Hyperparameter configuration for (a) grid search and (b) random search. 115
3.43. (a) A clamped beam under a uniformly distributed load $q(x)$. (b) Deflection curves for different loads. 116
3.44. Images for deflection curves of the beam example under different loads. 116
3.45. Comparison of the actual load (q_{act}) and the CNN predictions (q_{pred}). 118

4.1. Surrogate modeling framework for the effective contact area in rough surface contact: (a) The reference model f computes $\mathcal{A}_\mathrm{e}$ as a function of far-field displacement Δ and surface height field $\boldsymbol{z}$, requiring costly numerical simulations; (b) the surrogate model $\hat{f}_{\boldsymbol{w}}$ approximates this using a fast-to-evaluate regression model $\hat{h}_{\boldsymbol{w}}(\Delta, \boldsymbol{\vartheta})$, where $\boldsymbol{\vartheta} = g(\boldsymbol{z})$ is a vector of statistical parameters. The function g is a known deterministic mapping that enables dimensionality reduction of the input vector. 122
4.2. Data-driven surrogate modeling workflow to solve rough surface contact simulations. 124
4.3. Visualization of the database. (a) distribution of far-field displacement, and (b) scatter plot showing the target effective contact area against the feature far-field displacement. Each point represents a triplet consisting of a rough surface, a far-field displacement, and the corresponding effective contact area. Variations in effective contact area at a fixed far-field displacement are due to differences in the characteristics of the rough surfaces. 126
4.4. Classification of surrogate modeling techniques based on their characteristics. . 128
4.5. Comparison of the investigated baseline models to predict the effective contact area (%). Horizontal axes are the observations or actual values, while vertical axes denote the predictions. The red lines represent a perfect fit, which is used to qualify the model's accuracy. 133
4.6. Comparison of the investigated data-driven baseline models to predict the effective contact area for rough surface contact: (a) Accuracy metrics (%), (b) times for fit and prediction in % are given including the corresponding actual values in s. Note that each accuracy metric is scaled relative to the worst accuracy across all models to enable better comparisons. Similarly, each prediction and fit time measures are scaled relative to the maximum prediction and fit time across all models. 134
4.7. Comparison of the baseline models with the tuned models. Horizontal axes represent the observations or actual values, while vertical axes denote the predictions. The red lines represent a perfect fit, which is used to qualify the model's accuracy. 136
4.8. Comparison of the total computational cost for the BEM model and the Kernel Ridge surrogate model. For BEM, the cost is estimated using the average simulation time ($189.79\,\mathrm{s}$) per sample. The surrogate model cost includes four components: prediction time (t_p), training time (t_f), hyperparameter optimization time (t_h), and database generation time (t_d), with total cost $t_{\mathrm{KRR}} = t_p + t_f + t_h + t_d$. The surrogate model's overall cost is dominated by database generation (t_d), while training and hyperparameter optimization contribute marginally. The surrogate becomes more cost-efficient than BEM after approximately 15,893 simulations. 139

5.1. Illustration of the trade-off between data and physics. 144
5.2. The general representation of a physics-informed neural network for a BVP. . . 146
5.3. Approaches for solving inverse problems using PINNs: (a) Network inversion, (b) Data-driven discovery, (c) Learnable parameter optimization. 148
5.4. The general representation of variational physics-informed neural networks. . 152

5.5. An illustration of introduced test functions ϕ_k and their first-order derivatives $\partial\phi_k$ up to $k = 6$: (a) modified Legendre polynomials, (b) Legendre shape functions, and (c) Lagrange shape functions. 154
5.6. The general representation of element-based energy physics-informed neural networks (EBE-PINNs). 156
5.7. Comparison of ReLU and tanh activation functions for solving an ODE system. Plots include the comparison of predicted solutions with true or analytical solutions and corresponding ODE loss evolutions through epochs. 163

6.1. An illustration of the application of PINNs for dynamic one-dimensional Euler-Bernoulli beams. 169
6.2. An illustration of the application of PINNs for static one-dimensional Euler-Bernoulli beams. 169
6.3. Three investigated beam models with their descriptions and analytical solutions. 171
6.4. On the figures left, the predicted and true deflections through the coordinate x are compared for three different cases. On the figures right, the loss history for PDE and BCs is given through epochs. 173
6.5. An illustration of a simple supported beam having a non-uniformly distributed cross-section under an arbitrary load 174
6.6. Results for the complex beam example. On the left is the comparison of displacements obtained from the analytical solution and the predicted values, on the right, the evolution of the MSE for the PDE loss and BCs loss. 174
6.7. Results for the inverse problem. (a) Comparison of displacements obtained from the analytical solution and the predicted values, (b) comparison of the analytical and predicted loads, (c) evolution of the MSE for the PDE, BCs and the experimental data (EXPs). 176
6.8. Results for the inverse problem with a prior knowledge of the distributed load. (a) Comparison of displacements obtained from the analytical solution and the predicted values, (b) comparison of the analytical and predicted loads, (c) evolution of the MSE for the PDE, BCs and the experimental data (EXPs) 177
6.9. Results for the time-dependent beam example. The figure above shows the predicted solution, while the figures below compare the predicted solution to the analytical solution when one input dimension is kept constant. 178
6.10. Physics-informed neural networks using the displacement-based formulation to solve elastodynamics problems. 180
6.11. Physics-informed neural networks in the displacement-based approach for elastostatics problems. 185
6.12. Physics-informed neural networks in the mixed-variable formulation to solve elastodynamics problems. 189
6.13. Physics-informed neural networks in the mixed-variable form to solve elastostatics problems. 194
6.14. The Lam'e elasticity problem. (a) The full cylinder under internal pressure, and (b) the equivalent reduced model due to axisymmetry in geometry and loading, along with boundary conditions. 198

6.15. Results for the Lamé problem of elasticity with and without output transformation. (a,d) Comparison of normalized stresses obtained from the analytical solution and the predicted values, (b,e) comparison of normalized displacements, and (c,f) evolution of the MSE for the cumulative PDE loss and NBC loss. 200
6.16. Results of the Lamé problem of linear elasticity for the mixed-variable formulation. (a) Comparison of normalized stresses obtained from the analytical solution and the predicted values, (b) comparison of normalized displacements, and (c) evolution of the MSE for the cumulative PDE loss and NBC loss. 201
6.17. Identification of Young's modulus for the Lamé problem of elasticity. 202
6.18. A hollow sphere subjected to internal pressure. (a) The complete geometry, including radial dimensions. (b) One-eighth of the hollow sphere under internal pressure and Dirichlet boundary conditions due to the axisymmetric nature of both the geometry and the loading. (c) Dirichlet and Neumann boundary conditions to be enforced as hard constraints. (d) Neumann boundary conditions to be enforced as soft constraints. 204
6.19. Results for the hollow sphere subjected to internal pressure. 206
6.20. Comparison of the predicted values and analytical solutions for the hollow sphere subjected to internal pressure. (a) Comparison of stresses in the rr- and $\theta\theta$-directions, (b) comparison of stresses in the $\phi\phi$-direction, and (c) comparison of the radial displacements u_r. 207
6.21. A 2D block under time-dependent loading. (a) geometry, supports, loading, (b) all relevant boundary conditions. 207
6.22. Results for the 2D block under time-dependent loading at $t = 0.2$. Predicted and analytical solutions for the displacement (u_x, u_y) and stress ($\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$) components. Additionally, corresponding point-wise absolute errors are provided, which are defined as $E^*_{abs} = \text{abs}(\tilde{*} - *)$ 209
6.23. Comparisons between the predicted values and analytical solutions of the 2D block under time-dependent loading for different time steps. (a) Comparison of the displacement component u_x, (b) comparison of the displacement component u_y, and (c) comparison of the normal stress component σ_{yy}. 210
6.24. Predicted solutions of the hollow sphere subjected to time-dependent loading for three steps: $t = 0.25\,\text{s}$, $t = 0.5\,\text{s}$ and $t = 0.75\,\text{s}$. 213
6.25. Comparison between analytical and predicted solutions for the hollow sphere subjected to time-dependent loading at different time steps. (a) Radial stress σ_{rr}, (b) circumferential stress $\sigma_{\theta\theta}$, (c) circumferential stress $\sigma_{\phi\phi}$, and (d) radial displacement u_r. 214
6.26. Physics-informed neural networks in the mixed-variable form to solve normal contact problems. (a) Time-dependent, (b) quasi-static. 216
6.27. Illustration of the sign-based functions for enforcing constraints depending on the gap $\tilde{g}_n$ and contact pressure $\tilde{p}_n$. 217
6.28. Illustration of the Sigmoid-based function applied to the gap g_n and contact pressure p_n for different values of the steepness parameter δ, which controls the smoothness of the transition in the loss function. 219

6.29. Visualization of the Fischer-Burmeister NCP function based on the gap g_n and contact pressure p_n, where (a) is the 3D surface plot and (b) represents the projection of the function as 2D contour plot. 220
6.30. Illustration of the domination effect of p_n over g_n in the Fischer-Burmeister NCP function. (a) No domination when g_n and p_n are of similar scales. (b) Intermediate domination as the scale of p_n increases. (c) Strong domination where p_n significantly outweighs g_n, causing optimization challenges. 221
6.31. Contact problem between a linear-elastic block and a rigid flat surface: (a) geometry, supports, and loading, and (b) equivalent system including all relevant boundary conditions. 222
6.32. Predictions of the displacement component u_y, stresses σ_{yy} and σ_{xy} with corresponding absolute errors $E_{\text{abs}}^{u_y}$, $E_{\text{abs}}^{\sigma_{yy}}$, $E_{\text{abs}}^{\sigma_{xy}}$. The contact constraints $\mathcal{L}_{\text{KKT}}$ is enforced via three different methods. Absolute error is defined as $E_{\text{abs}}^{*} = \text{abs}(\tilde{*} - *)$. 224
6.33. The Hertzian contact problem between an elastic half-cylinder and a rigid flat surface (a) domain under uniform pressure on top and making use of symmetry, (b) accompanying boundary conditions. 225
6.34. PINNs as pure forward solver for the Hertzian contact problem. Comparison of stress and displacement components obtained by PINN and FEM. 227
6.35. Comparison of contact pressure distributions $p_c(x)$ obtained by analytical solution, PINN and FEM for case 1. 227
6.36. Comparison of contact pressure distributions $p_c(x)$ obtained by analytical solution, PINN and FEM for case 2. (a) PINNs as a data-enhanced forward solver, (b) distribution of data points generated through FEM simulations in the domain and on the boundary. 228
6.37. Identification of the applied external pressure on the half-cylinder for different initial guesses (a) $\tilde{p}_o = 0.1$, (b) $\tilde{p}_o = 5$, (c) $\tilde{p}_o = 20$. 229
6.38. An illustration of the procedure to include the applied external pressure as a neural network input. 230
6.39. Comparison of the contact pressure distribution obtained through a PINN-based surrogate model to the analytical solution. The PINNs are trained using different numbers of chunks. 231
6.40. Single patch contact problem: (a) geometry, constraints, and loading, (b) relevant boundary conditions including NBCs, DBCs, KKT, and FSCs. 232
6.41. Comparison of the PINN predictions and the analytical solution for the single contact patch test. Point-wise absolute error is defined as $E_{\text{abs}}^{*} = \text{abs}(\tilde{*} - *)$. . . 233
6.42. The Hertzian contact problem between an elastic quarter-cylinder and a rigid flat domain: (a) The full contact problem under load F, including geometry and dimensions, (b) the width of the contact zone, (c) the equivalent problem due to the axisymmetrical nature of both geometry and loading (d) all relevant boundary conditions. 235
6.43. The predictions of the plain vanilla PINN for the cylindrical contact example. Polar components are computed based on the cylindrical coordinates. 237
6.44. Comparison of the predictions with the analytical solution for the cylindrical contact problem between the plain vanilla PINN and data-enhanced PINN. . . . 238

6.45. The data-enhanced PINN predictions for the Hertzian contact example. Polar components are computed based on the cylindrical coordinates. 239
6.46. Element-based energy physics-informed neural networks (EBE-PINNs) solve quasi-static finite strain solid mechanics problems. 240
6.47. Element-based energy physics-informed neural networks (EBE-PINNs) solve quasi-static finite strain contact mechanics problems. 243
6.48. 2D bending beam under shear loading. (a) Geometry, boundary conditions, and applied traction, (b) corresponding mesh with quadrilateral elements. 245
6.49. Comparison of predicted and reference displacement components (u_x, u_y) for the 2D beam under shear loading between EBE-PINN and FEM. The bottom row shows the corresponding point-wise absolute errors, defined as $E^*_{abs} = \mathrm{abs}(\tilde{*}-*)$, where $\tilde{*}$ denotes the predicted quantity. 246
6.50. Comparison of predicted and reference Cauchy stress components (σ_{xx}, σ_{yy}, σ_{xy}) for the 2D beam under shear loading using the EBE-PINN and FEM. Similar to displacement results, corresponding point-wise absolute errors are defined as $E^*_{abs} = \mathrm{abs}(\tilde{*} - *)$, where $\tilde{*}$ denotes the predicted quantity. 248
6.51. Torsion of a prism. (a) Geometry, boundary conditions, and loading; (b) corresponding mesh with hexahedral elements. 249
6.52. Predicted and reference displacement components (u_x, u_y, u_z) for the torsion of the prism example shown in the deformed configuration. Results from EBE-PINN and FEM are shown in the first and second rows, respectively. The third row presents the corresponding point-wise absolute errors. 250
6.53. Predicted and reference Cauchy stress components (σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{xz}) for the torsion of the prism example. Results from EBE-PINN and FEM are shown in the first and second rows, respectively. The third row presents the corresponding point-wise absolute errors. 252
6.54. Comparison of displacement and Cauchy stress components along a line in the 3D torsion problem between FEM and EBE-PINNs. 253
6.55. Comparison of Green-Lagrange strain tensor components along a line in the 3D torsion problem between FEM and EBE-PINN. 253
6.56. Spherical contact problem between an elastic sphere and a rigid flat domain. (a) The full contact problem under load F, including geometry and dimensions, (b) the equivalent problem due to the axisymmetrical nature of both geometry and loading, including boundary conditions, (c) corresponding mesh with tetrahedral elements. 255
6.57. Comparison of displacement and Green-Lagrange strain components predicted by EBE-PINN and FEM for the 3D spherical contact problem. 256
6.58. Comparison of Cauchy stress components predicted by the EBE-PINN model and FEM for the 3D spherical contact problem. The top two rows show the normal stress components σ_{xx}, σ_{yy}, and σ_{zz}, while the bottom two rows present the shear stress components σ_{xy}, σ_{yz}, and σ_{xz}. 258

6.59. Torus contact instability example. (a) The full contact problem under displacement loading u_y including geometry and dimensions, (b) the equivalent problem due to the axisymmetrical nature of both geometry and loading, including boundary conditions, (c) the tetrahedral mesh used for the quarter torus contact instability example. 260
6.60. Demonstration of predicted displacement components (u_x, u_y, u_z) obtained using EBE-PINN for the 3D quarter torus contact instability example. 261
6.61. Demonstration of predicted Cauchy stress components (σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{xz}) obtained using EBE-PINN the 3D quarter torus contact instability example. 262

7.1. The conceptual workflow between hybrid digital twin, virtual twin, and digital twin. 264
7.2. Conceptual framework of a hybrid digital twin for a steel-reinforced concrete beam under a four-point bending test. 265
7.3. Schematic of the four-point bending test setup for the steel-reinforced concrete beam. The experiment aims to evaluate the flexural response of the beam under cyclic loading. Image adapted from [45]. 266
7.4. Strain measurements obtained from fiber-optical sensors during the four-point bending test under a peak load of $10\,\mathrm{kN}$. (a) The microstrain response of Fiber 1 on the compression side shows a typical parabolic bending profile. (b) The microstrain response of Fiber 2 on the tension side highlights regions of significant strain increase due to the presence of cracks and localized damage. 267
7.5. Temporal strain measurements recorded at the center of fiber 1 over a duration of 16 seconds. Each second contains ten individual measurement points, capturing the cyclic nature of the loading. 269
7.6. Schematic representation of the temporal reduced-order model (ROM) using a physics-informed neural network (PINN). 271
7.7. Comparison of the purely data-driven model and the physics-informed temporal ROM, including training data and sensor measurements. The data-driven model aligns well with the training data, but diverges outside the training interval, whereas the temporal ROM maintains accurate predictions beyond the training period. 272
7.8. Convergence of the identified natural frequency ω_0. The optimization begins with the *Adam* optimizer and switches to *L-BFGS-B*, resulting in rapid convergence to $\tilde{\omega}_0^2 \approx 9.74\,\mathrm{rad}^2\,\mathrm{Hz}^2$. 273
7.9. Schematic of the spatial ROM based on the PINNs in the mixed-variable formulation. 274
7.10. (a) Schematic representation of the relaxed system with simplified boundary conditions. (b) Detailed boundary conditions of the relaxed model, showing the fixed displacement at $x = 0$ and traction BCs on the left and right edges. 275
7.11. Strain predictions using the spatial ROM along fiber 1 (compression side, top row) and fiber 2 (tension side, bottom row). (a, b) Scenario 1: Only compression data is included. (c, d) Scenario 2: Both compression and tension data are included. (e, f) Scenario 3: Both compression and tension data are included, but the sensor loss weight for tension data is set to $w_{\mathrm{EXPs,T}} = 0.01$. 277

7.12. Strain predictions using the spatial ROM within the beam domain for the three scenarios. The strain distribution is visualized, showing how the model predictions adapt with varying sensor data contributions. 278

A.1. Case 1: Good generalization. 288
A.2. Case 2: Bad generalization. 288
A.3. Comparison of GP Regressor and Kernel Ridge under the assumption of both models having the same kernel including identical kernel parameters and the regularization parameter. 288

List of Tables

3.1. Comparison of bias and weights for Ridge Regression and Polynomial Regression with regularization for varying regularization parameters (λ). 61
3.2. Comparison of bias and weights for Lasso Regression and Polynomial Regression with Lasso regularization for varying regularization parameters (λ). 62
3.3. First and second derivatives of the introduced activation functions in neural networks. 96

4.1. The list of parameters of the generated raw database including material parameters, model parameters, statistical parameters, and the target quantity. Each row contains a parameter with corresponding symbol and the range of variation. . . 127
4.2. Investigated surrogate model hyperparameter optimization, including the model type, selected hyperparameters, the defined parameter search space, and the optimal parameter value. 130
4.3. Assessment of time required for performing hyperparameter optimizations. . . 131
4.4. Comparison of the baseline models with the tuned models. Accuracy metrics nMSE, nMAE, nMaxE, R^2 are evaluated on the test data. Improvement is calculated based on $100(m_t - m_b)/m_b$ where m_b is the evaluated performance metric of the baseline model and m_t is the evaluated performance metric of the tuned model. The cells with blue background show the model accuracy of the best baseline model, the cells with orange background denote those of the best-tuned model, while the cells with green background show the model accuracy and performance of the model considered "optimal" overall. The model accuracy is determined based on accuracy metrics, and the model performance is assessed by examining fit and prediction time. The fit (or training) time refers to the time required to train the models on the training dataset, whereas the prediction time is measured on the test dataset and represents the total time needed to make predictions for all test data points. 138

5.1. Gauss points ξ_i, η_j and weighting factors c_i of Gauss-Legendre quadrature for triangular elements. 157

6.1. Comparison of displacement-based and mixed-variable PINN approaches for elastodynamics and elastostatics in 2D and 3D, showing the maximum order of time and spatial derivatives, hard enforcement capabilities, and the number of loss terms and weights. 196
6.2. Comparison of PINN results with and without the defined output transformation for the Lamé elasticity problem. 199

6.3. Comparison of relative errors, training time and number of epochs for the inverse Lamé problem of elasticity problem used to identify Young's modulus. The relative absolute error is defined as $E^{*}_{\text{rel,abs}} = \text{abs}\left((\tilde{*} - *)/*\right)$. 203
6.4. Relative L_2 errors for displacement ($E^{\boldsymbol{u}}_{L_2}$) and stress ($E^{\boldsymbol{\sigma}}_{L_2}$) fields at different time steps for the 2D block under time-dependent loading. These errors are computed based on Fig. 6.23. 210
6.5. Relative L_2 errors for displacement ($E^{\boldsymbol{u}}_{L_2}$) and stress ($E^{\boldsymbol{\sigma}}_{L_2}$) fields at different time steps for the hollow sphere under a time-dependent loading. These errors are computed based on Fig. 6.25. 212
6.6. Comparison of hidden layer architectures, performance metrics, and error measurements for the 2D contact problem involving an elastic block and a rigid surface. The KKT constraint enforcement methods include (a) sign-based, (b) Sigmoid-based, and (c) Fischer-Burmeister approaches. 223
6.7. The structure of hidden layers, number of input neurons, training and test points, and loss weights are given for different cases of the Hertzian contact problem. The remaining loss weights are set as 1. 226
6.8. Comparison of relative L_2 errors, training and prediction time for case 1 and case 2. The term $\int_{\Omega_c} \tilde{p}_c$ denotes an integral of the predicted contact pressure $\tilde{p}_c$ over the potential contact boundary. 228
6.9. Comparison of relative errors, training time, and epochs for identifying the applied external pressure. The relative error is defined as $E = \text{abs}\left((\tilde{*} - *)/*\right)$. 230
6.10. Comparison of relative L_2 errors for the contact pressure distribution $p_c(\boldsymbol{x}, p)$ between the analytical solution and the predictions of our surrogate PINN models. Predictions are based on unseen pressure values. 231
6.11. Comparison of relative L_2 errors (%) between the predictions and the analytical solution for the plain vanilla PINN and the data-enhanced PINN. 236

Nomenclature

Representation of scalars, tensors and other quantities

q, Q	Scalar quantity
$\boldsymbol{q}$	Vector
$\boldsymbol{Q}$	Second-order tensor

Chapter 2: Computational Mechanics: Initial and Boundary Value Problems

A	Cross-sectional area
E	Young's modulus
ρ	Material density
q	Distributed load
x	Spatial coordinate
t	Time
$\delta()$	Dirac delta
I	Inertia
u	Vertical displacement
u_x	Beam slope
u_{xx}	Bending moment
u_t	Velocity
L	Beam length
$\boldsymbol{\sigma}$	Cauchy stress tensor
$\boldsymbol{u}$	Displacement field
$\ddot{\boldsymbol{u}}$	Acceleration field
$\hat{\boldsymbol{b}}$	Body force vector
$\boldsymbol{n}$	Unit outward normal vector in current configuration
$\hat{\boldsymbol{u}}$	Prescribed displacement field
$\partial\Omega_u$	Dirichlet boundary
$\hat{\boldsymbol{t}}$	Prescribed traction field
$\partial\Omega_\sigma$	Neumann boundary
Ω_0	Elastic body in the reference configuration
Ω	Elastic body in the deformed (current) configuration
$\boldsymbol{\varepsilon}$	Infinitesimal strain tensor
$\mathbb{C}$	Fourth-order elasticity tensor
λ, μ	Lamé parameters
$\boldsymbol{I}$	Identity tensor

δ_{ij}	Kronecker delta
ν	Poisson's ratio
$\boldsymbol{u}_0$	Initial displacement field
$\dot{\boldsymbol{u}}_0$	Initial velocity field
$\delta\hat{\boldsymbol{u}}$	Test function
$\mathcal{R}$	Residuum
$\boldsymbol{P}$	First Piola-Kirchhoff stress tensor
$\boldsymbol{N}$	Unit outward normal vector in reference configuration
$\boldsymbol{\varphi}$	Position vector mapping
$\boldsymbol{F}$	Deformation gradient
Ψ	Strain energy density function
$\boldsymbol{C}$	Right Cauchy-Green deformation tensor
J	Determinant of deformation gradient
$\boldsymbol{S}$	Second Piola-Kirchhoff stress tensor
$\boldsymbol{E}$	Green-Lagrange strain tensor
U	Internal energy
W_{ext}	External work
K	Kinetic energy
Π_{S}	Total potential energy for finite-strain problems
$\mathcal{L}$	Lagrangian function of a dynamic system
A_0	undeformed boundary surface
V_0	undeformed volume
g_n	Gap function
$\boldsymbol{t}_c$	Contact traction vector
$\boldsymbol{\tau}^{\xi}$	1. tangential direction in 3D
$\boldsymbol{\tau}^{\eta}$	2. tangential direction in 3D
p_n	Normal pressure
t_τ^ξ	Tangential traction component in ξ-direction
t_τ^η	Tangential traction component in η-direction
Π_{C}	Contact potential energy
$\boldsymbol{g}$	Gap vector
ϵ	User-defined and problem-specific penalty parameter
$\boldsymbol{\lambda}$	Lagrange multiplier
A_e	Effective contact area
n_c	Number of points in contact
g_l	Grid size
$\boldsymbol{z}$	Height field
H	Hurst exponent
Δ	Far-field displacement
F	Contact force
A_n	Normalized contact area

Chapter 3: Fundamentals of Machine Learning

T	Set of N training data pairs
$\boldsymbol{X}$	Input vector
$\boldsymbol{y}$	Target variable
$\boldsymbol{x}_*$	Unseen sample points
$\hat{y}$	Predicted output
$\hat{\boldsymbol{y}}$	Predicted output in vector form
b	Bias term (learnable parameters)
w	Weights (learnable parameters)
$\boldsymbol{\theta}$	Learnable parameters in vector form
J	Cost function in machine learning
η	Learning rate
n_{iter}	Number of iterations
N	Number of samples
d	Number of features
p	Polynomial order
λ	Regularization parameter
$\kappa(\cdot, \cdot)$	Kernel function
$\boldsymbol{K}$	Kernel matrix
ε	Margin for Support Vector Machines
ξ	Slack variable
N_s	Number of slack variables
C	Regularization parameter in Support Vector Machines
ϕ	Mapping or basis function
σ	Standard deviation
α	Model parameters (dual parameters)
σ_n^2	Noise variance
$\bar{\boldsymbol{f}}_*$	Predictive mean
$\mathbb{V}[\bar{\boldsymbol{f}}_*]$	Predictive variance
G	A weak learner
M	Number of weak learners
α_m	Contribution weight for a weak learner m
ψ	Activation function
L	Cost function in neural networks
$\boldsymbol{b}$	Bias term in vector form
$\boldsymbol{W}$	Weights in vector form
$\boldsymbol{z}$	Neural network input
δ	Error term in neural networks
$\boldsymbol{\nu}$	Local momentum vector or velocity
β	Momentum coefficient
ρ	Decay rate

Chapter 4: Application of Machine Learning Techniques to Rough Surface Contact Problems

$\boldsymbol{\vartheta}$	Statistical descriptors
$\boldsymbol{z}$	Height field input
$\hat{f}_{\boldsymbol{w}}$	Surrogate model
$\mathcal{A}_{\mathrm{e}}$	Target quantity
$\hat{\mathcal{A}}_{\mathrm{e}}$	Predicted target quantity
g	Known deterministic function
$\mathcal{D}$	Dataset
A_e	Effective contact area
$\hat{h}_{\boldsymbol{w}}$	Regression model
Δ	Far-field displacement input
nMSE	normalized mean square error
nMAE	normalized mean absolute error
nMaxE	normalized maximum error

Chapter 5: Combining Neural Networks with Physics: Physics-Informed Neural Networks

$\tilde{\boldsymbol{u}}$	Approximated (predicted) physics-informed neural network solution
$\mathcal{N}^L$	Network output in the output layer L
g	Prescribed boundary condition
s	Scaling parameter
h	Distance-to-boundary function
$\mathcal{L}_{\mathrm{PDE}}$	Losses due to partial differential equation (PDE)
$\mathcal{L}_{\mathrm{BCs}}$	Losses due to boundary conditions
$\mathcal{L}_{\mathrm{EXPs}}$	Losses due to experimental or measurement data
N_{rp}	Number of collocation points
N_{bp}	Number of boundary points
N_{ep}	Number of experimental data points
w_n	PDE loss weight
w_{bc}	Boundary condition loss weight
w_e	Experimental data loss weight
N_n	Number of total PDE loss weights
N_{bc}	Number of total boundary conditions weights
N_e	Number of total experimental PDE loss weights
λ	Unknown parameter to identify
$\boldsymbol{\phi}$	Test function over entire domain
$\boldsymbol{\phi}^e$	Localized and nonoverlapping test function
$\mathcal{R}^e$	Element-based residual
N_k	number of test functions
N^e	number of subdomains
$\mathbf{J}$	Jacobian matrix

c	Gauss weights
N^{gp}	Number of Gauss points
$\mathcal{L}^{\phi}_{\mathrm{PDE}}$	Overall variational loss
$\boldsymbol{\xi}$	Natural coordinates
$P()$	Legendre polynomial functions

Chapter 6: Application of Physics-Informed Neural Networks to Computational Mechanics

$\boldsymbol{x}$	Spatial coordinates
t	Time
$\mathcal{L}_{\mathrm{B}}$	Overall beam loss
$\mathcal{L}_{\mathrm{DBCs}}$	Losses due to Dirichlet boundary conditions
$\mathcal{L}_{\mathrm{NBCs}}$	Losses due to Neumann boundary conditions
$\mathcal{L}_{\mathrm{ICs}}$	Losses due to initial conditions
$\tilde{u}$	Predicted displacement
$\hat{u}$	Prescribed displacement
u^*	Additional measurement data
$\hat{u}_0$	Prescribed initial displacement
$\hat{\dot{u}}_0$	Prescribed initial velocity
$w_{\mathrm{bc,D}}$	Loss weight due to Dirichlet boundary condition
$w_{\mathrm{bc,N}}$	Loss weight due to Neumann boundary condition
$w_{\mathrm{ic,D}}$	Loss weight due to initial displacement
$w_{\mathrm{ic,V}}$	Loss weight due to initial velocity
$\tilde{\boldsymbol{u}}$	Predicted displacement in vector form
$\tilde{\boldsymbol{\sigma}}$	Predicted Cauchy stress (tensor)
$\mathcal{L}_{\mathrm{E}}$	Overall linear elasticity loss
$\mathcal{L}_{\mathrm{C}}$	Overall contact problem loss
$\mathcal{L}_{\mathrm{FSC}}$	Losses due to frictionless sliding conditions
$\mathcal{L}_{\mathrm{KKT}}$	Losses due to Karush-Kuhn-Tucker conditions
$w^{(\mathrm{KKT})}$	Loss weight due to Karush-Kuhn-Tucker conditions
$w^{(\mathrm{FST})}$	Loss weight due to frictionless sliding conditions
δ	Steepness parameter
ϕ_{FB}	Fischer-Burmeister function
$relErrorU$	Vector-based relative error for displacement
$relErrorS$	Vector-based relative error for displacement
$\boldsymbol{u^*}$	Additional measurement data in vector form
E_{abs}	Point-wise absolute error
m	Ineertial mass
k	Spring constant
ω_0	Natural frequency
ε	Longitudinal strain
$\tilde{\varepsilon}$	Predicted longitudinal strain

Operators and symbols

$\boldsymbol{\nabla} \cdot ()$	Divergence operator
$\mathrm{tr}(\cdot)$	Trace operator
$(\cdot)^{\top}$	Transpose of a vector, a matrix, or a tensor
$(\cdot)^{-1}$	Inverse of a vector, a matrix, or a tensor
$\cdot$	Dot or inner product
$\log$	Logarithm
$\det$	Determinant
$\odot$	Hadamard product or element-wise multiplication
$\oslash$	Hadamard division or element-wise division
$\mathcal{D}[\cdot]$	Differential operator
$\mathcal{B}[\cdot]$	Boundary operator
$(.)'$	User-defined output transformation

1. Introduction

1.1. Motivation

”Makine düşünebilir mi ve nasıl düşünebilir?”
–Can machines think and how can they think?–

- Cahit Arf, 1958/59 [5]

This profound question posed by Cahit Arf continues to resonate in modern computational science. While the concept of machines ”thinking” may have been speculative at the time, today's advancements in artificial intelligence (AI) and machine learning (ML) bring us closer to exploring and exploiting this capability in practical and innovative ways. This leads us to the next question: ”Can machines learn the physics that we seek?”

The field of computational engineering has reached a mature level particularly for complex real-world problems including solid and contact mechanics, fluid dynamics, structural analysis, and many others. In order to tackle such problems, traditional computational techniques, deeply rooted in physics-based principles, have provided a robust foundation for decades. However, their limitations become evident when faced with high-dimensional, nonlinear, and multi-scale problems that demand significant computational resources. Additionally, integration of data into traditional methods is not always straightforward and feasible with the fact that we live in an era dominated by data. These challenges or limitations are further compounded by the need to account for uncertainties in input data, necessitating the need for not deterministic but statistical approaches.

To address these challenges, data-driven techniques enabled by machine learning and neural networks have emerged as powerful and innovative tools. Unlike traditional approaches, machine learning models demonstrate great ability at identifying patterns, extracting insights, and making predictions from vast amounts of data. This capability allows them to bypass the need for explicit physical modeling in certain scenarios with a promising accuracy and a high computational efficiency. For example, ML-based surrogate models have demonstrated their potential to drastically reduce computation times while maintaining acceptable accuracy in tasks such as predicting the effective contact area in rough surface contact problems [143]. However, these purely data-driven methods are often referred to as *black-box* approaches, meaning that while they may produce accurate results, yet their underlying mechanisms remain largely hidden and difficult to understand. In other words, they somehow do the job, but we don't have a full understanding of how they work. This lack of transparency raises important questions about reliability and generalizability of the purely data-driven approaches that resonate with Cahit Arf's thought-provoking question posed decades ago: ”Can machines think, and how can they think?” In the context of computational engineering, this can be translated to: Can machines not only predict but also understand the underlying physical principles driving the target phenomena?

In order to lighten the ”black-box” nature of these purely data-driven approaches and achieve more reliable solutions, one promising possibility lies in combining data-driven methods with traditional physics-based approaches. This combined method, exemplified by hybrid methods and physics-informed neural networks (PINNs), integrates the interpretability and reliability of physics-based models with the adaptability and efficiency of data-driven techniques. These physics-enhanced or physics-informed methods are often referred to as *gray-box* approaches, as they can produce accurate and interpretable results by embedding physical laws into their formulation. Such integration opens new avenues for addressing computational engineering problems with accurate and efficient predictions, scalability, and robustness. By leveraging both data and domain knowledge (underlying physics), these hybrid methods overcome the limitations in both approaches, effectively bridging the gap between traditional techniques and purely data-driven methods.

To address these challenges, this thesis focuses on three interconnected directions: (i) the development and application of machine learning and deep learning techniques to model rough surface contact mechanics with improved efficiency; (ii) the development of PINNs for solving solid and contact mechanics problems under small strain assumptions, along with element-based energy PINNs for tackling large deformation scenarios; and (iii) the integration of sensor data into PINN-based frameworks to enable hybrid digital twin models for structural systems.

1.2. Modeling approaches for computational engineering problems

1.2.1. Physics-based modeling

Indeed, traditional physics-based computational techniques, such as the finite element method (FEM) [199] and boundary element method (BEM) [32], have served as a robust foundation for decades. Particularly for contact mechanics, these methods have proven to capture complex interactions between surfaces under various loading conditions. They enable precise computation of stress distributions, deformation, and contact areas, offering a detailed understanding of the underlying physics. However, they are computationally intensive, often requiring significant expertise to implement effectively. Additionally, they face challenges when incorporating additional data and inverse problems.

As this thesis does not aim to contribute directly to physics-based modeling techniques, we will not delve into their details here. For comprehensive discussions, we refer readers to foundational works such as [32, 126, 127, 129, 178, 199].

1.2.2. Purely data-driven modeling

In recent years, the rapid advancements in data-driven modeling techniques have opened up significant opportunities for solving complex problems across various fields. These approaches rely on experimental or synthetically generated data, deliberately omitting explicit physical knowledge of the system. Instead, they uncover underlying patterns within the data by utilizing machine learning and deep learning algorithms. Numerous applications of data-driven modeling tailored specifically for science and engineering are extensively documented in the literature

[27, 87, 109, 160]. These methods aim to deliver high accuracy while minimizing computational costs and prediction times.

Particularly in the field of beam mechanics, numerous studies have focused on the development and application of data-driven approaches for structural analysis and prediction. In the work by Beljkaš et al. [14], the authors employed artificial neural networks to predict the deflection of continuous beams with glass fiber-reinforced polymer (GFRP) reinforcement, achieving a good accuracy. Their study demonstrated the reliability of machine learning models in deflection estimation by training on an experimental database. Similarly, Preetham et al. [132] utilized machine learning techniques to predict midspan deflection. They implemented two machine learning approaches, specifically Support Vector Machines (SVM) and Artificial Neural Networks (ANN), for this purpose. Their study considered eight input parameters: beam length, beam breadth, beam depth, concrete compressive strength, load magnitude, tension steel area, characteristic steel strength, and shear span. The findings indicated that the predictions obtained using SVM were more accurate and aligned more closely with experimental results compared to those generated by ANN. Kaczmarek et al. [85] proposed an ANN-based approach for predicting reinforced concrete beam deflections, addressing the limitations of traditional methods like Eurocode 2. Their model, trained on experimental data, showed high accuracy, with applied bending moment and Young's modulus of concrete being the most influential parameters. Compared to standard calculations, the ANN predictions aligned more closely with experimental results, emphasizing its potential as a reliable structural analysis tool. On the other hand, Nie et al. [113] explored a deep learning approach to predict stress fields in 2D linear elastic cantilevered structures subjected to external static loads. They develop two Convolutional Neural Networks (CNNs) architectures: SCSNet, a single-channel input network, and StressNet, a multi-channel input network that incorporates structure geometry, external loads, and boundary conditions. Their findings indicate that StressNet achieves a mean relative error of 2.04% during testing, suggesting that deep learning models can serve as effective alternatives to traditional methods in structural design and topology optimization.

In the field of solid mechanics, Rocha et al. [141] introduced a method that combines reduced-order models with deep neural networks to select optimal boundary conditions for multiscale simulations in solid mechanics. This approach addresses the challenge of determining appropriate boundary conditions in homogenization problems, which is crucial for accurately capturing the effective properties of materials with heterogeneous microstructures. In the study by Hürkamp et al. [77], the authors proposed a cyber-physical production system framework that integrates FEM simulations with machine learning to create a digital twin for the manufacturing process of overmolded thermoplastic composites. They conducted parametric FEM studies to generate training data for various machine learning models, developing a surrogate model capable of predicting interface bond strength based on process parameters. The findings indicate that even minor deviations in process settings, such as temperature variations, can significantly impact structural integrity. In the study conducted by Maria Antony et al. [104], Linel2D-Net is introduced as a deep learning-based framework for solving 2D linear elastic boundary value problems on image-based domains. The model leverages CNNs to approximate displacement fields directly from input geometries, bypassing the need for FEM. By training on a dataset of precomputed solutions, the network learns to generalize across different material distributions and boundary conditions, enabling rapid inference. The results demonstrate that Linel2D-Net achieves high accuracy while significantly reducing computational costs compared to conventional

numerical solvers. In the research carried out by Heidenreich et al. [74], a transfer learning approach is proposed for Recurrent Neural Network (RNN)-based plasticity models to enhance their generalization across different material behaviors. The study focuses on leveraging pre-trained RNN models to accelerate learning and improve predictive accuracy for unseen loading conditions. By fine-tuning the models with limited new data, the approach reduces the need for extensive retraining while maintaining high fidelity in capturing material responses. The results indicate that transfer learning effectively enhances model adaptability, making it a promising technique for data-driven constitutive modeling in computational mechanics. Additional studies are comprehensively documented in the reviews by Yang [185] and Jin [83]

When it comes to contact mechanics, the development of data-driven methods has been significantly less explored, with a noticeable decline in the number of studies compared to solid and beam mechanics. Wu et al. [180] introduced a deep learning framework based on ANNs for predicting 3D contact force chains in granular materials. Their model leverages spatial features of particle interactions to accurately estimate force distributions under varying load conditions. The results demonstrate that the proposed approach outperforms traditional contact mechanics models in both computational efficiency and predictive accuracy. In terms of contact search, Oishi et al. [114] proposed a surface-to-surface contact detection method enhanced by deep learning to improve efficiency and accuracy. Their approach utilizes a neural network to predict potential contact regions, reducing the computational cost associated with traditional search algorithms. The results indicate that the deep learning-enhanced method significantly accelerates contact detection while maintaining high precision in complex geometries.

1.2.3. Hybrid modeling

As previously discussed, a promising approach to mitigating the black-box nature of purely data-driven methods and enhancing solution reliability is the combination of data-driven techniques with traditional physics-based models, commonly referred to as hybrid modeling.

The Fig. 1.1 presents an overview of hybrid modeling, which combines physics-based approaches with machine learning techniques. This integration leverages the strengths of both methodologies, addressing the limitations of purely data-driven or solely physics-based approaches.

One key aspect depicted in the figure is machine learning-assisted simulation, where data-driven techniques are incorporated into traditional physics-based simulations to accelerate computations and improve efficiency. This includes methods such as model order reduction, surrogate modeling, and neural network-based solvers, which approximate complex physical relationships while significantly reducing the computational cost of numerical solvers without compromising accuracy.

Another component is physics-assisted machine learning, where machine learning models are augmented with physical constraints, governing equations, or domain-specific knowledge. This category includes physics-informed neural networks, physics-guided feature engineering, and training data augmentation, all of which contribute to improved model generalization, reduced data dependency, and physically consistent predictions. These approaches make physics-assisted machine learning particularly useful for scientific and engineering applications requiring high fidelity and interpretability.

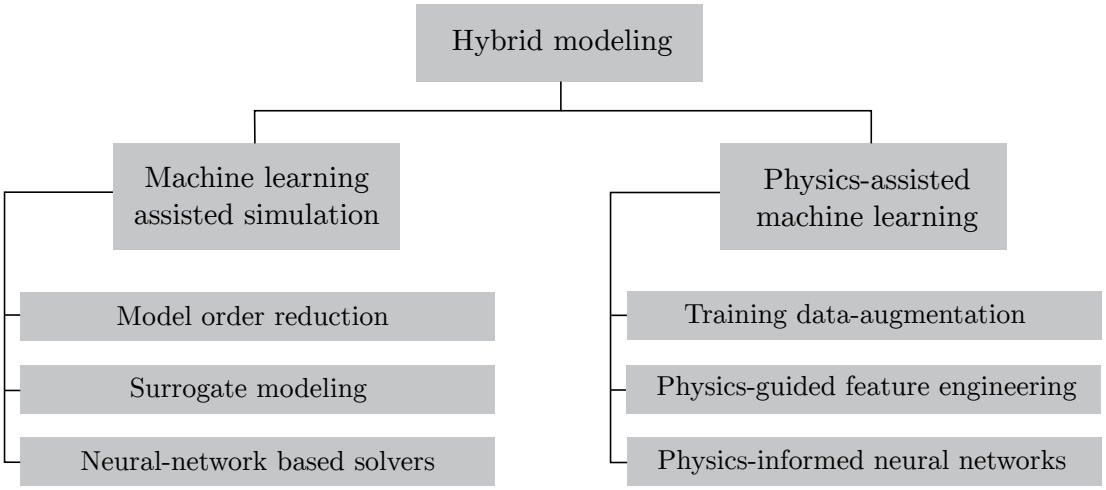


Figure 1.1.: An overview of hybrid modeling, illustrating the integration of physics-assisted machine learning and machine learning-assisted simulation, along with key subfields such as physics-informed neural networks, surrogate modeling, and model order reduction.

Physics-informed neural networks

Machine learning approaches usually require a large amount of simulation or experimental data, which might be challenging to acquire due to the complexity of simulations and the cost of experiments. Also, data scarcity can cause data-driven techniques to perform poorly in terms of accuracy. This is particularly true when using real-world observations that are noisy or datasets that are incorrectly labeled, as there is no physics-based feedback mechanism to validate the predictions. To tackle this problem, physics-informed neural networks (PINNs) have been developed. PINNs integrate boundary or initial boundary value problems and measurement data into the neural network's loss function to compensate for the lack of sufficient data and the black-box behavior of purely data-driven techniques [134]. In terms of forward problems, PINNs can serve as a partial differential equation (PDE) solver even in cases where domains are irregular. This is because PINNs utilize automatic differentiation and therefore do not require any connectivity of the sampling points, making them a mesh-free method [4]. Moreover, PINNs can break the curse of dimensionality when approximating functions in higher dimensions [68, 121]. Additionally, PINNs are a good candidate for addressing inverse problems due to the easy integration of measurement data [46].

PINNs have been widely applied across various fields of engineering and science to leverage their ability to incorporate physical laws into data-driven models. Their applications extend to geosciences [158], fluid mechanics [49, 135], optics and electromagnetics [15, 41, 88], as well as industrial sectors such as fatigue prognosis for wind turbine main bearings [191]. PINNs have also been integrated into hybrid digital twins for civil engineering structures, utilizing sensor data to improve structural monitoring and assessment [45], and have been explored for safeguarding critical infrastructure [34].

In the domain of solid mechanics, PINNs have been developed to address a range of problems, including linear elasticity, elastodynamics, and elastoplasticity [69], along with inverse problems

for parameter identification [18]. Rao et al. [136] proposed a mixed-variable formulation of PINNs for solving elastodynamic problems, drawing inspiration from hybrid finite element analysis [197]. Their approach incorporates both displacement and stress components as neural network outputs to impose boundary conditions as hard constraints, achieved through additional parallel networks. This formulation is suggested to enhance both accuracy and training efficiency.

Furthermore, Samaniego et al. [148] leveraged energy-based methods to formulate PINNs for various computational mechanics problems, including elastodynamics, hyperelasticity, and phase-field modeling of fracture. In a related study, Bai et al. [7] extended this concept to nonlinear contact mechanics and proposed an energy-based PINN framework for solving frictionless contact problems under large deformation. However, their formulation does not employ proper integral-based discretizations; instead, it relies on node-based integration, which substantially increases computational cost and poses significant challenges for extending the approach to three-dimensional problems. In a related study, Lu et al. [102] developed a physics-informed neural network approach with hard constraints (hPINNs) for topology optimization. They improved loss function formulation by integrating the penalty method and the augmented Lagrangian method to enforce inequality constraints as hard constraints. Additionally, equality constraints were explicitly imposed using output transformations, a concept initially introduced by Lagaris et al. [94] for simple domains.

Haghighat et al. [70] explored the use of PINNs in solid mechanics for inverse problems and surrogate modeling, employing a mixed-variable formulation with parallel networks to address linear elasticity. Their methodology was later extended to nonlinear elastoplasticity problems, incorporating classical *Karush-Kuhn-Tucker* (KKT) constraints. These constraints were initially enforced as soft constraints using a sign function, despite its discontinuous gradients. Building upon this work, Haghighat et al. [69] further utilized PINNs for constitutive model characterization and discovery by calibrating models based on macroscopic mechanical testing. To overcome the limitations of the sign function, they adopted the Sigmoid function to enforce KKT constraints, which provides smooth gradients but requires tuning an additional hyperparameter.

1.3. Scope and contribution

This thesis aims to address the gap between traditional computational methods and modern data-driven techniques by focusing on the following contributions:

- **Development and application of machine learning and neural networks to rough surface contact mechanics:** This research develops a surrogate modeling framework to predict the effective contact area in rough surface contact problems by integrating machine learning with high-fidelity Boundary Element Method (BEM) simulations. The study constructs an extensive database of rough topographies and material parameters, optimizes machine learning models through hyperparameter tuning, and evaluates their generalization capabilities using unseen test data, demonstrating accurate and efficient predictions. See also Sahin et al. [143].

- **Physics-informed neural networks for beam mechanics:** This work explores the application of PINNs to solve both static and dynamic beam problems based on Euler-Bernoulli

beam theory. The effectiveness of the approach is demonstrated through numerical examples, including static models, dynamic beam responses, and inverse problems such as load identification. The results highlight the potential of PINNs in accurately solving beam mechanics problems while reducing the computational cost compared to conventional numerical solvers.

- **Physics-informed neural networks for small deformation problems:** This study, following the work of Sahin et al. [144], applies PINNs to solve forward and inverse problems in solid and contact mechanics under small deformation assumptions. It investigates both displacement-based and mixed-variable formulations, with output transformations used to impose boundary conditions as hard constraints. Time-dependent solid mechanics have also been explored. An extension to 3D contact problems is presented in Sahin et al. [146].

- **Physics-informed neural networks for large deformation problems:** To address nonlinear solid mechanics at large deformations, we introduce the element-based energy PINN (EBE-PINN) formulation. Inspired by the finite element method, this approach partitions the domain into elements and enforces the principle of minimum potential energy, thereby ensuring physical consistency while improving the scalability of integral evaluations. This approach tackles geometrically nonlinear problems, such as hyperelasticity and contact mechanics, within a PINN framework.

- **Towards hybrid digital twins via physics-informed modeling enhanced by sensor data:** In the final part of this thesis, the combination of sensor data with physics-based simulations is explored to build surrogate models for structural components, such as reinforced concrete beams, following the methodology of Sahin et al. [145]. The approach allows for the incorporation of real-world experimental data, leading to the development of hybrid models. Such integration is intended to improve the accuracy and robustness of digital twins for structural health monitoring.

1.4. Structure of the thesis

This thesis is organized into multiple chapters, each addressing different aspects of computational mechanics, machine learning, and their integration through hybrid modeling.

Chapter 1 introduces the research topic, outlining the motivation, modeling approaches for computational engineering problems, objectives, and the key contributions of the thesis. Chapter 2 provides a fundamental background on computational mechanics, covering essential concepts such as beam mechanics, solid mechanics, contact mechanics and rough surface contact problems. It includes discussions on governing equations, boundary conditions, and classical numerical approaches used in computational mechanics. Chapter 3 explores fundamental concepts in machine learning, defining key terminologies and different types of learning approaches, such as supervised, unsupervised, and reinforcement learning. It also delves into various regression techniques, including k-nearest neighbors, linear regression, support vector machines, kernel-based methods, e.g., kernel ridge, tree-based models, such as decision trees, and neural networks, which serve as foundational tools for hybrid modeling. The primary challenges in machine learning are discussed, along with an exploration of hyperparameter optimization techniques.

Furthermore, a brief case study is presented to demonstrate the application of convolutional neural networks in beam mechanics. Chapter 4 discusses the application of machine learning and neural networks in computational mechanics, particularly in developing data-driven surrogate models for rough surface contact problems. The methodology for constructing predictive models, database generation, hyperparameter tuning, and accuracy evaluation is also detailed. Chapter 5 focuses on physics-informed neural networks (PINNs), outlining their formulation and different approaches such as element-based PINN formulations and PINNs with output transformation. Algorithmic challenges, including gradient vanishing, loss balancing, and optimization issues, are also discussed. Chapter 6 applies PINNs to a variety of problems in computational mechanics, including beam mechanics, solid mechanics, and contact mechanics. Specific applications include solving elastodynamic and elastostatic equations in solid mechanics and enforcing Karush-Kuhn-Tucker conditions in contact problems. Additionally, element-based energy PINNs are employed to address large deformation problems involving geometric and contact nonlinearities. Chapter 7 introduces the concept of hybrid digital twins, emphasizing their role in combining physics-informed machine learning techniques with sensor data. A proof of concept is presented for a steel-reinforced concrete beam, illustrating the integration of experimental sensor measurements with surrogate models. In this context, PINNs are employed as surrogate models to represent the mechanical behavior of reinforced concrete structures. Finally, Chapter 8 summarizes the key findings of the research and discusses potential future directions for improving purely data-driven, physics-enhanced and hybrid modeling approaches in computational engineering.

2. Computational Mechanics: Initial and Boundary Value Problems

2.1. Beam mechanics: Euler-Bernoulli beam theory

2.1.1. Overview

A beam is a structural element having one dimension that is much larger than the other two and capable of resisting the loads applied laterally to the beam axis. Since further investigations are pursued for the Euler-Bernoulli beam theory, known as the thin beam theory, the main assumptions and the governing equations of the Euler-Bernoulli are given in more detail.

The main hypothesis of the Euler-Bernoulli beam theory is that plane cross-sections which are orthogonal to the beam axis (neutral axis) stay still plane and orthogonal to the beam axis after deformation. The overall assumptions of the Euler beam theory:

- Plane cross-sections remain plane and perpendicular to the neutral axis of the beam after deformation.
- The deformations are small, so the equilibrium is computed on the reference configuration (Small deformation theory).
- Behavior of the beam is linear elastic and isotropic.
- Shear strains are neglected.

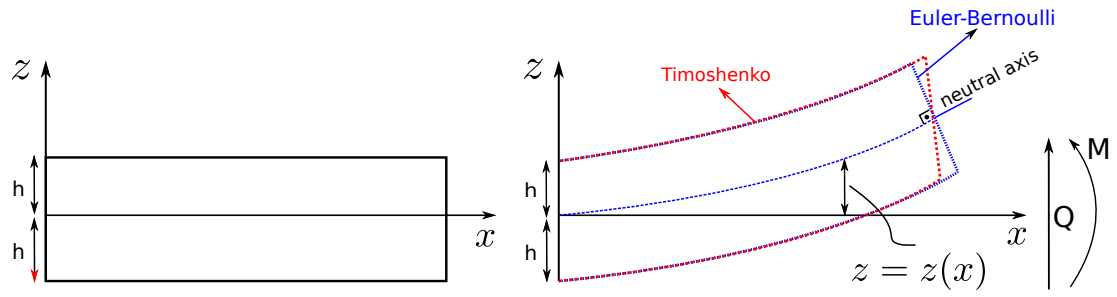


Figure 2.1.: Comparison between Euler-Bernoulli and Timoshenko beam theories.

As shown in Fig. 2.1, the Euler-Bernoulli beam theory assumes that plane cross-sections remain plane and perpendicular to the beam axis. In contrast, the Timoshenko beam theory allows

cross-sections to remain plane but not necessarily perpendicular to the axis, thereby accounting for shear deformation. Since shear effects are neglected in this work, the formulation is based on the Euler-Bernoulli (thin beam) theory.

2.1.2. Governing equations

2.1.2.1. Time-dependent beam equation

Consider the governing equation of an Euler-Bernoulli beam under the space-time dependent distributed load as (Fig. 2.2)

$$\rho A \frac{\partial^2 u(x,t)}{\partial t^2} + \frac{\partial^2}{\partial x^2}\left(EI \frac{\partial^2 u(x,t)}{\partial x^2}\right) = q(x,t), \quad x \in \Omega, t \in [0,T], \tag{2.1}$$

where $u(x,t)$ is the displacement field, Ω is a subset of $\mathbb{R}^D$, E is the uniform elastic modulus of the material, I is the moment of inertia of the cross-section of the beam, ρ is the material density, A is the cross-sectional area, and $q(x,t)$ is distributed space-time varying load. Eq. 2.1 is also known as the *Euler-Langrange* equation [172] used mainly in vibration problems. The first term describes the dissipated kinetic energy where ρA is the area density or the mass per unit length, and the second term characterizes the potential energy due to internal forces.

The concentrated load q can be considered as a special case of the distributed load as follows:

$$q(x,t) = P(t)\delta\left(x - x_0\right), \tag{2.2}$$

where δ is the *Dirac* delta function, and $P(t)$ is the time-dependent load applied at position x_0.

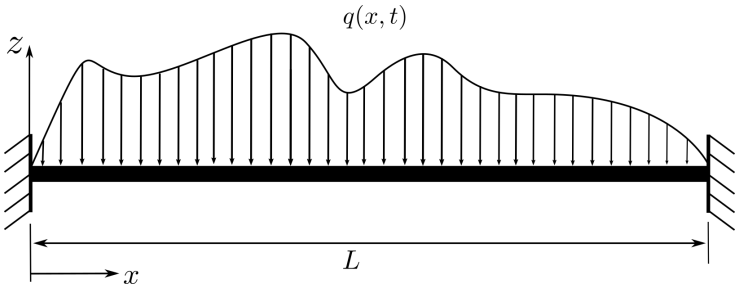


Figure 2.2.: A fixed-fixed supported beam under a space-time dependent distributed load.

2.1.2.2. Static beam equation

By neglecting the time-dependent terms in Eqs. 2.1 and 2.2, the governing equation is obtained for the static case can be written as

$$\frac{\partial^2}{\partial x^2}\left(EI \frac{\partial^2 u(x)}{\partial x^2}\right) = q(x), \quad x \in \Omega, \tag{2.3}$$

where $q(x)$ denotes the static load distribution. In the case of a concentrated load, $q(x)$ takes the form

$$q(x) = P\delta\left(x - x_0\right). \tag{2.4}$$

2.1.3. Boundary conditions and beam supports

To determine the unknown displacement field $u(x,t)$, the governing equation of the Euler-Bernoulli beam must be solved using together with the prescribed initial and boundary conditions. Since the governing equation is fourth order in space (see Eq. 2.1), four boundary conditions are required:

- $u(x,t) \rightarrow$: transverse displacement in the z-direction,
- $u_x(x,t) \rightarrow$: slope of the beam,
- $u_{xx}(x,t) \rightarrow$: bending moment,
- $u_{xxx}(x,t) \rightarrow$: shear force,

evaluated at $x \in \Omega$ and $t \in [0,T]$.

Similarly, two initial conditions are required, since Eq. 2.1 is second order in time:

- $u(x,0) \rightarrow$: displacement in the z-direction at $t = 0$,
- $u_t(x,0) \rightarrow$: time derivative of the displacement (velocity) at $t = 0$,

for all $x \in \Omega$. Note that initial conditions are not considered in the static beam formulation, as the time-dependent terms vanish.

Three primary support conditions are considered in this study:

- Cantilever beam,
- Simply supported beam,
- Fixed support (clamped beam).

2.1.3.1. Cantilever beam

A cantilever beam is fixed supported at one end and free at the other as depicted in Fig. 2.3. No displacement and rotation are allowed at the fixed edge. No bending moment and no shear force at the free edge are generated.

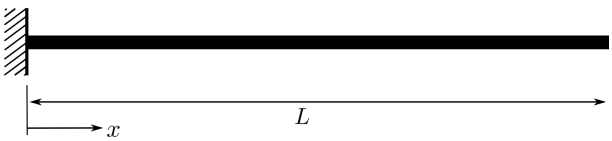


Figure 2.3.: An illustration of a fixed supported beam.

Boundary conditions of a cantilever beam are given as

$$\begin{aligned} &u(x,t) = 0 \quad and \quad u_x(x,t) = 0 \quad at\ x = 0, t = [0,T], \\ &u_{xx}(x,t) = 0 \quad and \quad u_{xxx}(x,t) = 0 \quad at\ x = L, t = [0,T]. \end{aligned} \tag{2.5}$$

2.1.3.2. Simply supported beam

A simply supported beam is pinned at both ends, as illustrated in Fig. 2.4. At the supports, transverse displacement is constrained, while rotation is permitted. Consequently, no bending moments develop at the pinned boundaries.

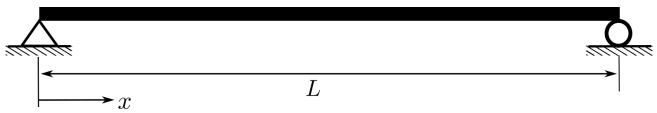


Figure 2.4.: An illustration of a simply supported beam.

Boundary conditions of a simply supported beam are given as

$$\begin{aligned} u(x,t) = 0 \quad and \quad u_{xx}(x,t) = 0 \quad at\ x = 0, t = [0,T], \\ u(x,t) = 0 \quad and \quad u_{xx}(x,t) = 0 \quad at\ x = L, t = [0,T]. \end{aligned} \tag{2.6}$$

2.1.3.3. Clamped beam

A clamped beam, also known as a fixed beam, is rigidly supported at both ends as depicted in Fig. 2.5. This implies that at both supports, the beam cannot move vertically or horizontally (no displacement) and cannot rotate (no angular movement). Such a support condition results in zero degrees of freedom at the ends, ensuring the highest level of rigidity and stability for the beam.

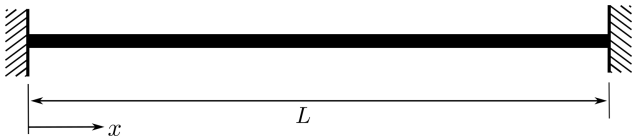


Figure 2.5.: An illustration of a clamped beam.

Boundary conditions of a clamped beam are given as

$$\begin{aligned} u(x,t) = 0 \quad and \quad u_{x}(x,t) = 0 \quad at\ x = 0, t = [0,T], \\ u(x,t) = 0 \quad and \quad u_{x}(x,t) = 0 \quad at\ x = L, t = [0,T]. \end{aligned} \tag{2.7}$$

2.2. Solid mechanics

2.2.1. Overview

Solid mechanics is the branch of continuum mechanics that investigates the behavior of solid materials under external effects, such as external forces or external displacements. It covers various types of materials and responses, from rigid bodies to deformable objects.

Linear elasticity is a fundamental subfield of solid mechanics that deals with structures that are inspected under small deformation theory. The physical interpretation is that the body returns to the undeformed or to the original shape when the external effect is removed. Since deformations are negligibly small in small deformation theory, displacement and strain fields are connected to each other linearly. In other words, the body deforms linearly under the applied load.

In contrast, finite strain theory is required when the assumptions of small deformation are no longer valid. This occurs when the displacements and rotations are sufficiently large such that geometric nonlinearities and nonlinear strain-displacement relations must be considered. In this case, the response of the material is no longer proportional to the applied load, and restoring the original configuration after unloading may not be guaranteed. Nonlinear models are therefore essential for accurately capturing the mechanical behavior of structures subjected to large deformations, such as soft materials, biological tissues, and structural components under extreme loading.

2.2.2. Infinitesimal strain theory: Linear elasticity

2.2.2.1. Governing equations

Let us consider the equations of initial boundary value problem (IBVP) under infinitesimal strain theory (small deformation assumption). The governing equations are given by

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \hat{\boldsymbol{b}} = \rho \ddot{\boldsymbol{u}} \qquad \text{in } \Omega \times [0, T], \qquad \rightarrow \text{(BE)} \tag{2.8}$$

$$\boldsymbol{u} = \hat{\boldsymbol{u}} \qquad \text{on } \partial\Omega_u \times [0, T], \qquad \rightarrow \text{(DBC)} \tag{2.9}$$

$$\boldsymbol{\sigma} \cdot \boldsymbol{n} = \hat{\boldsymbol{t}} \qquad \text{on } \partial\Omega_\sigma \times [0, T], \qquad \rightarrow \text{(NBC)} \tag{2.10}$$

where $\boldsymbol{\sigma}$ denotes the Cauchy stress tensor, $\boldsymbol{u}$ is the displacement vector representing the so-called primal variable, and $\ddot{\boldsymbol{u}} = \partial^2 \boldsymbol{u} / \partial t^2$ is the acceleration, which is defined as the second derivative of $\boldsymbol{u}$ with respect to time t. Additionally, $\hat{\boldsymbol{b}}$ denotes the body force vector, ρ is the material density, and $\boldsymbol{n}$ is the unit outward normal vector.

Prescribed displacements are represented by $\hat{\boldsymbol{u}}$ on Dirichlet boundary $\partial\Omega_u$, and $\hat{\boldsymbol{t}}$ denotes prescribed tractions on Neumann boundary $\partial\Omega_\sigma$. Abbreviations BE, DBC and NBC denote the balance equation, Dirichlet boundary condition and Neumann boundary condition, respectively. Given Neumann and Dirichlet boundary conditions are depicted in Fig. 2.6. Here, the deformation for a point $\boldsymbol{X}$ is defined from the reference configuration at time t_0 to the current configuration at time t.

Since the problem is time dependent, a proper set of initial conditions is expressed as

$$\boldsymbol{u}(\boldsymbol{X}, t = 0) = \hat{\boldsymbol{u}}_0(\boldsymbol{X}_0) \qquad \text{on } \Omega_0, \qquad \rightarrow \text{(IC)} \tag{2.11}$$

$$\dot{\boldsymbol{u}}(\boldsymbol{X}, t = 0) = \hat{\dot{\boldsymbol{u}}}_0(\boldsymbol{X}_0) \qquad \text{on } \Omega_0. \qquad \rightarrow \text{(IC)} \tag{2.12}$$

Here, the velocity is denoted as $\dot{\boldsymbol{u}} = \partial \boldsymbol{u} / \partial t$, which is defined as the first derivative of $\boldsymbol{u}$ with respect to time t. Initial conditions are applied to the whole domain only at the reference configuration Ω_0. Similarly, the abbreviation IC denotes the initial condition. Note that governing equations of linear elasticity are formulated using the current configuration.

The kinematic equation (KE) and constitutive equation (CE) for the deformable body are expressed as

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\boldsymbol{\nabla} \boldsymbol{u} + \boldsymbol{\nabla} \boldsymbol{u}^\top), \qquad \rightarrow \text{(KE)} \tag{2.13}$$

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon}. \qquad \rightarrow \text{(CE)} \tag{2.14}$$

Here, $\boldsymbol{\varepsilon}$ is the infinitesimal strain tensor and $\mathbb{C}$ is the fourth-order elasticity tensor. In the specific case of linear isotropic elasticity, the constitutive equation can be expressed via Hooke's law as

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}(\boldsymbol{\varepsilon})\boldsymbol{I} + 2\mu\boldsymbol{\varepsilon}, \tag{2.15}$$

where λ and μ are the Lamé parameters, $\operatorname{tr}(\cdot)$ is the *trace* operator to sum strain components on the main diagonal and $\boldsymbol{I}$ is the identity tensor.

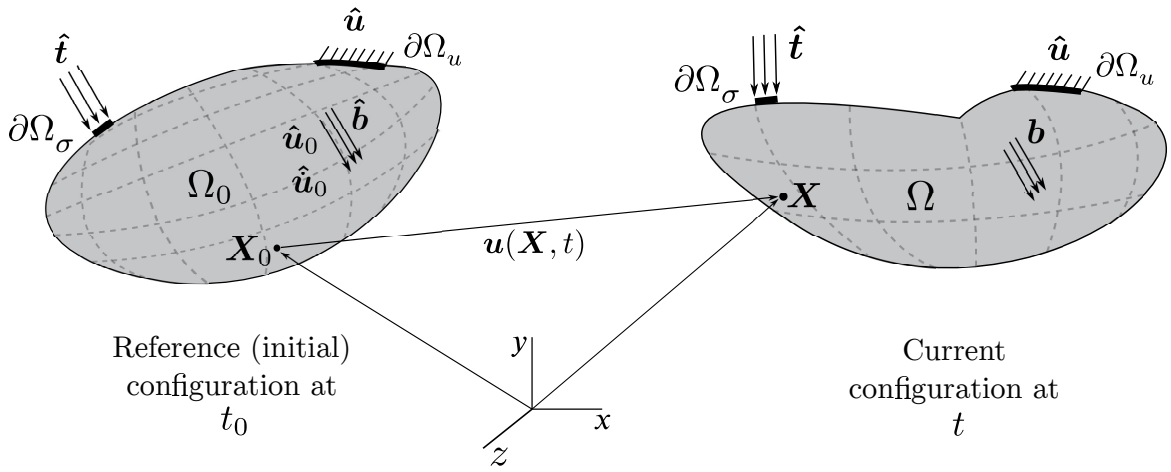


Figure 2.6.: Illustration of reference configuration and current configuration of deformed object. In the reference configuration, the elastic body is denoted by Ω_0, and in the deformed (current) configuration, it is represented by Ω_t.

3D formulation

Using the Einstein summation convention, we can write

- Equilibrium equation (Eq. 2.8):

$$\sigma_{ij,j} + b_i = \rho\partial_{tt}u_i \tag{2.16}$$

- Kinematic equation (Eq. 2.13):

$$\varepsilon_{ij} = \frac{1}{2}(u_{j,i} + u_{i,j}) \tag{2.17}$$

- Constitutive equation or Hooke's law (Eq. 2.14):

$$\sigma_{ij} = \lambda\delta_{ij}\varepsilon_{kk} + 2\mu\varepsilon_{ij}, \tag{2.18}$$

where $i, j = x, y, z$ represent the coordinates. The term δ_{ij} is the Kronecker delta, which is defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.19}$$

This function serves as a convenient way to select components of a tensor.

The equilibrium equations (BE) in 3D Cartesian system can be expressed as follows:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z} + \hat{b}_x = \rho \frac{\partial^2 u_x}{\partial t^2}, \tag{2.20}$$

$$\frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{yz}}{\partial z} + \hat{b}_y = \rho \frac{\partial^2 u_y}{\partial t^2}, \tag{2.21}$$

$$\frac{\partial \sigma_{zx}}{\partial x} + \frac{\partial \sigma_{zy}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + \hat{b}_z = \rho \frac{\partial^2 u_z}{\partial t^2}. \tag{2.22}$$

Similarly, kinematic equations can be written as

$$\varepsilon_{xx} = \frac{1}{2}\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_x}{\partial x}\right) = \frac{\partial u_x}{\partial x}, \tag{2.23}$$

$$\varepsilon_{yy} = \frac{1}{2}\left(\frac{\partial u_y}{\partial y} + \frac{\partial u_y}{\partial y}\right) = \frac{\partial u_y}{\partial y}, \tag{2.24}$$

$$\varepsilon_{zz} = \frac{1}{2}\left(\frac{\partial u_z}{\partial z} + \frac{\partial u_z}{\partial z}\right) = \frac{\partial u_z}{\partial z}, \tag{2.25}$$

$$\varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right), \tag{2.26}$$

$$\varepsilon_{yz} = \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right), \tag{2.27}$$

$$\varepsilon_{xz} = \frac{1}{2}\left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}\right). \tag{2.28}$$

Final set of equations, constitutive equations, can be written using Lamé constants can be summarized as

$$\sigma_{xx} = \lambda\left(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}\right) + 2\mu\varepsilon_{xx}, \tag{2.29}$$

$$\sigma_{yy} = \lambda\left(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}\right) + 2\mu\varepsilon_{yy}, \tag{2.30}$$

$$\sigma_{zz} = \lambda\left(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}\right) + 2\mu\varepsilon_{zz}, \tag{2.31}$$

$$\sigma_{xy} = 2\mu\varepsilon_{xy}, \tag{2.32}$$

$$\sigma_{xz} = 2\mu\varepsilon_{xz}, \tag{2.33}$$

$$\sigma_{yz} = 2\mu\varepsilon_{yz}. \tag{2.34}$$

These set of equations collectively describe the relationship between stresses, strains, and displacements in a three-dimensional elastic body. Note that the stress tensor has symmetry, meaning that $\sigma_{ij} = \sigma_{ji}$, such as $\sigma_{xy} = \sigma_{yx}$. This implies that the components of the stress tensor are equal when their indices are swapped.

2D formulation

Plane strain

Consider an elasticity problem as illustrated in Fig. 2.7. In this scenario, the length of the problem in the z-direction is significantly larger than in the x and y directions. Under an external

force in y-direction, we can assume that the strain in z-direction will be negligible compared to the strains in the other directions. Such problems are called plane strain in elasticity.

Most common application cases of plane strain are long tunnels and thick walls, where one dimension is significantly greater than the others. Assumption of negligible strain components in z-direction leads to

$$\varepsilon_{zz} = \varepsilon_{yz} = \varepsilon_{xz} = 0. \tag{2.35}$$

In a 2D Cartesian system, the equilibrium equations (Eq. 2.20-2.22) reduce to the following forms

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \hat{b}_x = \rho \frac{\partial^2 u_x}{\partial t^2}, \tag{2.36}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \hat{b}_x = \rho \frac{\partial^2 u_y}{\partial t^2}. \tag{2.37}$$

Similarly, the constitutive equations for plane strain, expressed in terms of Lamé constants, can be written as

$$\sigma_{xx} = (\lambda + 2\mu)\varepsilon_{xx} + \lambda \varepsilon_{yy}, \tag{2.38}$$

$$\sigma_{yy} = \lambda \varepsilon_{xx} + (\lambda + 2\mu)\varepsilon_{yy}, \tag{2.39}$$

$$\sigma_{xy} = 2\mu \varepsilon_{xy}. \tag{2.40}$$

Since the strains in the z-direction are assumed to be zero (as stated in Eq. 2.35), the out-of-plane stress component σ_{zz} can be determined using the following relationship

$$\sigma_{zz} = \nu(\sigma_{xx} + \sigma_{yy}), \tag{2.41}$$

where ν denotes the Poisson's ratio, which can be expressed using Lamé constants as $\nu = \frac{\lambda}{2(\lambda+\mu)}$.

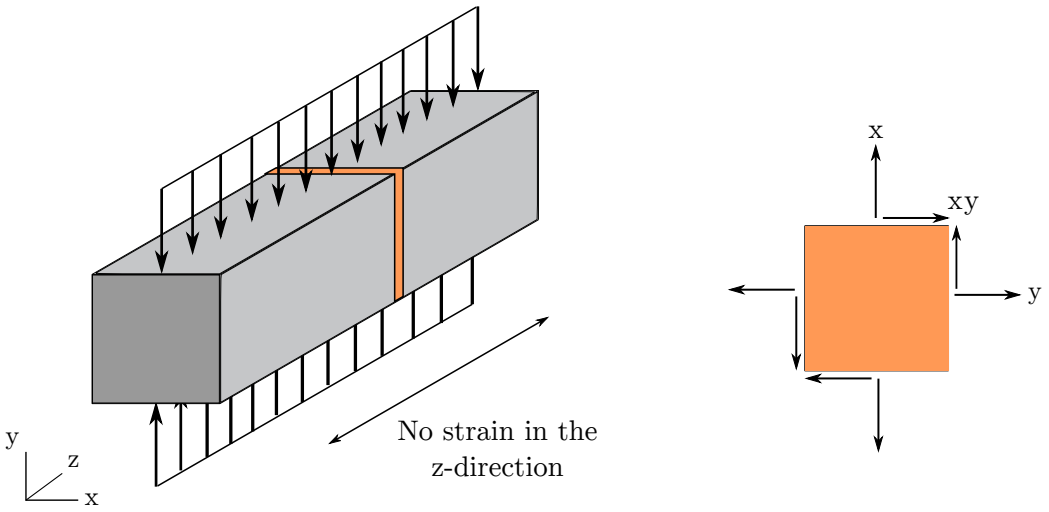


Figure 2.7.: Plane strain configuration.

Plane stress

In the plane stress configuration, no stress components in the out-of-plane direction are allowed, meaning that stress in the direction perpendicular to the plane is assumed to be zero. As shown in

Fig. 2.8, height in z-direction significantly smaller than other directions in this case. Most common application cases of plane stress are thin plates or shells. Assumption of stress components in z-direction are zero leads to

$$\sigma_{zz} = \sigma_{yz} = \sigma_{xz} = 0. \tag{2.42}$$

The equilibrium equations remain the same as plane strain. However, the constitutive equations for plane strain do not hold anymore. For simplification, we will define constitutive equations using Young's modulus (E) and Poisson's ratio (ν) expressed as follows

$$\sigma_{xx} = \frac{E}{1-\nu^2}(\varepsilon_{xx} + \nu\varepsilon_{yy}), \tag{2.43}$$

$$\sigma_{yy} = \frac{E}{1-\nu^2}(\nu\varepsilon_{xx} + \varepsilon_{yy}), \tag{2.44}$$

$$\sigma_{xy} = \frac{E}{1+\nu}\varepsilon_{xy}. \tag{2.45}$$

The stress components can be also computed in terms of Lamé constants λ and μ by substituting E and ν with the following expressions

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}, \tag{2.46}$$

$$\nu = \frac{\lambda}{2(\lambda + \mu)}. \tag{2.47}$$

Since the stresses in the z-direction are assumed to be zero (as stated in Eq. 2.42), the strain component in the z-direction ε_{zz} can be determined using the following relationship

$$\varepsilon_{zz} = -\frac{\nu}{E}(\sigma_{xx} + \sigma_{yy}). \tag{2.48}$$

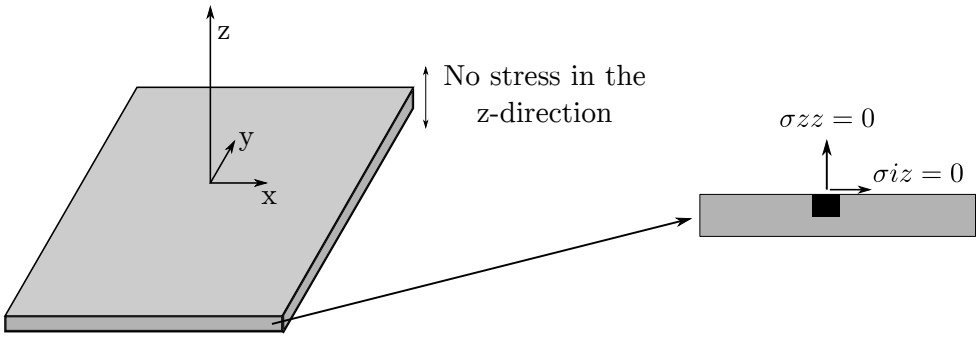


Figure 2.8.: Plane stress configuration.

Boundary and initial conditions

Boundary conditions

In the IBVP of solid mechanics, boundary conditions are essential to ensure that the problem is well-posed and has a unique solution. As mentioned earlier, there are two primary types of

boundary conditions commonly encountered: Dirichlet boundary conditions (DBC), also known as displacement boundary conditions, and Neumann boundary conditions (NBC), also known as traction or stress boundary conditions.

The Dirichlet boundary conditions prescribe the displacement at specific parts of the boundary, denoted as $\partial\Omega_u$. These conditions are imposed when the displacement field is constrained or controlled, for instance when a portion of the boundary is fixed or moved according to a known function. The general form of the DBC can be expressed as

$$\boldsymbol{u} = \hat{\boldsymbol{u}}(\boldsymbol{x}, t), \tag{2.49}$$

where $\hat{\boldsymbol{u}}$ is the prescribed displacements. Often, the DBCs are set to $\boldsymbol{u} = \boldsymbol{0}$ (homogeneous BC). However, they can be homogeneous or nonhomogeneous. The homogeneous DBCs are applied for the fixed portions of the boundary, while nonhomogeneous DBCs require non-zero values or prescribed functions. Nonhomogeneous DBCs could represent a boundary that is moving or deforming over time. In Cartesian coordinates, the displacement components can be expressed as

$$u_x = \hat{u}_x(\boldsymbol{x}, t), \tag{2.50}$$
$$u_y = \hat{u}_y(\boldsymbol{x}, t), \tag{2.51}$$
$$u_z = \hat{u}_z(\boldsymbol{x}, t). \tag{2.52}$$

where $\hat{u}_x, \hat{u}_y, \hat{u}_z$ are the components of the prescribed displacements in the x-, y-, z-directions.

Neumann boundary conditions specify the traction (or stress) on a part of the boundary denoted as $\partial\Omega_\sigma$ These boundary conditions define external forces or stresses applied to the surface of the body (in 3D) or to the edge of the domain (in 2D) such as pressure, a distributed load, or a line load. The general form of the NBC can be expressed as

$$\boldsymbol{\sigma} \cdot \boldsymbol{n} = \hat{\boldsymbol{t}}(\boldsymbol{x}, t), \tag{2.53}$$

where $\hat{\boldsymbol{t}}$ denotes the applied traction vector. On free surfaces, the traction is zero, meaning that $\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{0}$. Similarly, the NBCs can be expressed in Cartesian form as follows

$$\sigma_{xx} n_x + \sigma_{xy} n_y + \sigma_{xz} n_z = \hat{t}_x(\boldsymbol{x}, t), \tag{2.54}$$
$$\sigma_{yx} n_x + \sigma_{yy} n_y + \sigma_{yz} n_z = \hat{t}_y(\boldsymbol{x}, t), \tag{2.55}$$
$$\sigma_{zx} n_x + \sigma_{zy} n_y + \sigma_{zz} n_z = \hat{t}_z(\boldsymbol{x}, t). \tag{2.56}$$

where n_x, n_y, n_z are the components of the prescribed traction vectors in the x-, y-, z-directions.

On specific planes, such as boundaries align with coordinate axes, the NBCs simplify the following forms:

- A surface normal to the x-axis (at $x = x_0$) has $n_x = \pm 1$, $n_y = 0$ and $n_z = 0$, thereby, Eqs. 2.54-2.56 simplify to

$$\sigma_{xx} - \hat{t}_x(x = x_0, t) = 0, \tag{2.57}$$
$$\sigma_{yx} - \hat{t}_y(x = x_0, t) = 0, \tag{2.58}$$
$$\sigma_{zx} - \hat{t}_z(x = x_0, t) = 0. \tag{2.59}$$

- A surface normal to the y-axis (at $y = y_0$) has $n_x = 0$, $n_y = \pm 1$ and $n_z = 0$, thereby, Eqs. 2.54-2.56 simplify to

$$\sigma_{xy} - \hat{t}_x(y = y_0, t) = 0, \tag{2.60}$$
$$\sigma_{yy} - \hat{t}_y(y = y_0, t) = 0, \tag{2.61}$$
$$\sigma_{zy} - \hat{t}_z(y = y_0, t) = 0. \tag{2.62}$$

- A surface normal to the z-axis (at $z = z_0$) has $n_x = 0$, $n_y = 0$ and $n_z = \pm 1$, thereby, Eqs. 2.54-2.56 simplify to

$$\sigma_{xz} - \hat{t}_x(z = z_0, t) = 0, \tag{2.63}$$
$$\sigma_{yz} - \hat{t}_y(z = z_0, t) = 0, \tag{2.64}$$
$$\sigma_{zz} - \hat{t}_z(z = z_0, t) = 0. \tag{2.65}$$

Based on the problem definition, it might be convenient to express NBCs in terms of normal (n) and tangential directions (ξ and η). By using traction vector decomposition, traction vector can be expressed as

$$\hat{\boldsymbol{t}} = p_n \boldsymbol{n} + t_\tau^\xi \boldsymbol{\tau}^\xi + t_\tau^\eta \boldsymbol{\tau}^\eta, \tag{2.66}$$

where p_n denotes the normal pressure, t_τ^ξ is the tangential traction component in ξ-direction, and t_τ^η is the tangential traction component in η-direction. The direction vectors n, ξ, η are perpendicular to each other, meaning that $n \cdot \xi = 0$, $n \cdot \eta = 0$ and $\eta \cdot \xi = 0$. Therefore, each traction vector component can be obtained in terms of normal and tangential directions by multiplying the corresponding direction vector such as

$$\hat{\boldsymbol{t}} \cdot \boldsymbol{n} = (p_n \boldsymbol{n} + t_\tau^\xi \boldsymbol{\tau}^\xi + t_\tau^\eta \boldsymbol{\tau}^\eta) \cdot \boldsymbol{n} = p_n, \tag{2.67}$$
$$\hat{\boldsymbol{t}} \cdot \boldsymbol{\tau}^\xi = (p_n \boldsymbol{n} + t_\tau^\xi \boldsymbol{\tau}^\xi + t_\tau^\eta \boldsymbol{\tau}^\eta) \cdot \boldsymbol{\tau}^\xi = t_\tau^\xi, \tag{2.68}$$
$$\hat{\boldsymbol{t}} \cdot \boldsymbol{\tau}^\eta = (p_n \boldsymbol{n} + t_\tau^\xi \boldsymbol{\tau}^\xi + t_\tau^\eta \boldsymbol{\tau}^\eta) \cdot \boldsymbol{\tau}^\eta = t_\tau^\eta, \tag{2.69}$$

Components of these equations can be expressed as

$$\hat{t}_x n_x + \hat{t}_y n_y + \hat{t}_z n_z = p_n, \tag{2.70}$$
$$\hat{t}_x \xi_x + \hat{t}_y \xi_y + \hat{t}_z \xi_z = t_\tau^\xi, \tag{2.71}$$
$$\hat{t}_x \eta_x + \hat{t}_y \eta_y + \hat{t}_z \eta_z = t_\tau^\eta. \tag{2.72}$$

Indeed, equation sets (Eqs. 2.70-2.72) can be further simplified by inserting Eqs. 2.54-2.56. However, this would lead to long expressions.

Initial conditions

Initial conditions for the IBVP of solid mechanics describe the initial state of the solid at the start of the problem (at $t = 0$). These conditions are used in transient problems involving the time-dependent phenomena. Typically, the ICs consist of prescribed displacements and velocities.

The general form of the displacement IC is

$$\boldsymbol{u}(\boldsymbol{x}, t = 0) = \hat{\boldsymbol{u}}_0(\boldsymbol{x}, t = 0), \tag{2.73}$$

where $\hat{\boldsymbol{u}}_0$ prescribes the initial displacement at $t = 0$. The displacement IC is applied over the initial configuration Ω_0, representing the geometry of the body at $t = 0$. In Cartesian coordinates, components of displacement ICs can be expressed as follows

$$u_x(\boldsymbol{x}, t = 0) = \hat{u}_{0x}, \tag{2.74}$$
$$u_y(\boldsymbol{x}, t = 0) = \hat{u}_{0y}, \tag{2.75}$$
$$u_z(\boldsymbol{x}, t = 0) = \hat{u}_{0z}. \tag{2.76}$$

The velocity describes the rate of change of displacement field with respect to time and the velocity ICs can be expressed as follows

$$\dot{\boldsymbol{u}}(\boldsymbol{x}, t = 0) = \hat{\dot{\boldsymbol{u}}}_0, \tag{2.77}$$

where $\hat{\dot{\boldsymbol{u}}}_0$ denotes the prescribed initial velocity field at $t = 0$. Similarly, in Cartesian coordinates, components of velocity ICs can be written as follows

$$\dot{u}_x(\boldsymbol{x}, t = 0) = \hat{\dot{u}}_{0x}, \tag{2.78}$$
$$\dot{u}_y(\boldsymbol{x}, t = 0) = \hat{\dot{u}}_{0y}, \tag{2.79}$$
$$\dot{u}_y(\boldsymbol{x}, t = 0) = \hat{\dot{u}}_{0y}. \tag{2.80}$$

Indeed, initial conditions have physical interpretations. For instance, the initial displacement defines the configuration of the investigated at the beginning of the analysis at $t = 0$, while the initial velocity gives the direction and how fast the body is moving at $t = 0$.

Remark 2.2.1: Consistency of boundary conditions

Remark 1: In many solid mechanics problems, it is common to apply both Dirichlet and Neumann boundary conditions simultaneously, also known as mixed boundary conditions. However, these boundary conditions must be applied in a *mutual manner* for the same degree of freedom on a given boundary. However, it must be kept in mind that Dirichlet and Neumann boundary conditions cannot be imposed in the same direction on the same portion of the boundary. For instance, if the displacement in the x-direction on a particular boundary is prescribed as $u_x = 0$ (DBC), it causes to inconsistency to also apply a traction or stress boundary condition in the x-direction (NBC) on that same boundary. This will cause an over-constrained problem, delivering an ill-posed system.
Remark 2: In time-dependent (transient) problems, both Dirichlet BCs and displacement ICs include the displacement field, and they must be applied in a consistent manner to ensure that the solution is physically and mathematically sound. Any displacement value imposed by the Dirichlet boundary conditions at $t = 0$ should match with the initial displacement prescribed by the initial conditions. For instance, if the initial displacement is defined as $u_x(\boldsymbol{x}, t = 0) = \hat{u}_{0x}$ at time $t = 0$, the Dirichlet boundary condition should also reflect this same displacement value at the corresponding boundary.

2.2.2.2. Displacement-based formulation: Navier equations

In the displacement formulation, the goal is to eliminate strains and stresses from the equilibrium equation and seek a reduced set of equations involving only displacements as the primary variable. By substituting the constitutive equation (Eq. 2.14) into the equilibrium equation (Eq. 2.8), we can rewrite the equilibrium equation as

$$\boldsymbol{\nabla} \cdot (\mathbb{C} : \boldsymbol{\varepsilon}) + \hat{\boldsymbol{b}} = \rho \ddot{\boldsymbol{u}}. \tag{2.81}$$

Taking this a step further, we substitute the kinematic equation (Eq. 2.13) into Eq. 2.81 to derive a new form of the equilibrium equation that depends only on displacements such that

$$\boldsymbol{\nabla} \cdot \left(\mathbb{C} : \left(\frac{1}{2} \left(\boldsymbol{\nabla} \boldsymbol{u} + \boldsymbol{\nabla} \boldsymbol{u}^{\top} \right) \right) \right) + \hat{\boldsymbol{b}} = \rho \ddot{\boldsymbol{u}}. \quad \rightarrow \text{(NE)} \tag{2.82}$$

For linear isotropic elasticity, where the stress-strain relationship is given by $\boldsymbol{\sigma} = \lambda \ \mathrm{tr}(\boldsymbol{\varepsilon}) \boldsymbol{I} + 2\mu \boldsymbol{\varepsilon}$, the equilibrium equation simplifies further to

$$(\lambda + \mu) \boldsymbol{\nabla} \cdot (\boldsymbol{\nabla} \boldsymbol{u}) + \mu \boldsymbol{\nabla}^2 \boldsymbol{u} + \hat{\boldsymbol{b}} = \rho \ddot{\boldsymbol{u}}. \quad \rightarrow \text{(NE)} \tag{2.83}$$

This final equilibrium formulation is known as the Navier equations (NE). Based on the displacement formulation, we now construct a Tonti's diagram [165] to solve linear elasticity problems using displacements as the primary variable, as shown in Fig. 2.9. In this method, secondary (slave) variables are intermediate quantities derived from the primary variables, specifically $\boldsymbol{\sigma}^u$, $\boldsymbol{\varepsilon}^u$, $\dot{\boldsymbol{u}}$, and $\ddot{\boldsymbol{u}}$.

Similarly, the NE (Eq. 2.83) can be written in 3D Cartesian coordinate system as

$$(\lambda + 2\mu) \frac{\partial^2 u_x}{\partial x^2} + \mu \left(\frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \right) + (\lambda + \mu) \left(\frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial x \partial z} \right) + \hat{b}_x = \rho \frac{\partial^2 u_x}{\partial t^2}, \tag{2.84}$$

$$(\lambda + 2\mu) \frac{\partial^2 u_y}{\partial y^2} + \mu \left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial z^2} \right) + (\lambda + \mu) \left(\frac{\partial^2 u_x}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial y \partial z} \right) + \hat{b}_y = \rho \frac{\partial^2 u_y}{\partial t^2}, \tag{2.85}$$

$$(\lambda + 2\mu) \frac{\partial^2 u_z}{\partial z^2} + \mu \left(\frac{\partial^2 u_z}{\partial x^2} + \frac{\partial^2 u_z}{\partial y^2} \right) + (\lambda + \mu) \left(\frac{\partial^2 u_x}{\partial x \partial z} + \frac{\partial^2 u_y}{\partial y \partial z} \right) + \hat{b}_z = \rho \frac{\partial^2 u_z}{\partial t^2}. \tag{2.86}$$

As depicted in Fig. 2.9, the Tonti diagram provides a concise overview of the governing equations of linear elasticity, specifically for the displacement-based formulation. This formulation incorporates Eqs. 2.83, 2.13, and 2.14.

The primary variable $\boldsymbol{u}$ is used to fulfill the Dirichlet boundary conditions (DBCs) and the initial conditions (ICs) using the prescribed quantities: initial displacements $\hat{\boldsymbol{u}}_0$ and prescribed displacements on Dirichlet boundaries, $\hat{\boldsymbol{u}}$. Additionally, the first-order time derivative of $\boldsymbol{u}$ is applied to meet the ICs associated with the prescribed velocity field $\hat{\dot{\boldsymbol{u}}}$.

Since the stress field is not considered as a primary variable in this approach, it is derived from $\boldsymbol{u}$ using the kinematic equation (KE) and constitutive equation (CE). Neumann boundary conditions (NBCs) are then enforced based on the derived stress field and the prescribed traction field $\hat{\boldsymbol{t}}$.

Finally, the Navier equation (NE) is fulfilled through the computed acceleration field $\ddot{\boldsymbol{u}}$ (a secondary variable) and along with the prescribed body force $\hat{\boldsymbol{b}}$.

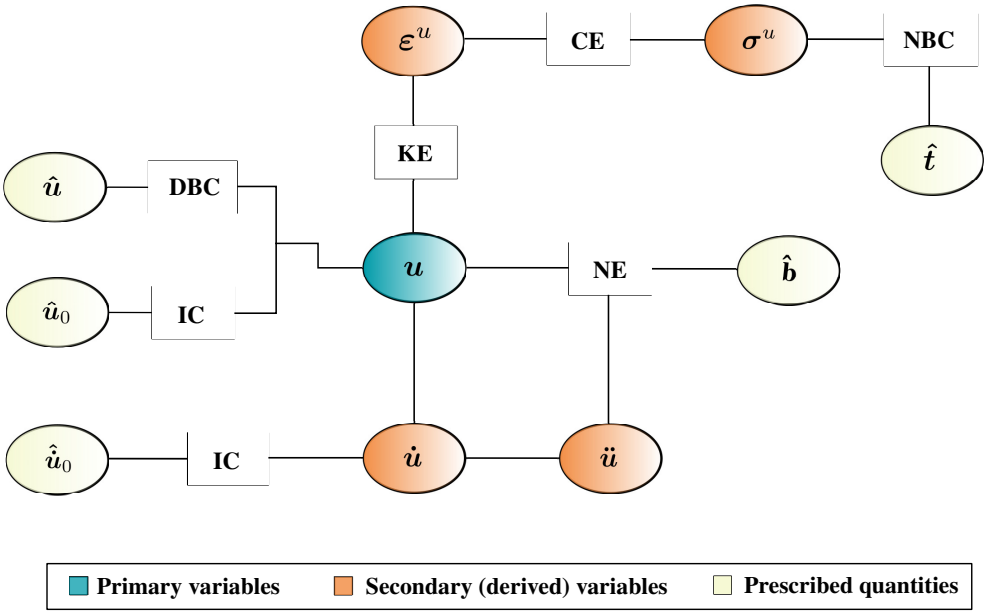


Figure 2.9.: Tonti's diagram for the displacement-based formulation of linear elasticity problems.

2.2.2.3. Mixed-variable formulation: the Hellinger-Reissner principle

One way to solve the governing equations of linear elasticity is based on a mixed-variable formulation in which stresses, displacements and strains can be approximated independently. Thus, mixed-variable formulations allow stress, displacement and strain fields to be primary (master) variables [197]. Inspired by the Hellinger-Reissner (HR) principle [139, 197], we construct a Tonti's diagram to solve linear elasticity problems based on two primary variables: displacements $\boldsymbol{u}$ and stresses $\boldsymbol{\sigma}$ (see Fig. 2.10). The secondary (slave) variables are intermediate and they are derived from the primary variables, i.e., $\boldsymbol{\sigma}^u$, $\boldsymbol{\varepsilon}^u$, $\dot{\boldsymbol{u}}$ and $\ddot{\boldsymbol{u}}$.

As shown in Fig. 2.10, the Tonti diagram summarizes the governing equations of linear elasticity problems (see Eqs. 2.8-2.14). The stress-to-stress coupling (SS) between the primary variable $\boldsymbol{\sigma}$ and the secondary variable $\boldsymbol{\sigma}^u$, defined as $\boldsymbol{\sigma} = \boldsymbol{\sigma}^u$, ensures that the two master fields remain consistent using the constitutive equation such that

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon}^{\boldsymbol{u}}. \tag{2.87}$$

2.2.2.4. Weak formulation

The weak (or variational) formulation of the governing equations is obtained by projecting the strong form of the equilibrium equations onto a space of admissible test functions. This approach facilitates the use of numerical methods by relaxing the differentiability requirements on the solution and transforming the differential equations into integral form.

Virtual displacement (virtual work) formulation

The virtual displacement formulation is derived based on the balance equation (Eq. 2.8) and the Neumann boundary condition (Eq. 2.10). Let $\delta\hat{\boldsymbol{u}}$ be a test function fulfilling homogeneous

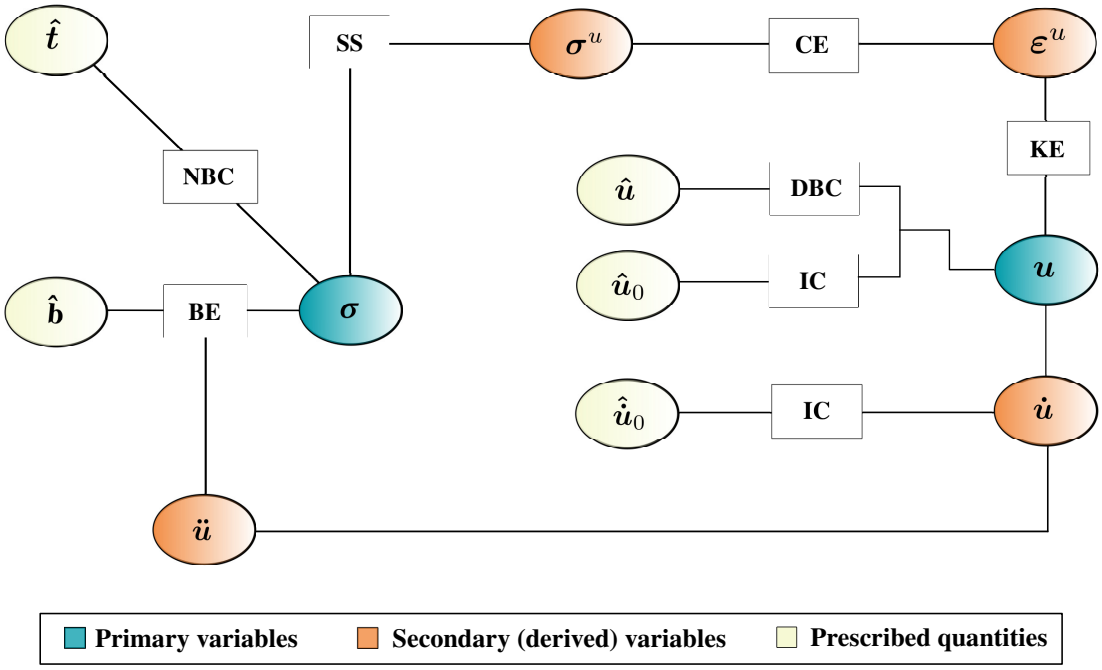


Figure 2.10.: Tonti's diagram based on Hellinger-Reissner principle for linear elasticity problems with small strain theory.

Dirichlet boundary conditions such that $\delta\hat{\boldsymbol{u}} = \mathbf{0}$ on $\partial\Omega_u$. Multiplying these equations by $\delta\hat{\boldsymbol{u}}$ and integration over the volume and on the Neumann boundary, residuum $\mathcal{R}$ is defined as

$$\mathcal{R} := \int_\Omega (\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \hat{\boldsymbol{b}})\cdot\delta\hat{\boldsymbol{u}}\, dV + \int_{\partial\Omega_\sigma} (\hat{\boldsymbol{t}} - \boldsymbol{\sigma}\cdot\boldsymbol{n})\cdot\delta\hat{\boldsymbol{u}}\, dA = 0. \tag{2.88}$$

For further simplification, let us consider the term $\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}}$. By application of the product rule for divergence, the corresponding integral term can be written as

$$\int_\Omega \boldsymbol{\nabla}\cdot(\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}})\, dV = \int_\Omega \boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}}\, dV + \int_\Omega \boldsymbol{\nabla}\delta\hat{\boldsymbol{u}} : \boldsymbol{\sigma}\, dV. \tag{2.89}$$

By applying the Divergence theorem to the term $\boldsymbol{\nabla}\cdot(\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}})$, which reduces the volume integral of the divergence over the region to the surface integral of a vector field over a closed surface, we obtain

$$\int_\Omega \boldsymbol{\nabla}\cdot(\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}})\, dV = \int_{\partial\Omega_u} \delta\hat{\boldsymbol{u}}\cdot\boldsymbol{\sigma}\cdot\boldsymbol{n}\, dA + \int_{\partial\Omega_\sigma} \delta\hat{\boldsymbol{u}}\cdot\boldsymbol{\sigma}\cdot\boldsymbol{n}\, dA. \tag{2.90}$$

Since $\delta\hat{\boldsymbol{u}} = \mathbf{0}$ on $\partial\Omega_u$, the second term vanishes. Inserting Eq. 2.90 into Eq. 2.89 leads to

$$\int_{\partial\Omega_\sigma} \delta\hat{\boldsymbol{u}}\cdot\boldsymbol{\sigma}\cdot\boldsymbol{n}\, dA = \int_\Omega \boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\cdot\delta\hat{\boldsymbol{u}}\, dV + \int_\Omega \boldsymbol{\nabla}\delta\hat{\boldsymbol{u}} : \boldsymbol{\sigma}.\, dV \tag{2.91}$$

Likewise, by merging equations 2.91 and 2.88, we derive the following expression

$$\mathcal{R} = \int_{\partial\Omega_\sigma} \delta\hat{\boldsymbol{u}}\cdot\boldsymbol{\sigma}\cdot\boldsymbol{n}\, dA - \int_\Omega \boldsymbol{\nabla}\delta\hat{\boldsymbol{u}} : \boldsymbol{\sigma}\, dV + \int_\Omega \hat{\boldsymbol{b}}\cdot\delta\hat{\boldsymbol{u}}\, dV + \int_{\partial\Omega_\sigma} (\hat{\boldsymbol{t}} - \boldsymbol{\sigma}\cdot\boldsymbol{n})\cdot\delta\hat{\boldsymbol{u}}\, dA, \tag{2.92}$$

and the cancellation of Neumann terms leads to the finalized weak form defined as

$$\mathcal{R} = -\int_\Omega \boldsymbol{\nabla}\delta\hat{\boldsymbol{u}} : \boldsymbol{\sigma}\, dV + \int_\Omega \hat{\boldsymbol{b}} \cdot \delta\hat{\boldsymbol{u}}\, dV + \int_{\partial\Omega_\sigma} \hat{\boldsymbol{t}} \cdot \delta\hat{\boldsymbol{u}}\, dA. \tag{2.93}$$

2.2.3. Finite strain theory: Hyperelasticity

2.2.3.1. Governing equations

Let us consider the equations of initial boundary value problem (IBVP) in finite strain theory as

$$\begin{aligned} \boldsymbol{\nabla}\cdot\boldsymbol{P} + \hat{\boldsymbol{b}}_0 &= \rho\ddot{\boldsymbol{u}} \qquad \text{in } \Omega_0 \times [0,T], \\ \boldsymbol{u} &= \hat{\boldsymbol{u}} \qquad \text{on } \partial\Omega_u \times [0,T], \\ \boldsymbol{P}\cdot\boldsymbol{N} &= \hat{\boldsymbol{t}}_0 \qquad \text{on } \partial\Omega_{\sigma,0} \times [0,T], \end{aligned} \tag{2.94}$$

where $\boldsymbol{P}$ denotes the first Piola-Kirchhoff stress tensor, $\hat{\boldsymbol{b}}_0$ is the body force per unit undeformed volume, $\boldsymbol{N}$ is the unit outward normal to the boundary in the reference configuration, and $\hat{\boldsymbol{t}}_0$ represents the tractions on the undeformed domain.

The motion of the body is given by the mapping between the position vector in the reference system ($\boldsymbol{X}$) and the position vector in the current configuration ($\boldsymbol{x}$) such as

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{x} = \boldsymbol{X} + \boldsymbol{u}(\boldsymbol{X}, t). \tag{2.95}$$

Consequently, the deformation gradient can be defined as

$$\boldsymbol{F} = \nabla_X \boldsymbol{\varphi}(\boldsymbol{X}) = \boldsymbol{I} + \nabla_X \boldsymbol{u}(\boldsymbol{X}, t). \tag{2.96}$$

In the hyperelastic formulation, the material behavior is derived from a strain energy density function $\Psi(\boldsymbol{F})$, which determines the constitutive relation as

$$\boldsymbol{P} = \frac{\partial\Psi}{\partial\boldsymbol{F}}. \tag{2.97}$$

A commonly used model is the compressible Neo-Hookean material, where the strain energy density [19] is expressed as

$$\Psi(\boldsymbol{F}) = \frac{\mu}{2}(\mathrm{tr}(\boldsymbol{C}) - 3) - \mu\log J + \frac{\lambda}{2}(\log J)^2, \tag{2.98}$$

where μ and λ are the Lamé parameters, $J = \det\boldsymbol{F}$ is the determinant of the deformation gradient, and $\boldsymbol{C} = \boldsymbol{F}^\top\boldsymbol{F}$ is the right Cauchy-Green deformation tensor.

3D formulation

For 3D problems, the deformation gradient (Eq. 2.96) can be written in terms of the displacement components as

$$\boldsymbol{F} = \begin{bmatrix} F_{xx} & F_{xy} & F_{xz} \\ F_{yx} & F_{yy} & F_{yz} \\ F_{zx} & F_{zy} & F_{zz} \end{bmatrix} = \begin{bmatrix} 1+\dfrac{\partial u_x}{\partial X} & \dfrac{\partial u_x}{\partial Y} & \dfrac{\partial u_x}{\partial Z} \\ \dfrac{\partial u_y}{\partial X} & 1+\dfrac{\partial u_y}{\partial Y} & \dfrac{\partial u_y}{\partial Z} \\ \dfrac{\partial u_z}{\partial X} & \dfrac{\partial u_z}{\partial Y} & 1+\dfrac{\partial u_z}{\partial Z} \end{bmatrix}, \tag{2.99}$$

where X, Y, Z are the reference coordinates.

Similarly, the right Cauchy-Green deformation tensor $\boldsymbol{C} = \boldsymbol{F}^\top \boldsymbol{F}$ can be expressed as

$$\begin{aligned}\boldsymbol{C} &= \begin{bmatrix} C_{xx} & C_{xy} & C_{xz} \\ C_{yx} & C_{yy} & C_{yz} \\ C_{zx} & C_{zy} & C_{zz} \end{bmatrix} \\ &= \begin{bmatrix} F_{xx}^2 + F_{yx}^2 + F_{zx}^2 & F_{xx}F_{xy} + F_{yx}F_{yy} + F_{zx}F_{zy} & F_{xx}F_{xz} + F_{yx}F_{yz} + F_{zx}F_{zz} \\ F_{xy}F_{xx} + F_{yy}F_{yx} + F_{zy}F_{zx} & F_{xy}^2 + F_{yy}^2 + F_{zy}^2 & F_{xy}F_{xz} + F_{yy}F_{yz} + F_{zy}F_{zz} \\ F_{xz}F_{xx} + F_{yz}F_{yx} + F_{zz}F_{zx} & F_{xz}F_{xy} + F_{yz}F_{yy} + F_{zz}F_{zy} & F_{xz}^2 + F_{yz}^2 + F_{zz}^2 \end{bmatrix}.\end{aligned}$$

The strain energy density function as provided in Eq. 2.98 requires the trace of $\boldsymbol{C}$ which can be obtained as

$$\begin{aligned}\mathrm{tr}(\boldsymbol{C}) &= C_{xx} + C_{yy} + C_{zz} \\ &= F_{xx}^2 + F_{yx}^2 + F_{zx}^2 + F_{xy}^2 + F_{yy}^2 + F_{zy}^2 + F_{xz}^2 + F_{yz}^2 + F_{zz}^2.\end{aligned} \tag{2.100}$$

By inserting Eq. 2.100 into Eq. 2.98, we can easily calculate the strain energy density function. Since we will employ the minimum potential energy principle (see Section 2.2.3.3) in the following sections, the components of the strong form of the large deformation theory (Eq. (2.94)) will not be explicitly expressed.

2D plane strain formulation

For 2D problems under plane strain assumptions, the deformation gradient (Eq. 2.96) can be written in terms of the in-plane displacement components as

$$\boldsymbol{F} = \begin{bmatrix} F_{xx} & F_{xy} \\ F_{yx} & F_{yy} \end{bmatrix} = \begin{bmatrix} 1 + \dfrac{\partial u_x}{\partial X} & \dfrac{\partial u_x}{\partial Y} \\ \dfrac{\partial u_y}{\partial X} & 1 + \dfrac{\partial u_y}{\partial Y} \end{bmatrix}, \tag{2.101}$$

The right Cauchy-Green deformation tensor $\boldsymbol{C} = \boldsymbol{F}^\top \boldsymbol{F}$ is given by:

$$\boldsymbol{C} = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{yx} & C_{yy} \end{bmatrix} = \begin{bmatrix} F_{xx}^2 + F_{yx}^2 & F_{xx}F_{xy} + F_{yx}F_{yy} \\ F_{xy}F_{xx} + F_{yy}F_{yx} & F_{xy}^2 + F_{yy}^2 \end{bmatrix}.$$

The strain energy density function in Eq. 2.98 requires the trace of $\boldsymbol{C}$, which becomes:

$$\begin{aligned}\mathrm{tr}(\boldsymbol{C}) &= C_{xx} + C_{yy} \\ &= F_{xx}^2 + F_{yx}^2 + F_{xy}^2 + F_{yy}^2.\end{aligned} \tag{2.102}$$

By inserting Eq. 2.102 into Eq. 2.98, the strain energy density function can be directly evaluated. Since the problem is now restricted to 2D, the expression for the strain energy density becomes

$$\Psi(\boldsymbol{F}) = \frac{\mu}{2}\left(\mathrm{tr}(\boldsymbol{C}) - 2\right) - \mu \log J + \frac{\lambda}{2}\left(\log J\right)^2, \tag{2.103}$$

where the subtraction of 2 (instead of 3 as in the 3D case) accounts for the reduced dimensionality under the plane strain assumption. For plane stress conditions, it is quite challenging to obtain the strain energy density function.

While both plane strain and plane stress formulations are used in 2D formulation, the plane strain model is generally more straightforward to implement for hyperelastic materials. This is because it is essentially a 3D formulation with the strain in the out-of-plane direction constrained to zero. In practice, this condition can be enforced by simply setting $F_{zz} = 1$ in the deformation gradient.

In contrast, the plane stress model requires that the out-of-plane stress vanishes (i.e., $\sigma_{zz} = 0$), which is a more complex condition to impose in nonlinear elasticity. As a result, the plane stress formulation for hyperelasticity involves additional algebraic manipulation or iterative numerical procedures, making it significantly more involved than the plane strain case. Therefore, in this work, hyperelasticity for 2D problems is considered under the plane strain assumption.

2.2.3.2. Stress measures

In finite strain hyperelasticity, different stress measures are used depending on whether quantities are expressed in the reference or current configuration. These stress tensors are all derived from the strain energy density function Ψ, which depends on the deformation gradient $\boldsymbol{F}$ or the right Cauchy-Green tensor $\boldsymbol{C}$.

Second Piola-Kirchhoff stress tensor: This stress measure is symmetric and defined in the reference configuration. It is energetically conjugate to the Green-Lagrange strain tensor and is given by

$$\boldsymbol{S} = 2\frac{\partial \Psi}{\partial \boldsymbol{C}}. \tag{2.104}$$

For the compressible Neo-Hookean material, applying the chain rule to $\Psi(\boldsymbol{C})$ leads to

$$\boldsymbol{S} = \mu \boldsymbol{I} - \mu \boldsymbol{C}^{-1} + \lambda \log J \boldsymbol{C}^{-1}, \tag{2.105}$$

where $\boldsymbol{C}^{-1}$ is the inverse of the right Cauchy-Green tensor. Moreover, the Green-Lagrange strain tensor can be expressed as

$$\boldsymbol{E} = \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I}). \tag{2.106}$$

First Piola-Kirchhoff stress tensor: This tensor is a two-point stress measure, mapping vectors in the reference configuration to the current configuration. It is expressed as

$$\boldsymbol{P} = \boldsymbol{F}\ \boldsymbol{S}. \tag{2.107}$$

Unlike the second Piola-Kirchhoff tensor, $\boldsymbol{P}$ is generally not symmetric. It appears in the weak form of the balance of linear momentum and is directly used in finite element formulations based on the reference configuration.

Cauchy (true) stress tensor: The Cauchy stress tensor is defined in the current (deformed) configuration and represents the actual physical stress acting on the deformed body. It is obtained from the first Piola-Kirchhoff stress tensor as follows

$$\boldsymbol{\sigma} = \frac{1}{J}\boldsymbol{P}\ \boldsymbol{F}^{\top} = \frac{1}{J}\boldsymbol{F}\ \boldsymbol{S}\ \boldsymbol{F}^{\top}. \tag{2.108}$$

This transformation ensures that the Cauchy stress tensor is symmetric and correctly represents the force per unit area in the deformed configuration.

Remark 2.2.2: Compressibility vs. Incompressibility

The Neo-Hookean strain energy density function provided in Eq. (2.98) is valid for *compressible* materials, where the volume may change under deformation. However, many solid materials are nearly or fully incompressible.
For incompressible materials, the volume remains constant, which implies that the determinant of the deformation gradient satisfies

$$\det(\boldsymbol{F}) = J = 1 \quad \Rightarrow \quad \log J = 0.$$

Substituting this condition into the strain energy density yields

$$\Psi(\boldsymbol{F}) = \frac{\mu}{2}\left(\operatorname{tr}(\boldsymbol{C}) - 3\right).$$

Accordingly, the second Piola-Kirchhoff stress tensor becomes

$$\boldsymbol{S} = \mu \boldsymbol{I} - \mu \boldsymbol{C}^{-1}.$$

2.2.3.3. Minimum potential energy principle

The total energy of a hyperelastic body in finite strain theory consists of contributions from internal energy, external work (due to body forces and tractions), and kinetic energy (in dynamic cases).

The internal energy stored in the material due to deformation is given by the following relation

$$U = \int_{\Omega_0} \Psi(\boldsymbol{F})\, dV_0, \tag{2.109}$$

where the strain energy density function is integrated over the undeformed volume V_0.

The external potential energy, which accounts for the work done by prescribed tractions and body forces, is expressed as:

$$W_{\text{ext}} = \int_{\partial\Omega_{\sigma,0}} \hat{\boldsymbol{t}}_0 \cdot \boldsymbol{u}\, dA_0 + \int_{\Omega_0} \hat{\boldsymbol{b}}_0 \cdot \boldsymbol{u}\, dV_0. \tag{2.110}$$

Similarly, the applied traction $\hat{\boldsymbol{t}}_0$ is integrated over the undeformed boundary surface A_0 and the body force $\hat{\boldsymbol{b}}_0$ is integrated over the entire domain, i.e. V_0.

For dynamic problems, the kinetic energy of the system is expressed as

$$K = \int_{\Omega_0} \frac{1}{2}\rho \dot{\boldsymbol{u}} \cdot \dot{\boldsymbol{u}}\, dV_0. \tag{2.111}$$

The total potential energy $\Pi_{\text{S}}[\boldsymbol{u}]$ is then defined differently for static and dynamic settings. For static problems (no inertia), it is defined as

$$\Pi_{\text{S}}[\boldsymbol{u}] = U - W_{\text{ext}}. \tag{2.112}$$

The principle of minimum potential energy states that the equilibrium displacement field minimizes the total potential energy such as

$$\delta \Pi_{\mathrm{S}}[\boldsymbol{u}] = 0. \tag{2.113}$$

For dynamic problems, the governing equations of motion can be derived from Hamilton's principle of stationary action. The Lagrangian function of a dynamic system can be expressed as

$$\mathcal{L}[\boldsymbol{u}] = K - U + W_{\text{ext}}, \tag{2.114}$$

and the motion of the system satisfies:

$$\delta \int_0^T \mathcal{L}[\boldsymbol{u}] dt = 0, \tag{2.115}$$

where the time integral $[0, T]$. Note that in the following sections, we focus exclusively on static problems in the context of finite strain hyperelasticity.

2.3. Contact mechanics

2.3.1. Overview

Contact mechanics is a specialized subfield of solid mechanics that examines the interactions between surfaces in contact, focusing on their mechanical response under deformation. It plays a critical role across diverse fields such as tribology (e.g., analysis of wear and lubrication), biomechanics (e.g., joint articulation and dental contact mechanics), structural engineering (e.g., analysis of bearings, bolted joints, and beam-to-solid interactions), and various manufacturing processes (e.g., metal forming, machining, stamping, and welding).

In this thesis, we investigate macro-scale contact problems considering both small and large deformations, under the assumption of frictionless contact. This allows us to isolate normal contact behavior without complexities from tangential effects.

2.3.2. Frictionless contact under small deformations

2.3.2.1. Governing equations

Consider a three-dimensional contact problem between an elastic body and a stationary rigid obstacle, as depicted in Fig. 2.11. In the reference configuration, the elastic body is denoted by Ω_0, and in the deformed (current) configuration, it is represented by Ω_t. The rigid obstacle remains fixed in its configuration, labeled as Ω_r. Fig. 2.11c shows a scenario where the two bodies are in contact.

The surface of the elastic body is divided into three distinct regions: the Dirichlet boundary $\partial\Omega_u$, where the displacements are prescribed; the Neumann boundary $\partial\Omega_\sigma$, where external tractions are applied; and the potential contact boundary $\partial\Omega_c$, where contact conditions may be enforced. The actual contact surface forms a subset of $\partial\Omega_c$ and is determined as part of the solution process.

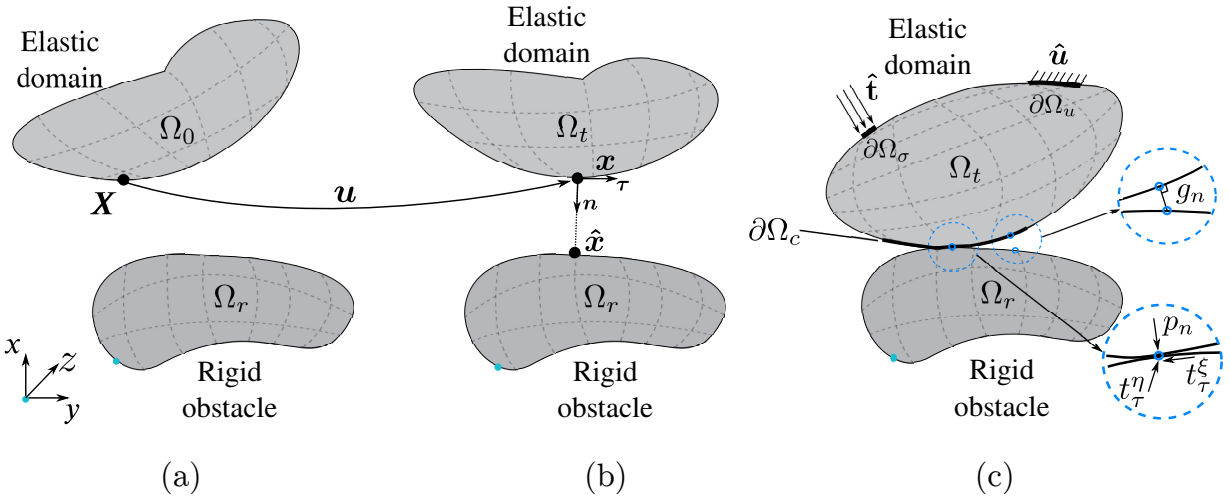


Figure 2.11.: Illustration of a contact problem between an elastic body and a rigid obstacle. (a) Reference configuration, (b) deformed (current) configuration, (c) boundary conditions, showing the gap g_n, tangential traction $t_{\boldsymbol{\tau}}$, and contact pressure p_n.

Recall the governing equations of linear elasticity for completeness as

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}+\hat{\boldsymbol{b}}=\rho\ddot{\boldsymbol{u}} \qquad \text{in } \Omega\times[0,T], \qquad \rightarrow \text{(BE)} \tag{2.116}$$

$$\boldsymbol{u}=\hat{\boldsymbol{u}} \qquad \text{on } \partial\Omega_u\times[0,T], \qquad \rightarrow \text{(DBC)} \tag{2.117}$$

$$\boldsymbol{\sigma}\cdot\boldsymbol{n}=\hat{\boldsymbol{t}} \qquad \text{on } \partial\Omega_\sigma\times[0,T]. \qquad \rightarrow \text{(NBC)} \tag{2.118}$$

The displacement vector $\boldsymbol{u}$ describes the motion of the elastic body from the reference configuration $\boldsymbol{X}$ to the current configuration $\boldsymbol{x}$ (see Fig. 2.11(a, b)), and is defined as follows

$$\boldsymbol{u}=\boldsymbol{x}-\boldsymbol{X}. \tag{2.119}$$

The gap function g_n measures the normal distance between the elastic and rigid bodies in the current configuration, and is given by

$$g_n=-\boldsymbol{n}\cdot(\boldsymbol{x}-\hat{\boldsymbol{x}}). \qquad \rightarrow \text{(GF)} \tag{2.120}$$

Here, $\hat{\boldsymbol{x}}$ represents the closest point projection of $\boldsymbol{x}$ onto the surface of the rigid body Ω_r (see Fig. 2.11b). The contact traction vector $\boldsymbol{t}_c$ can be expressed as

$$\boldsymbol{t}_c=p_n\boldsymbol{n}+t_\tau^\xi\boldsymbol{\tau}^\xi+t_\tau^\eta\boldsymbol{\tau}^\eta, \tag{2.121}$$

where

$$\boldsymbol{t}_c=\boldsymbol{\sigma}\cdot\boldsymbol{n} \qquad \text{on } \partial\Omega_c. \qquad \rightarrow \text{(CST)} \tag{2.122}$$

According to Cauchy's stress theorem (CST), the stress tensor $\boldsymbol{\sigma}$ maps the normal vector $\boldsymbol{n}$ to the traction vector $\boldsymbol{t}_c$. It is assumed that the boundary vectors $\boldsymbol{n}$, $\boldsymbol{\tau}^\eta$, and $\boldsymbol{\tau}^\xi$ can be computed in the reference configuration under the assumption of small deformations (linear elasticity) [152]. Since contact constraints are defined in the current configuration, the traction components p_n, t_τ^ξ, and t_τ^η are obtained by traction vector decomposition (TVD) of the contact traction vector $\boldsymbol{t}_c$ as

$$p_n=\boldsymbol{t}_c\cdot\boldsymbol{n}, \quad t_\tau^\xi=\boldsymbol{t}_c\cdot\boldsymbol{\tau}^\xi, \quad t_\tau^\eta=\boldsymbol{t}_c\cdot\boldsymbol{\tau}^\eta. \qquad \rightarrow \text{(TVD)} \tag{2.123}$$

Note that in 2D contact problems, only one tangential traction component is present, i.e. t_τ. For normal contact problems, the classical Karush-Kuhn-Tucker (KKT) conditions, also known as the Hertz-Signorini-Moreau (HSM) conditions, and the frictionless sliding conditions (FSC) are defined as

$$\left.\begin{aligned} g_n &\geqslant 0, && \text{(a)} \\ p_n &\leqslant 0, && \text{(b)} \\ p_n g_n &= 0, && \text{(c)} \end{aligned}\right\} \quad \rightarrow \text{(KKT)} \tag{2.124}$$

$$\left.\begin{aligned} t_\tau^\xi &= 0, && \text{(a)} \\ t_\tau^\eta &= 0. && \text{(b)} \end{aligned}\right\} \quad \rightarrow \text{(FSCs)} \tag{2.125}$$

The system of equations in (2.124) represents the classical Karush-Kuhn-Tucker (KKT) conditions. These conditions capture the essential aspects of a normal contact problem, ensuring non-penetration, physical admissibility of the contact pressure, and the complementarity between the gap and the contact pressure.

The first condition (2.124a) enforces the non-penetration constraint, which is a fundamental requirement in contact mechanics. It ensures that the elastic body does not interpenetrate the rigid obstacle. In the case of two bodies being in contact, the gap g_n becomes zero, indicating that the surfaces are in physical contact without any overlapping. When $g_n > 0$, the bodies are separated by a positive gap, and no contact force is exerted.

The second condition (2.124b) imposes that the contact pressure p_n is negative. This implies that the contact pressure acts only in compression and not in tension, meaning no adhesive or tensile forces are allowed in the contact zone. This condition is essential for ensuring a physically realistic response, as tensile stresses would indicate an unphysical adhesion between the contacting surfaces.

The third condition (2.124c) is the complementarity constraint, which states that the contact pressure and the gap cannot be simultaneously non-zero. This implies that if the bodies are in contact ($g_n = 0$), the contact pressure p_n must be non-zero (compressive). Conversely, if the bodies are not in contact ($g_n > 0$), the contact pressure must be zero. This complementarity condition distinguishes between active contact (when there is physical interaction between the surfaces) and inactive contact (when the surfaces are separated). It also highlights the nonlinear nature of contact problems, as the status of each point on the potential contact surface can change during the analysis.

The frictionless sliding conditions (FSCs) in (2.125) describe the behavior of the tangential traction components at the contact interface. These conditions, denoted as $t_\tau^\xi = 0$ and $t_\tau^\eta = 0$, indicate that the tangential components of the contact traction vector vanish. This reflects the assumption of a frictionless contact interface, where no resistance is expected against relative tangential movement between the surfaces.

Together, the KKT conditions and the frictionless sliding conditions form a complete set of constraints that govern the behavior of the contact interface in this problem. The KKT conditions handle the normal contact behavior, ensuring that the surfaces do not interpenetrate and that the contact pressure remains physically admissible. The FSCs address the tangential behavior, enforcing a zero tangential traction in the case of frictionless contact.

For further details on constitutive models for contact mechanics, including frictional interactions, see [126, 127, 129, 178].

2.3.2.2. Mixed-variable contact formulation

As discussed in Section 2.2.2.3, where the mixed-variable formulation for solving the equations of linear elasticity was introduced, we now extend this approach to address contact mechanics under small deformation, including frictionless sliding conditions. This extension incorporates the principles of contact mechanics into the framework of the mixed-variable formulation.

Fig. 2.12 illustrates Tonti's diagram based on the Hellinger-Reissner (HR) principle for contact problems formulated under linear elasticity with frictionless sliding conditions. The connections in the diagram associated with linear elasticity have already been detailed in Fig. 2.10. Thus, this discussion focuses on the elements and connections specific to contact mechanics, which are highlighted in red.

Inspired by the Hellinger-Reissner principle, we develop a Tonti's diagram tailored for contact problems. This diagram operates with two primary variables: the displacements $\boldsymbol{u}$ and the stresses $\boldsymbol{\sigma}$. Secondary, or slave, variables are derived from these primary variables and include $\boldsymbol{\sigma}^u$, $\boldsymbol{\varepsilon}^u$, $\dot{\boldsymbol{u}}$, $\ddot{\boldsymbol{u}}$, p_n, g_n, $\boldsymbol{t}_c$, t_τ^ξ, and t_τ^η. These secondary variables serve as intermediates in the solution process.

The gap function (GF), as defined in Eq. 2.120, computes the separation between the two bodies based on the displacement field. Similarly, the contact traction $\boldsymbol{t}_c$ is determined using Cauchy's stress theorem (CST) from the stress field, as expressed in Eq. 2.122. The contact traction is then decomposed into its normal component p_n and tangential components t_τ^ξ and t_τ^η.

- The computed gap g_n and the normal pressure p_n are used to fulfill the Karush-Kuhn-Tucker (KKT) conditions, as given in Eq. 2.124. These conditions ensure non-penetration, admissible contact pressure, and complementarity.

- The tangential traction components t_τ^ξ and t_τ^η are employed to enforce the frictionless sliding conditions, as specified in Eq. 2.125.

Together, these formulations provide a comprehensive framework for solving contact problems in small deformation elasticity with frictionless sliding conditions.

2.3.3. Frictionless contact under large deformations

Governing equations

In terms of large deformations, this thesis focuses on the frictionless contact problems using the minimal potential energy principle. For quasi-static contact problems, the total potential energy can be expressed as

$$\Pi = \Pi_{\mathrm{S}} + \Pi_{\mathrm{C}}, \tag{2.126}$$

where, Π_{S} denotes the finite strain potential energy defined in Eq. 2.112 and Π_{C} represents the contact potential energy. As the finite strain theory for the hyperelastic materials has been explained in Section 2.2.3, we will dive into the formulation of only the contact potential in this section. There are several ways to formulate the contact potential. In this work, we will only dive into penalty-based and augmented Lagrange-based contact formulations [124, 126].

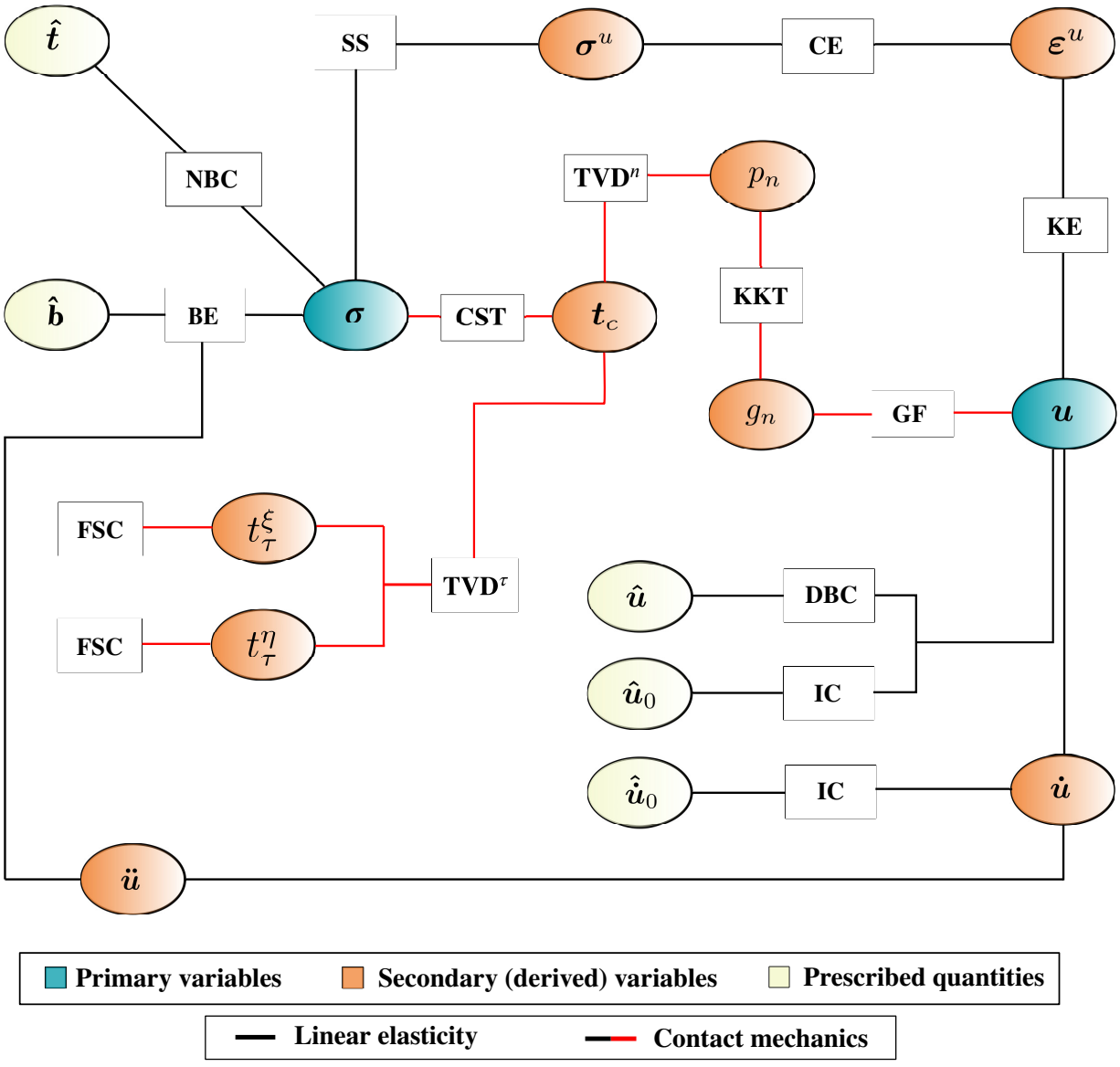


Figure 2.12.: Tonti's diagram of Hellinger-Reissner (HR) principle applied to linear elastic contact problems with frictionless sliding conditions.

Penalty-based contact potential

The penalty-based contact potential can be expressed as

$$\Pi_{\mathrm{C}} = \frac{1}{2} \int_{\partial\Omega_c} \epsilon \langle -\boldsymbol{g} \rangle^2 \, dA_0. \tag{2.127}$$

Here, ϵ is a user-defined and problem-specific penalty parameter. The term $\boldsymbol{g}$ denotes the relative displacement between two bodies, commonly referred to as the gap vector. Similar to the gap function defined in Eq. 2.120, the gap vector between an elastic body and a rigid obstacle, as illustrated in Fig. 2.11, is expressed as

$$\boldsymbol{g} = \boldsymbol{x} - \hat{\boldsymbol{x}}, \tag{2.128}$$

with

$$\boldsymbol{x} = \boldsymbol{X} + \boldsymbol{u}. \tag{2.129}$$

Here, $\hat{\boldsymbol{x}}$ is the closest point projection of $\boldsymbol{x}$ onto the surface of the rigid body Ω_r (see Fig. 2.11b), $\boldsymbol{u}$ is the displacement vector which describes the motion of the elastic body from the reference configuration $\boldsymbol{X}$ to the current configuration $\boldsymbol{x}$. Note that the gap vector formulation is also frequently applied in mesh tying problems.

Augmented Lagrange-based contact potential

The augmented Lagrange-based contact potential [124] can be expressed as

$$\Pi_{\mathrm{C}} = \frac{1}{2\epsilon} \int_{\partial\Omega_c} \langle -\boldsymbol{\lambda}^{(k)} - \epsilon \boldsymbol{g} \rangle^2 \, dA_0, \tag{2.130}$$

where $\boldsymbol{\lambda}^{(k)}$ is the Lagrange multiplier at iteration k, updated iteratively based on the Uzawa algorithm [95]. This approach augments the standard penalty method by adaptively updating the Lagrange multiplier, thereby improving the enforcement of the contact constraint while avoiding the need for excessively large penalty parameters.

The augmented Lagrange-based contact potential formulation is presented in Algorithm 1. The procedure begins with the initialization $\boldsymbol{\lambda}^{(0)} = 0$, which corresponds to the classical penalty-based contact formulation. During training, the Lagrange multiplier is updated periodically using an Uzawa-type scheme to improve constraint enforcement. Specifically, every i_{freq} iterations, the multiplier is updated according to $\boldsymbol{\lambda}^{(k+1)} = -\epsilon \langle -\boldsymbol{g}^{(k)} \rangle$, where $\langle \cdot \rangle$ denotes the Macaulay bracket, ensuring that only interpenetration is penalized. The update frequency is controlled by the user-defined parameter i_{freq}, which can, for instance, be associated with loading steps or displacement increments. After each multiplier update, the counter k is incremented to track the update history At every step, regardless of the update, the contact potential is computed as $\Pi_{\mathrm{C}} = \frac{1}{2\epsilon} \int_{\partial\Omega_c} \langle -\boldsymbol{\lambda}^{(k)} - \epsilon \boldsymbol{g} \rangle^2 \, dA_0$, which combines penalty and Lagrange terms into a single variationally consistent and differentiable energy density. This approach allows for progressive enforcement of the contact constraint while preserving smooth optimization behavior.

Algorithm 1 Augmented lagrange-based contact potential

Inputs:

ϵ: user-defined and problem-specific penalty parameter

n_{iter}: number of training iterations

i_{freq}: frequency to activate the Uzawa update

Output:

Π_{C}: Contact potential

procedure AUGMENTEDLAGRANGECONTACTPOTENTIAL($\epsilon, n_{\text{iter}}, i_{\text{freq}}$)

Set the counter $k = 0$ for update step

Initialize the lagrange multiplier $\boldsymbol{\lambda}^{(0)} = 0$

for each iteration $i = 1 : n_{\text{iter}}$ **do**

if $(i \bmod i_{\text{freq}}) = 0$ **then**

Update the lagrange multiplier

$$\boldsymbol{\lambda}^{(k+1)} = -\epsilon\langle -\boldsymbol{g}^{(k)} \rangle$$

$k = k + 1$

end if

Calculate the contact potential

$$\Pi_{\text{C}} = \frac{1}{2\epsilon} \int_{\partial\Omega_c} \langle -\boldsymbol{\lambda}^{(k)} - \epsilon \boldsymbol{g} \rangle^2 \, dA_0$$

end for

return Π_{C}

end procedure

2.4. Rough surface contact problems

At the micro scale, we shift our attention to rough surface contact problems, which are essential for understanding the behavior of real-world surfaces that are never perfectly smooth. These rough surface contact models serve as foundational examples for data-centric approaches in solving contact mechanics problems as we will explain in Chapter 4. Therefore, a detailed overview on rough surface contact mechanics is provided due to its importance in the development of data-centric methods.

2.4.1. Overview

The vast majority of real surfaces, either natural or artificial, only collide within a small set of touching points or touching zones when brought into contact under moderate pressure. This defines a so-called effective contact area much smaller than the nominal contact area observed at a scale comparable to the surface extension in space. The reason for this behavior is that surfaces are affected by finely spaced irregularities and deviations from the ideal shape that include scratches, pits, grooves and, in general, jaggedness resulting from external factors or machining processes, with characteristic dimensions spanning several different scales. These imperfections are referred to, in the literature, as *surface roughness*, for which a comprehensive theory relevant to surface metrology can be found in Whitehouse [173].

The correct assessment of the effective contact area is a problem of paramount importance in numerous engineering and physics applications, since this quantity governs many coupled multi-physics phenomena happening at the contact interface, either mechanical, thermal or electrical. The excellent review presented in the work of Vakis et al. [167] accurately describes this complex scenario, referred to by the authors as a true *"paradise of multiphysics"*.

As a direct consequence of the utmost importance of this topic, a vast ecosystem of solution methods has developed over the last decades, which could be broadly categorized into analytical and numerical approaches. The first modern quantitative approach belongs to the first class and dates back to the seminal work of Greenwood and Williamson [66]. They developed the first multi-asperity model for the analysis of the relationship between real contact area and normal reaction force in a rough contact, and paved the way for all the subsequent analytical models whose detailed descriptions would, unfortunately, fall outside the scope of the presented publication. The interested reader is referred to Barber [9, §6] for a thorough list and detailed breakdown of the most important analytical models that have been developed during the last eighty years. These models have all proven to be excellent tools to gain a deep and fundamental understanding of the problem under examination, and they are still used nowadays given their accuracy and speed of deployment. Nonetheless, they are characterized by some intrinsic limitations that make them unsuitable concerning various specific contexts. Analytical models are limited to derive results in a global sense, meaning that they can not deliver local information for specific points of the interface and, furthermore, have to rely on strong assumptions regarding surface geometry. To overcome their limitations, and thanks to the dramatic increase in terms of computational resources available, several numerical models have been developed parallel to analytical models for approximating solutions of rough contact problems.

Conventional models for rough contact analysis

As of today, three main approaches, two based on continuum mechanics and one on atomistic models, are commonly used as computational workhorses for the numerical treatment of rough contact problems. Within the first category fall the Boundary Element Method (BEM), was first used in a paper by Brebbia and Dominguez [31] and specifically elaborated for rough contact by Xu and Jackson [184], and the Finite Element Method (FEM), for which related comprehensive computational contact schemes have been proposed by Wriggers [179], Popp et al. [126], Wriggers and Popp [177] for example with suitable numerical algorithms and solvers contributed by Adams [2], Wiesner et al. [174, 175] or Mayr and Popp [107].

In BEM models, the rough contact problem is cast as the equivalence between the interface displacement field and a Boundary Integral Equation (BIE) coincident with the spatial convolution over the contact domain between a Green's function and an unknown contact traction field. In BEM, the problem complexity is reduced by one dimension, thus making it a very efficient tool, capable of delivering very accurate information on local displacements and contact tractions, see for example Pohrt and Li [122], Vollebregt [170], or Bemporad and Paggi [16] for a review of different possible implementations of the method. Since global quantities can be directly derived, it also proved itself a suitable tool for the comparison of the results delivered by different analytical models. Systematic research on this subject can be found in Paggi and Ciavarella [115], Carbone and Bottiglione [38], and Yastrebov et al. [189]. Despite this method being capable of analyzing any kind of surface without a priori assumptions on the asperities distribution, one drawback is that it is mostly limited to infinitesimal strain and linear material laws. Extensions of the method can be found to account for bulk plasticity, viscoelasticity or generic conditions for which a Green's function is not available in analytical form, see e.g. Jacq et al. [81], Putignano and Carbone [133], and Zhao et al. [194] for respective exemplary instances of these three cases. They unfortunately remain limited niche exceptions.

On the other hand, the FEM is much more versatile, since it can deal with the desired material law, finite deformations, finite size domains and complex boundary conditions. As a drawback, every part of the domain has to be discretized, so that the higher computational cost must be justified by specific requirements for the problem to be solved. The last twenty years have seen a continuous growth of the use of the FEM applied to the context under examination. In this regard, the first example can be traced back to Hyun et al. [79] for the adhesiveless frictionless contact between elastic bodies with rough surfaces, later extended by Pei et al. [119] to account for an elastoplastic response. The FEM has also been employed by Yastrebov et al. [190] for the solution of rough contact with finite plastic deformations and in Carvalho et al. [39] for the solution of Signorini problems involving deformable bodies in contact with geometrically complex rigid surfaces. Recently, a new method has been developed by Paggi and Reinoso [117] for a simplified yet accurate analysis of rough contact problems using FEM, in which the rough surface needs not to be explicitly modeled, but its shape is embedded in a layer of interface finite elements. The method, named *MPJR* (eMbedded Profile for Joint Roughness), was originally designed for the solution of two-dimensional adhesive rough contact problems, and has later been extended in Bonari et al. [23] to include friction and in Bonari et al. [22] to solve three-dimensional contact. The framework aims at bypassing some shortcomings that are faced when solving rough contact problems using FEM, namely the care required in the mesh generation process to avoid negative Jacobians and the difficulty related to the explicit discretization of the rough geometry.

Molecular Dynamics (MD) methods fall within the last category of numerical models. They can trace the kinematics of every single atom by computing its interaction with all particles in the domain, using a predefined potential. Since every atom has to be explicitly modeled, the dynamic interaction time scales are inherently low, and the method is therefore particularly suitable for the analysis of very limited space and time domains; cf. [167, §2.5]. To overcome this, the elastic potentials of a single layer of atoms can be normalized to incorporate the elastic interactions proper to a semi-infinite solid. This Green's Function Molecular Dynamics (GFMD) approach, developed by Campañá et al. [37], proved to deliver very accurate solutions, as demonstrated by the results of the recent contact mechanics challenge issued by Müser et al. [112].

To leverage the advantages of one method over its drawbacks, multi-scale approaches can be defined, in which different models are employed for the solution of different scales of the problem, with information exchange enforced across the scales. In Bonari et al. [21], FEM is used at the macro-scale, where a perfectly smooth interface is considered, while BEM is employed for the solution of a micro-scale rough contact problem. A bidirectional coupling between the two scales is enforced in terms of mean-plane separation.

2.4.2. Problem formulation

A rough surface contact problem typically refers to a class of engineering challenges that involves the interaction between two surfaces which are conformal at the macro-scale, but non-conformal at smaller scales, given the presence of micro-scale textures, defects, or, in general, asperities [84].

The study of these problems stemmed from the attempt to explain Coulomb's law of friction - specifically, the observation that the tangential force Q required for the relative sliding of two bodies in contact is proportional to the normal force P that pushes them together, and independent of the observable contact area [28]. However, at the microscopic level, real surfaces are rough and contact occurs only at discrete asperities, forming the effective contact area A_e which is typically much smaller than the nominal contact area. This effective contact area plays a crucial role in friction, wear, and sealing mechanisms, as it governs the actual load-bearing capacity and energy dissipation at the interface [168]. Therefore, its correct assessment is essential for the analysis of physical phenomena related to friction, wear, and sealing, i.e., the class of problems investigated in the field of tribology [168]. On top of that, thanks to electro- and thermo-mechanical analogies [8], the effective contact area also plays a crucial role in the study of electrical and thermal exchanges at the interface level.

Many analytical models have been developed for the identification of the effective contact area in this context, but they rely on strong assumptions for the statistical distribution of the elevation field describing the rough surface and, most importantly, they deliver a global response that can not describe the solution at every point of the contact domain. In contrast, numerical models can not only estimate the effective contact area but also determine the contact status of each interface point–that is, identify which points are in contact and which are not.

A common numerical analysis approach is based on the hypothesis of scale separation. Consider the two bodies B_i, $i \in \{1, 2\}$, represented in Fig. 2.13, brought into contact by the action of an external load, which can be an applied force or an imposed act of motion Δ, the latter going under the name of *far-field displacement*. We assume that the shapes of the contacting surfaces $\mathcal{S}_i$, which correspond to portions of the boundaries ∂B_i of the bodies, match or conform to each other, enabling the creation of a finite nominal contact area. In this case, a nominally smooth

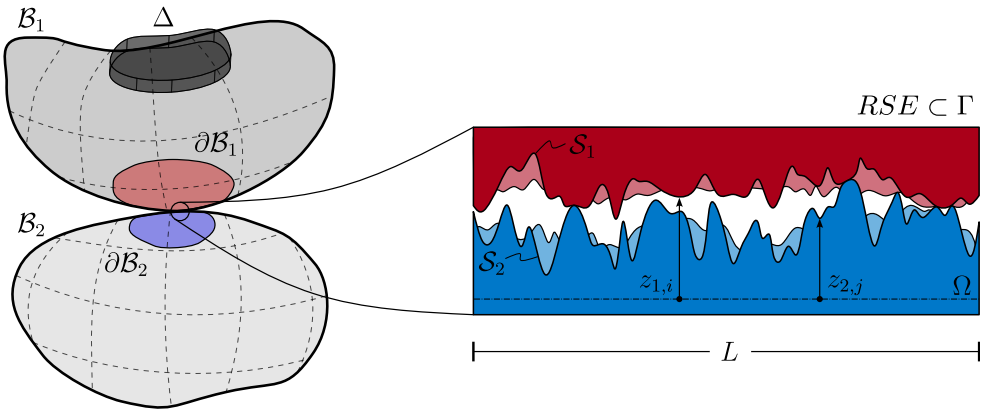


Figure 2.13.: Graphical representation of the problem under consideration. At a macroscopic scale, two deformable bodies come into contact under the action of an external imposed displacement Δ. At the microscopic scale, roughness features can be revealed, resulting in an effective contact area A_e much smaller than the nominal contact area that is assumed at the macroscopic scale.

contact interface Γ can be defined, encompassing $\mathcal{S}_i$ along with the underlying portions of the material bulk.

We now assume that the contacting surfaces are microscopically rough, and focus the attention on a subset of Γ small enough to make the roughness characteristics relevant. This subset is referred to as a Representative Surface Element (RSE). In most cases analyzed in the literature, the RSE is modelled as a square patch with size L, discretized on a Cartesian grid with n points per side and grid size $g_l = L/(n-1)$. On this grid, roughness is described by two discrete height fields $\boldsymbol{z}_i \in \mathbb{R}^{n\times n}$, i.e., a height value for both rough surfaces is assigned to every node of the grid, calculated from a reference datum plane Ω for each of the surfaces $\mathcal{S}_i$.

If the macroscopic, nominally smooth contact interface Γ is flat, Ω can be considered as part of $\mathcal{S}_i$. Otherwise, it can be regarded as the tangent plane at the point under consideration: given the hypothesis of scale separation, we assume that the local radius of curvature R satisfies $R >> L$, where R is the common local radius of curvature for both S_i.

If this approach is chosen, a variable for the number of points in contact, n_c, can be introduced and the effective contact area for the RSE expressed as:

$$A_e = 100\frac{n_c\, g_l^2}{L^2}\%. \tag{2.131}$$

If the bodies in contact are linear elastic, the solution of the contact problem in terms of A_e, or, equivalently, n_c, is only governed by the relative normal displacements $\boldsymbol{u}_i$ of opposite points belonging to the two sides of Γ.[8, §2.2] Thanks to this property, the problem can be simplified and re-formulated as a contact problem involving a rough rigid surface characterized by a *combined height field* $\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{z}_i) \in \mathbb{R}^{n\times n}$ and a semi-indefinite linear elastic half-space with *composite mechanical parameters* $E = E(E_i)$ and $\nu = \nu(\nu_i)$ cf. Fig. 2.14. However, given the assumptions of linearity, and since for small values of A_e a well acknowledged result in tribology is the proportionality between contact area and imposed load, it can be concluded that the actual pattern

of contact spots is dictated by the surfaces topography only, and not by value of the composite Young's modulus.

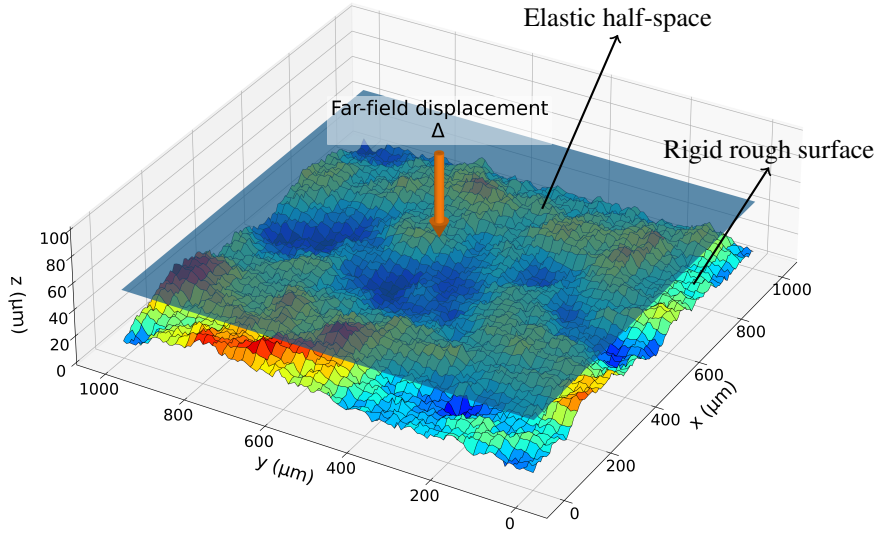


Figure 2.14.: Schematic depicting the standard rough surface contact problem. Since a composite formulation is applied, the rough surface is assumed to be rigid while the flat surface is linear elastic with composite material properties.

2.4.3. Random Midpoint Displacement algorithm

The Random Midpoint Displacement (RMD) algorithm is a well-established method for producing surfaces with realistic properties. Originally introduced by Fournier et al. [54], it is commonly used to generate self-affine fractal surfaces [120]. This algorithm is a recursive process that starts from the known heights of four points located at the corners of a square grid made of a single cell with lateral side L. The cell is then split in four, resulting in a grid with nine nodes and four smaller cells with side $L/2$. Then, a height is assigned to each of the five newly created nodes. The height of the four nodes on the edges is given by the mean value of the heights of the two adjacent corner nodes plus a random perturbation extracted from a Gaussian distribution with zero mean and standard deviation σ_0. The height of the central node is given by the mean value of the four corner nodes plus a random perturbation extracted from the same Gaussian distribution $\mathcal{N}(0, \sigma_0)$ used for the edge nodes. The process is repeated as many times as desired, and for every iteration the standard deviation associated with the Gaussian distribution is scaled by a factor 2^{-H}, where H is the generalized Hurst exponent. In this study, a range between 0.5 and 0.8 is chosen for H. This range includes typical values observed in many artificial surfaces [120] and in various natural phenomena that exhibit fractal self-affinity [10, 17, 103]. The key point of the algorithm is the scaling law of the standard deviation: it can be proven that for this value the height variation between grid points separated by a distance $1/2^n$ is proportional to $1/2^{nH}$, as

required by the definition of self-affine processes [10]. An illustrative example of how the RMD algorithm works is given in Fig. 2.15. In the current study, surfaces with both 128×128 and 256×256 elements per side have been investigated, thus employing seven and eight iterations of the RMD algorithm, respectively.

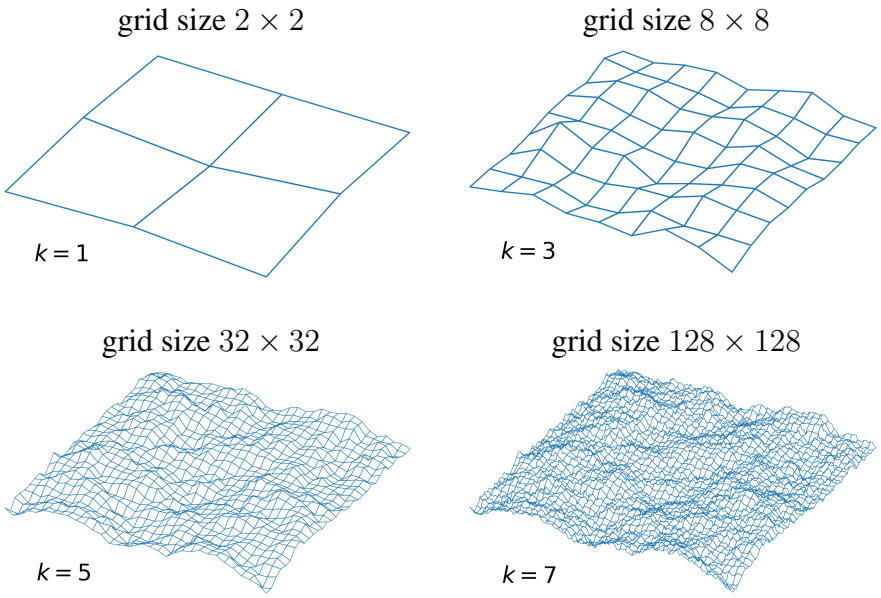


Figure 2.15.: An illustration of the *Random Midpoint Displacement* algorithm. A higher interaction count k results in a finer discretization of the generated self-affine fractal surface.

2.4.4. Statistical characterization of topographies

Statistical characterization of rough surfaces plays an important role in identifying hidden patterns [192]. This result has been achieved by applying the Random Process Theory (RPT), previously employed in the study of random noise signals, to the surfaces under examination. Functional relationships can be drawn between the statistical parameters of the surfaces and the features of the related contact problem. As an example, many analytical models predict that the ratio between effective contact area and applied load is inversely proportional to the mean square profile slope, which quantifies the average steepness of the surface asperities and is mathematically defined as the expectation of the squared derivative of the surface profile [116]. Furthermore, many legacy asperity-based normal contact models, most of them stemming from the pioneering works of Greenwood and Williamson [66], hinge on the distribution of the surface heights.

The statistical characterization can be performed by considering the surface as a 3D entity, sampling the height field and the summits (maxima in 3D) distributions, or by considering the surface as a global assembly of multiple 2D section cuts, or profiles, and then analyzing each profile's heights, slopes and peaks (maxima in 2D) distribution [25]. Peaks are measured on profiles generated by a profilometer which moves in both the x- and y-direction through the

scan length of the topography as depicted in Fig. 2.16a. A profile can be characterized by peak heights z_p, the maximum of peaks $z_\mathrm{p}^\mathrm{max}$ and the mean of peaks $\bar{z}_\mathrm{p}$, see Fig. 2.16b. The statistical parameters employed in this study are listed in Table 4.1. These parameters will later serve as the foundation for developing data-driven methods, as introduced in Section 4.3.3.

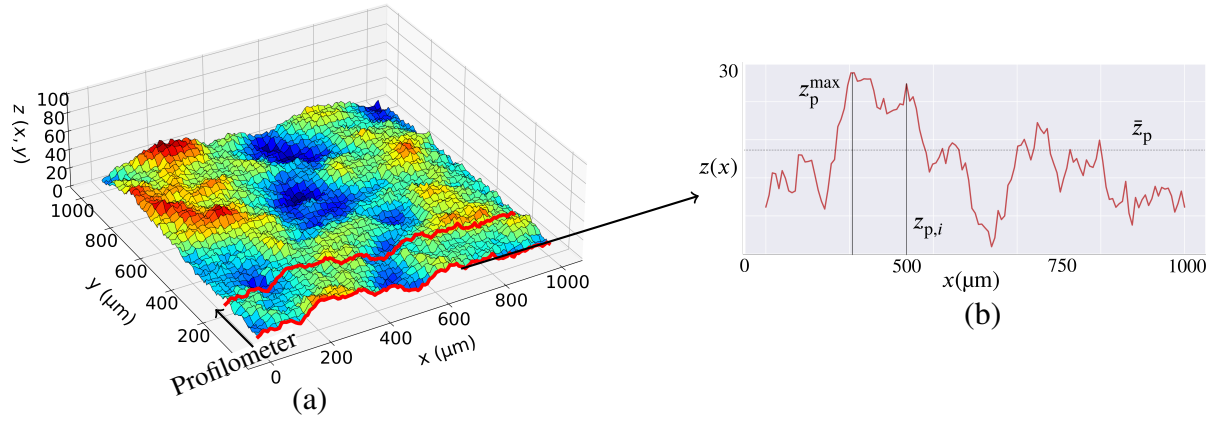


Figure 2.16.: (a) 3D representation of a rough topography. (b) A corresponding 2D profile characterized by specific statistical parameters, such as peak heights $z_{\mathrm{p},i}$ with accompanying maximum and mean values, $z_\mathrm{p}^\mathrm{max}$, $\bar{z}_\mathrm{p}$, respectively.

2.4.5. State of the art: BEM-based numerical simulations

As stated in Section 2.4.2, the solution of the rough contact problem requires the determination of surface displacements. The Boundary Element Method applied to rough contact problems [184] is a robust numerical workflow to obtain boundary displacements u at the frontier of a deformable body, in this case a linear elastic half-space. The formulation of the continuous problem starts from the closed-form convolution of an applied traction field p with a Green function that can be expressed as the following Boundary Integral Equation (BIE):

$$u(x,y) = \frac{1-\nu^2}{\pi E} \int_\mathcal{S} \frac{p(\xi,\eta)}{\sqrt{(x-\xi)^2+(y-\eta)^2}} \,\mathrm{d}\xi d\eta. \tag{2.132}$$

Assuming the action of the vertical far-field displacement Δ, that pushes the bodies in contact, an indentation level $\overline{u}$ can be defined, that quantifies the interference of the rough surface and the half-space, Fig. 2.17. The introduction of this quantity allows to state the contact problem as an infinite dimensional Linear Complementarity Problem (LCP):

$$\begin{aligned} g(x,y) \geq 0, \quad p(x,y) \geq 0, \\ g(x,y)p(x,y) = 0, \end{aligned} \tag{2.133}$$

where $g = u - \overline{u}$. Equations (2.132) and (2.133) above can be discretized on the Cartesian grid introduced in Section 2.4.2, resulting in the finite dimensional LCP:

$$\begin{aligned} \boldsymbol{g} \geq 0, \quad \boldsymbol{p} \geq 0, \\ \boldsymbol{g} \cdot \boldsymbol{p} = 0, \quad \boldsymbol{u} = \boldsymbol{H} \cdot \boldsymbol{p}, \end{aligned}$$

where the discretized field variables $\boldsymbol{w} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{p} \in \mathbb{R}^{n\times n}$ are coincident with the integral average value of the equivalent continuous quantities, evaluated in correspondence of the center of each cell g_c. Under linear elasticity assumptions it can be proven that $\boldsymbol{H}$ is symmetric and positive definite and that the problem has a unique solution. The values to be used in the definition of the matrix $\boldsymbol{H}$ are dependent on the discretization framework employed. The interested reader is pointed to, among the others, [16, 122] for possible choices of the influence coefficients.

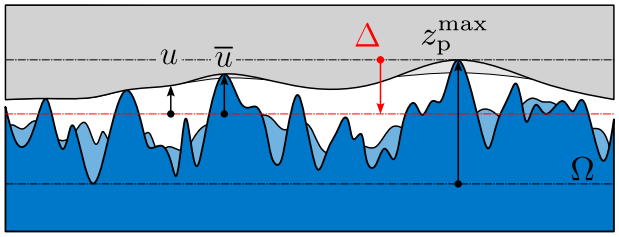


Figure 2.17.: Under the action of the vertical far-field displacement Δ, the deformable bodies are pushed into contact. To properly formulate the LCP problem, an interference $\overline{\boldsymbol{u}}$ can be defined, equal, at each node, to $\boldsymbol{z} - (z_{\mathrm{p}}^{\max} - \Delta)$.

Several open-source codes are available to solve rough contact problems using the formulation presented above, e.g. *MIRCO* (co-developed by the authors themselves) [17], *Tamaas* [57] and *MultiFEBE* [24]. Since we aim at simulating linear elastic frictionless contact between a rigid rough indenter and an elastic half-space, *MIRCO* is used to run the numerical simulations. Fig. 2.18 presents example results from contact simulations using two different surface topographies created by the RMD, and two indentation levels, $\Delta_0 = 25\,\mu\mathrm{m}$ and $\Delta_1 = 35\,\mu\mathrm{m}$. For the same topographies, given in Fig. 2.18a and Fig. 2.18b, increasing the indentation level leads to a larger effective contact area, as shown in Fig. 2.18d and Fig. 2.18e. On the other hand, for the same indentation level (see Figures 2.18a and 2.18c), two different topographies can lead to very different effective contact areas. The surface in Fig. 2.18c is much smoother and consequently more points get in contact as can be seen from the resulting contact force distributions in Figures 2.18d and 2.18f.

Furthermore, to illustrate the accuracy of *MIRCO*, a benchmark example, namely the contact problem between an elastic half-space and a smooth paraboloid rigid indenter,[123] is investigated. Analytical solutions for the contact force F and the normalized contact area A_n can be obtained as:

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{R\Delta^3} \quad \text{and} \quad A_n = 100\frac{\pi R\Delta}{L^2}, \tag{2.134}$$

where R is the minimum radius of curvature of the paraboloid. For our specific setup, we set $R = 5{,}000\,\mu\mathrm{m}$, $L = 1{,}000\,\mu\mathrm{m}$, and Δ linearly varying from 0 µm to 1 µm. Fig. 2.19 shows a comparison between the analytical solution and two numerical solutions obtained using BEM with two different grid discretizations of 128×128 and 256×256 points per side. As shown in Fig. 2.19a, the contact force computed using BEM matches the analytical solution well for both grid resolutions. In contrast, the computed contact area slightly underestimates the analytical solution at higher resolution, and is even smaller with the coarser grid, as illustrated in Fig. 2.19b. These deviations stem from the method used to calculate the effective contact area. A simple

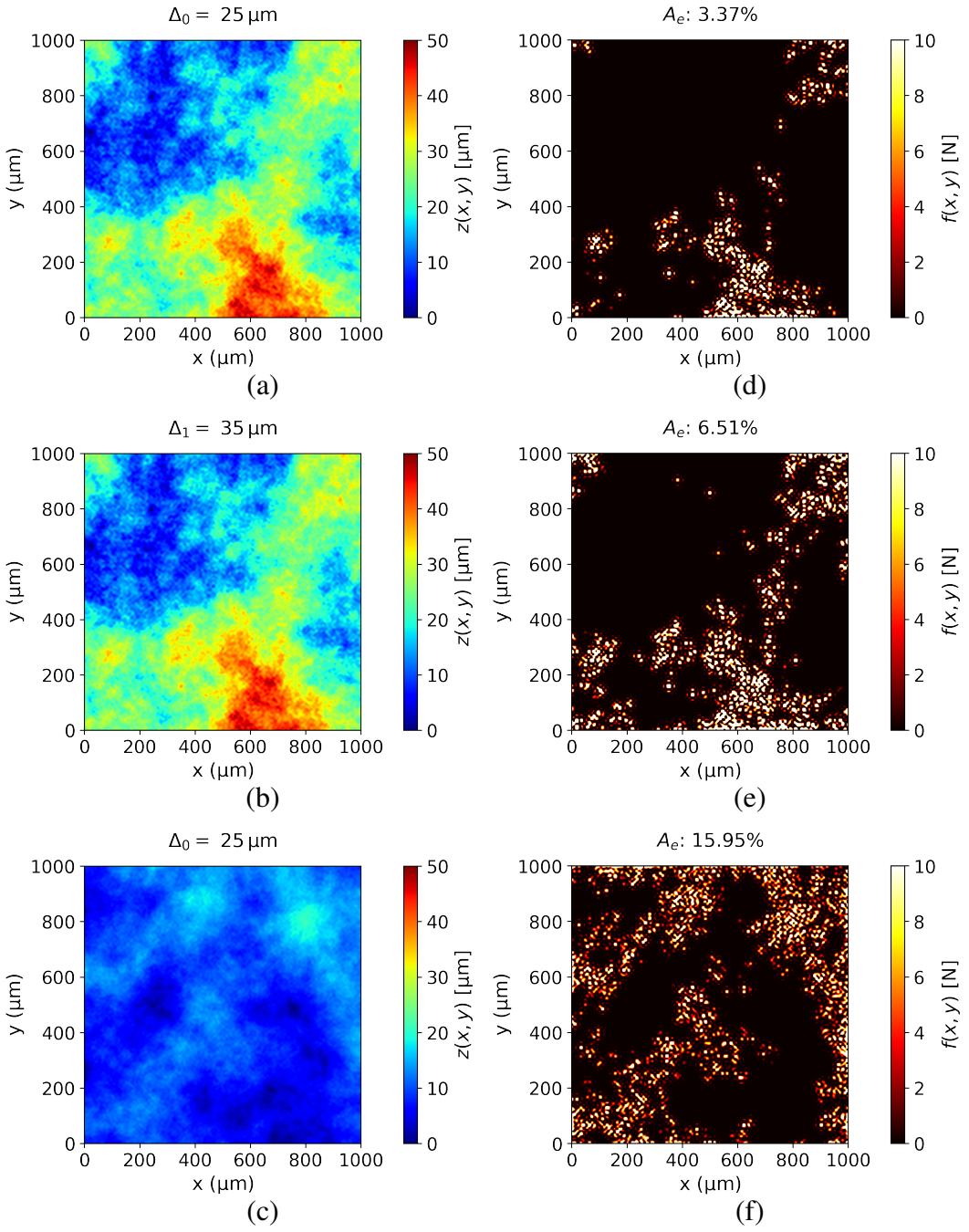


Figure 2.18.: The top row of images (Fig. a-c) shows two surface topographies generated by the RMD. Dark blue indicates the deepest valleys, while dark red represents the highest summits. The bottom row of images (Fig. d-f) shows the resulting contact force per grid point f, for two far-field displacements $\Delta_0 = 25\,\mu\text{m}$ and $\Delta_1 = 35\,\mu\text{m}$. The corresponding effective contact area A_e is also shown.

method is used to estimate the contact area by dividing the number of contact nodes by the total number of nodes on the micro-scale surface. As a result, the finer grid leads to lower errors and smoother results.

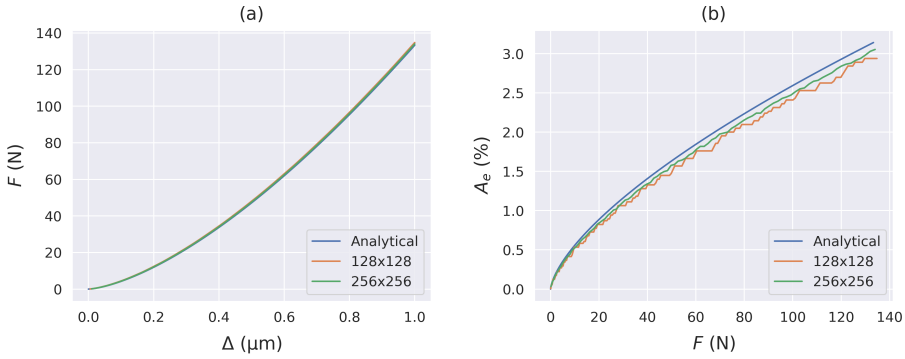


Figure 2.19.: Comparison between analytical and BEM solutions for the paraboloid rigid indenter example. BEM models have surfaces discretized on a grid with 128×128 and 256×256 points per side, respectively.

Although BEM is an efficient method, simulations can still become computationally expensive. Fig. 2.20a presents the simulation times for different combinations of far-field displacements and surface realizations. Each point represents the simulation time for a randomly generated surface and a specific value of the far-field displacement. All surfaces are discretized on a 128×128 grid. The simulation time varies significantly across different far-field displacements. Additionally, there is considerable variability in simulation time even for the same far-field displacement, which can be attributed to differences in surface properties. Generally, larger far-field displacements and smoother surfaces result in longer simulation times. Notably, both higher far-field displacements and smoother surfaces are associated with increased contact areas. Fig. 2.20b illustrates this trend by showing the relationship between simulation time and effective contact area. Clearly, larger contact areas lead to longer simulation times.

The simulation time for a single run can reach up to approximately $1{,}750\,\mathrm{s}$. While relatively low compared to other large-scale models, this still makes multi-query analyses computationally expensive. Such analyses-including parameter studies, sensitivity analysis, uncertainty quantification, and parameter identification-are crucial for many real-world applications.[30, 47, 176] For example, parameter studies and sensitivity analysis explore the influence of variables like grid size or far-field displacement, while uncertainty quantification evaluates how input uncertainties affect model outputs. Parameter identification, on the other hand, aims to infer model parameters from data. All these tasks require running the model many times—often several thousands—making them potentially prohibitively expensive due to the computational cost of individual BEM simulations.

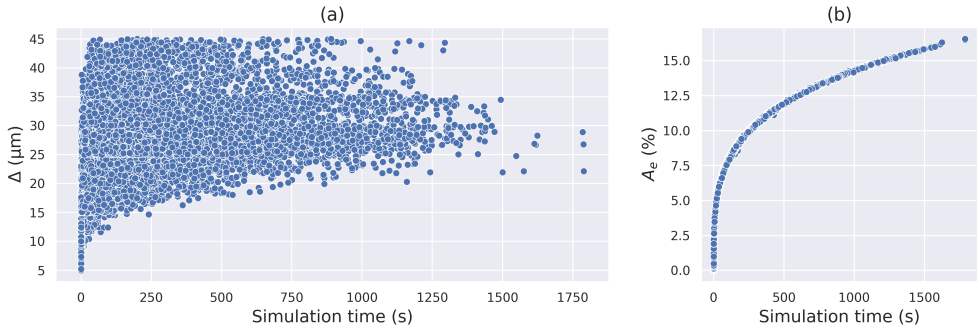


Figure 2.20.: Comparison of simulation time for rough surface contact simulations with a grid size of 128×128. (a) shows the effect of different far-field displacements and surface realisations on the overall simulation time. Each point represents a randomly generated surface and a specific far-field displacement. (b) The correlation between effective contact area and simulation time.

3. Fundamentals of Machine Learning

Machine learning is a field of artificial intelligence (AI) which focuses on developing computer systems that learn and improve from experience [108, 149]. Rather than being controlled by explicit instructions for certain tasks, machine learning systems utilize data to grasp patterns, connections, and insights, which can subsequently be used to make predictions and decisions, or execute tasks autonomously.

As illustrated in Fig. 3.1, conventional programming or computing involves supplying both the input data and the well-defined function to compute the model output which is the unknown quantity. This well-defined function serves as a mapping between the input data and the model output. On the other hand, machine learning aims to reveal the underlying pattern to predict the unknown target function. Here, the model is trained using both input and output quantities to learn these patterns.

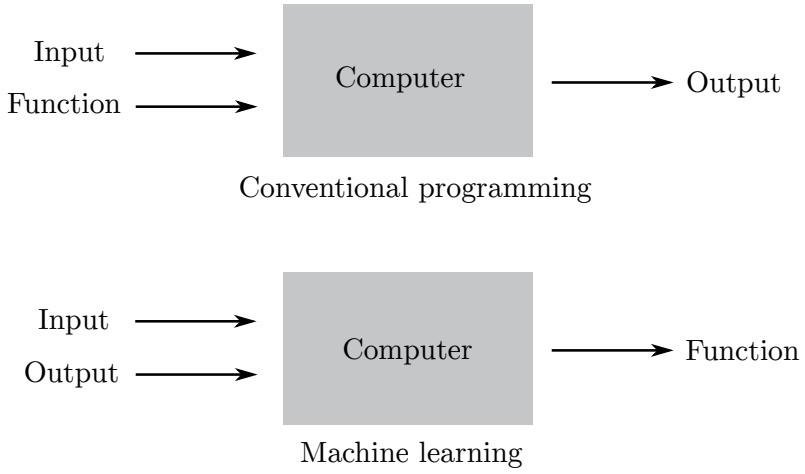


Figure 3.1.: Comparison of classical programming and machine learning.

3.1. Machine learning terminology

To ensure a smooth transition before we delve into machine learning terminology in the following sections, let us briefly explain some commonly used basic terms.

- **Model:** A mathematical representation or computer programs to capture the underlying behavior or pattern in data.
- **Features:** The input variables or attributes used by the model.
- **Target (label):** The output variable or the model predictions.

- **Training set:** It is used to train the model to obtain the optimal network parameters. Usually, the training set comprises 60-80% of the entire dataset.
- **Validation set:** The dataset used during training to tune hyperparameters of the model. Typically, it refers to 10-20% of the dataset.
- **Test set:** The dataset used for evaluating model performance after training. In other words, it is used to show how well the model generalizes the unseen samples. Typically, the test set represents 10-20% of the dataset.
- **Model parameters:** Learnable model parameters that are optimized during training.
- **Overfitting:** The trained model performs exceptionally well on training data but poorly on unseen data. In other words, the model mimics the training data, including noise, delivering poor performance on test data.
- **Underfitting:** A situation where the trained model is too simple, it cannot capture the underlying
- **Regularization:** Techniques used to tackle overfitting by introducing a penalty term to the loss function.
- **Loss (cost) function:** A measure of how accurately the predictions align with the actual values in the training set.
- **Optimizer:** A process used to compute optimal model parameters that minimize the loss function.
- **Epoch:** During the model training, refers to a single pass using the entire training dataset.
- **Iteration:** A single step of an optimizer to update the model parameters.
- **Batch size:** The number of training samples that a model processes in a single iteration.
- **Hyperparameters:** Parameters that specify and control the learning process; they are usually set before training and can be optimized during training as well.
- **Cross-Validation:** A technique used during training to tune hyperparameters of the model.
- **Performance metric:** A measure of how accurately the predictions align with the actual values in the test set.

3.2. Types of learning systems

Machine learning approaches can be classified into various groups based on the following criteria:

- **Learning type:** Learning system is based on how the labeled or unlabeled data is utilized, in other words availability of human supervision (supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning)

- **Output type:** Whether the predicted output quantity belongs to a class or is a continuous value (classification or regression)
- **Model on the fly:** The decision criterion is whether the model is trained on the entire dataset at once (batch learning), or the model is trained incrementally by providing it with data instances (online learning)
- **Model type:** The decision criterion is whether to build a model by learning the pattern, i.e., optimizing the model parameters (model-based learning), or to use specific instances to mimic the training data points (instance-based learning)

This classification can be further expanded by including other common model characteristics. Nevertheless, in this thesis, the discussion is restricted to categorizations according to the learning type.

3.2.1. Learning type

Machine learning models based on learning type are differentiated depending on the type and amount of supervision. This branch mainly includes supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

3.2.1.1. Supervised learning

Supervised learning [100, 108] is a type of machine learning in which the model learns the underlying pattern utilizing the labeled dataset or the known output quantities. In supervised learning, the training data includes input-output pairs, where inputs denote the model features, and outputs represent the labels or output quantities.

An illustrative example of supervised learning is provided in Fig. 3.2. Training data consists of labeled samples, such as cats or dogs. The machine learning algorithm learns the underlying pattern within this training set to distinguish whether a given sample is a dog or a cat in new samples. This example demonstrates a classification problem in supervised learning. However, supervised learning also encompasses regression problems. A key characteristic of supervised learning is that all samples in the training set are labeled.

Some of the key supervised learning algorithms covered in this thesis are given as:

- k-Nearest Neighbors
- Linear Regression including derivatives
- Decision Tree
- Ensemble methods: Random Forest
- Boosting algorithms: AdaBoost, Gradient Boost, XGBoost
- Kernel-based methods: Kernel Ridge Regressor, Gaussian Process (GP) Regressor and Support Vector Machines

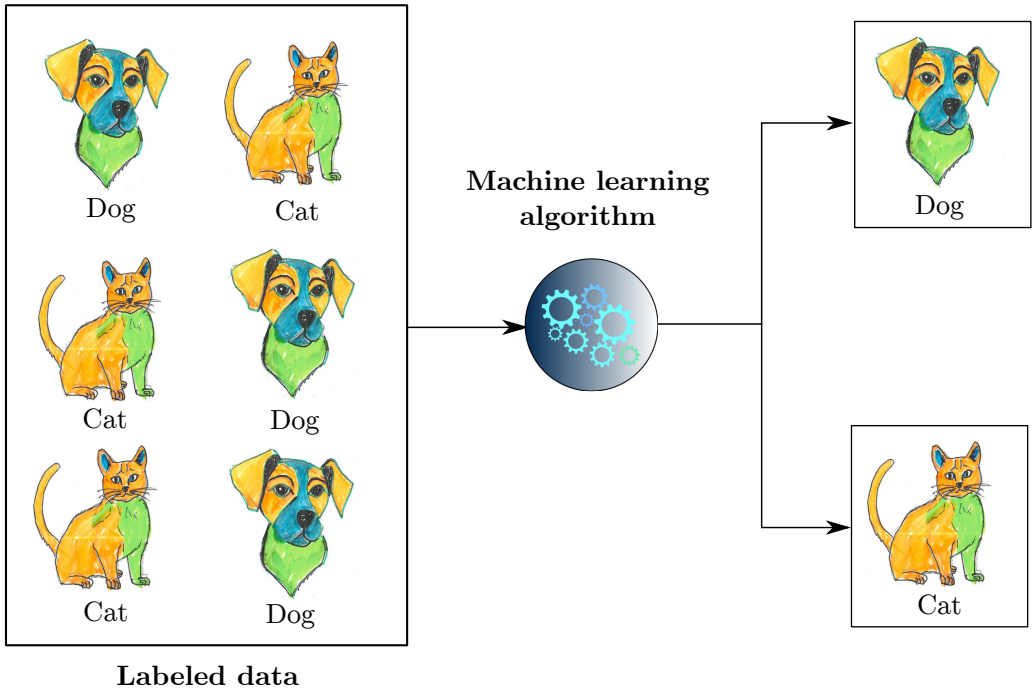


Figure 3.2.: An illustrative example for supervised learning (classification).

- Neural networks

In this thesis, we focus solely on regression models, as they serve as methods for the examples in the upcoming chapters.

3.2.1.2. Unsupervised learning

In unsupervised learning [72], training data is not labeled, thus the machine learning algorithm attempts to identify underlying patterns and structures within the dataset by creating and analyzing similarities and differences among the samples, i.e., clustering of outputs.

As demonstrated in Fig. 3.3, the training dataset does not have any labeling, meaning there is no information about the output quantities. The expectation is that the machine learning algorithm will identify the underlying patterns by extracting features based on similarities and differences within the samples. For instance, the model can form two distinct clusters, one for samples collecting features that belong to cats (e.g., ear shape and whiskers), and another for samples collecting features common between dogs, such as body type and snout.

Most common unsupervised learning algorithms are given as:

- Clustering:
  - K-means Clustering,
  - Hierarchical Clustering.
- Dimensionality reduction:
  - Principal Component Analysis (PCA),

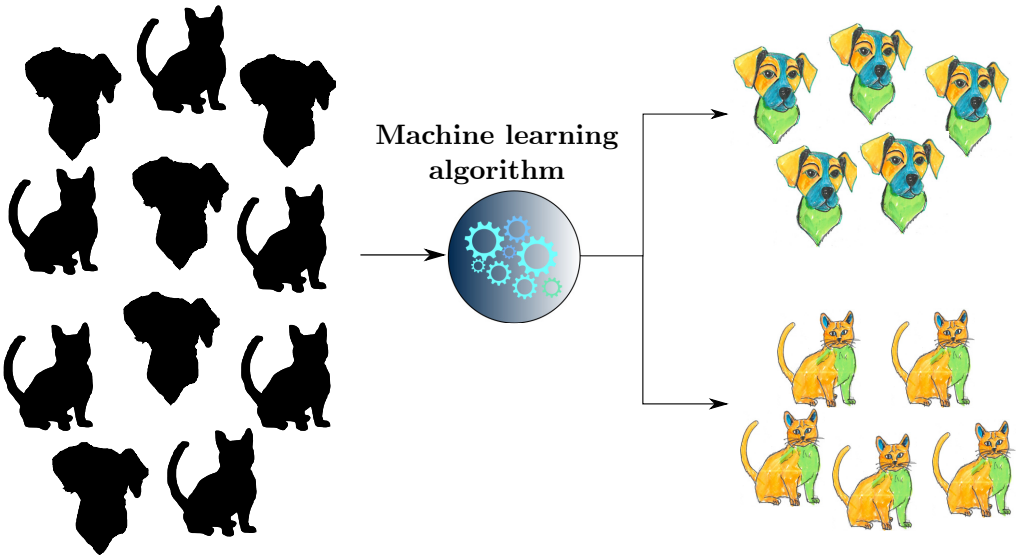


Figure 3.3.: An illustrative example for unsupervised learning (clustering).

  - t-Distributed Stochastic Neighbor Embedding (t-SNE).

- Association:
  - Apriori algorithm,
  - Eclat algorithm.
- Anomaly detection:
  - One-Class SVM,
  - Isolation Forest.

3.2.1.3. Semi-supervised learning

Semi-supervised learning [196], often referred to as weak supervision, combines elements of both supervised and unsupervised learning by leveraging a small portion of labeled data together with a larger set of unlabeled data. One of the simplest approaches to implement semi-supervised learning is *self-learning* (also known as self-training). The procedure can be outlined as follows:

- **Step 1**: Train the model using the available labeled data (supervised learning stage). This allows the model to capture the underlying patterns present in the labeled dataset.
- **Step 2**: Use the trained model to predict labels for a subset of the unlabeled data. Only a small portion is selected in order to minimize the risk of propagating errors.
- **Step 3**: Combine the original labeled data with the newly labeled samples to form an updated training set.
- **Step 4**: Retrain the model on the expanded dataset, thereby improving its ability to recognize patterns in both the original and newly labeled data.

- **Step 5**: Repeat Steps 2–4 iteratively until a satisfactory proportion of the unlabeled data has been assigned labels.

As outlined in the steps of semi-supervised learning, this approach is inherently iterative and may be computationally expensive. Nevertheless, it considerably reduces the manual effort required for data labeling, since the algorithm automatically assigns labels to the unlabeled data during training.

3.2.1.4. Reinforcement learning

In reinforcement learning [162], an agent learns the underlying pattern in data based on obtained rewards by letting it interact with an environment. The aim is to maximize the positive reward based on the actions the agent makes by trial and error.

An illustrative sketch of how reinforcement learning works is given in Fig. 3.4. The environment has an initial state or observation, which is a representation of the current situation or condition within the environment. Based on the current state and the policy function, the agent selects an action to take. Here, the policy function can be considered as a set of guidelines the agent follows to select actions. Based on the current state and the action, the environment updates the state and delivers a reward to the agent. The reward provides an assessment of the quality of the current action, indicating how good or bad the action taken is. In the next step, this computed reward and the updated state are delivered to the agent. Based on the received reward, the machine learning algorithm updates the parameters of the agent policy to maximize the positive reward or minimize the negative reward. The machine learning algorithm in the context of reinforcement learning includes approaches such as Q-Learning and Deep-Q-Networks. Based on the updated policy function and the change in the state, the agent takes a new action. Similar to semi-supervised learning, this is an iterative process. Ultimately, the agent learns which actions lead to higher positive rewards and which do not.

Since reinforcement learning algorithms learn directly from interacting with the environment, they can easily adapt to new states or conditions. However, this process requires numerous interactions, making it time-consuming and expensive. Due to its self-learning and continuous learning capability, reinforcement learning has been employed in various fields including the game industry, robotics, healthcare and engineering.

3.2.2. Output type

Machine learning models can also be classified based on the type of output. This categorization typically distinguishes between classification and regression tasks.

3.2.2.1. Classification

In classification tasks, the model predicts a discrete label or class for a given input. The output space is finite and consists of a predefined set of categories. Classification is commonly used in engineering applications such as crack detection in structural components [183], short-term traffic flow forecasting [99], and fault diagnosis in induction motors [6].

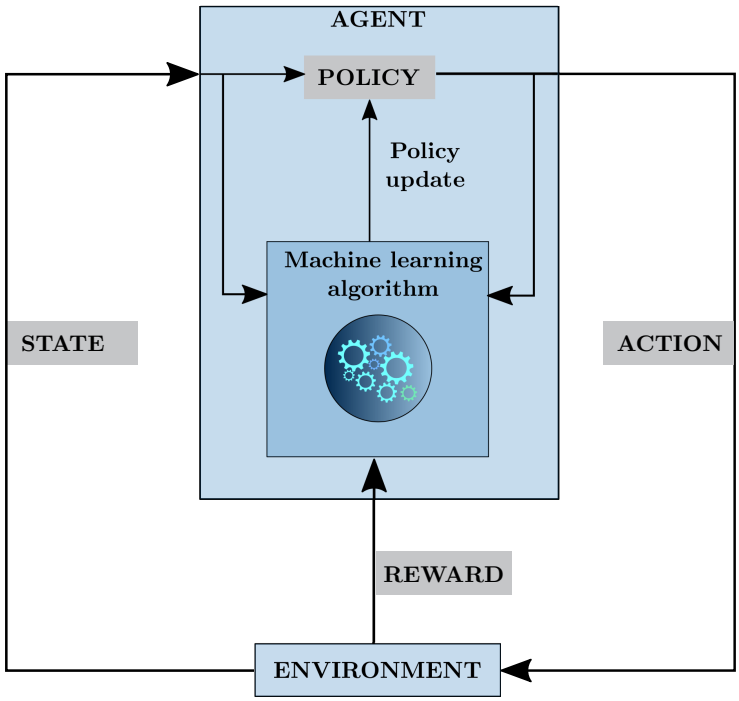


Figure 3.4.: An illustrative example of reinforcement learning.

3.2.2.2. Regression

In regression tasks, the model predicts a continuous numerical value based on the input features. The output is not limited to a finite set of categories, but can take on any value within a range. Regression is widely used in engineering applications such as predicting stress, strain, and deformation fields in materials and structural systems [106], forecasting the remaining useful life (RUL) of mechanical components [155], among many others.

3.3. Regression in machine learning

Regression in machine learning is a supervised learning technique used to predict continuous outcomes. The goal of a regression algorithm is to uncover the relationship between input features and a quantitative target variable. For example, forecasting stock prices is a regression task because the target value, the price, is continuous rather than categorical.

In this thesis, we will not cover all regression models, but rather focus on the most commonly used ones. The illustration in Fig. 3.5 presents the regression models discussed in the following sections, and additionally provides a categorization of these models. Tree-based models aim to extract features by partitioning the feature space into smaller domains based on splitting rules. The model can consist of a single tree (Decision Tree) or it can contain multiple trees, known as an ensemble method, such as Random Forest. In boosting algorithms, multiple weak learners (decision trees) are trained sequentially to achieve higher model accuracy in such a way that each sequential tree improves the errors of its predecessor. Linear Regression generates linear models between the input and output spaces through the optimal model parameters. In other words,

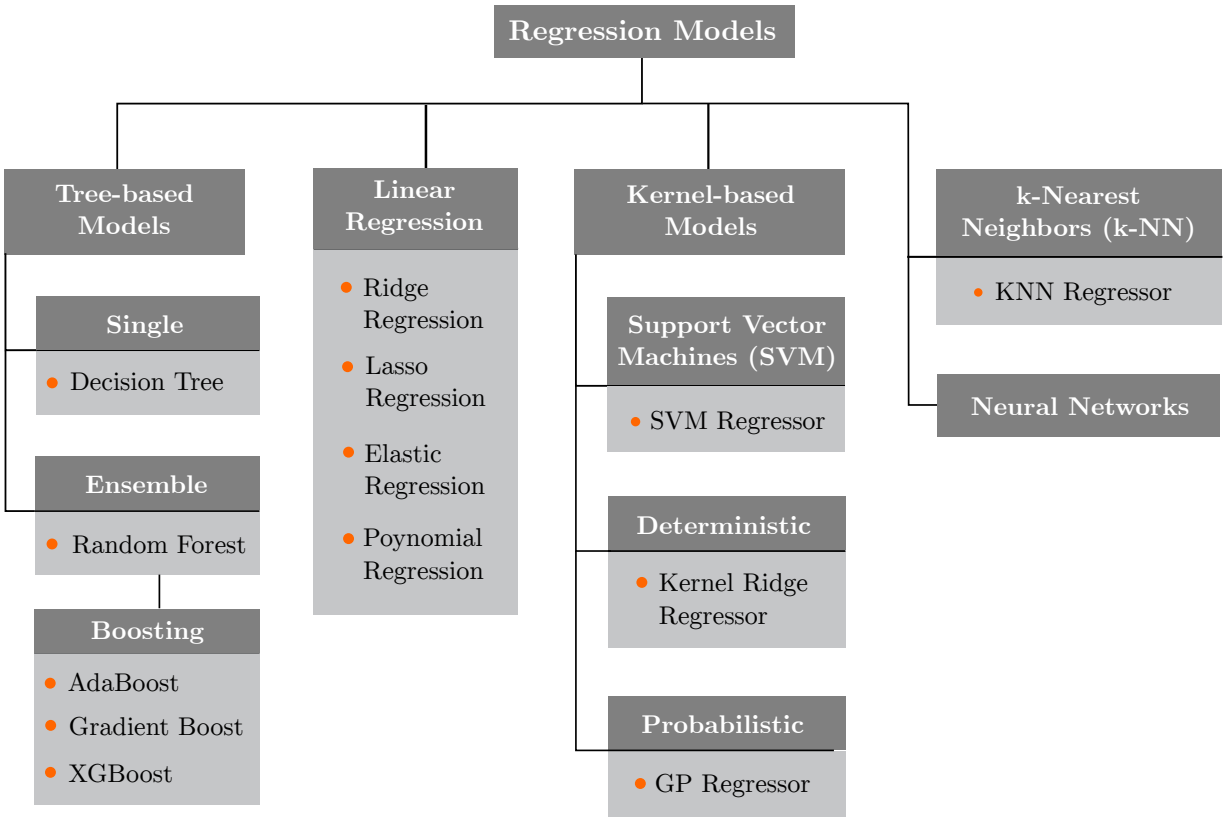


Figure 3.5.: List and categorization of regression models in machine learning.

the output quantity can be expressed as a linear combination of the input quantity. This branch contains regularized models to avoid overfitting, such as Ridge, Lasso, and ElasticNet Regression, and also a polynomial form for capturing nonlinear relationships, such as Polynomial Regression. Kernel-based methods utilize kernel functions (often referred to as the kernel trick[110]), where the kernel is deterministic, i.e., Kernel Ridge Regressor, and the kernel is probabilistic as in GP Regressor. If the same kernel and hyperparameters are used, the mean function of a GP Regressor is identical to the Kernel Ridge prediction. Support Vector Machines (SVM), a type of kernel-based method, are employed to determine the optimal hyperplane that separates different feature spaces. In the case of SVM regression, the objective is to identify the best-fitting hyperplane that encompasses the majority of data points while maintaining a predefined margin. The k-Nearest Neighbors (k-NN) algorithm is an instance-based learning method that uses specific samples to mimic the training data. k-NN Regressor predicts the target quantity by finding the k nearest data points via a distance metric and averaging the corresponding target values. Neural networks extract the underlying pattern hidden in data by utilizing connected layers containing neurons with activation functions. Since neural networks constitute the core models used in this thesis, they will be explained in Sec. 3.4 in detail.

3.3.1. k-Nearest Neighbors

The k-Nearest Neighbors (k-NN Regressor) algorithm [44, 53] is regarded as the simplest machine learning algorithm. It is an instance-based learning method, meaning that it does not involve explicit learning, but rather mimics the training data. As it does not include learning or training, it is also known as *lazy algorithm*. The concept behind k-NN Regressor (the k-NN Regressoralgorithm for regression problems) is first to identify the k-Nearest neighbors and then compute the average of their corresponding target values to obtain the model prediction. In another saying, the average of predictions of the k-Nearest Neighbors algorithm is set as the k-NN Regressor prediction. A pseudocode for k-NN Regressor is given in Algorithm 2, along with explanations for each step.

As shown in the k-NN Regressor algorithm, k is a hyperparameter that has to be optimized. Fig. 3.6 shows the effect of the different nearest neighbors k. Subplots demonstrate the k-NN regression fit for $k = 1$ (left), $k = 3$ (center), and $k = 10$ (right). For smaller k values, the model tends to overfit, meaning that the model mimics training data. For instance, with $k = 1$, the training error is zero since the model perfectly memorizes the training data. This leads to a model with high variance, delivering poor performance on unseen data. On the other hand, increasing k leads to underfitting, meaning that k-NN Regressor can not generalize the given data anymore. As k increases, the prediction curve becomes smoother, indicating a trade-off between bias and variance. Therefore, to avoid both underfitting and overfitting, the optimal k has to be determined, such as through cross-validation (see Sections 3.5 and 3.6 for more details on overfitting and underfitting).

As seen in the Algorithm 2, k-NN Regressor is not an efficient method since it requires the calculation of distance metric for every new unseen data. In other words, each prediction is considered as a new training. On the other hand, in the case of multiple features, those with larger magnitudes can dominate the model prediction due to the distance function. Therefore, feature scaling techniques can be applied to avoid this domination.

Algorithm 2 The k-NN Regressor algorithm

Inputs:
I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
k: the number of nearest neighbors
$\boldsymbol{x}_*$: unseen sample points
$d(\bar{\boldsymbol{x}}, \boldsymbol{x})$: a distance metric
Output:
$\hat{\boldsymbol{y}}$: predicted output

procedure KNNREGRESSOR($I, k, \boldsymbol{x}_*, d$)
Initialize the distance array D
for each training pair $(\boldsymbol{x}_i, y_i) \in I$ **do**
$D_i = d(\boldsymbol{x}_*, \boldsymbol{x}_i)$ ▷ Compute the distance between $\boldsymbol{x}_*$ and $\mathbf{x}$
end for
sort($\boldsymbol{D}$) ▷ Sort the distance array (ascending)
ind = slice($\boldsymbol{D}[:k]$) ▷ Get the indices of k-nearest neighbors
$\hat{\boldsymbol{y}}_s$ = slice($\hat{\boldsymbol{y}}[ind]$) ▷ Get corresponding y values of nearest neighbors
$\hat{\boldsymbol{y}}$ = mean(y_s) ▷ Compute the average $\hat{\boldsymbol{y}}$ of k-nearest neighbors
return $\hat{\boldsymbol{y}}$
end procedure

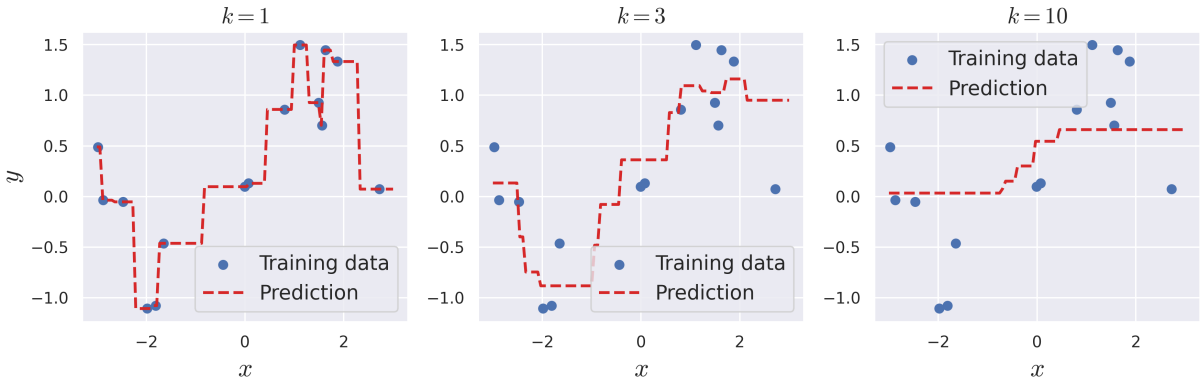


Figure 3.6.: Comparison of results of k-NN Regressor for different values of k. Subplots demonstrate the k-NN regression fit for $k = 1$ (left), $k = 3$ (center), and $k = 10$ (right).

3.3.2. Linear Regression

The Linear Regression [48] algorithm can be considered also one of the simplest machine learning approaches. The idea is to explain the output quantity as a linear combination of the input quantity as follows

$$\hat{y}_i = b + w_1 x_{i1} + \dots w_d x_{id}. \tag{3.1}$$

Here, $\hat{y}_i$ denotes the model prediction for the sample i, where $i \in [1, N]$, x_{id} denote the features with d number of features, and learnable parameters of the model are denoted as b (the bias term) and w (the weights). The equation of linear regression (Eq. 3.1) can be expressed also in vector form such as

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta},$$

where, $\boldsymbol{X}$ represents the input vector such as $\boldsymbol{X} = [\mathbf{1}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_d]$ and $\boldsymbol{\theta}$ is the learnable parameters vector such as $\boldsymbol{\theta} = [b, w_1, \dots, w_d]^\top$. $\top$ is the transpose operator. To facilitate matrix-vector multiplication, an artificial "*1*" is appended to the first column of the input vector. For example, if we have a dataset with three features and three samples, the variables appear as follows:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{13} \\ 1 & x_{31} & x_{32} & x_{13} \end{bmatrix}, \quad \hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 = b \\ \theta_1 = w_1 \\ \theta_2 = w_2 \end{bmatrix}. \tag{3.2}$$

In linear regression, the aim is to find the most accurate approximation of the actual values $\boldsymbol{y}$, such that

$$\boldsymbol{y} \approx \hat{\boldsymbol{y}}(\boldsymbol{\theta}).$$

To fulfill this approximation, optimal parameters have to be determined. Thus, the cost function or the loss function is introduced. In linear regression, the cost function quantifies the disparity between the predicted and actual values in the training data set. There are distinct types of cost functions based on the purpose (see 3.4.3.4 for details). The *Mean Squared Error* (MSE) is commonly used in linear regression. So the MSE cost function for linear regression (in cumulative form) is defined as

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - (\theta_0 + \theta_1 x_{i1} + \dots + \theta_n x_{id})\right)^2,$$

and it can be expressed in the vector form as

$$J(\boldsymbol{\theta}) = \frac{1}{2N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}). \tag{3.3}$$

The aim is to find the optimal parameters that minimize the cost function. This can be achieved in two ways: Normal Equation (the direct method) and Gradient Descent (the iterative method).

Normal Equation is a direct method that is developed based on the derivatives of the cost function. By taking derivative of the cost function $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ and equating the derivative to zero, we obtain the Normal Equation to get optimal parameters such as

$$\boldsymbol{\theta} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}.$$

The Normal Equation provides a closed-form solution for determining the optimal parameter values. However, this approach involves several matrix operations, including multiplication, transposition, and inversion, which can become computationally expensive when dealing with large datasets with many samples and features. Moreover, if the matrix product $\boldsymbol{X}^\top \boldsymbol{X}$ is ill-conditioned (for example, when it has a high condition number), its inversion may lead to numerical instability and inaccurate results.

In contrast, the Gradient Descent method is an iterative process to minimize the cost function. The algorithm seeks to find the optimal $\boldsymbol{\theta}$ by moving in the direction of the negative gradient of J. The update rule is given by

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}), \tag{3.4}$$

where η is the learning rate, which controls the step size of each iteration. A detailed pseudocode for Linear Regressor using Gradient Descent is given in Algorithm 3. The algorithm takes as inputs the training data $\boldsymbol{X}$ and $\boldsymbol{y}$, the cost function (mse), the learning rate η, the number of iterations n_{iter} and a set of unseen test samples. The objective is to estimate the parameter vector $\boldsymbol{\theta}$ of the linear model by minimizing the cost function, and subsequently compute the predictions based on the optimal model parameters.

Algorithm 3 Linear Regression using Gradient Descent (Vectorized)

Inputs:
I: training data in vector form $\boldsymbol{X}$, $\boldsymbol{y}$
J: cost function (mse)
η: learning rate
n_{iter}: number of iterations
$\boldsymbol{x}_*$: unseen sample points
Output:
$\hat{\boldsymbol{y}}$: predicted output

procedure LINEARREGRESSION($I, \eta, n_{\text{iter}}, \boldsymbol{x}_*$)
Initialize $\boldsymbol{\theta}$ ▷ Initialize parameter vector
Get the number of samples N
for each iteration **do**
$\hat{\boldsymbol{y}} \leftarrow \boldsymbol{X}\boldsymbol{\theta}$ ▷ Compute the predictions
$\nabla J \leftarrow \frac{1}{N}\boldsymbol{X}^T(\hat{\boldsymbol{y}} - \boldsymbol{y})$ ▷ Get the gradient of the cost function
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J$ ▷ Update parameters using gradient descent
end for
$\hat{\boldsymbol{y}} = [1 \;\; \boldsymbol{x}_*]\boldsymbol{\theta}$ ▷ Compute the prediction based on optimal θ
return $\hat{\boldsymbol{y}}$
end procedure

As explained above, the Normal Equation method delivers an analytical solution to determine the optimal parameter, so it does not involve any hyperparameters. However, this approach becomes computationally expensive when handling large datasets, as it involves multiple matrix

operations, particularly matrix inversion. This situation can potentially cause computational challenges, when particularly for ill-conditioned matrices. Thus, the Normal Equation method is more appropriate for small to medium-sized datasets. In contrast, the Gradient Descent does not face such limitations concerning dataset size, making it suitable for also large-sized datasets. However, it is an iterative process and requires careful selection of hyperparameters, such as the learning rate. For example, choosing a very small learning rate may result in slow convergence, while a very large learning rate can cause the solution to diverge.

3.3.2.1. Polynomial Regression

Polynomial regression is a special type of linear regression where the output is predicted using p^{th} order polynomial such as

$$\hat{y}_i = b + w_1 x_i^1 + \dots w_d x_i^p. \tag{3.5}$$

Although polynomial regression is a nonlinear function of x, it still remains linear with respect to the parameters. Therefore, it is classified as a linear regression type. In general, higher-order terms are simply treated as additional features. In other words, the model does not recognize that x^2 has a correlation with x. Note that Eq. 3.5 shows the polynomial regression formulation for a single feature. However, it can be easily extended by adding polynomial terms for other features as well.

Fig. 3.7 depicts a comparison of Linear Regression and Polynomial Regression on training data. The left figure shows that the prediction of a linear model cannot accurately capture the underlying behavior as the model is not sufficiently complex. This phenomenon is also known as underfitting. On the other hand, plots on the right figure show the results of Polynomial Regression for different polynomial orders p. Regression models with increasing the polynomial order can fit the training data better. The order-2 polynomial fit captures the pattern roughly, showing a better fit compared to the linear model. Increasing $p = 5$ improves the model, delivering a generalizable model. The order-20 polynomial fit captures the training data very closely, however potentially leading to overfitting. See Section 3.5.3 for more details on underfitting and overfitting.

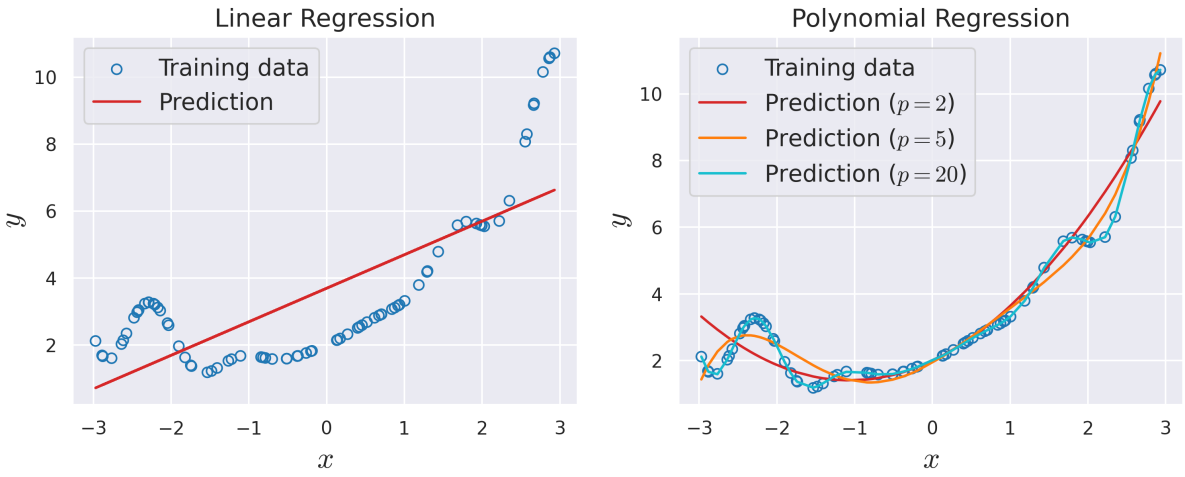


Figure 3.7.: Comparison of Linear Regression and Polynomial Regression training data.

3.3.2.2. Shrinkage methods

Shrinkage methods are used to control the model complexity by incorporating additional terms into the cost function using a technique known as *regularization*. This approach forces the model parameters to shrink towards zero. In other words, the additional terms serve as punishers to prevent the model from learning the training data too closely. Therefore, the key motivation to use a shrinkage method is to prevent overfitting and reduce the variance in the model complexity. There are mainly three types of shrinkage models: Ridge Regression, Lasso Regression and Elastic Regression.

Ridge Regression

Recall that the cost function for Linear Regression is defined as

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} \theta_j x_{ij} \right)^2 . \tag{3.6}$$

In Ridge Regression [75], the cost function includes an additional regularization term beside the Eq. 3.6 and it is expressed as

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} \theta_j x_{ij} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2, \tag{3.7}$$

where λ is the regularization parameter. Similar to Linear Regression, Ridge Regression aims to minimize the loss function. However, the additional regularization term penalizes the loss to prevent the model learns data too closely. Thus, this forces the learning parameters towards zero. Note that the regularization term of the cost function does not include the bias term (θ_0). The regularization parameter λ is a hyperparameter that has to be determined separately from the optimization process. The only requirement for choosing λ is that it must be positive, i.e. $\lambda > 0$ to ensure an increase in the loss. When $\lambda = 0$, the regularization term has no influence, so Ridge Regression becomes identical to Linear Regression. As λ approaches infinity, the regularization term dominates the cost function. In essence, the model prediction then heavily depends on the determined bias term alone, which might cause the underfitting phenomenon. This is precisely why the regularization component typically excludes the bias term. If the bias term were included in the regularization term, when $\lambda \to \infty$, the overall model prediction would tend towards zero.

Fig. 3.8 shows two subfigures illustrating the application of Ridge regression and Polynomial Regression with Ridge regularization. Ridge Regression contains three fits based on different regularization strengths. As λ increases, model complexity decreases since the weight terms approach to zero. This is also valid for Polynomial Regression fits. In both cases, increasing λ leads that model prediction becoming the computed bias term. Additionally, regularization avoids model overfitting by reducing the model variance (trade-off between variance and bias).

Table 3.1 provides a comparison of bias and weights for Ridge Regression and Polynomial Regression with regularization for varying regularization parameters (λ). For Ridge Regression, higher λ values lead to smaller weights since the minimization of the cost function forces the learning parameters towards zero. Similarly, for Polynomial Regression with regularization,

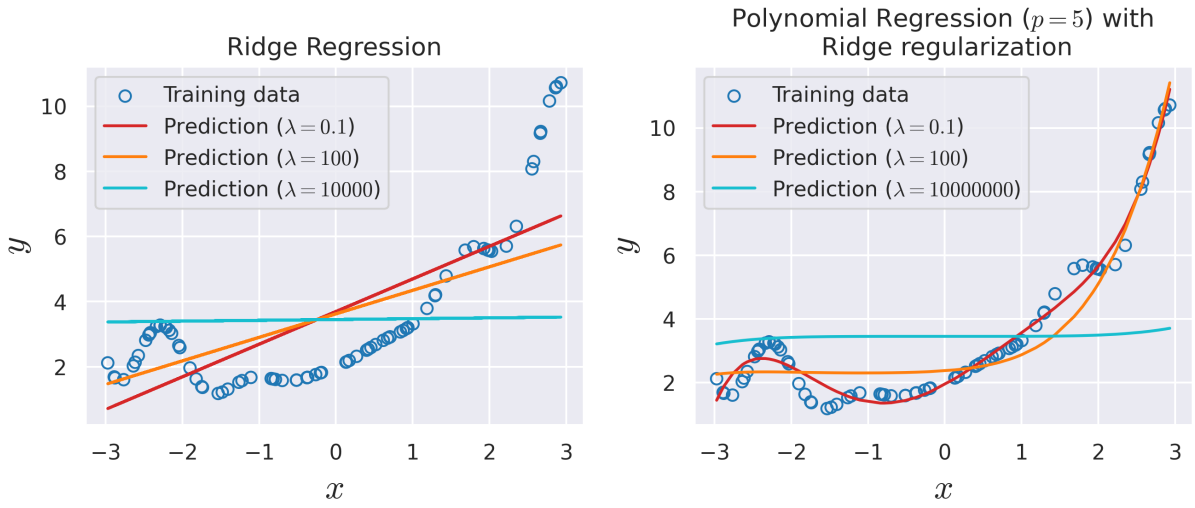


Figure 3.8.: The left figure shows Ridge Regression with varying regularization parameters and Polynomial Regression with Ridge regularization ($p = 5$) is given in the right figure.

higher λ values result in significantly smaller weights across polynomial terms. For both models, applying extensively large λ values causes model underfitting and predictions become the bias term.

	Regularization	Bias term	Weights
Ridge Regression	$\lambda = 0.1$	$\theta_0 = 3.69$	$\boldsymbol{\theta} = [1.00]$
	$\lambda = 100$	$\theta_0 = 3.62$	$\boldsymbol{\theta} = [0.72]$
	$\lambda = 10{,}000$	$\theta_0 = 3.45$	$\boldsymbol{\theta} = [0.002]$
Poly. Reg. with Ridge regularization	$\lambda = 0.1$	$\theta_0 = 1.93$	$\boldsymbol{\theta} = [1.33, 0.54, -0.28, -0.002, 0.037]$
	$\lambda = 100$	$\theta_0 = 2.37$	$\boldsymbol{\theta} = [0.18, 0.18, 0.10, 0.04, 0.01,]$
	$\lambda = 10{,}000{,}000$	$\theta_0 = 3.45$	$\boldsymbol{\theta} = [2.48 \times 10^{-5}, 2.78 \times 10^{-5}, 1.53 \times 10^{-4}, 2.21 \times 10^{-4}, 1.08 \times 10^{-3}]$

Table 3.1.: Comparison of bias and weights for Ridge Regression and Polynomial Regression with regularization for varying regularization parameters (λ).

Lasso Regression

The main drawback of Ridge Regression is that it shrinks all the parameters towards zero, but none of them will be set as zero unless $\lambda = \infty$. In other words, the model considers all features to make a prediction. This is undesirable in cases where feature selection is important, as the model does not automatically eliminate irrelevant or less significant variables. To address this limitation, Lasso Regression [163] was introduced, with the cost function expressed as

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{d} \theta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{d} |\theta_j|. \tag{3.8}$$

The key difference between Eq. 3.7 and Eq. 3.8 is that the regularization term is based on $|\theta_j|$ instead of θ_j^2. This subtle modification allows certain weights to become exactly zero. Fig. 3.9 presents two subfigures demonstrating Lasso regression and Polynomial Regression with Lasso regularization. Compared to Fig. 3.8, increasing λ results in some weights being reduced to zero, even in complex models. Table 3.2 further illustrates this effect through a qualitative comparison. For $\lambda = 5$ and $\lambda = 10$, only the polynomial terms corresponding to order $p = 4$ and $p = 5$ contribute to the weights, indicating that those two features are the most significant.

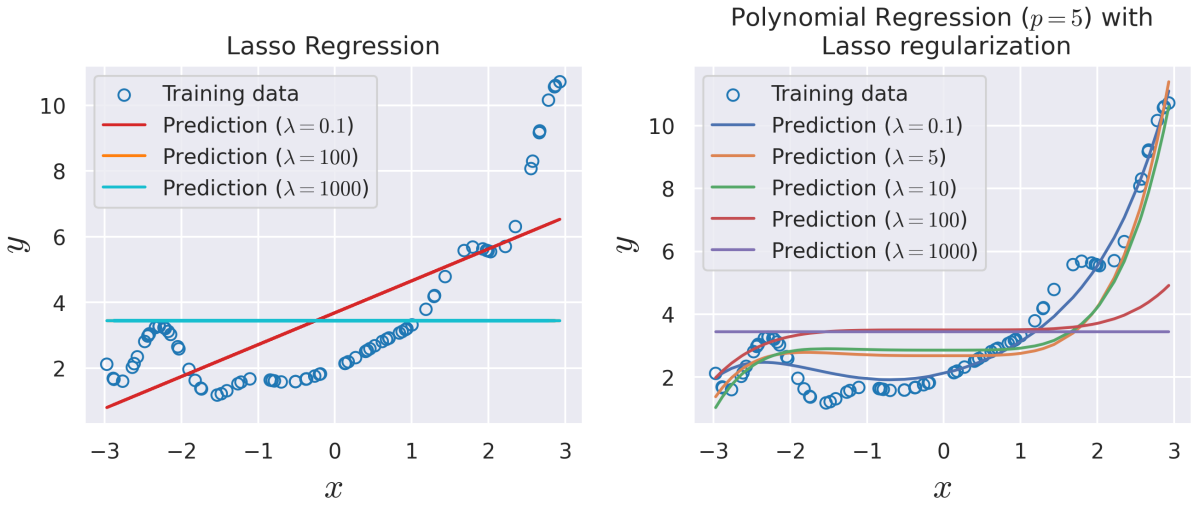


Figure 3.9.: The left figure shows Lasso Regression with varying regularization parameters and Polynomial Regression with Lasso regularization ($p = 5$) is given in the right figure.

	Regularization	Bias term	Weights
Ridge Regression	$\lambda = 0.1$	$\theta_0 = 3.69$	$\boldsymbol{\theta} = [0.97]$
	$\lambda = 100$	$\theta_0 = 3.45$	$\boldsymbol{\theta} = [0.0]$
	$\lambda = 1{,}000$	$\theta_0 = 3.45$	$\boldsymbol{\theta} = [0.0]$
Poly. Reg. with Lasso regularization	$\lambda = 0.1$	$\theta_0 = 2.12$	$\boldsymbol{\theta} = [0.58, 0.41, 0., 0.01, 0.013]$
	$\lambda = 5$	$\theta_0 = 2.68$	$\boldsymbol{\theta} = [0., 0., 0., 0.05, 0.02]$
	$\lambda = 10$	$\theta_0 = 2.86$	$\boldsymbol{\theta} = [0., 0., 0., 0.04, 0.02]$
	$\lambda = 100$	$\theta_0 = 3.51$	$\boldsymbol{\theta} = [0., 0., 0., 0., 0.006]$
	$\lambda = 1{,}000$	$\theta_0 = 3.45$	$\boldsymbol{\theta} = [0., 0., 0., 0., 0.]$

Table 3.2.: Comparison of bias and weights for Lasso Regression and Polynomial Regression with Lasso regularization for varying regularization parameters (λ).

ElasticNet Regression

As explained above, both Ridge and Lasso Regression have advantages and disadvantages. For instance, the Ridge Regression method keeps the contributions from all features, while Lasso

Regression considers only the contributions coming from the most important features. On the other hand, Lasso regression is much more sensible to the range of λ, meaning that small increase in λ might lead to the underfitting phenomenon drastically. At this point, ElasticNet comes into play as a middle ground. The cost function for ElasticNet Regression is expressed as

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \theta_0 - \sum_{j=1}^{d}\theta_j x_{ij}\right)^2 + r\lambda\sum_{j=1}^{d}|\theta_j| + \frac{1-r}{2}\lambda\sum_{j=1}^{d}\theta_j^2, \tag{3.9}$$

where r is the ratio for controlling contributions between Ridge Regression and Lasso Regression. When $r = 0$, ElasticNet Regression becomes Ridge Regression, when $r = 1$, it is equivalent to Lasso Regression. Thus, r can be chosen based on the purpose.

3.3.3. Kernel-based methods

Kernel-based methods utilize kernels to handle nonlinear behavior in datasets. These methods can be investigated under three different types: Kernel Ridge, Gaussian Process (GP) Regressor and Support Vector Machines: SVM Regressor. Kernel Ridge can be considered a deterministic approach, indicating that it provides a single-point estimate for predictions without quantifying the uncertainty. On the other hand, GP Regressor is a probabilistic method that provides a mean prediction and measure of uncertainty (variance) for each prediction. Support Vector Machines (SVM), a kernel-based method, aim to find the optimal hyperplane for feature separation. In SVM regression, the goal is to fit a hyperplane that includes most data points within a set margin.

3.3.3.1. Support Vector Machines: SVM Regressor

In Support Vector Machines (SVM) [169], the aim is to find the optimal hyperplane that deviates from the actual values by at most defined margin, while also keeping the hyperplane as flat as possible. In SVM Regressor, the goal is to estimate the best hyperplane that fits the maximum number of data points within a specified margin.

Recall the model prediction as

$$\hat{y}_i = \theta_0 + \sum_{j=1}^{d}\theta_j x_{ij}. \tag{3.10}$$

As we seek the flattest hyperplane, weights must be as small as possible as defined in Eq. 3.10. To achieve this, the optimization problem can be constructed as

$$\min_{\theta_j}\left(\frac{1}{2}\sum_{j=1}^{d}\theta_j^2\right), \tag{3.11}$$

subject to

$$-\varepsilon \leq y_i - \hat{y}_i \leq \varepsilon. \tag{3.12}$$

Eq. 3.10 forces the weights to become smaller, so flatness condition, while Eq. 3.12 ensures that the predicted output quantity deviates from the actual values by at most defined margin ε (Fig. 3.10).

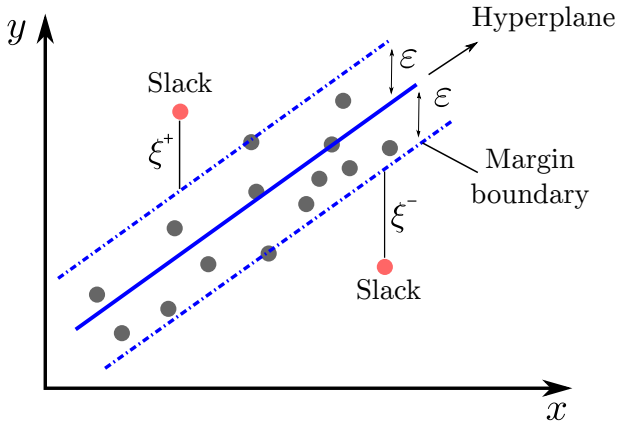


Figure 3.10.: An illustration for Support Vector Machine (SVM) Regressor.

It is not always feasible to find a function (Eq. 3.10) that approximates the unknown solution while simultaneously satisfying the constraints in Eqs. 3.11 and 3.12 for all data pairs (x_i, y_i). To address this limitation, slack variables (ξ) are introduced, allowing for certain violations of the margin. Specifically, ξ^+ corresponds to deviations above the ε-tube, whereas ξ^- corresponds to deviations below the ε-tube. In other words, some data points are allowed to lie outside the margin boundary. Incorporating slack variables yields the following cost function

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{j=1}^{d}\theta_j^2 + C\sum_{i=1}^{N_s}(\xi_i^+ + \xi_i^-), \tag{3.13}$$

subject to the following constraints

$$\begin{aligned} -\varepsilon - \xi_i^- \leq y_i - \hat{y}_i \leq \varepsilon + \xi_i^+, \\ \xi_i^- \geq 0, \quad \xi_i^+ \geq 0. \end{aligned} \tag{3.14}$$

Here, C is the regularization parameter to control the trade-off between model fitting the data closely (overfitting) and a simple model which tolerates slack variables utmost (underfitting). The N_s term denotes the number of slack variables.

As provided in Eq. 3.10, the approximated function is linear. To extend SVM Regressor for nonlinear regression problems, a kernel trick is used. A kernel function κ is defined as

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j),$$

where ϕ is a mapping or basis function. The kernel trick allows the computation of the dot product in the original input space. For a problem with N samples and d number of features, the kernel matrix is defined as

$$\boldsymbol{K} = \begin{bmatrix} \sum_{j=1}^{d}\phi(x_{1j})\phi(x_{1j}) & \sum_{j=1}^{d}\phi(x_{1j})\phi(x_{2j}) & \dots & \sum_{j=1}^{d}\phi(x_{1j})\phi(x_{Nj}) \\ \sum_{j=1}^{d}\phi(x_{2j})\phi(x_{1j}) & \sum_{j=1}^{d}\phi(x_{1j})\phi(x_{2j}) & \dots & \sum_{j=1}^{d}\phi(x_{2j})\phi(x_{Nj}) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{d}\phi(x_{Nj})\phi(x_{1j}) & \sum_{j=1}^{d}\phi(x_{Nj})\phi(x_{2j}) & \dots & \sum_{j=1}^{d}\phi(x_{Nj})\phi(x_{Nj}) \end{bmatrix}. \tag{3.15}$$

The kernel matrix can alternatively be expressed in terms of the vector representation of the mapping functions as

$$\boldsymbol{K} = \begin{bmatrix} \phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_1) & \phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_2) & \dots & \phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_N) \\ \phi(\boldsymbol{x}_2)^T\phi(\boldsymbol{x}_1) & \phi(\boldsymbol{x}_2)^T\phi(\boldsymbol{x}_2) & \dots & \phi(\boldsymbol{x}_2)^T\phi(\boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}_N)^T\phi(\boldsymbol{x}_1) & \phi(\boldsymbol{x}_N)^T\phi(\boldsymbol{x}_2) & \dots & \phi(\boldsymbol{x}_m)^T\phi(\boldsymbol{x}_N) \end{bmatrix}. \tag{3.16}$$

Note that kernel matrix $\boldsymbol{K}$ is symmetric. Common kernel functions are listed as follows:

- Linear kernel: $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$
- Polynomial kernel: $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + c)^d$
- Radial basis kernel (rbf): $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{2\sigma^2}}$
- Sigmoid kernel: $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\alpha \boldsymbol{x}_i \cdot \boldsymbol{x}_j + c)$

Here, c represents a constant in the polynomial kernel with the polynomial order d, and σ is the standard deviation. An example is given in Fig. 3.11 to show the effect of kernels in the SVM Regressor. Since input space consists of a single feature, the prediction of the SVM Regressor with the linear kernel shows almost an identical behavior to Linear Regressor models. Models with polynomial and sigmoid kernels don't capture the pattern in data. The regression model with the RBF kernel is the most promising one. In these models, the regularization parameter is set as $C = 1$ and the margin is set as $\varepsilon = 0.1$.

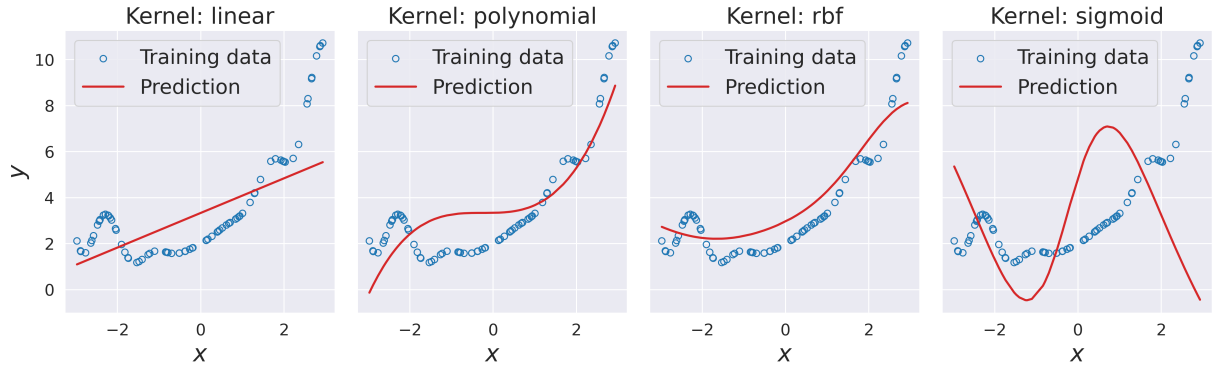


Figure 3.11.: Comparison of training data and predictions of SVM Regressor models based on different kernels.

3.3.3.2. Kernel Ridge Regressor

Kernel Ridge Regressor [171] extends Ridge Regression by incorporating the kernel trick. The cost function of Kernel Ridge is given as

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \sum_{i=1}^{N}\alpha_i\kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{3.17}$$

Here, α are the dual model parameters, κ denotes the kernel function, and λ is the regularization parameter. In contrast to Ridge Regression (Eq. 3.7), the prediction is expressed not directly as a function of the input x, but in terms of the kernel function κ, which defines the feature mapping. Eq. 3.17 can also be expressed in vector form as

$$J(\boldsymbol{\alpha}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^\top(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}) + \frac{\lambda}{2}\boldsymbol{\alpha}\boldsymbol{K}\boldsymbol{\alpha}. \tag{3.18}$$

The first term in Eq. 3.18 penalizes the mismatch between actual values and the kernelized predictions, while the second term represents the regularization to avoid the model learning the training data too closely. The optimal model parameters which minimizes the cost function can be expressed as

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}. \tag{3.19}$$

Here, $\boldsymbol{I}$ is the identity matrix. Optimal parameters are subsequently used to compute the predictions as follows

$$\hat{\boldsymbol{y}} = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*), \tag{3.20}$$

where κ is calculated based on the test points $\boldsymbol{x}_*$. A detailed pseudocode for the Kernel Ridge model is given in Algorithm 4. The algorithm takes as inputs a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the regularization parameter λ, kernel function κ, and unseen sample points $\boldsymbol{x}_*$. The goal is to estimate the model parameters α by minimizing the Kernel Ridge cost function (Eq. 3.18) and compute the predictions based on the optimal model parameters. Note that Kernel Ridge is equivalent to Ridge Regression (Section 3.3.2.2) when the kernel function is chosen as a linear kernel.

As mentioned earlier, there are different types of kernels. Fig. 3.12 shows a comparison of the training data and predicted outputs of Kernel Ridge models based on distinct kernels. Models with polynomial and RBF kernels show a good fit. In contrast, Kernel Ridge with linear and sigmoid kernels produces predictions that do not align with the training data at all. In these models, the regularization parameter is set as $\lambda = 1$.

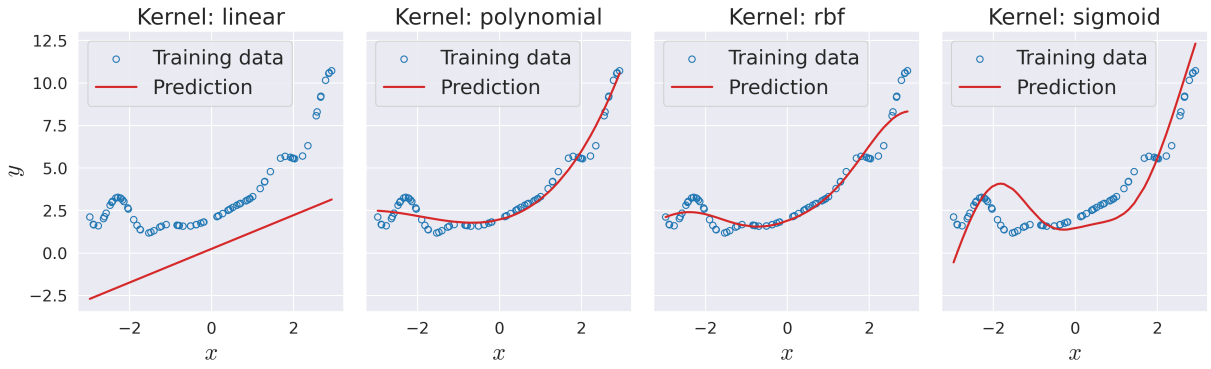


Figure 3.12.: Comparison of training data and predictions of Kernel Ridge models based on different kernels.

Algorithm 4 Kernel Ridge Regressor

Inputs:
I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
λ: regularization parameter
κ: kernel function
$\boldsymbol{x}_*$: unseen sample points
Output:
$\hat{\boldsymbol{y}}$: predicted output

procedure KERNELRIDGEREGRESSOR($I, \lambda, \kappa, \boldsymbol{x}_*$)
Get the number of samples N
for $i \in 1 : N$ **do**
for $j \in 1 : N$ **do**
$\boldsymbol{K}[i][j] = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ ▷ Get the kernel matrix
end for
end for
$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$ ▷ Get the optimal model parameters
$\hat{\boldsymbol{y}} = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*)$ ▷ Make the prediction
return $\hat{\boldsymbol{y}}$
end procedure

3.3.3.3. Gaussian Process Regressor

Gaussian Process (GP) Regressor [111, 138] is a strong Bayesian method that provides a probabilistic approach to estimate the predictions with quantifying uncertainty by the mean and the variance of the estimated quantity. The marginal likelihood (also called the log marginal likelihood or evidence) serves as the cost function in GP Regressor and is defined as

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}) = -\frac{1}{2}\boldsymbol{y}^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} - \frac{1}{2} \log \det(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}) - \frac{N}{2} \log 2\pi, \tag{3.21}$$

where $\boldsymbol{K}$ denotes the kernel (covariance) matrix, σ_n^2 represents the noise variance, $\boldsymbol{I}$ is the identity matrix. The first term ($\frac{1}{2}\boldsymbol{y}^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}$) of the GP Regressor cost function measures how well model predictions fit training data. The second term ($\frac{1}{2} \log \det(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})$) is the regularization term that penalizes model complexity based on the determinant of the model parameters. The third term ($\frac{N}{2} \log 2\pi$) is a constant used for normalization.

Similar to Kernel Ridge Regressor, the optimal parameter vector $\boldsymbol{\alpha}$ that minimizes this cost function can be expressed as

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}. \tag{3.22}$$

The predictive mean $\bar{f}(\boldsymbol{x}_*)$ at a new test input $\boldsymbol{x}_*$ is then given by

$$\bar{f}(\boldsymbol{x}_*) = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*), \tag{3.23}$$

and the corresponding predictive variance $\mathbb{V}[\bar{f}(\boldsymbol{x}_*)]$ is expressed as

$$\mathbb{V}[\bar{f}(\boldsymbol{x}_*)] = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) - \kappa(\boldsymbol{X}, \boldsymbol{x}_*)^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \kappa(\boldsymbol{X}, \boldsymbol{x}_*). \tag{3.24}$$

A detailed pseudocode for the GP Regressor is given in Algorithm 5. The algorithm takes as inputs a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, kernel function κ, the noise variance σ_n^2, unseen sample points $\boldsymbol{x}_*$. The goal is to estimate the model parameters α of the model by minimizing the GP Regressor cost function (Eq. 3.21) and compute the predictive mean, $\bar{\boldsymbol{f}}_*$, and the predictive variance, $\mathbb{V}[\bar{\boldsymbol{f}}_*]$, based on the optimal model parameters.

Algorithm 5 Gaussian Process (GP) Regressor

Inputs:

I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
κ: kernel function
σ_n^2: noise variance
$\boldsymbol{x}_*$: unseen sample points

Output:

$\bar{\boldsymbol{f}}_*$: predictive mean
$\mathbb{V}[\bar{\boldsymbol{f}}_*]$: predictive variance

procedure GPREGRESSOR($I, \kappa, \sigma_n^2, \boldsymbol{x}_*$)
  Get the number of samples N
  for $i \in 1 : N$ **do**
    for $j \in 1 : N$ **do**
      $\boldsymbol{K}[i][j] = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ ▷ Get the kernel matrix
    end for
  end for
  $\boldsymbol{\alpha} = (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}$ ▷ Get the optimal model parameters
  $\bar{\boldsymbol{f}}_* = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*)$ ▷ Get the predictive mean
  $\mathbb{V}[\bar{\boldsymbol{f}}_*] = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) - \kappa(\boldsymbol{X}, \boldsymbol{x}_*)^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \kappa(\boldsymbol{X}, \boldsymbol{x}_*)$ ▷ Get the predictive variance
  return $\bar{\boldsymbol{f}}_*, \mathbb{V}[\bar{\boldsymbol{f}}_*]$
end procedure

Fig. 3.13 illustrates a comparison between the training data and the predicted outputs of GP Regressor models, which vary based on different levels of noise variance σ_n^2. As σ_n^2 increases, the noise level rises, resulting in predictions that do not align well with the actual data. Making an analogy to Kernel Ridge, σ_n^2 serves a role similar to the regularization parameter λ in Kerne Ridge. In Kernel Ridge Regressor, λ controls the regularization strength to prevent overfitting. Similarly, σ_n^2 in Gaussian Process Regressor mitigates overfitting by accounting for the noise or uncertainty in the data.

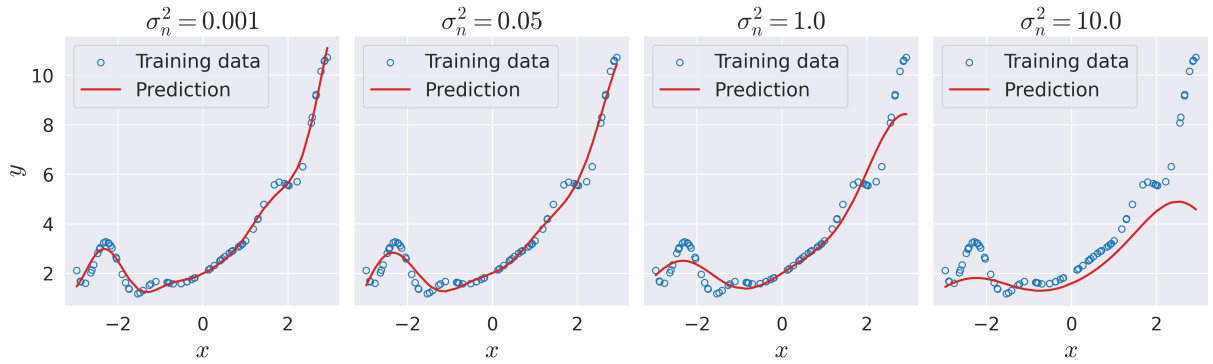


Figure 3.13.: Comparison of training data and predictions of GP Regressor models based on different noise variance values σ_n^2.

3.3.4. Tree-based models

Tree-based models are a type of nonparametric machine learning algorithm that partitions the feature space into distinct, non-overlapping regions with similar response values by applying a series of splitting rules. Tree-based models are simple to understand, interpret, and visualize.

The main components of tree-based models are given in Fig. 3.14. The top node represents the root node containing the entire training data. The root node splits into internal nodes depending on the split criterion. The common splitting criterion for regression problems is the mean squared error (mse), which measures the variance from the split. Internal nodes are also known as decision nodes. If the internal node does not have further splitting, then it is called a leaf node. Maximum depth (max depth) represents the maximum number of levels in a tree. A comprehensive discussion of the components of tree-based methods can be found in the work of Murphy [111].

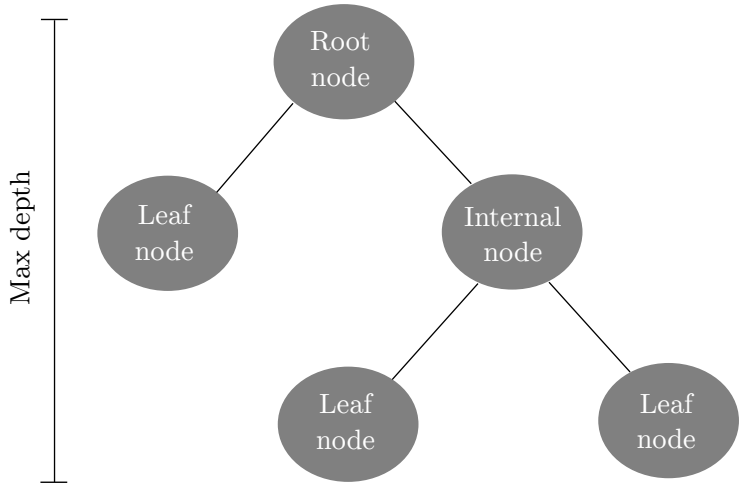


Figure 3.14.: A simple decision tree illustrating the components of tree-based models.

3.3.4.1. Decision Tree

Decision Tree [181] is the simplest tree-based method. It operates by repeatedly partitioning the data into smaller domains based on feature values, ultimately creating a tree structure where each leaf node represents a predicted value.

The Decision Tree algorithm begins at the root node and evaluates all possible splits for each feature. It chooses the split that minimizes the chosen splitting criterion of the target variable in the resulting subsets. The process is repeated recursively for each child node. Since this thesis focuses on regression problems, we will consider the mse as the splitting criterion.

The main algorithm of the Decision Tree for regression problems can be expressed as follows:

- **Start with the entire training dataset**: The algorithm begins at the root node, including all the data.
- **Identify the best splitting feature**: The main challenge of the Decision Tree is that the algorithm needs to find the best splitting feature based on the given splitting criterion or the error metric, which is the mse. Thus, for each possible feature and corresponding split point, the algorithm calculates the error metric and it selects the feature including the split point, minimizing the prediction error.
- **Breeding the child nodes**: Now the selected feature is used to split the dataset into the two child nodes as shown in Fig. 3.14. Each child node represents a subset of the data.
- **Recursive splitting**: The process is repeated for each child node, with further splitting of the data to create more partitions so that the algorithm can identify the hidden pattern in the dataset.
- **Apply stopping criteria**: Splitting continues until a predefined stopping criterion is met, such as maximum depth or a minimum number of samples in a node.
- **Leaf node predictions**: Once splitting is done, each leaf node is assigned a final prediction, which is typically the average of the target values within that leaf node.

Fig. 3.15 gives a comparison between the training data and the predicted outputs of Decision Tree models that are trained using different maximum depths. Increasing maximum depth leads to more accurate predictions that capture the underlying pattern in the data. For instance, when the max depth is set to 1, the root node generates only two child nodes, delivering only two different predictions, which is highly underfitting the dataset. On the other hand, when the max is set to 8, the algorithm might create $2^8 = 256$ child nodes, which can cause overfitting.

Figure 3.15 illustrates a comparison between the training data and the predicted outputs of Decision Tree models trained with varying maximum depths.

As the maximum depth increases, the model becomes capable of capturing more complex patterns in the data. For instance, with a maximum depth of 1, the tree consists of only a root node and two child nodes, resulting in just two distinct predictions. This simplicity causes the model to underfit the data, failing to represent the underlying trends properly.

On the other hand, when the maximum depth is set to 8, the tree can produce up to $2^8 = 256$ leaf nodes, enabling it to learn highly detailed patterns. While this can improve accuracy on the training data, as shown in Fig. 3.15, it can lead to overfitting.

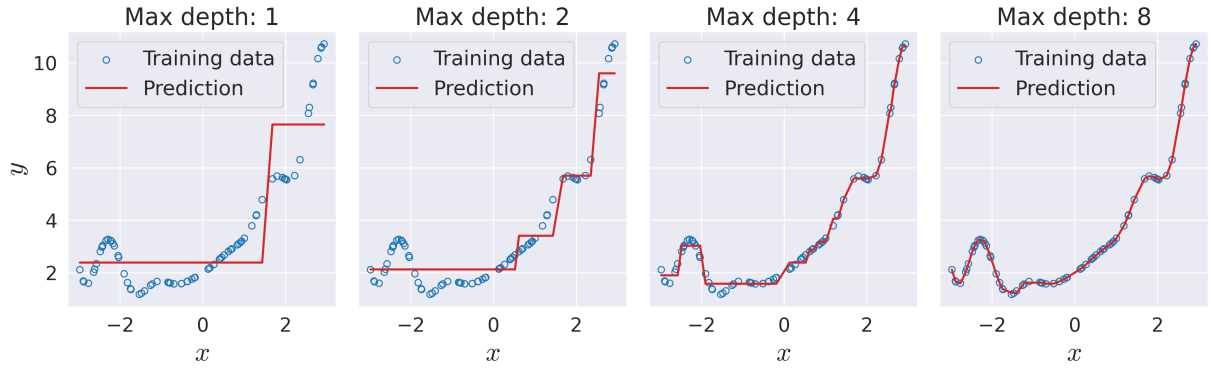


Figure 3.15.: Comparison of the predictions of the Decision Tree models trained with different maximum depths.

The main benefit of the Decision Trees is that they are easy to understand and interpret as they are decision-making approaches. Also, they provide insights about feature importance, enabling to identify the features contributing most to predictions. However, Decision Trees are prone to overfitting and they are sensitive to outliers. Additionally, identifying the best splitting feature can be computationally expensive for large datasets.

3.3.4.2. Random Forest

Random Forest [33] is an ensembling method which combines multiple decision trees. The main steps of the Random Forest algorithm can be expressed as follows (see Fig. 3.16):

- **Start with the entire training dataset**: The algorithm begins including all the dataset.
- **Bootstrap sampling**: The dataset is randomly sampled with bootstraps to create multiple subsets. Each subset is slightly different from the original dataset, ensuring diversity among the trees. This step enables that each tree extracts a different feature in an artificial way.
- **Decision Trees**: Each bootstrap sample is used to train an independent decision tree to extract features based on the random bootstrap samples. Additionally, each tree outputs an individual prediction.
- **Ensembling of Trees**: The individual predictions from the trees are combined into an ensemble prediction.
- **Majority voting (averaging)**: The final prediction is determined by majority voting for classification tasks or averaging for regression tasks.

Fig. 3.17 illustrates the effect of the number of estimators (trees) and the maximum depth of the trees on the performance of a Random Forest model in a regression task. The plots are organized in a grid, with rows representing different maximum depths of the trees and columns representing the number of estimators (1, 5, 20, and 100). In the top row, where the maximum depth is set to 1, the trees are shallow and can only produce piecewise constant predictions, resulting in a step-like appearance. Increasing the number of estimators improves the model's performance

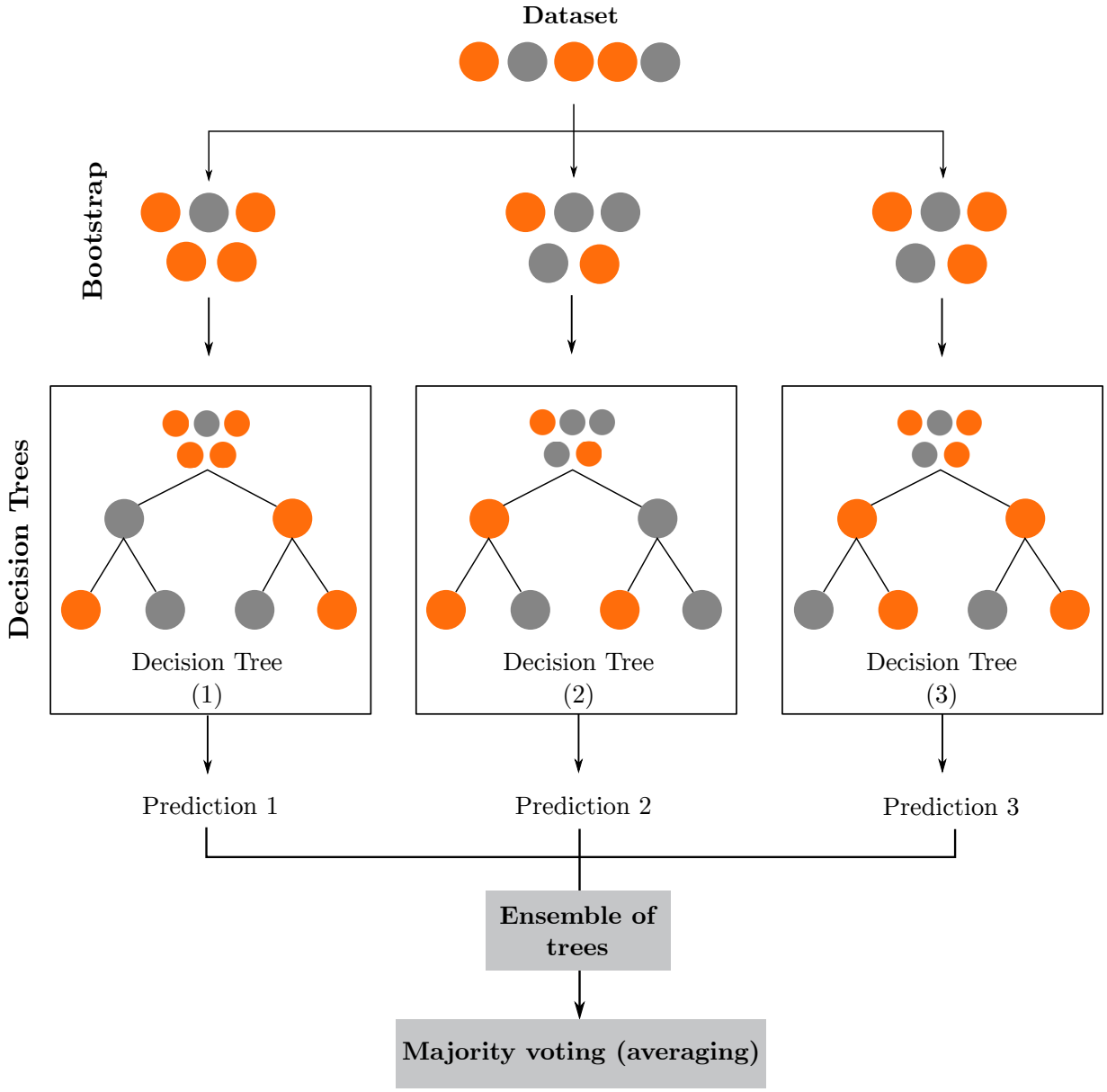


Figure 3.16.: Workflow of the Random Forest algorithm.

slightly, but the shallow depth limits its ability to capture complex relationships in the data. In the bottom row, with a maximum depth of 5, the trees can model more complex patterns. As the number of estimators increases, the predictions become smoother and more aligned with the true data distribution, demonstrating reduced variance due to the ensemble averaging effect. However, adding more trees does not improve the accuracy of the model since the training data consists of a single input feature. Thus, better accuracy can be achieved by adding more depths to individual Decision Trees. To conclude, deeper trees (with greater depth) provide more flexibility to capture complex patterns, and increasing the number of trees enhances the model's generalization by reducing variance.

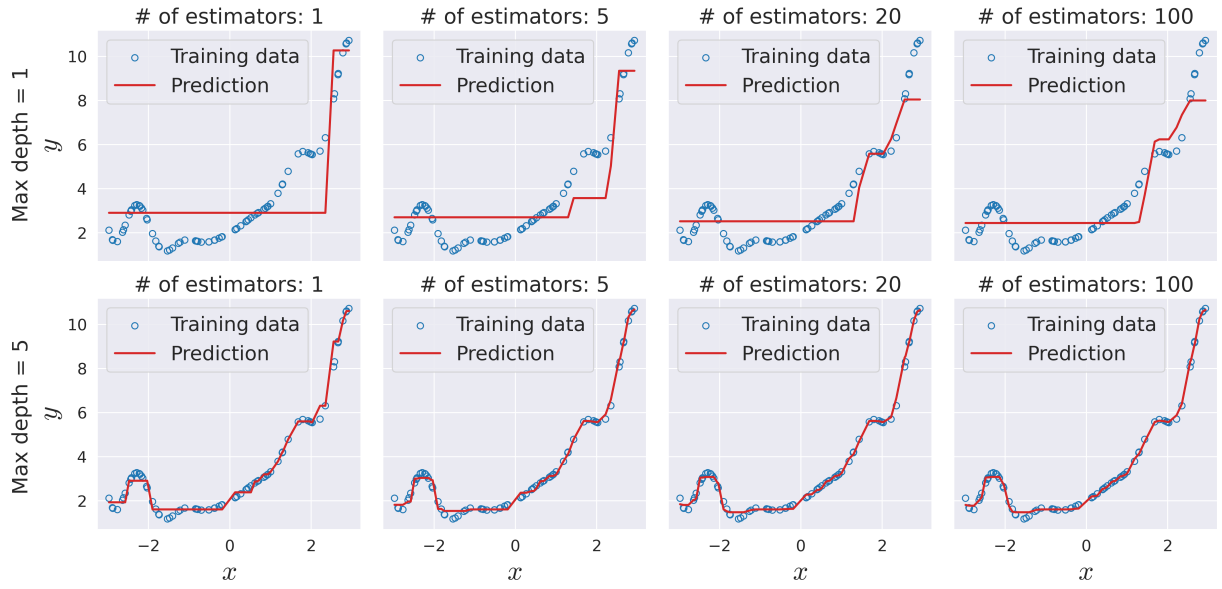


Figure 3.17.: Comparison of the predictions of the Random Forest models trained with different maximum depths and number of Decision Trees (# of estimators).

3.3.4.3. AdaBoost

AdaBoost [55], short for Adaptive Boosting, is a boosting ensemble approach which combines multiple weak learners in a sequential way to get a powerful "committee" [72]. A weak learner is a model whose accuracy is only slightly higher than that of random guessing. The idea behind AdaBoost is to build multiple sequential models on the subset of the dataset and each subsequent model learns from the previous model's errors. Moreover, since Decision Trees are usually considered as weak learners, the AdaBoost algorithm is explained under tree-based models (Section 3.3.4). However, any machine learning model can be used as long as it is a weak learner.

A detailed pseudocode for the AdaBoost algorithm for regression tasks is presented in Algorithm 6, which has been adapted from [72]. The algorithm takes as inputs a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, a set of M weak learners $\{G_1, G_2, \ldots, G_M\}$ and unseen sample points $\boldsymbol{x}_*$. Initially, equal weights are assigned each data point in the training set, ensuring that all samples are equally important at the beginning. The algorithm iteratively trains weak learners to adjust the weights of the data points after each iteration. For every weak learner, the total

weighted error is calculated, emphasizing the importance of samples that were poorly predicted. A contribution weight (α_m) is then computed for the learner, based on its performance, which determines its influence in the final prediction. The weights of the data points are updated to prioritize those with higher prediction errors, ensuring subsequent learners focus more on high error cases. In this pseudocode, an exponential approach is employed. Next, updated weights are normalized and they are now used to create a new dataset, where each sample's weight defines a range (bin). Here, the algorithm selects n_{rnd} random numbers from $[0, 1]$, and for each, the sample whose bin it falls into is selected. Misclassified samples, having higher weights, occupy larger bins, making them more likely to be selected. The new dataset has the same number of samples as the original, but misclassified samples may appear multiple times. At the end of all iterations, the final prediction for unseen data is obtained by combining the predictions of all weak learners, weighted by their contributions (α_m).

Algorithm 6 AdaBoost for Regression Tasks

Inputs:

I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$

G: a set of M weak learners $\{G_1, G_2, \ldots, G_M\}$

$\boldsymbol{x}_*$: unseen sample points

Output:

$\hat{\boldsymbol{y}}$: predicted output

procedure AdaBoost($I, G, \boldsymbol{x}_*$)

$w_i = \frac{1}{N} \rightarrow$ for $i = 1, 2, \ldots, N$ ▷ Initialize the observation weights

for $m \in 1 : M$ **do**

(a) Train a weak learner G_m using the current weights w_i

(b) $\text{err}_m = \dfrac{\sum_{i=1}^{N} w_i |y_i - G_m(\boldsymbol{x}_i)|}{\sum_{i=1}^{N} w_i}$ ▷ Compute the weighted error

(c) $\alpha_m = \frac{1}{2} \ln(\frac{1 - \text{err}_m}{\text{err}_m})$ ▷ Compute the learner contribution α

(d) $w_i \leftarrow w_i \exp(\alpha_m |y_i - G_m(\boldsymbol{x}_i)|)$ ▷ Update the weights

(e) $w_i \leftarrow \dfrac{w_i}{\sum_{i=1}^{N} w_i}$ ▷ Normalize the weights

(f) Create a new weighted dataset $\{\boldsymbol{x}_i, y_i\}$ based on the updated weights.

end for

$\hat{\boldsymbol{y}} = \sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x}_*)$ ▷ Get the prediction

return $\hat{\boldsymbol{y}}$

end procedure

Fig. 3.18 demonstrates the effect of the number of weak learners (M) on the performance of AdaBoost for a regression task. In the left figure ($M = 1$), the model is clearly underfitting, since a single weak learner cannot capture the non-linear relationship in the data. In the middle figure ($M = 5$), the model begins to approximate the data better, as the additional weak learners allow the ensemble to capture more complexity. Finally, in the right figure ($M = 50$), the model closely fits the data, indicating that with sufficient weak learners, the AdaBoost algorithm can effectively model the underlying structure of the data.

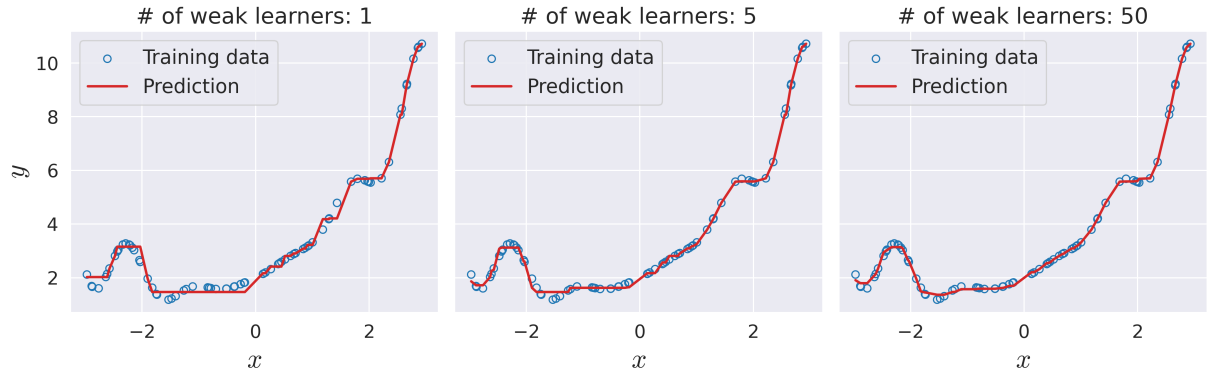


Figure 3.18.: Comparison of the predictions of the AdaBoost models trained with different number of weak learners (Decision Trees).

3.3.4.4. Gradient Boost

Gradient Boost [56] is another ensemble boosting algorithm where the main idea is inspired from the gradient descent algorithm. Compared to AdaBoost, which adjusts the observation weights of training samples based on the computed weighted errors, Gradient Boost minimizes a predefined loss function, such as mean squared error, by sequentially adding weak learners to the ensemble. Each weak learner, a decision tree, is trained to correct the errors (pseudo residuals) of the previous model by fitting to the negative gradient of the loss function at each step. Thus, compared to AdaBoost and many other classical machine learning algorithm, the intermediate predictions of the learning algorithm are not the target variables but the pseudo residuals or the errors.

A detailed pseudocode for the Gradient Boost algorithm for regression tasks is given in Algorithm 7, which has been similarly adapted from [72]. Initially, the model $G_0(\boldsymbol{x})$ is initialized to a constant value that minimizes the chosen loss function J across all training samples. When the mean squared error is chosen as the loss function, then this constant is equal to average of the actual target variables. Next, we iterate a loop for a given number of M weak learners. In each iteration m, the algorithm computes the pseudo residuals r_{im}, which represent the negative gradient of the loss function with respect to the predictions of the previous model G_{m-1}. A weak learner is then fitted to these pseudo residuals, partitioning the input space into terminal regions R_{km}. This means that we consider the leaves of the weak learner decision tree as terminal regions. For each region, the algorithm determines the optimal leaf output γ_{km} by minimizing the loss. Note that this summation at step (c) takes into account the previous prediction $G_{m-1}(\boldsymbol{x})$ and it runs over samples lying on the corresponding leaf. Similarly, if the mean squared error is the

chosen loss function, then the γ value that minimizes the loss function will be the average of the outputs of the corresponding leaf. Next, the model is updated by adding the predictions of the weak learner, scaled by a learning rate η, to the current ensemble. Note that the learning rate is used to avoid overfitting. Finally, the ensemble $G_m(\boldsymbol{x})$ is used to predict the output for unseen data points.

Algorithm 7 Gradient Boost for Regression Tasks

Inputs:

I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
G: a set of M weak learners $\{G_1, G_2, \ldots, G_M\}$
J: a differentiable regression loss function $J(y_i, G(\boldsymbol{x}_i))$
$\boldsymbol{x}_*$: unseen sample points

Output:

$\hat{\boldsymbol{y}}$: predicted output

procedure GRADIENTBOOST($I, G, J, \boldsymbol{x}_*$)

$G_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{i=1}^{N} J(y_i, \gamma)$ ▷ Initialize the model with a constant value

for $m \in 1 : M$ **do**

(a) For $i = 1, 2, \ldots, N$, compute the pseudo residuals as

$$r_{im} = -\left[\frac{\partial J(y_i, G(\boldsymbol{x}_i))}{\partial G(\boldsymbol{x}_i)}\right]_{G(\boldsymbol{x}_i)=G_{m-1}(\boldsymbol{x}_i)}$$

(b) Fit a regression tree to the r_{im} giving terminal regions R_{km}, for $k = 1, 2, \ldots, K_m$.

(c) For $k = 1, 2, \ldots, K_m$, compute the optimal leaf outputs

$$\gamma_{km} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{km}} J(y_i, G_{m-1}(\boldsymbol{x}_i) + \gamma)$$

(d) Update $G_m(\boldsymbol{x}_i) = G_{m-1}(\boldsymbol{x}_i) + \eta \sum_{k=1}^{K_m} \gamma_{km} \boldsymbol{I}(\boldsymbol{x} \in R_{km})$

end for

$\hat{\boldsymbol{y}} = G_M(\boldsymbol{x}_*)$ ▷ Get the prediction

return $\hat{\boldsymbol{y}}$

end procedure

An illustrative example of the Gradient Boost algorithm is given in Fig. 3.19, which compares the predictions of a regression model trained with varying numbers of weak learners: 1, 5, and 50. With only a single weak learner, the model underfits and fails to capture the underlying pattern in the data. As the number of weak learners increases to 5, the model starts to capture more patterns in the data and the model accuracy increases. With 50 weak learners, the model reaches a certain level of model complexity and it closely fits the training data.

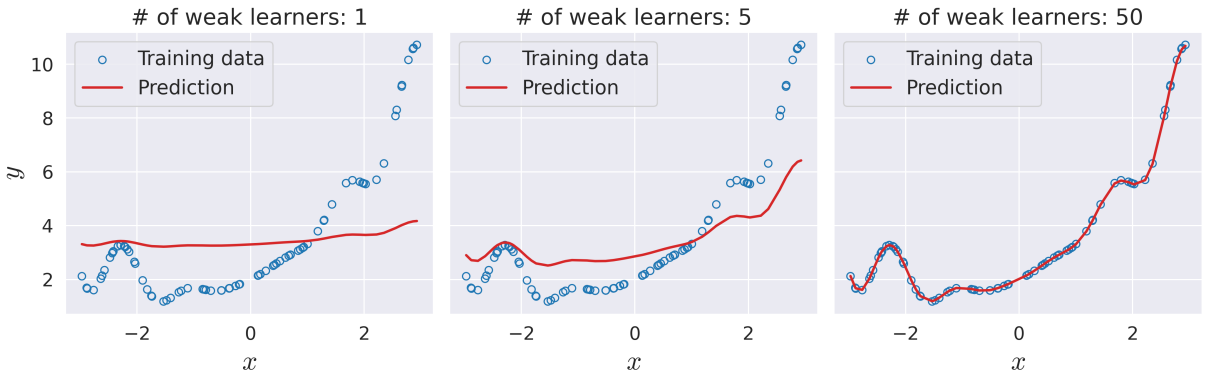


Figure 3.19.: Comparison of the predictions of the Gradient Boost models trained using different numbers of weak learners.

3.3.4.5. XGBoost

XGBoost [40] is the extended version of Gradient Boosting, designed to enhance both efficiency and performance by introducing advanced features. It incorporates regularization techniques, such as L_1 and L_2 regularizations, directly into the objective function to control the complexity of the model and prevent overfitting [40]. Unlike traditional Gradient Boosting, XGBoost leverages second-order optimization, using both first-order gradients and second-order derivatives (Hessians) to make more precise updates during the learning process [56]. Additionally, XGBoost implements features such as column sampling, weighted quantile sketching for handling sparse data, and parallel processing to speed up training [40]. These optimizations make XGBoost highly efficient, scalable, and accurate, particularly for large datasets and complex machine learning problems.

A detailed pseudocode for the XGBoost algorithm for regression tasks is given in Algorithm 8. The algorithm starts by initializing the model with a constant value that minimizes the loss function across all training samples. For each weak learner, the algorithm calculates the first-order gradient (g_{im}) and second-order gradient (h_{im}) of the loss function for each training sample. These measures are used to select splits that maximize gain. In this pseudocode, we omit the details of the splitting strategy and refer to Choi [43] for further reading. Similar to Gradient Boost, a regression tree is then fitted to predict these gradients, generating terminal regions (R_{km}) such as leaves. For each region, the optimal leaf output, (γ_{km}), is computed by minimizing a regularized objective function that incorporates the first- and second-order gradients with a penalty term. Finally, the model is updated by adding the contributions of the new tree, scaled by a learning rate (η). This iterative process continues until all m weak learners are trained. Finally, the ensemble of weak learners is used to make predictions for unseen data ($\boldsymbol{x}_*$).

An illustrative example of the XGBoost algorithm is given in Fig. 3.20. Similar to the Gradient Boost example, the figure compares the predictions of a regression model trained with varying numbers of boosting trees: 1, 5, and 50. Note that the terms *"boosting tree"* and *"weak learner"* refer to the same concept. The left figure ($M = 1$) illustrates underfitting, where a single weak learner fails to capture the underlying pattern in the data, resulting in an overly simplistic prediction. In the middle figure, with 5 boosting trees, the model begins to approximate the data more effectively, as the additional weak learners allow the model to capture more complexity.

Algorithm 8 XGBoost for Regression Tasks

Inputs:

I: a set of N training data pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
G: a set of M weak learners $\{G_1, G_2, \ldots, G_M\}$
J: a differentiable regression loss function $J(y_i, G(\boldsymbol{x}_i))$
λ: regularization term controlling complexity of trees
$\boldsymbol{x}_*$: unseen sample points

Output:

$\hat{\boldsymbol{y}}$: predicted output

procedure XGBOOST($I, G, J, \lambda, \boldsymbol{x}_*$)

$G_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{i=1}^{N} J(y_i, \gamma)$ ▷ Initialize the model with a constant value

for $m \in 1 : M$ **do**

(a) For $i = 1, 2, \ldots, N$, compute

$$g_{im} = \left[\frac{\partial J(y_i, G(\boldsymbol{x}_i))}{\partial G(\boldsymbol{x}_i)}\right]_{G(\boldsymbol{x}_i)=G_{m-1}(\boldsymbol{x}_i)}, \quad h_{im} = \left[\frac{\partial^2 J(y_i, G(\boldsymbol{x}_i))}{\partial G(\boldsymbol{x}_i)^2}\right]_{G(\boldsymbol{x}_i)=G_{m-1}(\boldsymbol{x}_i)}$$

(b) Fit a regression tree to the g_{im} residual values and create terminal regions R_{km}, for $k = 1, 2, \ldots, K_m$.

(c) Compute the optimal leaf outputs

$$\gamma_{km} = \arg\min_{\gamma} \left\{ \sum_{\boldsymbol{x}_i \in R_{km}} \left(g_{im}\gamma + \frac{1}{2} h_{im}\gamma^2 \right) + \frac{1}{2}\lambda\gamma \right\}$$

(d) Update the model:

$$G_m(\boldsymbol{x}_i) = G_{m-1}(\boldsymbol{x}_i) + \eta \sum_{k=1}^{J_m} \gamma_{km} \boldsymbol{I}(\boldsymbol{x} \in R_{km})$$

end for

$\hat{\boldsymbol{y}} = G_M(\boldsymbol{x}_*)$ ▷ Get the final prediction

return $\hat{\boldsymbol{y}}$

end procedure

Finally, when M is set to 50, the model achieves a good fit, with the prediction line closely following the training data, indicating that the ensemble has successfully learned the underlying structure of the data. Compared to the Gradient Boost example shown in Fig. 3.19, for the same number of boosting trees, the XGBoost model demonstrates slightly better results.

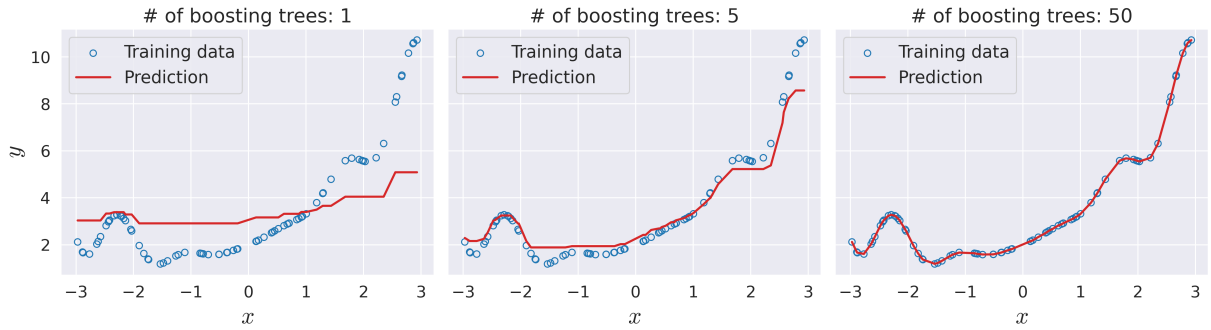


Figure 3.20.: Comparison of the predictions of the XGBoost models trained using different numbers of boosting trees.

3.4. Neural networks

Neural networks are a subclass of deep learning inspired by the structure and function of the human brain. Essentially, they can be considered an extended version of linear regression by incorporating non-linear activation functions and multiple interconnected layers, which enable them to capture complex patterns within data. Each layer comprises neurons that transform the input data through weighted connections, allowing the network to learn relationships that go beyond the capabilities of simple linear models.

Fig. 3.21 presents the key components of a neural network. The first (input) layer receives the input data, $\{z_1, z_2, \ldots, z_d\}$, with each neuron corresponding to one of the known input features. Here, d represents the input dimension. Hidden layers, located between the input and output layers, extract features and learn underlying patterns; these layers may include multiple sub-layers and neurons. Each neuron is associated with learnable parameters-namely, weights (w) and biases (b)-which are optimized during training. Similar to linear regression, these weights and biases form the network parameters, denoted by $\boldsymbol{\theta}$. Moreover, each neuron applies an activation function (ψ) to introduce non-linearity into the model. Finally, the output layer delivers predictions, denoted by $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_t\}$, where t denotes the output dimension.

The network output is computed through a series of recursive, element-wise operations that propagate the input through the hidden layers. Specifically, the computation is performed as follows:

$$\begin{aligned} \text{input layer} &\rightarrow \mathcal{N}^1(\boldsymbol{z};\boldsymbol{\theta}) = \boldsymbol{z}, \\ \text{hidden layers} &\rightarrow \mathcal{N}^l(\boldsymbol{z};\boldsymbol{\theta}) = \psi(\boldsymbol{\theta}^l \cdot \mathcal{N}^{l-1}(\boldsymbol{z};\boldsymbol{\theta})), \text{ for } 2 \leqslant l \leqslant L-1, \\ \text{output layer} &\rightarrow \hat{\boldsymbol{y}} = \psi(\boldsymbol{\theta}^L \cdot \mathcal{N}^{L-1}(\boldsymbol{z};\boldsymbol{\theta})), \end{aligned} \tag{3.25}$$

In this formulation, the input layer simply passes the raw input data, $\boldsymbol{z}$, to the network. For each hidden layer (from $l = 2$ to $L - 1$), the output is computed by applying an activation function to

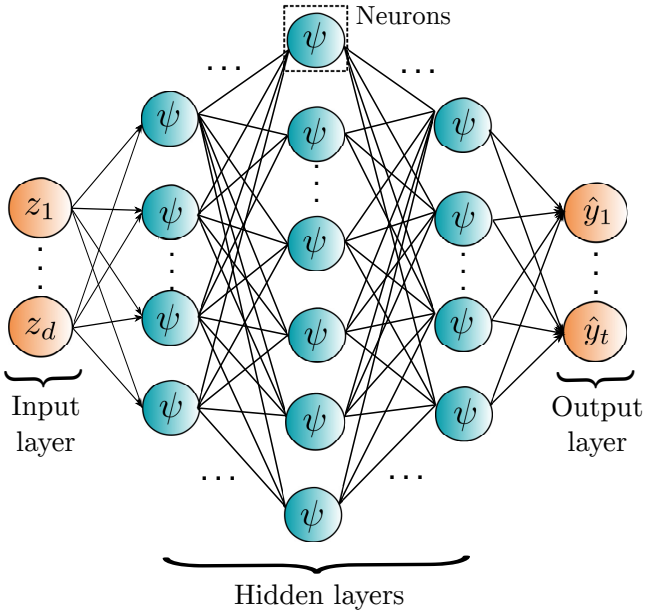


Figure 3.21.: An illustration to show the components of a neural network.

the weighted sum of the previous layer's outputs, where $\boldsymbol{\theta}^l$ encapsulates the parameters (weights and biases) of layer l. The final output layer applies its activation function to produce the network prediction, $\hat{\boldsymbol{y}}$. Here, L denotes the total number of layers (with the output layer counted as one of the hidden layers), and $\mathcal{N}^l(\boldsymbol{z};\boldsymbol{\theta})$ represents the output of the l-th layer. Note that no activation function is applied in the input layer; nonlinear activation functions are used exclusively in the hidden and output layers.

As an example, the computation at an arbitrary neuron of a hidden layer is presented in Fig. 3.22. First, the input layer is multiplied by the weights and the bias is added to form the weighted sum in the next layer expressed as

$$z = b + z_1 w_1 + z_2 w_2 + z_3 w_3. \tag{3.26}$$

Then, a chosen activation function is applied to introduce nonlinearity such as

$$a = \psi(z). \tag{3.27}$$

This calculation can be derived using the cumulative approach, as shown in Fig. 3.22a. Alternatively, it can be formulated in vector form, which requires the artificial addition of a bias neuron. In the vectorized version, the network parameters for layer l are defined as

$$\boldsymbol{\theta} = [b \quad w_1 \quad \dots \quad w_{n'}],$$

and the augmented input vector is given by

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ z_1 \\ \vdots \\ z_{n'} \end{bmatrix},$$

where n' denotes the number of neurons in the previous layer $(l-1)$, as illustrated in Fig. 3.22b. Note that the product $\{\boldsymbol{\theta z}\}$ can be decomposed into $\{\boldsymbol{W z}+\boldsymbol{b}\}$. This vectorized formulation simplifies the computation by integrating the weights and bias into a single operation, and will be used in the following section. Note that in this decomposed form, the input vector $\boldsymbol{z}$ omits the constant term 1.

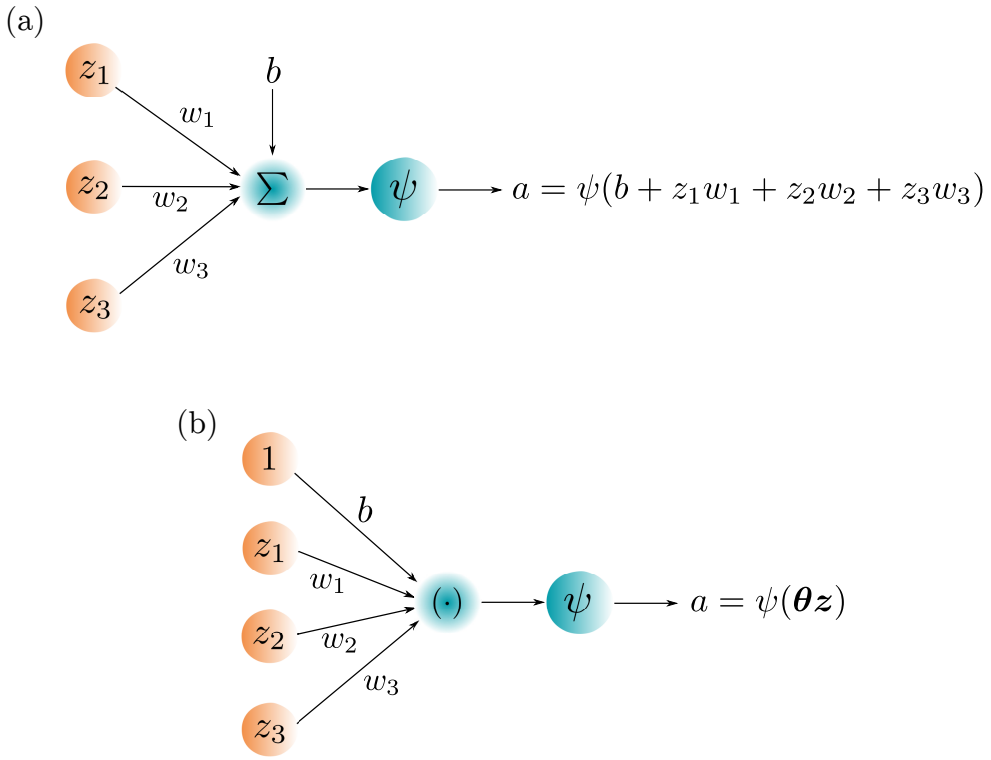


Figure 3.22.: Computation at a hidden neuron shown in two approaches. (a) The cumulative method computes the weighted sum of inputs plus a bias and applies an activation function. (b) The vectorized method performs the same calculation using augmented input vectors.

3.4.1. Forward propagation and backpropagation

Forward propagation is the way input data propagates from the input layer to the output layer through neurons and hidden layers. The main goal of forward propagation is to obtain the network prediction so to calculate the loss function. On the other hand, backpropagation is the process of moving from the output layer to the input layer to obtain derivatives of the loss function with respect to network parameters.

To illustrate the computation scheme of forward propagation and backpropagation, consider the simple architecture shown in Fig. 3.23. The provided network has two inputs z_1 and z_2, one hidden layer containing two neurons and one output quantity $\hat{y}$. The activation function ψ is chosen as $\psi(\star) = e^{\star}$.

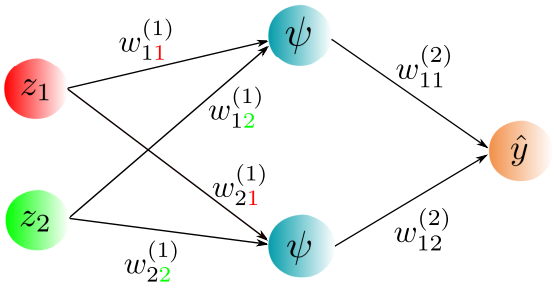


Figure 3.23.: Schematic of a simple neural network architecture, including weight and bias terms.

The input vector $\boldsymbol{z}$ is defined as

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \tag{3.28}$$

and the weights for the first layer are given by

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, \tag{3.29}$$

while the weights for the second layer are defined as

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.6 \end{bmatrix}. \tag{3.30}$$

The corresponding bias terms are specified as

$$b_1^{(1)} = 0.1, \quad b_2^{(1)} = 0.2, \quad b_1^{(2)} = 0.3. \tag{3.31}$$

Here, the first lower indices represent the neuron number and the second indices denote the input dimension. For instance, the term $w_{11}^{(1)}$ denotes the weight of the top (first) neuron which is coming from the input z_1 as depicted in Fig. 3.23. Additionally, the superscript indicates the hidden layer in which the computation takes place; for example, the (1) in $w_{11}^{(1)}$ signifies that the weight associated with the first hidden layer.

By applying Eq. 3.26 to obtain calculations at each neuron, we get the weighted sum $\boldsymbol{z}^{(1)}$ as

$$\begin{aligned} z_1^{(1)} &= b_1^{(1)} + z_1 w_{11}^{(1)} + z_2 w_{12}^{(1)} = 0.1 + 1 \cdot 0.1 + 2 \cdot 0.2 = 0.6 \\ z_2^{(1)} &= b_2^{(1)} + z_1 w_{21}^{(1)} + z_2 w_{22}^{(1)} = 0.2 + 1 \cdot 0.3 + 2 \cdot 0.4 = 1.3 \end{aligned} \tag{3.32}$$

Also, the weighted sum can be obtained in a compact matrix-vector notation such as

$$\begin{aligned} \boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{z} + \boldsymbol{b}^{(1)} &= \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \\ &= \begin{bmatrix} 0.6 \\ 1.3 \end{bmatrix}. \end{aligned}$$

Next, the activation function ψ is applied for each neuron to get output vector of hidden layer as

$$\boldsymbol{a}^{(1)} = \psi(\boldsymbol{z}^{(1)}) = \begin{bmatrix} e^{0.6} \\ e^{1.3} \end{bmatrix} = \begin{bmatrix} 1.82 \\ 3.67 \end{bmatrix}. \tag{3.33}$$

The resulting vector $\boldsymbol{a}^{(1)}$ at the hidden layer becomes now the input for next layer. Following the same way, we can get the weighted sum in vector form for the output layer as

$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)} = [0.5 \quad 0.6] \begin{bmatrix} 1.82 \\ 3.67 \end{bmatrix} + 0.3 = 3.41 \tag{3.34}$$

Similarly, by applying the activation function, we obtain the neural network prediction as

$$\hat{y} = \boldsymbol{a}^{(2)} = \psi(\boldsymbol{z}^{(2)}) = e^{3.41} = 30.27 \tag{3.35}$$

A generalized computation scheme to obtain outputs at a random neuron j lying in the layer l can be expressed as

$$a_j^{(l)} = \psi(\boldsymbol{z}_j^{(l)}) = \psi\left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right). \tag{3.36}$$

Additionally, this expression can be reformulated in a compact matrix-vector notation as

$$\boldsymbol{a}^{(l)} = \psi(\boldsymbol{z}^{(l)}) = \psi(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}). \tag{3.37}$$

So far, the input data has been propagated from the input layer to the output layer to generate the network prediction (i.e., through forward propagation). Before exploring backpropagation, a loss function must be defined. Similar to linear regression, the mean squared error (MSE) in its vector form is selected as the neural network cost function, and it is expressed as follows

$$L(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{2N}(y - \hat{y})^\top (y - \hat{y}). \tag{3.38}$$

Here, y denotes the actual or known output quantity. Assume that a single sample is used, delivering $N = 1$. Now, we will apply chain rule to obtain derivatives of the cost function with respect to network parameters including weights and biases. Let us start with generalization of the partial derivative of L with respect to weights at layer l, called as $\boldsymbol{W}^{(l)}$. It can be expressed using the chain rule as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(l)}} \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{W}^{(l)}}. \tag{3.39}$$

For simplification, we redefine the first component of the partial derivative as

$$\delta^{(l)} = \frac{\partial L}{\partial \boldsymbol{z}^{(l)}}, \tag{3.40}$$

which is also known as the error term. Since the resulting vector $\boldsymbol{z}$ is a direct function of corresponding network parameters, it can be easily further simplified such as

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial (\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{a}^{(l-1)^\top}. \tag{3.41}$$

Note that $\boldsymbol{a}^{(l-1)}$ has been transposed for the dimension consistency. Similarly, the partial derivative of L with respect to weights at layer l, called as $\boldsymbol{b}^{(l)}$, can be obtained as

$$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(l)}} \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}}, \tag{3.42}$$

where

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial (\boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})}{\partial \boldsymbol{b}^{(l)}} = 1. \tag{3.43}$$

The challenging part when it comes to simplifying the error term δ (Eq. 3.40). Let us first simplify the error term for the output layer by applying the chain rule

$$\delta^{(2)} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} \tag{3.44}$$

$$= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{z}^{(2)}}, \tag{3.45}$$

where

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y} - y), \tag{3.46}$$

and

$$\frac{\partial \hat{y}}{\partial \boldsymbol{z}^{(2)}} = \frac{\partial \psi(\boldsymbol{z}^{(2)})}{\partial \boldsymbol{z}^{(2)}} = e^{\psi(\boldsymbol{z}^{(2)})} = \hat{y}. \tag{3.47}$$

Recalling the Eqs. 3.39 and 3.42, we define the partial derivative of L with respect to network parameters at the output layer as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{W}^{(2)}} = \delta^{(2)} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{W}^{(2)}}, \tag{3.48}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{b}^{(2)}} = \delta^{(2)}. \tag{3.49}$$

Inserting Eqs. 3.44, 3.41, 3.43 into Eqs. 3.48 and 3.49 leads to the final forms of the partial derivatives with respect to weights and biases at the output layer as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = (\hat{y} - y) \hat{y} {\boldsymbol{a}^{(1)}}^{\top}, \tag{3.50}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = (\hat{y} - y) \hat{y}. \tag{3.51}$$

For our example, assume that $y = 1$. By substituting the prediction $\hat{y}$ from Eq. 3.35 and the output vector $\boldsymbol{a}^{(1)}$ from Eq. 3.33, the partial derivatives can be computed as follows

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = (30.27 - 1) \cdot 30.27 \cdot \begin{bmatrix} 1.82 & 3.67 \end{bmatrix} = \begin{bmatrix} 1612.52 & 3251.62 \end{bmatrix}, \tag{3.52}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = (30.27 - 1) \cdot 30.27 = 886.00. \tag{3.53}$$

Let us continue to the next layer. Similar to the previous step, we calculate the partial derivative of the cost function with respect to weights $\boldsymbol{W}^{(1)}$ as follows

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(1)}} \frac{\partial \boldsymbol{z}^{(1)}}{\partial \boldsymbol{W}^{(1)}} = \delta^{(1)} {\boldsymbol{a}^{(0)}}^{\top}, \tag{3.54}$$

where $\boldsymbol{a}^{(0)}$ denotes $\boldsymbol{z}$. The error term $\delta^{(1)}$ can be defined using the chain rule as

$$\delta^{(1)} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(1)}} \odot \frac{\partial \boldsymbol{a}^{(1)}}{\partial \boldsymbol{z}^{(1)}} = \delta^{(2)} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(1)}} \odot \frac{\partial \boldsymbol{a}^{(1)}}{\partial \boldsymbol{z}^{(1)}}, \tag{3.55}$$

using the error term $\delta^{(2)}$, which is computed at the output layer. Here, $\odot$ is the Hadamard product, or element-wise multiplication. The second term can be further simplified as

$$\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(1)}} = \frac{\partial (\boldsymbol{W}^{(2)} \boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)})}{\partial \boldsymbol{a}^{(1)}} = {\boldsymbol{W}^{(2)}}^{\top} \tag{3.56}$$

Similarly, we employ transposition for the dimension consistency. The last term of Eq. 3.55 can be calculated as

$$\frac{\partial \boldsymbol{a}^{(1)}}{\partial \boldsymbol{z}^{(1)}} = \frac{\partial \psi(\boldsymbol{z}^{(1)})}{\partial \boldsymbol{z}^{(1)}} = e^{\psi(\boldsymbol{z}^{(1)})}. \tag{3.57}$$

Inserting Eqs. 3.55, 3.56 and 3.57 into Eq. 3.54 leads to the final form of the partial derivative of the cost function with respect to $\boldsymbol{W}^{(1)}$ expressed as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \delta^{(2)} {\boldsymbol{W}^{(2)}}^{\top} \odot e^{\psi(\boldsymbol{z}^{(1)})} \boldsymbol{z}^{\top}. \tag{3.58}$$

Similarly, the final form of the partial derivative of the cost function with respect to $\boldsymbol{b}^{(1)}$ is

$$\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \delta^{(2)} {\boldsymbol{W}^{(2)}}^{\top} \odot e^{\psi(\boldsymbol{z}^{(1)})}. \tag{3.59}$$

Recall each individual component of final partial derivative forms as

$$\delta^{(2)} = 886.00, \tag{3.60}$$

$${\boldsymbol{W}^{(2)}}^{\top} = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}, \tag{3.61}$$

$$e^{\psi(\boldsymbol{z}^{(1)})} = \begin{bmatrix} 1.82 \\ 3.67 \end{bmatrix}, \tag{3.62}$$

$$\boldsymbol{z}^{\top} = \begin{bmatrix} 1 & 2 \end{bmatrix}. \tag{3.63}$$

Inserting equations above into Eq. 3.59, partial derivatives with respect to weights and biases can be computed as

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = 886.00 \cdot \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix} \odot \begin{bmatrix} 1.82 \\ 3.67 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} \tag{3.64}$$

$$= \begin{bmatrix} 886.00 \cdot 0.5 \cdot 1.82 \\ 886.00 \cdot 0.6 \cdot 3.67 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 806.26 & 1612.52 \\ 1625.81 & 3251.62 \end{bmatrix}, \tag{3.65}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = 886.00 \cdot \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix} \odot \begin{bmatrix} 1.82 \\ 3.67 \end{bmatrix} \tag{3.66}$$

$$= \begin{bmatrix} 886.00 \cdot 0.5 \cdot 1.82 \\ 886.00 \cdot 0.6 \cdot 3.67 \end{bmatrix} = \begin{bmatrix} 806.26 \\ 1625.81 \end{bmatrix}. \tag{3.67}$$

The backpropagation algorithm can be summarized as follows: For any random layer l, the partial derivative of the cost function L with respect to weights $\boldsymbol{W}^{(l)}$ and biases $\boldsymbol{b}^{(l)}$

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)} \boldsymbol{a}^{(l-1)^\top},$$
$$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)},$$

where

$$\delta^{(l)} = \boldsymbol{W}^{(l+1)} \delta^{(l+1)} \odot \frac{\partial \psi(\boldsymbol{z}^{(l)})}{\partial \boldsymbol{z}^{(l)}}. \tag{3.68}$$

When the arbitrary layer l is the output layer, then the error term δ becomes

$$\delta^{(L)} = (\hat{y} - y) \odot \frac{\partial \psi(\boldsymbol{z}^{(L)})}{\partial \boldsymbol{z}^{(L)}}. \tag{3.69}$$

Remark 3.4.1:

- Remember that backpropagation is not the learning algorithm or the optimization process itself. It just refers to method of computing gradients.
- Avoid using a non-vectorized implementation of backpropagation, as it can be overly cumbersome and error-prone.
- Note that both forward and backward propagation can be performed using an augmented network parameter matrix, $\boldsymbol{\theta}$, which integrates both weights and biases. Since the aim is to explain how to propagate input through network layers step by step, we presented network parameters as two separate vector groups.

3.4.2. Learning algorithm and optimizers

So far, we have covered all the required components for constructing a neural network, including the computation of gradients of the loss function with respect to trainable network parameters are computed. Now, we need a learning algorithm to update network parameters based on the computed gradients.

A learning algorithm, or optimizer, is a method to update the weights and biases of a neural network to minimize the error between the predicted output and the actual target values. Optimizers play a crucial role in the training process. Thus, various strategies are employed to achieve efficient learning, avoiding local minima, improving speed or training time, and accuracy in finding the optimal set of parameters.

3.4.2.1. Common optimizers

Gradient Descent: The Gradient Descent (GD) algorithm is the most basic optimizer, which updates the network parameters in the negative direction of the gradients. The update rule is defined as

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}-1}}, \tag{3.70}$$

where η denotes the learning rate and i_{iter} is the epoch or iteration number. Learning rate is a hyperparameter that controls the step size of the updates. The pseudocode in Algorithm 9 provides a vectorized implementation of Gradient Descent. In this implementation, both the weights and bias terms are updated simultaneously at each iteration. The process continues until either the maximum number of iterations, n_{iter}, is reached or the updates converge.

Algorithm 9 Gradient Descent (Vectorized)

Inputs:
L: a differentiable loss function
$\boldsymbol{W}_0$: initialized weights
$\boldsymbol{b}_0$: initialized bias terms
η: learning rate
n_{iter}: max. number of iterations
Output:
$\boldsymbol{W}, \boldsymbol{b}$: Updated weights and bias terms

procedure GRADIENTDESCENT($L, \boldsymbol{W}_0, \boldsymbol{b}_0, \eta, n_{\text{iter}}$)
Set the iteration counter $i_{\text{iter}} = 0$
while $i_{\text{iter}} < n_{\text{iter}}$ **do**
$\boldsymbol{W}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{W}_{i_{\text{iter}}} - \eta \dfrac{\partial L}{\partial \boldsymbol{W}_{i_{\text{iter}}}}$ ▷ Update weights using gradient descent
$\boldsymbol{b}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{b}_{i_{\text{iter}}} - \eta \dfrac{\partial L}{\partial \boldsymbol{b}_{i_{\text{iter}}}}$ ▷ Update bias terms using gradient descent
$i_{\text{iter}} = i_{\text{iter}} + 1$ ▷ Increase the epoch number
end while
return $\boldsymbol{W}, \boldsymbol{b}$ ▷ Get the optimized weights and bias terms
end procedure

Stochastic Gradient Descent: One of the primary drawbacks of Gradient Descent (GD) is its computational cost. In GD, the gradient of the loss function is computed using the entire training dataset, meaning that all samples contribute to the calculation of the partial derivatives with respect to the weights and bias terms. However, this can be computationally expensive for large datasets. In contrast, Stochastic Gradient Descent (SGD) computes the gradient using only a single, randomly selected training sample (or a small batch) at each iteration. Although this makes SGD much faster per iteration, it can lead to more oscillatory updates, as the gradient direction may vary significantly between steps. As a result, while SGD often achieves faster convergence, it tends to be less stable compared to GD.

Momentum Optimization: Recall update rule of Gradient Descent (Eq. 3.70): $\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}$. This rule updates the network parameters by subtracting a fraction of the gradient computed in the previous iteration. However, when the local derivatives are very small, the resulting updates become negligible, leading to a very slow convergence rate. To overcome

this issue, the following modified update rule is proposed.

$$\boldsymbol{\nu}_{i_{\text{iter}}+1} \leftarrow \beta \boldsymbol{\nu}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.71}$$

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} + \boldsymbol{\nu}_{i_{\text{iter}}+1}. \tag{3.72}$$

Here, the $\boldsymbol{\nu}$ denotes the local momentum vector or the velocity, β is a hyperparameter called the momentum coefficient, which is set between 0 (high friction) and 1 (low friction). It also determines the rate of the exponential decay.

It can be verified that if the gradients do not change through iterations, the optimizer has reached the terminal velocity, meaning that $\boldsymbol{\nu}_{i_{\text{iter}}+1} = \boldsymbol{\nu}_{i_{\text{iter}}}$. With this assumption, recall the update rule (Eq. 3.71) to rewrite

$$\begin{aligned} \boldsymbol{\nu}_{i_{\text{iter}}} &= \beta \boldsymbol{\nu}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \\ \boldsymbol{\nu}_{i_{\text{iter}}}(1-\beta) &= -\eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \\ \boldsymbol{\nu}_{i_{\text{iter}}} &= -\frac{\eta}{(1-\beta)} \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}. \end{aligned} \tag{3.73}$$

Also, as shown in Eq. 3.73, choosing β larger than 1 might cause to amplification of weights rather than decay.

Momentum optimization can be easily integrated with Gradient Descent and its variants to speed up the optimization process.

Nesterov Accelerated Gradient: Nesterov Accelerated Gradient (NAG) is an enhanced version of standard momentum-based optimization. Instead of computing the gradient at the current network parameters $\boldsymbol{\theta}$, the gradient is evaluated at a slightly ahead position $\boldsymbol{\theta} + \beta \boldsymbol{\nu}$, where β is the momentum coefficient. This approach anticipates the future position of the parameters and adjusts the update accordingly. The NAG update rule is defined as

$$\boldsymbol{\nu}_{i_{\text{iter}}+1} \leftarrow \beta \boldsymbol{\nu}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}} + \beta \boldsymbol{\nu}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.74}$$

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} + \boldsymbol{\nu}_{i_{\text{iter}}+1}. \tag{3.75}$$

The main advantage of NAG is that it converges faster by anticipating the next step, since it can adjust the velocity vector in advance.

AdaGrad: In standard gradient-based methods, the learning rate remains fixed throughout training. In contrast, the Adaptive Gradient (AdaGrad) algorithm adjusts the learning rate for each parameter based on the frequency and magnitude of its gradients. The update rule for AdaGrad is defined as

$$\boldsymbol{s}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{s}_{i_{\text{iter}}} + \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}} \odot \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.76}$$

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}} \oslash \sqrt{\boldsymbol{s}_{i_{\text{iter}}+1} + \boldsymbol{\varepsilon}}, \tag{3.77}$$

where $\odot$ and $\oslash$ denote element-wise multiplication and division, respectively. In Eq. 3.76, the squared gradients are accumulated in $\boldsymbol{s}$, and in Eq. 3.77, the gradient is scaled down by dividing by $\sqrt{\boldsymbol{s}_{i_{\text{iter}}+1} + \varepsilon}$, with ε serving as a smoothing term to prevent division by zero. This mechanism effectively adapts the learning rate so that parameters with larger gradients receive smaller updates, while those with smaller gradients receive relatively larger updates.

The main drawback of AdaGrad is that the learning rate might scale down monotonically and eventually become too small for further learning, and the algorithm might stop before reaching the global minimum.

RMSProp: Root Mean Square Propagation (RMSProp) is an adaptive learning algorithm designed to mitigate the diminishing learning rate issue found in AdaGrad. It addresses this problem by accumulating only the gradients from recent iterations, rather than all past gradients. The update rule for RMSProp can be expressed as:

$$\boldsymbol{s}_{i_{\text{iter}}+1} \leftarrow \rho \boldsymbol{s}_{i_{\text{iter}}} + (1-\rho)\frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}} \odot \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.78}$$

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} - \eta \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}} \oslash \sqrt{\boldsymbol{s}_{i_{\text{iter}}+1} + \boldsymbol{\varepsilon}}, \tag{3.79}$$

where ρ denotes the decay rate. Unlike AdaGrad, RMSProp uses an exponential decay of recent squared gradients (Eq. 3.78). This prevents the learning rate from shrinking too fast, leading the algorithm to further training to reach the global minimum.

Adam: Adam (short for Adaptive Moment Estimation) is an adaptive optimization algorithm that combines two other optimization algorithms, that are moment-based optimization and RMSProp. Similar to moment-based optimization, Adam maintains an exponential decay of past gradients, and like RMSProp, it monitors an exponential decay of past squared gradients. The update rule for Adam is expressed as

$$\boldsymbol{r}_{i_{\text{iter}}+1} \leftarrow \rho_1 \boldsymbol{r}_{i_{\text{iter}}} + (1-\rho_1)\frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.80}$$

$$\boldsymbol{s}_{i_{\text{iter}}+1} \leftarrow \rho_2 \boldsymbol{s}_{i_{\text{iter}}} + (1-\rho_2)\frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}} \odot \frac{\partial L(\boldsymbol{\theta}_{i_{\text{iter}}})}{\partial \boldsymbol{\theta}_{i_{\text{iter}}}}, \tag{3.81}$$

$$\hat{\boldsymbol{r}}_{i_{\text{iter}}} \leftarrow \frac{\boldsymbol{r}_{i_{\text{iter}}+1}}{1-\beta_1^{i_{\text{iter}}}}, \tag{3.82}$$

$$\hat{\boldsymbol{s}}_{i_{\text{iter}}} \leftarrow \frac{\boldsymbol{s}_{i_{\text{iter}}+1}}{1-\beta_2^{i_{\text{iter}}}}, \tag{3.83}$$

$$\boldsymbol{\theta}_{i_{\text{iter}}+1} \leftarrow \boldsymbol{\theta}_{i_{\text{iter}}} - \eta \hat{\boldsymbol{r}}_{i_{\text{iter}}} \oslash \sqrt{\hat{\boldsymbol{s}}_{\boldsymbol{i}_{\text{iter}}} + \boldsymbol{\varepsilon}}. \tag{3.84}$$

As explained above, Eq. 3.80 denotes the moment-based optimization part of Adam since it keeps track of past gradients, while Eq. 3.81 represents the RMSProp related part of Adam since the formulation contains an exponential decay of past squared gradients. Here, i_{iter} is the current iteration number. Also, the Adam optimizer can be further improved by using the Nesterov trick, which takes into account the momentum term while calculating gradients (see Eq. 3.74). This optimizer is called Nadam.

3.4.3. Hyperparameters of neural networks

For neural networks, the overall architecture and its components determine the data flow, the training procedure, and ultimately the quality of the predictions. Consequently, hyperparameters play a critical role in the performance of the neural network. The key hyperparameters include:

- Layers and neurons
- Activation function
- Network initializer
- Cost (loss) function
- Optimizer
  - Learning rate
  - Epochs
  - Batch size

3.4.3.1. Layers and neurons

As previously explained in Section 3.4, but worth mentioning again for completeness of the section, layers and neurons serve as the fundamental components of neural networks. Each layer is composed of neurons that perform computations via activation functions. Typically, the architecture comprises an input layer, one or more hidden layers, and an output layer. For a more detailed explanation of these components, please refer to Section 3.4.

So far, we have not addressed network topology and the total number of trainable parameters. Network topology refers to the arrangement of hidden layers and neurons, which is characterized by the network's *depth* and *width*. Network depth specifies the number of hidden layers–a shallow neural network contains only a few hidden layers, whereas a deep neural network comprises many. In contrast, network width denotes the number of neurons in each layer. Although wider networks generally have a greater capacity to capture complex patterns, they also involve more neurons, which increases the computational demands.

The total number of trainable parameters (i.e., both weights and biases) in a fully connected neural network is given by

$$\sum_{i=1}^{L}(n_{h_i-1}+1)\cdot n_{h_i}, \tag{3.85}$$

where L is the number of layers, including all hidden layers and the output layer, n_{h_i} denotes the number of neurons in the l^{th} layer. As an example, consider a neural network with three input neurons, 2 hidden layers containing 2 and 3 neurons, respectively, and an output layer with five neurons. By applying the formula in Eq. 3.85, the total number of parameters can be determined as follows: $(3+1)\cdot 2+(2+1)\cdot 3+(3+1)\cdot 5=8+9+20=37$.

The number of trainable parameters reflects the model's capacity and complexity, indicating its potential to learn complex patterns. However, a higher number of parameters also demands more computational resources and may increase the risk of overfitting, making it essential to choose the network architecture carefully.

3.4.3.2. Activation function

Activation functions are crucial components of neural networks. They add nonlinearity to the network, allowing it to capture hidden patterns through layers and connections within the data. Without activation functions, a neural network would turn into a basic linear regression model, regardless of how many layers it has. Thus, activation functions play an important role in neural networks. The most common activation functions are: linear, sigmoid (logistic), hyperbolic tangent (tanh), rectified linear unit (ReLU), Leaky ReLU, exponential linear unit (ELU), parametric linear unit (PReLU) and Swish.

Linear activation: The linear activation function can be considered the simplest one among the activation functions. It can be expressed as

$$\psi(x) = x. \tag{3.86}$$

The linear activation function does not change the output, meaning that it does not introduce any nonlinearity for the following layer. The output of linear activation spans the entire range from $-\infty$ to ∞. It is usually used in the output layer. Fig. 3.24 illustrates the linear activation function along with its first and second derivatives.

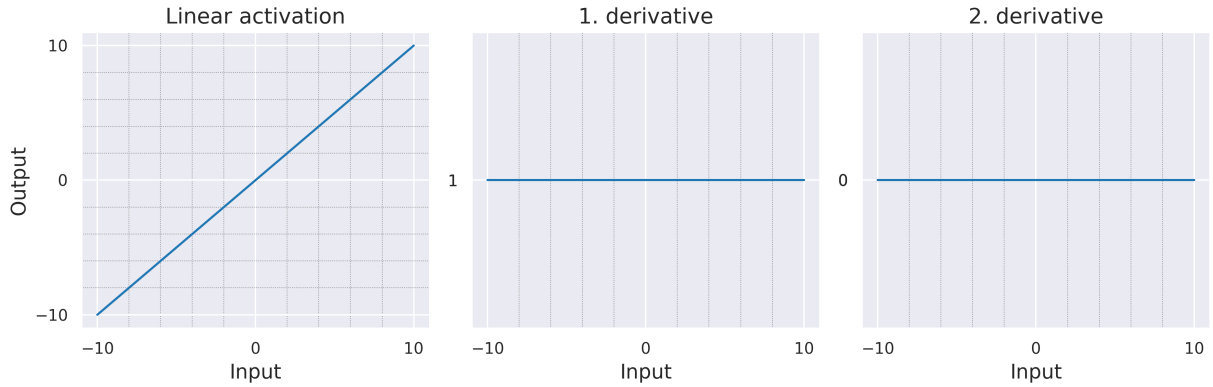


Figure 3.24.: The linear activation function and its derivatives.

Sigmoid (logistic) activation: The sigmoid (logistic) activation is a popular activation function, and it is expressed as

$$\psi(x) = \frac{1}{1 + e^{(-x)}}. \tag{3.87}$$

The output falls within the interval $(0, 1)$, which can be interpreted as a probability function. The sigmoid activation function and its derivatives are illustrated in Fig. 3.25. Near the origin, it exhibits large gradients, while these gradients drastically decrease by moving away from the origin, which might cause the vanishing gradients. Also, the sigmoid function is quite nonlinear.

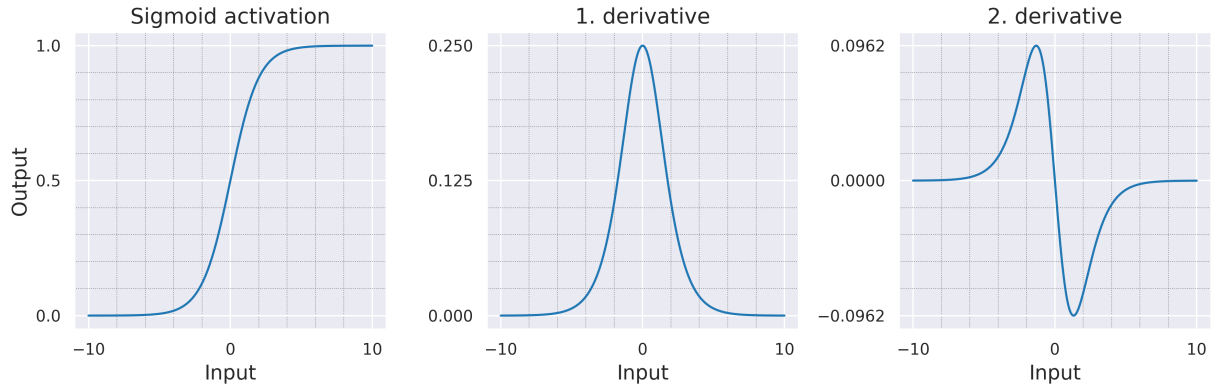


Figure 3.25.: The sigmoid activation function and its derivatives.

Hyperbolic tangent (tanh) activation: The hyperbolic tangent (tanh) activation function is expressed as

$$\psi(x) = \frac{e^{(x)} - e^{(-x)}}{e^{(x)} + e^{(-x)}}. \tag{3.88}$$

The output falls within the range of $(-1, 1)$, which is centered around the origin. It also suffers from vanishing gradients for large inputs. The activation function and its derivatives are given in Fig. 3.26. The sigmoid and tanh activation functions are quite similar, but the tanh activation exhibits a more spiky behavior.

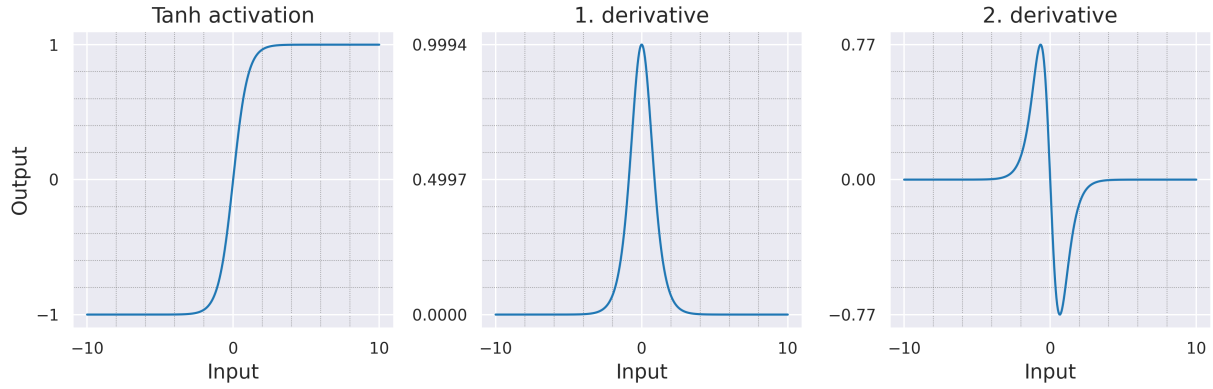


Figure 3.26.: The hyperbolic tangent (tanh) activation function and its derivatives.

Rectified linear unit (ReLU) activation: As mentioned earlier, the sigmoid and tanh activation functions suffer from vanishing gradients. To mitigate this problem, the rectified linear unit (ReLU) activation has been introduced and it can be expressed as

$$\psi(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{if } x \leq 0. \end{cases} \tag{3.89}$$

As shown in Fig. 3.27, the ReLU activation is a piecewise function, meaning that it outputs zero for negative inputs, while it is linear for positive inputs. However, the ReLU activation function

introduces the issue of dying nodes for negative inputs. Also, the range for output is $(0, \infty)$. The ReLU activation is widely used in deep learning applications due to its simplicity regarding implementation, and its computational efficiency.

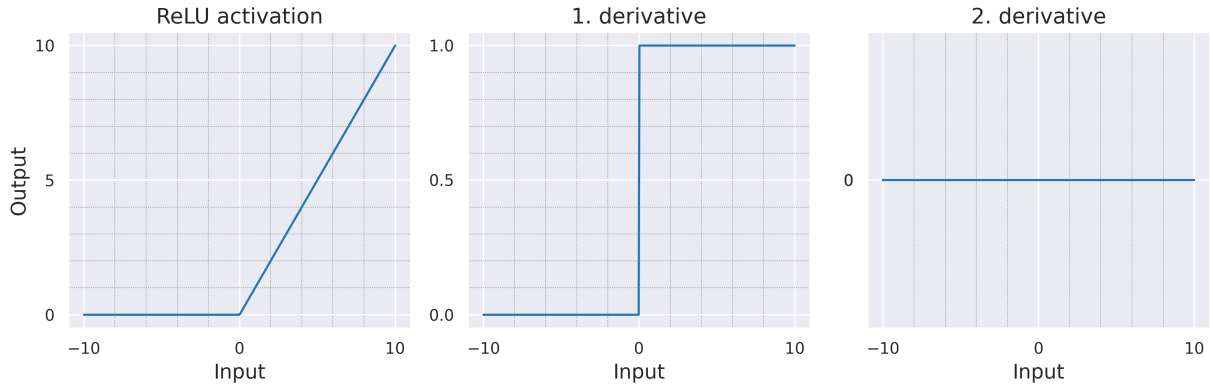


Figure 3.27.: Rectified linear unit (ReLU) activation function and its derivatives.

Leaky ReLU activation: The Leaky ReLU activation function is a modified version of ReLU to mitigate the issue of dying nodes for negative inputs. It is defined as

$$\psi(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{if } x \leq 0. \end{cases} \tag{3.90}$$

Here, a small positive value of 0.01 is introduced to ensure that neurons produce a nonzero output when x is negative. The Leaky ReLU activation function allows outputs to range from $-\infty$ to ∞. Similarly, Fig. 3.28 illustrates the Leaky ReLU activation function along with its derivatives.

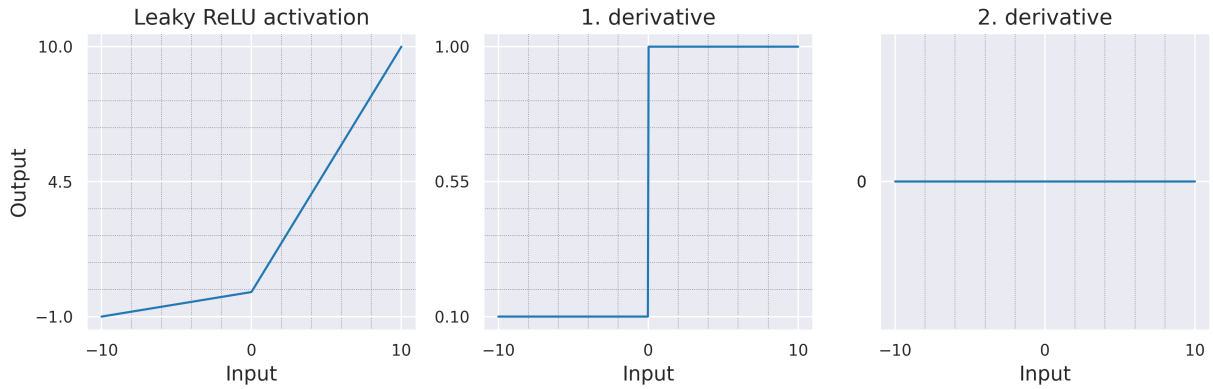


Figure 3.28.: The Leaky ReLU activation function and its derivatives.

Parametric ReLU (PReLU) activation: The parametric ReLU (PReLU) activation function can be considered as a more generalized version of Leaky ReLU and it is expressed as

$$\psi(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{if } x \leq 0, \end{cases} \tag{3.91}$$

where α is a hyperparameter instead of small constant 0.01 compared to Leaky ReLU. The output of the parametric ReLU activation spans the entire range from $-\infty$ to ∞. The main benefit here is that the parameter α can be optimized during training as well, which might outperform Leaky ReLU. Fig. 3.29 illustrates the PReLU activation function along with its first and second derivatives.

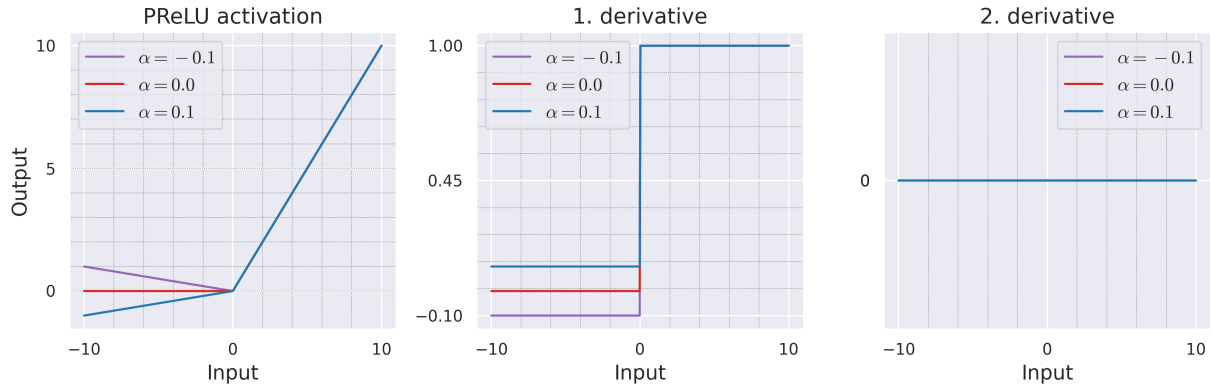


Figure 3.29.: The parametric RELU (PReLU) activation function and its derivatives for different α values.

Exponential linear unit (ELU) activation: The exponential linear unit (ELU) activation function can be expressed as

$$\psi(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha(e^x - 1) & \text{if } x \leq 0. \end{cases} \tag{3.92}$$

Unlike other ReLU-based variants, ELU has nonzero second-order derivatives, which make it a smoother function. This smoothness improves optimization stability and supports more effective gradient-based learning, especially in deep neural networks. Furthermore, its output range $(-\alpha, \infty)$ guarantees nonzero responses for negative inputs, thereby alleviating the vanishing gradient issue.

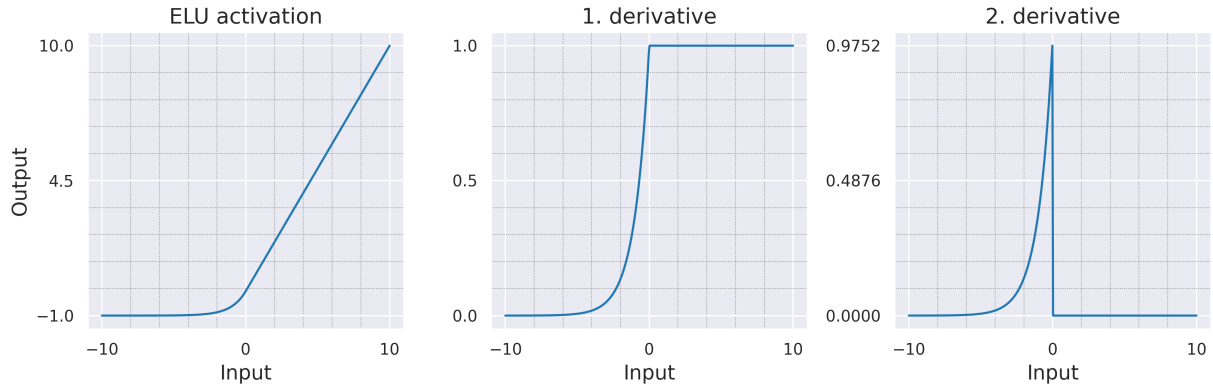


Figure 3.30.: The exponential linear unit (ELU) activation function and its derivatives for $\alpha = 1$.

Swish activation: The swish activation function is defined as

$$\psi(x) = x\frac{1}{1+e^{-\beta x}}, \tag{3.93}$$

where β is either a constant or a trainable parameter. When $\beta = 1$, the Swish function becomes equivalent to the sigmoid linear unit (siLU). Compared to traditional activation functions, Swish is both smooth and non-monotonic, allowing it to preserve small negative values instead of truncating them to zero, as seen in ReLU-based functions. This property enhances gradient flow and improves convergence in deep networks. However, the function is computationally more expensive than simpler alternatives like ReLU due to the presence of the sigmoid component.

Fig. 3.31 illustrates the Swish activation function along with its first and second derivatives for $\beta = 0.1$. The first plot shows the activation function, which smoothly transitions through zero while allowing negative inputs. The second plot depicts its first derivative, which remains nonzero, ensuring stable gradient propagation. The third plot presents the second derivative, highlighting the curvature variations that contribute to its adaptive learning behavior.

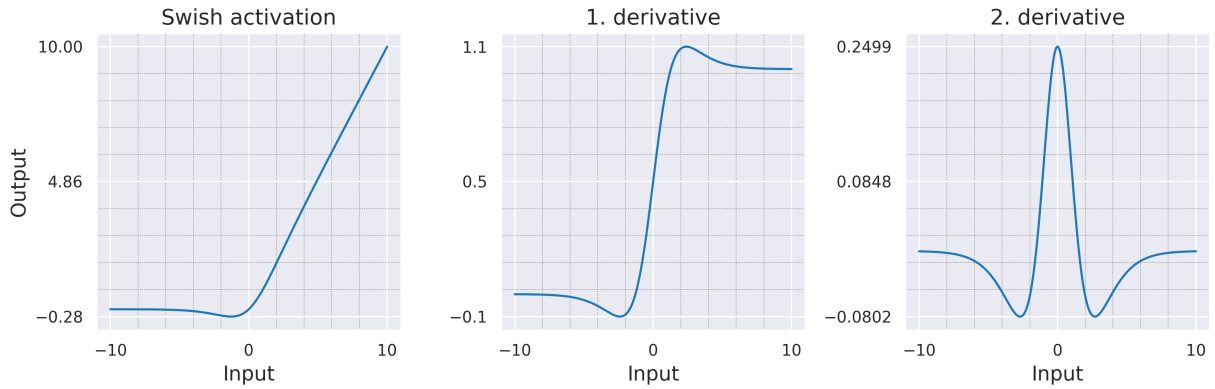


Figure 3.31.: The Swish activation function and its derivatives for $\beta = 0.1$.

The formulations for the first and second derivatives of the introduced activation functions are provided in Table 3.3. These derivatives play a crucial role in the optimization process, influencing gradient-based learning algorithms such as backpropagation. Understanding their behavior helps in selecting appropriate activation functions based on the specific needs of a model, such as mitigating the vanishing gradient problem or improving convergence speed, as we will see in the following chapters.

For instance, the linear activation has constant first and second derivatives, making it unsuitable for capturing complex patterns. Sigmoid and tanh functions introduce nonlinearity, but their derivatives diminish for extreme input values, leading to the vanishing gradient problem. ReLU and its variants (Leaky ReLU and Parametric ReLU) address this issue by maintaining nonzero gradients for positive inputs, although their second derivatives are zero, reducing adaptability. ELU extends ReLU by ensuring smooth gradients even for negative inputs, while Swish, with its non-monotonic behavior, maintains nonzero gradients across the input range, improving convergence in deep networks. Ultimately, the choice of activation function depends on the model's needs, balancing smoothness, gradient flow, and computational efficiency.

Activation function	**1. derivative**	**2. derivative**
linear	$\psi'(x) = 1$	$\psi''(x) = 0$
sigmoid	$\psi'(x) = \sigma(x)(1-\sigma(x))$	$\psi''(x) = \sigma(x)(1-\sigma(x))(1-2\sigma(x))$
tanh	$\psi'(x) = 1 - \tanh^2(x)$	$\psi''(x) = -2\tanh(x)(1-\tanh^2(x))$
ReLU	$\psi'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	$\psi''(x) = 0$
Leaky ReLU	$\psi'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0.01 & \text{if } x \leq 0 \end{cases}$	$\psi''(x) = 0$
PReLU	$\psi'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$	$\psi''(x) = 0$
ELU	$\psi'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$	$\psi''(x) = \begin{cases} 0 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$
Swish	$\psi'(x) = \frac{1 + e^{-\beta x} + \beta x e^{-\beta x}}{(1+e^{-\beta x})^2}$	$\psi''(x) = \frac{\beta e^{-\beta x}\left[(1+e^{-\beta x})+(1-\beta x)(1+e^{-\beta x})+2\beta x e^{-\beta x}\right]}{(1+e^{-\beta x})^3}$

Table 3.3.: First and second derivatives of the introduced activation functions in neural networks.

Remark 3.4.2:

As discussed in backpropagation (Section 3.4.1), conventional neural networks require only first-order derivatives for updating trainable parameters. However, as we will explore in the following chapters, physics-informed neural networks necessitate higher-order derivatives–such as second and even third-order–due to the nature of the target physics. Consequently, second-order derivatives are also introduced in this section.

3.4.3.3. Network initializer

Initialization schemes for neural networks are techniques employed to assign the initial values to trainable network parameters, namely, weights and biases, before starting the training of a neural network. Proper initialization is a crucial step since it can significantly affect the convergence performance and speed. The most common initialization schemes are: zero initialization, random initialization, Xavier (Glorot) initialization, He initialization, and LeCun initialization.

Zero initialization: All network parameters are initialized to zero. It is a simple approach to implement, but it leads to poor performance due to the symmetry problem [97]. Specifically, when all weights are initialized to the same value, neurons within the same layer receive identical updates, preventing the network from learning diverse features.

Random initialization: Network parameters are initialized using a random or uniform Gaussian distribution [97]. Random initialization helps avoid the symmetry problem present in zero initialization. However, a poorly chosen variance in the distribution may lead to large gradients or vanishing signals, negatively affecting model stability and performance.

Xavier (Glorot) initialization: In Xavier (Glorot) initialization [61], the goal is to maintain a consistent variance in trainable network parameters, ensuring smooth information flow during training. The Xavier initializer assigns weights using a normal distribution with a mean of zero and a variance determined by the number of input and output units in each layer. The variance of the distribution is defined as $\sigma^2 = \frac{2}{n_{in}+n_{out}}$, where n_{in} is the number of input neurons in the layer and n_{out} is the number of output neurons. Xavier initialization is effective for networks utilizing linear, tanh, logistic, and softmax activation functions.

He initialization: : He initialization [73] is an extension of Xavier initialization designed for deep networks that use ReLU and its variants. It follows a similar principle but adjusts the variance based only on the number of input units instead of averaging input and output units. The weight parameters are initialized using a normal distribution with a mean of zero and a variance defined as $\sigma^2 = \frac{2}{n_{in}}$.

LeCun initialization: : LeCun initialization [97] is a variant of Xavier initialization, optimized for networks using sigmoid and tanh activation functions. When $n_{out} = n_{in}$, LeCun initialization is equivalent to Xavier. The trainable parameters are drawn from a normal distribution with a mean of zero and a variance $\sigma^2 = \frac{1}{n_{in}}$.

Remark 3.4.3:

- Initialization schemes such as Xavier, He, and LeCun are typically applied only to the weights of the neural network, while bias terms are usually initialized to zero or a small constant.
- Similar to zero initialization, a neural network can be initialized using any constant value. However, this approach results in poor performance for the same reasons as zero initialization. When all trainable parameters are assigned identical values, neurons within the same layer receive identical gradient updates, preventing effective feature extraction and leading to a symmetry problem that hinders learning.

3.4.3.4. Cost (loss) function

The cost function, also known as the loss function, is a fundamental component of neural networks that measures how well the model predictions align with the actual values. In other words, it quantifies the difference between predicted and true values. During training, the cost function is minimized to optimize network parameters, enabling the model to learn underlying patterns within the data.

Cost functions are generally classified into two categories based on the type of output: regression and classification costs. Regression cost functions are used when the output is a continuous variable, while classification cost functions are applied when the output represents categorical values. In this section, we focus on the most commonly used cost functions for regression tasks.

Additionally, loss functions serve as evaluation metrics for assessing the performance of a trained model. After the model is trained, they can also be used to measure a model generalization capability by evaluating its performance on test data.

Mean absolute error (MAE) : Mean absolute error (MAE), also referred to as L1 loss, is defined as

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \tag{3.94}$$

where N is the number of training points, y denotes the actual target value, and $\hat{y}$ is the prediction.

Since MAE does not square the errors, it is less sensitive to outliers. On the other hand, it is not differentiable at zero, which can cause problems during optimization. MAE is mainly used for regression tasks.

Mean squared error (MSE) : Mean squared error (MSE), also referred to as L2 loss, is the most widely used loss function for regression tasks. It is defined as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{3.95}$$

MSE is more sensitive to outliers compared to the Mean Absolute Error (MAE) because it squares the errors. As a result, larger errors receive more severe penalties due to the squaring operation.

However, one of the key advantages of MSE is that it is a smooth and differentiable function, making it particularly useful for training neural networks.

Huber loss: Huber loss combines the advantages of MSE and MAE, making it more robust to outliers while maintaining differentiability. The cost function is defined as:

$$L_\delta = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta \cdot (|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{if } |y_i - \hat{y}_i| > \delta. \end{cases} \tag{3.96}$$

Here, δ serves as a threshold that determines the transition between the MSE and MAE components of Huber loss. When the absolute error is below δ, the function behaves like MSE, ensuring smooth gradient updates for small errors. For larger errors, it switches to MAE, reducing sensitivity to outliers by preventing excessive gradient magnitudes. Thus, Huber loss can be considered a hybrid approach that leverages the strengths of both MSE and MAE. However, it introduces an additional hyperparameter, δ, which needs to be carefully tuned to balance robustness and sensitivity to errors.

3.4.3.5. Optimizer

The most commonly used optimizers have been discussed in Section 3.4.2.1. Therefore, in this part, we will only cover key hyperparameters related to the optimization process, i.e., learning rate, epochs, and batch size.

Learning rate : The learning rate (η) is a crucial hyperparameter in optimization, determining the step size for updating trainable parameters during the minimization of the loss function. Selecting an appropriate value for η is essential, as it directly impacts both convergence speed and overall performance. A higher learning rate results in larger parameter updates, which can accelerate the learning process. However, if η is too large, the optimizer may overshoot the minimum of the loss function, causing instability or even divergence. On the other hand, a lower learning rate leads to smaller updates, delivering more precise convergence but potentially prolonging the training process.

Fig. 3.32 illustrates the impact of different learning rates on the convergence of the loss function. When using a small learning rate, such as $\eta = 0.01$, the optimization process remains stable but progresses slowly. Increasing the learning rate to $\eta = 0.1$ accelerates convergence while maintaining stability. However, with a higher learning rate, such as $\eta = 0.95$, the training process exhibits oscillations, leading to instability. Moreover, setting the learning rate to $\eta = 1.2$ results in divergence, preventing the model from converging.

Epochs : An epoch defines one complete pass in neural networks through the entire training dataset, meaning that input is propagated through the network to obtain the loss (forward propagation) Moreover, network parameters are updated to minimize the loss (backpropagation). Training a neural network usually requires multiple epochs since network parameters are first initialized randomly. Thus, it takes several epochs to obtain optimal network parameters.

The optimal number of epochs to train a model depends on the model complexity and the trade-off between the train-validation-test loss. Complex models include more parameters. Thus,

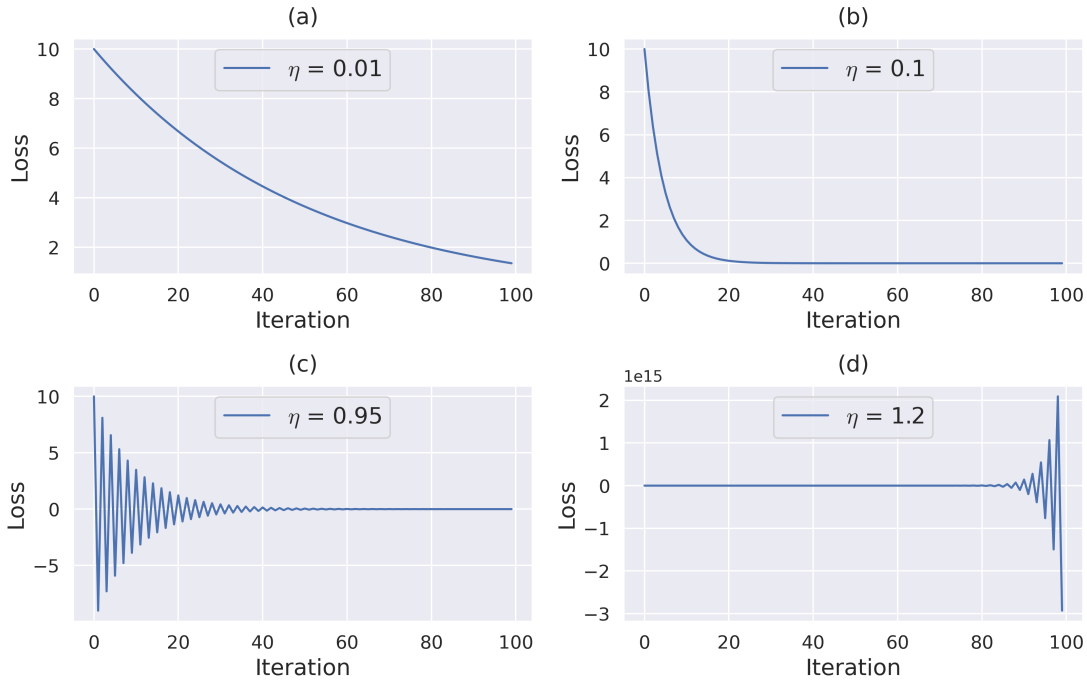


Figure 3.32.: Effect of different learning rates on the convergence of the loss function.

they require more epochs to obtain the optimal parameters. The trade-off between the train-validation-test will be explained in Section 3.5.3.

Batch size : Batch size refers to the number of training samples a model processes in a single iteration. In other words, batch size controls the number of samples that pass through in a single pass. A smaller batch size leads to faster training. However, the optimization process might not be stable since the weight and bias updates might be noisy. On the other hand, larger batch sizes require more extensive storage, but they provide much more stable updates.

Remark 3.4.4:

The terms epoch and iteration do not have the same meaning, although they amount to the same thing when the entire dataset is used during training in one complete pass. Consider the following example to understand the difference between epoch and iteration using batch size. The total number of sample training points is given as 64, the batch size is defined as 32, and the number of iterations is 2. In the first iteration, the neural network takes the first 32 samples. It passes them to the output layer through the forward pass, computes the loss, performs a backpropagation to obtain gradients, and updates the network parameters to minimize loss. Then, the same process is repeated in the second iteration using the following 32 samples. As a result, the neural network goes through the entire data set once, but using two iterations. Thus, in this example, the number of epochs is ”1”, while ”2” iterations are performed.

3.4.4. Regularization in neural networks

Regularization is a crucial step in training neural networks to prevent overfitting, where the trained model performs exceptionally well on training data but poorly on unseen data. In other words, the model mimics the training data. Overfitting occurs when a model learns to capture noise or random variations in the training data rather than the underlying pattern. To tackle overfitting, regularization introduces constraints or modifications to the learning procedure. These constraints can be considered as penalties. The most widely used regularization techniques in neural networks are:

- Weight decay
- Dropout
- Data augmentation
- Early stopping

3.4.4.1. Weight decay

Weight decay is a technique for tackling overfitting by adding a penalty or regularization term to the loss function. The general form of weight decay is expressed as

$$\hat{L} = L + \lambda f(\boldsymbol{W}), \tag{3.97}$$

where λ is the regularization parameter and $f(\boldsymbol{W})$ denotes the type of regularization function. There are two main types of regularization: L1 and L2. In L1 regularization, the penalty term is proportional to the magnitude of the weights, and the cost function is defined as

$$\hat{L} = L + \lambda \boldsymbol{W}. \tag{3.98}$$

L1 regularization is particularly effective for feature selection, as it can shrink certain weights to exactly zero, effectively removing less important features. It is worth noting that L1 regularization is employed in Lasso regression (see Section 3.3.2.2). In contrast, L2 regularization is defined as

$$\hat{L} = L + \lambda \boldsymbol{W}^\top \boldsymbol{W}. \tag{3.99}$$

Compared to L1 regularization, L2 regularization uses the square of weights in the regularization function. Thus, a significant contribution to the loss function comes from larger weights. This leads to more penalization of larger weights compared to smaller weights and leads to smoother models in terms of weight distribution. Similarly, L2 regularization is used in Lasso Regression (see Section 3.3.2.2).

Remark 3.4.5:

Note that the regularization function only applies to the weights, not the bias term. The reason is that when the regularization parameter is set to large values, minimizing the regularized cost function forces weights to approach zero. The trained network output is zero if the bias term is also included in the regularization function. This is why the bias term is excluded from regularization.

3.4.4.2. Dropout

Dropout is a regularization technique in which a fraction of activation functions within a layer are set to zero. In other words, certain neurons are temporarily ignored during both the forward and backward passes of training, meaning they do not contribute to learning patterns from the data. Dropout is typically applied to the input layer and hidden layers.

An illustrative example is provided in Fig. 3.33, where some neurons are deactivated. However, this deactivation should not be mistaken for removal; dropout does not modify the network's structure. Instead, the activations of selected neurons are set to zero, preventing their corresponding weights from being updated. This approach ensures that computations using matrix-vector notation remain valid. Additionally, dropout is generally applied to weight parameters while leaving bias terms unaffected.

The selection of deactivated neurons is usually random. However, the proportion of neurons dropped can be controlled via a hyperparameter known as the **dropout rate**, which takes values within the range $[0, 1]$. This parameter represents the probability of dropping a neuron during training.

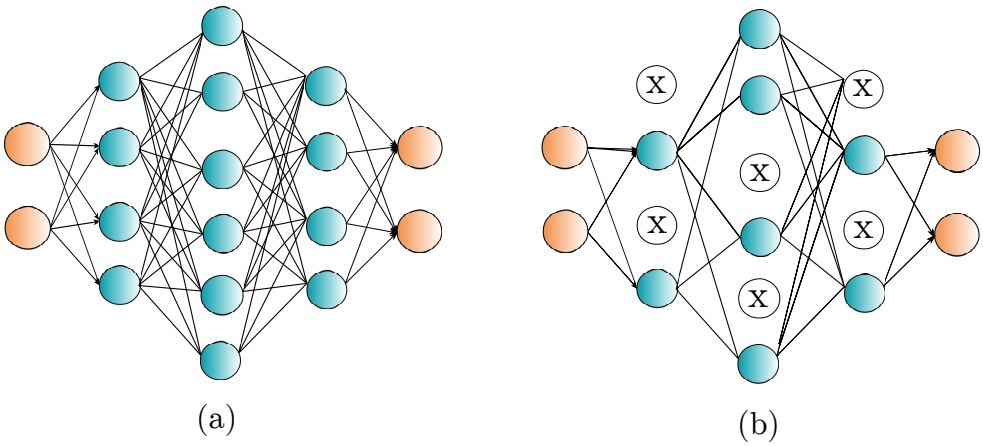


Figure 3.33.: Illustration of dropout in a fully connected neural network. (a) Without dropout, where all neurons are active. (b) With dropout, where a subset of neurons is randomly deactivated during training.

3.4.4.3. Data augmentation

Data augmentation generates a new dataset based on the existing dataset using transformation techniques. This allows for increasing the diversity of the dataset without collecting new samples. In other words, data augmentation can generate artificial data based on actual data.

Data augmentation is widely used in image processing and convolutional neural networks. The main techniques are:

- **Geometric transformations**: rotations, translations, scaling, flips, cropping, padding
- **Color space transformations**: brightness, contrast, saturation adjustments
- **Noise injection**: introducing random noise to the data

3.4.4.4. Early stopping

Early stopping [131] is a regularization technique that terminates the training process when the test loss no longer improves. While the training loss decreases through iterations, extending the training too far can cause the model to overfit, the model can no longer generalize the test data, since the model mimics the training data, including noise. To prevent this, training is intentionally stopped. An illustrative example of how early stopping works is shown in Fig. 3.34.

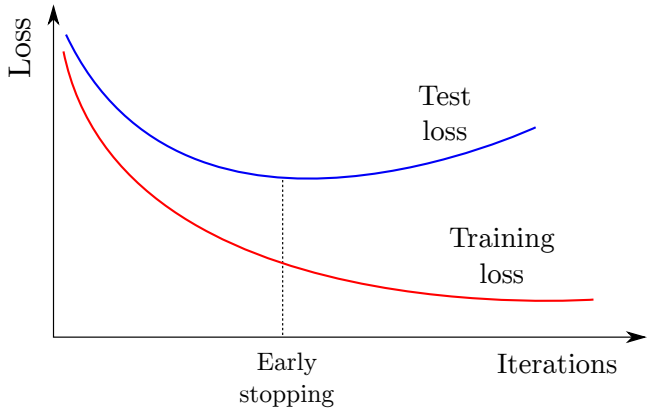


Figure 3.34.: Illustration of early stopping.

To further enhance the generalization performance of neural networks, several other regularization techniques can be employed, such as entropy regularization [65], batch normalization [80], and noise injection [3]. These approaches aim to reduce overfitting and improve model robustness.

3.4.5. Architecture types

3.4.5.1. Fully connected neural networks

A fully connected neural network (FCNN), also known as a dense neural network, is a neural architecture in which each neuron in a given layer is connected to all neurons in the next layer. Previous sections have already covered the key characteristics and details of FCNNs.

3.4.5.2. Convolutional neural networks

A convolutional neural network (CNN) is designed primarily for data with a grid-like topology, such as images. CNNs are multi-layered structures that are capable of extracting complex features in data.

The primary motivation behind CNNs can be explained using Fig. 3.35. Consider a grid-like topology having an input dimension of 100×100. This will require a neural network containing $10{,}000$ neurons only in the input layer (Fig. 3.35a). The hidden layers would need thousands of neurons to capture hidden patterns in each input, leading to millions of trainable parameters. Consequently, this would impose an extensive computational demand and cause overfitting since the model might contain millions of parameters even for a simple task. To tackle this issue,

convolutional layers are introduced to extract features that will be fed to the hidden layers (3.35b). These convolutional layers automatically extract features by applying convolutional filters.

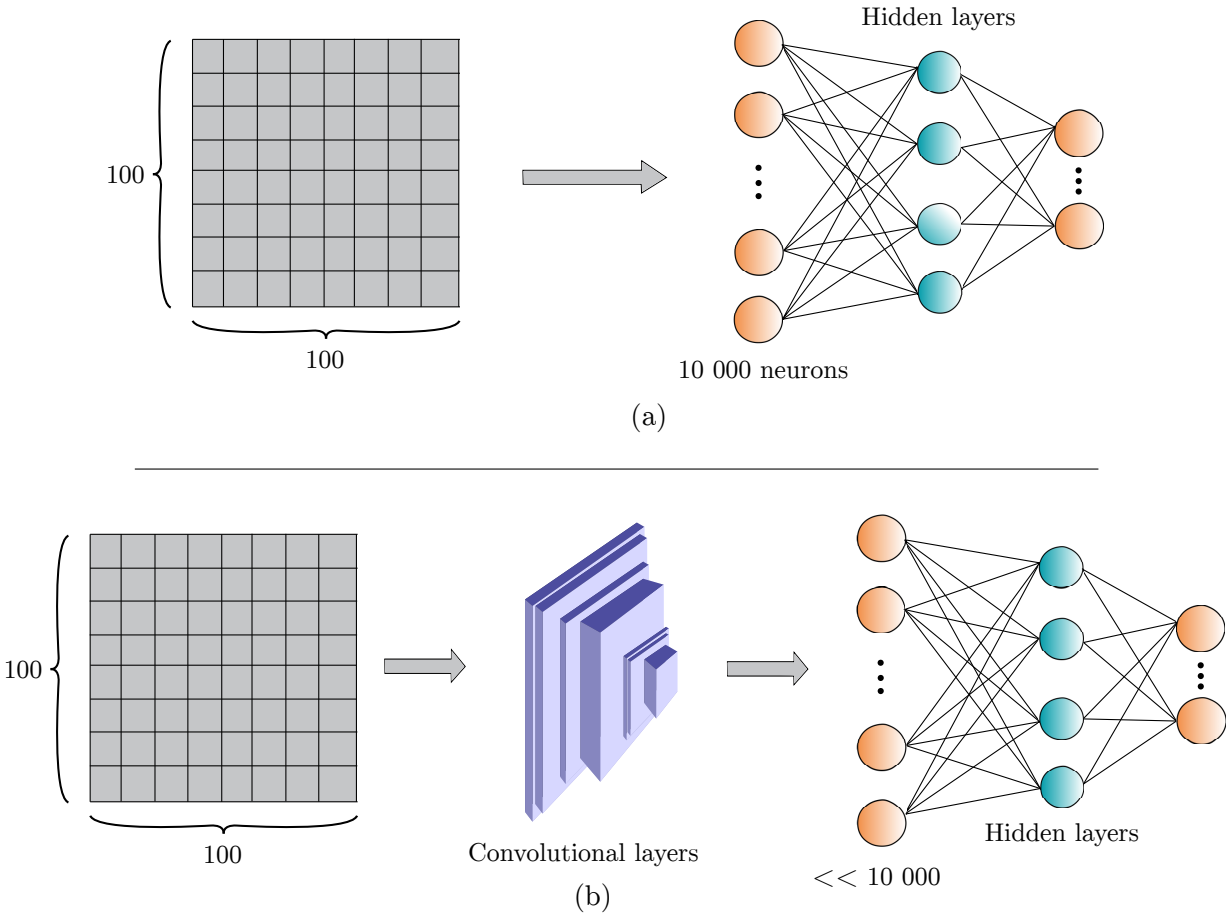


Figure 3.35.: (a) An illustration of a deep neural network for a grid-like topology and (b) adding convolutional layers to convert the network into a CNN.

Compared to traditional machine learning models that rely on manually defined features, CNNs can automatically extract features from the input, making them much more efficient. Thus, they are widely used for object detection, computer vision, image classification, and many other tasks.

CNN architecture: A CNN consists of several layers designed to extract the underlying patterns (features) in data and make predictions on new, unseen data. These layers include:

- **Input layer**: Contains the pixel values of an image or a matrix of a grid-like topology.
- **Convolutional layers**: Apply convolution operations to the input to extract features.
- **Pooling layers**: Perform downsampling on the feature maps to reduce spatial dimensions.
- **Fully connected layers**: Flatten the extracted features and use them for classification or regression tasks.

- **Output layer**: Provides the final prediction or target output.

Input layer

The input layer stores the pixel values of an image or a matrix of a grid-like structure, such as surface topology (height distribution). Images are three-dimensional structures consisting of height, width, and depth. For color images, the depth is 3, representing the RGB channels (red, green, blue), while grayscale images have a depth of 1, as they contain only density values. In Fig. 3.36, a portion of a grayscale image is given with corresponding pixel values. These pixel values range from 0 to 255, where zero represents black and 255 denotes white, with varying shades of gray in between.

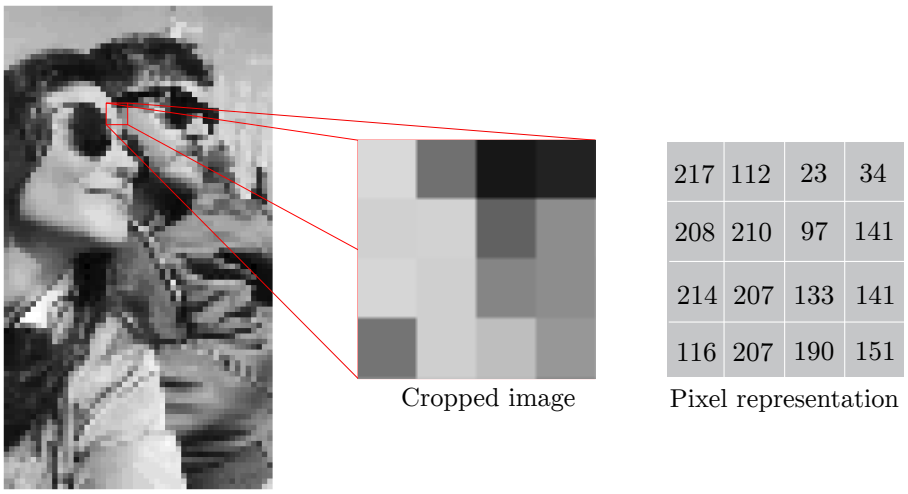


Figure 3.36.: Visualization of a grayscale image along with the pixel values of a selected region.

Convolutional layers

The main task of convolutional layers is to perform convolution operations on the input to extract features. This is achieved by using a set of filters, also known as kernels. The idea is to pass this filter over the entire image, identifying features like edges, curves, and other patterns. In other words, filters contain learnable parameters adjusted during training to capture these features.

The process of how a convolution layer performs operations is given in Fig. 3.37. The kernel matrix is initially placed at the top-left corner of the pixel matrix, and a dot product is applied to obtain between the kernel and the corresponding region in the convoluted matrix or the feature map. Next, the kernel shifts one pixel to the right, repeating the exact computation. We repeat this process until the entire input matrix has been covered.

In this example, the kernel moves one pixel at a time, a stride of 1. However, the kernel does not necessarily have to slide pixel by pixel. The stride can be set to any positive integer. Increasing the number of strides reduces the size of the output feature map, as fewer regions of the input matrix are covered.

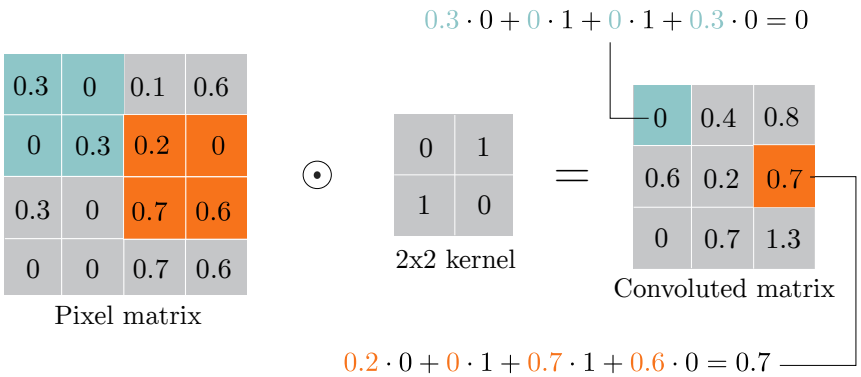


Figure 3.37.: An illustration of convolution operation on a 4×4 pixel matrix using a 2×2 filter.

Additionally, the dimensions of the input matrix decreased from 4×4 to 3×3 since no padding was used in this example. Padding means adding extra pixels (usually zeros) around the borders of the pixel matrix to preserve its original dimensions. With padding, the convoluted matrix maintains the same spatial size as the input matrix, ensuring no information is lost at the edges of the image.

Note that an activation function is applied after the convolution operation to introduce a nonlinearity to the model (see Section 3.4.3.2). The ReLU activation function is commonly the preferred choice for CNNs.

Pooling layers

Pooling in CNNs is a method of downsampling operation on the feature maps to reduce spatial dimensions. This process lowers computational costs and enhances the model's robustness by retaining essential features while discarding less significant details. The two most common types of pooling are max pooling and average pooling. Max pooling selects the highest value within the defined pooling window. On the other hand, average pooling computes the average of the pixel values within the window.

An illustration of the effects of max pooling and average pooling is provided in Fig. 3.38. In this example, a pooling window of size 2×2 scans the input matrix. The stride is set to 2, meaning that the window shifts by two pixels both horizontally and vertically. While max pooling extracts the maximum value from each window, average pooling calculates the mean of the pixel values within the pooling window. In both cases, pooling reduces the dimensions of the input matrix from 4×4 to 2×2, thereby lowering computational costs.

Fully connected layers

After convolutional and pooling layers, the extracted features are flattened into a one-dimensional vector. This vector is then fed into fully connected layers, which bridge the convolutional layers and the output layer. These fully connected layers are responsible for interpreting the extracted features.

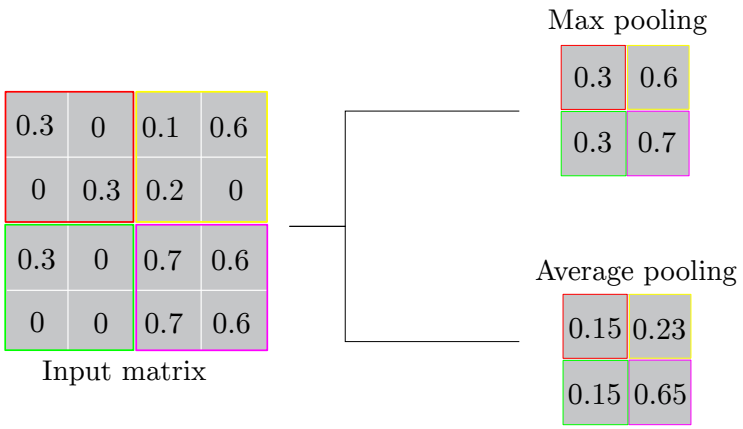


Figure 3.38.: Types of pooling.

Remark 3.4.6: How to calculate the number of trainable parameters in CNNs?

To calculate the number of trainable parameters in CNNs, we need to consider each type of layer separately. For a convolutional layer, the number of trainable parameters is determined by the following formula:

$$(w_f \times w_h \times n_p + 1) \times n_c$$

where w_f denotes the width of the filter, w_h denotes the height of the filter, n_p is the number of filters in the previous layer, n_c represents the number of filters in the current layer, and the "+1" accounts for the bias term associated with each filter.
On the other hand, for a fully connected layer and the output layer, the number of trainable parameters is computed as:

$$(n_{\text{input}} + 1) \times n_{\text{output}}$$

where n_{input} is the number of neurons (or units) in the previous layer, n_{output} denotes the number of neurons (or units) in the current layer. Similarly, the "+1" accounts for the bias term associated with each neuron in the fully connected layer.
Note that the input and pooling layers contain no trainable parameters.

3.5. Main challenges of machine learning

Despite significant advancements in machine learning, several challenges must be addressed to build robust models. These challenges arise from various aspects, including data quality, computational complexity, model interpretability, and generalization to unseen data.

These challenges can be broadly categorized into issues related to data, model complexity, and computational resources. Data-related challenges include ensuring high-quality and sufficient training data, dealing with missing values, handling imbalanced datasets, and preprocessing data effectively. Model-related challenges involve selecting appropriate architectures, tuning hyperparameters, preventing overfitting, and ensuring model interpretability. Finally, computational

challenges arise from the need for scalable algorithms, high-performance hardware, and efficient training techniques, especially when working with large datasets and deep neural networks.

3.5.1. Quantity and quality of data

The accuracy of a machine learning model heavily depends on the quantity and quality of the data used for training. A well-structured and sufficiently large dataset enables the model to learn meaningful patterns, whereas inadequate or poor-quality data can lead to unreliable predictions and poor generalization to unseen data.

Data quantity: A fundamental requirement for machine learning is having enough training data to capture underlying patterns and avoid overfitting. Insufficient data may cause the model to memorize training examples rather than learn generalizable features, leading to poor performance on new inputs. However, simply increasing the dataset size does not always guarantee better results, as the quality and diversity of the data also play a crucial role. In cases where data is limited, techniques such as data augmentation, synthetic data generation, or transfer learning can be used to compensate for the lack of labeled samples.

Data quality: Even with a large dataset, poor data quality can significantly degrade model performance. Issues such as missing values, noisy samples, redundant information, and mislabeled data can introduce bias and inconsistencies in the learning process. To ensure high-quality data, proper data cleaning, anomaly detection, and preprocessing techniques must be applied. Additionally, domain expertise is often required to curate datasets, remove irrelevant features, and validate data accuracy.

Data imbalance: A common challenge in real-world applications is imbalanced datasets, where some classes have significantly more samples than others. This can lead to biased models that favor the majority class while failing to recognize minority class patterns. Several techniques can help address this issue, including oversampling the minority class, undersampling the majority class, or using cost-sensitive learning approaches that assign different weights to different classes.

Data diversity and representativeness: For a model to generalize well, the training data must be diverse and representative of real-world scenarios. If the dataset lacks variation, the model may perform well only on specific conditions but fail when exposed to unseen distributions. Collecting data from diverse sources, performing exploratory data analysis (EDA), and using domain knowledge to identify biases are crucial steps in ensuring the dataset representativeness.

Overall, the quantity and quality of data are critical factors that determine the effectiveness of a machine learning model.

3.5.2. Data preprocessing

Data is a crucial and foundational element in machine and deep learning. Therefore, special handling is required before diving into the machine learning process. Data processing can

transform raw data into a clean and usable form suitable for machine learning models. The main steps for data preprocessing are:

- Data cleaning
- Encoding categorical features
- Transformation or feature scaling
- Feature selection
- Dimensionality reduction
- Splitting the dataset

Data cleaning: Data cleaning is the process of identifying and handling missing or corrupted, incorrect, and duplicated parts of a dataset, such as missing values, noise/anomaly, and duplicated data. For instance, some cells in a given dataset might contain missing values (NaNs), which must either be removed or imputed to ensure the data is usable. However, simply removing rows with missing values can cause a significant loss of information, as deleting an entire row discards all the data in that row, even if only a single value is missing. To mitigate this, imputation is often preferred over removal. Imputation involves filling in the missing values using either local (neighboring) or global information from the dataset. A standard method replaces missing values with the column's mean, median, or mode at which the missing value occurs. Alternatively, neighboring values can be used for imputation through forward or backward fill techniques, where missing values are replaced based on adjacent data points.

For example, in the case of a dataset containing two missing values, one approach could involve removing the rows containing the missing values (Fig. 3.39b). In this scenario, if an entire row (such as row 5) consists only of NaNs, it is reasonable to remove that row. However, removing rows containing only a single missing value, such as rows 1 and 4, would significantly lose valuable information. Instead, imputation can preserve the dataset's integrity. For instance, the missing value in row 2 can be replaced with the mean of the corresponding feature (column), while a backward fill can be used to replace the missing value in row 4 (Fig. 3.39b). This method retains useful information from other features and minimizes the effect of missing data.

Additionally, noise, outliers, and duplicates must be detected and removed. Noise and outliers can be removed from a dataset using monitoring and statistical methods. On the other hand, duplicates cause the model to be biased; thus, they can be removed using similar methods.

Encoding categorical features: Encoding is the process of converting categorical values or text-based features into numerical representations, as machine learning models require numerical input. Categorical feature encoding is typically performed using two main methods: label encoding and one-hot encoding.

In label encoding, each unique category is assigned a numerical value as an integer. However, this approach has a significant drawback: the assigned numbers can introduce an unintended hierarchical relationship between categories. For example, the model may mistakenly interpret higher numerical values as more significant than lower ones, leading to biased predictions. As

	Feature 1	Feature 2	Feature 3
Sample 1	2	4	1
Sample 2	NaN	7	2
Sample 3	3	1	9
Sample 4	7	NaN	5
Sample 5	NaN	NaN	NaN

(a)

Feature 1	Feature 2	Feature 3
2	4	1
3	1	9

(b)

Feature 1	Feature 2	Feature 3
2	4	1
4	7	2
3	1	9
7	1	5

(c)

Figure 3.39.: Handling with missing values: (a) row data with NaN, (b) removing the rows containing NaNs, (c) removing a row, and imputation by mean and backward fill.

a result, the model may incorrectly assume that a category with a more considerable numerical value holds greater importance, even though categorical values are inherently unordered. This can cause larger integers to influence the model's decisions disproportionately compared to smaller ones.

To address this issue, one-hot encoding is often used. The core idea of this approach is to create a new binary feature column for each category. In this method, a value of "1" is assigned to indicate the presence of a category in a sample, while all other category columns are marked as "0." However, this approach can lead to the "curse of dimensionality", particularly when dealing with a large number of unique categories.

Transformation or feature scaling: Feature scaling transforms numerical features to comparable scales—either by mapping them to a fixed range or by standardizing them. This removes any implicit hierarchy among features, so that large-magnitude features do not dominate smaller ones and each feature contributes more evenly to model performance.

Normalization

This method rescales a feature to a specified interval using its minimum and maximum values. For example,

$$\boldsymbol{X}_s = \frac{\boldsymbol{X} - \min(\boldsymbol{X})}{\max(\boldsymbol{X}) - \min(\boldsymbol{X})} \quad \text{(range } [0, 1]\text{)}, \tag{3.100}$$

and

$$\boldsymbol{X}_s = 2\,\frac{\boldsymbol{X} - \min(\boldsymbol{X})}{\max(\boldsymbol{X}) - \min(\boldsymbol{X})} - 1 \quad \text{(range } [-1, 1]\text{)}. \tag{3.101}$$

More generally, any desired range $[a, b]$ can be obtained via

$$\boldsymbol{X}_s = a + \frac{b - a}{\max(\boldsymbol{X}) - \min(\boldsymbol{X})}\big(\boldsymbol{X} - \min(\boldsymbol{X})\big). \tag{3.102}$$

Note that min-max normalization is highly sensitive to outliers.

Standardization

Unlike normalization, standardization does not constrain features to a specific range. Instead, it transforms the data into a normal distribution with a mean of 0 and a standard deviation of 1. The transformation formula is expressed as

$$\boldsymbol{X}_s = \frac{\boldsymbol{X} - \mu}{\sigma}, \tag{3.103}$$

where μ denotes the mean value of features and σ is the standard devition. Compared to normalization, standardization is more robust to outliers since it centers the features. However, one drawback of standardization is that it does not restrict the values within a specific range. As a result, standardized features can still exhibit large magnitudes, which might cause domination over the small magnitudes during training.

Feature selection: It is the process of selecting the features in a dataset to reduce the problem dimensionality. The core idea is to pick up the most relevant and significant features to speed up the training time and improve the model performance by reducing the model complexity. For instance, Lasso Regressor automatically performs feature selection using L1 regularization (see Section 3.3.2.2). Note that feature selection completely removes the determined features from the dataset.

Dimensionality reduction: Like feature selection, dimensionality reduction is an approach that aims to reduce the number of features in a dataset. However, while feature selection involves choosing a subset of the original features, dimensionality reduction lowers the number of features while retaining important properties of the original data. This approach addresses the curse of dimensionality by compressing the data into lower dimensions, thereby enhancing computational performance and interpretability. Among the various techniques available, one of the most commonly used dimensionality reduction techniques is Principal Component Analysis (PCA).

Splitting the dataset: Properly splitting a dataset is crucial for building, optimizing, and evaluating a machine learning model. A typical choice is to split the dataset into three subsets: the training set, validation set, and test set These terms have already been explained in the terminology part (Section 3.1).

A typical split strategy is 70% of the dataset for training, 15% for validation, and 15% for testing, though the ratios depend on the dataset. If the model has no hyperparameters to optimize, the validation set may be unnecessary, allowing a more significant portion of the dataset to train the model.

On the other hand, for large datasets containing many samples, the training set ratio can be increased since the model can substantially capture the hidden patterns. This improves the model's generalization ability and enhances its performance on the test set.

3.5.3. Underfitting vs. overfitting

In machine learning, overfitting is a phenomenon where a trained model has ideal predictions on the training set but fails to generalize to new, unseen data. In other words, the model not only captures the hidden behavior in the data but also learns the noise. As a result, the model with overfitting mimics the training data, including random fluctuations, causing poor performance on test data.

On the other hand, underfitting occurs when the model is too simple and cannot identify the underlying patterns in the data. This means that it does not perform poorly only on new data but also produces poor predictions even on the training set. Therefore, a model with overfitting cannot capture the underlying behavior in data, causing large errors.

These two phenomena are best understood in the context of the bias-variance tradeoff.

Bias and variance tradeoff: The bias-variance tradeoff is a fundamental concept in machine learning to help explain the relationship between underfitting and overfitting, as illustrated in Fig. 3.40. It describes how prediction errors can be decomposed into two main components: bias and variance.

- Bias defines the error when the model assumptions are too simple. Therefore, the model predictions do not align with the target values. Therefore, a model with high bias tends to be underfitting since the model cannot capture the underlying patterns in the data. Such models are prone to making inaccurate predictions on the training and test. For instance, a linear model might have a high bias in capturing underlying patterns in data defined using a cubic function since it is too simple.

- Variance, on the other hand, refers to the model's sensitivity to variations or fluctuations in the training data. A model with high variance tends to overfit, meaning that the model also captures the noise, which causes a highly sensitive model. Thus, the model cannot generalize, delivering poor predictions on new, unseen data. For example, a third-order polynomial model, which aims to learn the hidden behavior of linear problems, might have high overfitting since it might also capture the noise and fluctuations. In other words, the model will act on those outliers as a part of actual behavior and make unrealistic predictions for those data points.

Balancing bias and variance is essential for building models that generalize well to both training and test data. An effective model attains an appropriate level of complexity: it is expressive enough to capture meaningful patterns in the training set, yet not so sensitive that it reacts strongly to noise or random fluctuations.

To remedy underfitting, one can increase model capacity, introduce additional informative features, or train for more iterations. Conversely, to avoid overfitting, one may reduce model complexity, apply regularization, remove less relevant features, and use cross-validation to guide model selection and hyperparameter tuning.

Cross-validation : Cross-validation is a technique in machine learning to improve the model accuracy and generalization. Cross-validation's core idea is to first split the training data into a given number of subsets, namely *folds*. While the model is trained on some of those folds, its

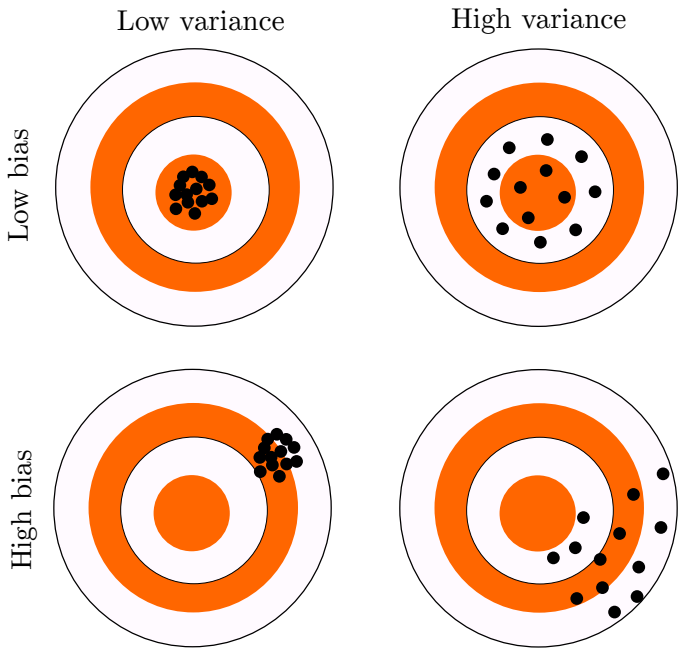


Figure 3.40.: Bias-variance tradeoff.

accuracy is tested on the rest. The primary purpose of cross-validation is to avoid overfitting and select hyperparameters during training.

The common cross-validation techniques are k-fold cross-validation, stratified k-fold cross-validation, and leave-one-out cross-validation.

K-fold cross-validation

K-fold cross-validation is one of the simplest and most commonly used techniques. It follows these steps:

- Split the dataset into k equally sized folds.
- Assign $k-1$ folds as a pseudo training set and the remaining fold as a pseudo test set.
- For each iteration:
  - Train the model on $k-1$ folds and evaluate it on the remaining fold.
- In the next iterations, rotate the test fold, ensuring each fold is used for validation exactly once.
- Compute the average performance across all k iterations to obtain a final evaluation metric.

Fig. 3.41 illustrates the concept of k-fold cross-validation with $k = 5$, resulting in five iterations. The model is trained on four of the five folds at each iteration and tested on the remaining fold. Once all five iterations are completed, the test performance is averaged to estimate the model's generalization ability.

One question might arise: What happens to the final model after cross-validation? The models trained during cross-validation are not used as the final model. Instead, after completing cross-validation, the model should be retrained on the entire dataset to produce the final, ready-to-deploy version. This is because the primary purpose of cross-validation is to assess generalization performance and optimize hyperparameters rather than to create a deployable model. We refer the reader to [59] for other cross-validation techniques.



Figure 3.41.: An illustration of 5-fold cross-validation.

3.6. Hyperparameter optimization

In machine learning, hyperparameter optimization, or tuning, selects the optimal parameters to improve the model performance. For instance, optimization parameters, such as learning rate, or network parameters, e.g., number of layers and neurons can be optimized during training. Standard hyperparameter optimization techniques are manual search, grid search, random search, and Bayesian optimization.

Manual search: In machine learning, a manual search is a process of tuning the model hyperparameters manually. It is not an automated process. Therefore, it requires a manual construction of all possible combinations of the target hyperparameters. Therefore, manual research is quite an intuitive process requiring experience.

Grid search: Grid search, also known as exhaustive search, is a brute-force approach that systematically explores all possible combinations of hyperparameters for a given machine learning model. It evaluates each combination to identify the optimal set of hyperparameters that maximizes model performance. However, this method is computationally expensive, especially when dealing with multiple hyperparameters.

For instance, consider a model with five hyperparameters, each having 10 possible values. In this case, grid search generates a total of $10 \times 10 \times 10 \times 10 \times 10 = 100{,}000$ configurations. Consequently, $100{,}000$ unique models must be trained and evaluated to determine the optimal hyperparameter setting, making the approach impractical for large search spaces.

The main advantage of grid search is that it guarantees that each possible configuration is considered to find the optimal configuration. Therefore, it can be practical when the search space is relatively small.

Random search: Compared to grid search, which exhaustively evaluates every possible combination of hyperparameters within the search space, random search involves randomly sampling combinations of hyperparameters of the defined search space. Thus, this approach significantly reduces the computational intensity. However, unlike grid search, it does not guarantee that every possible configuration is performed during optimization. Consequently, within the explored search space, the selected model might not be the optimal model. An illustration of the difference between grid search and random search is given in Fig. 3.42.

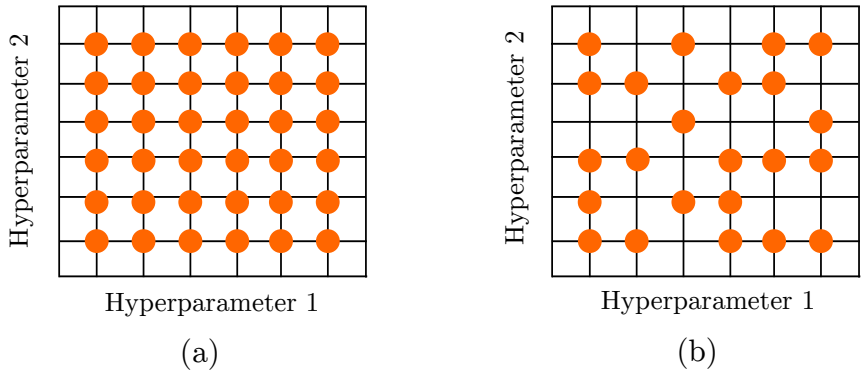


Figure 3.42.: Hyperparameter configuration for (a) grid search and (b) random search.

For a comprehensive review of Bayesian optimization methods, we refer the reader to [159].

3.7. Illustrative example: Using CNNs to predict load in beam models

As a use case example, we implement a CNN model to predict the load in a beam model. The beam configuration is shown in Fig. 3.43a, where a clamped beam is subjected to a uniformly distributed load $q(x)$ with magnitude q. The analytical solution for the beam displacement is given by

$$w(x) = -\frac{qx^2}{24EI}(x - L)^2, \tag{3.104}$$

where Young's modulus is set to $E = 1$, the moment of inertia to $I = 1$, and the beam length to $L = 1$. In this formulation, w represents the displacement, while q denotes the applied load. Note that this example is presented without specific units.

Since the applied load is unknown in this example, the aim is to predict load values based on images that depict deflection curves. The effect of the load on the displacement, or the deflection curve, is illustrated in Fig. 3.43b. As expected, a smaller load results in a less pronounced deflection, whereas larger load values cause greater beam deformation.

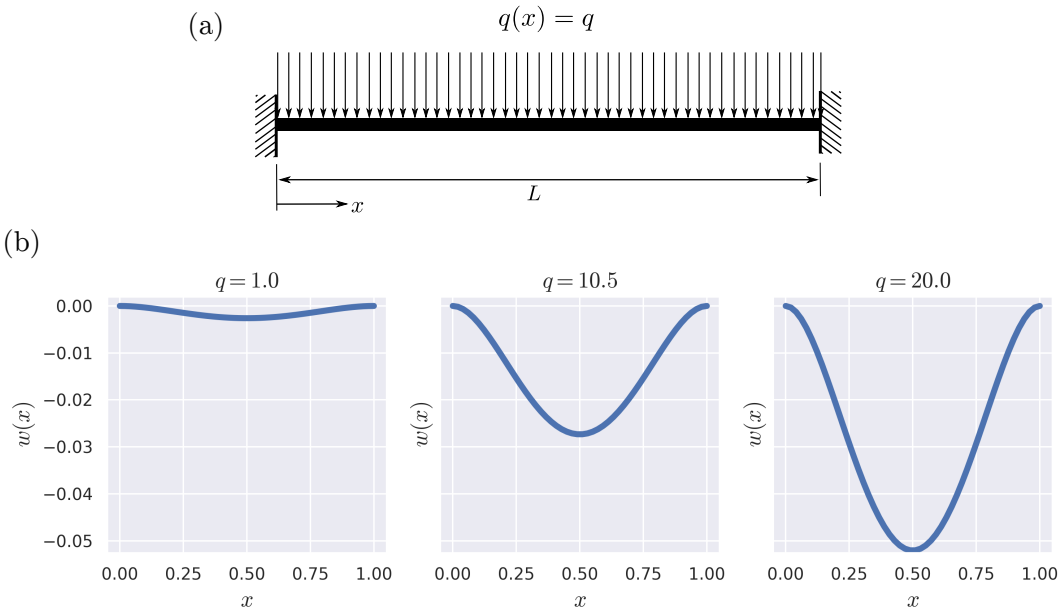


Figure 3.43.: (a) A clamped beam under a uniformly distributed load $q(x)$. (b) Deflection curves for different loads.

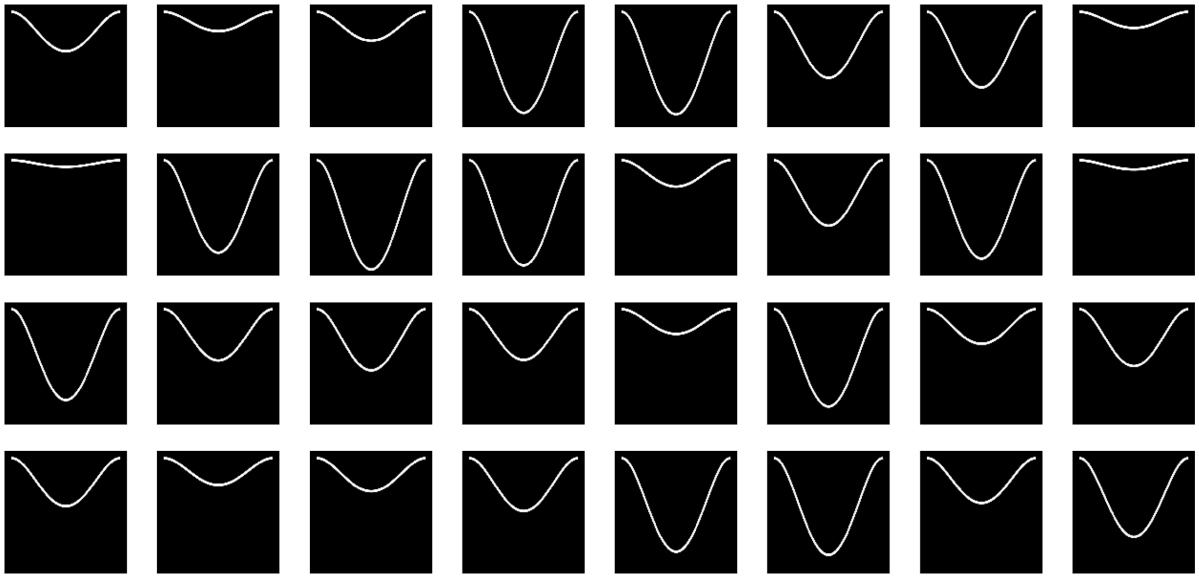


Figure 3.44.: Images for deflection curves of the beam example under different loads.

We generate 200 deflection images to construct a CNN, some shown in Fig. 3.44. These images are obtained by applying different load values ranging from 1 to 20.

First, the 200 images are split into three subsets: a training set with 162 images, a validation set with 18 images, and a test set with 20 images. Each image has dimensions of 128×128.

Next, we construct a CNN model with the following characteristics:

- **Input layer**: The input shape is defined by the image dimensions, specifically image height (128), image width (128), and channels (1). Since the images are grayscale, they contain a single channel.
- **First convolutional layer**: This layer extracts important features from the input image by employing 32 filters, each of size 3×3, to the input image. The ReLU activation function is used to introduce non-linearity. The convolutional layer is followed by a max pooling layer with a 2×2 pool size to reduce the spatial dimensions of the output, helping to decrease the computational intensity and prevent overfitting.
- **Second convolutional layer**: Similar to the first layer, this layer applies 64 filters of a window size 3×3. Again, the ReLU activation is used to capture more complex features from the image. Another max pooling layer with a 2x2 pool size is applied to further reduce the spatial size of the feature maps.
- **Flattening layer**: The output from the convolutional and pooling layers is a multi-dimensional tensor. Thus, the flattening layer converts this multi-dimensional output into a one-dimensional vector, which can then be passed to fully connected layers.
- **Fully connected layer (Dense layer)**: This layer consists of 128 neurons and uses the 'ReLU' activation function. It processes the flattened output and helps in learning non-linear combinations of the features extracted by the convolutional layers.
- **Output layer**: For the task of regression, the output layer contains a single neuron. This neuron outputs the continuous value denoting the prediction for the image, namely the load. In the output layer, no activation function is applied.

The deployed CNN model has $7{,}391{,}873$ trainable parameters. Adam optimizer is utilized for training with a learning rate of 0.001. The mean squared error is chosen as the cost function, a proper loss function for predicting continuous values.

Training is performed for 500 epochs with a batch size of 16, which speeds up the training and avoids memory problems due to large matrices. This configuration helps accelerate the training process while still maintaining robust learning. The total training is approximately $80\,\mathrm{s}$.

A comparison between the actual load (q_{act}) and the CNN predictions (q_{pred}) is presented in Fig. 3.45. As shown on the left side of the figure, the predicted values closely align with the actual load values, regardless of the load magnitude. Two randomly selected samples illustrate that the CNN architecture effectively captures the underlying patterns in the displacement curves to identify the applied load.

Further improvements can be achieved through hyperparameter optimization to enhance model accuracy and by refining the model complexity to reduce computational time.

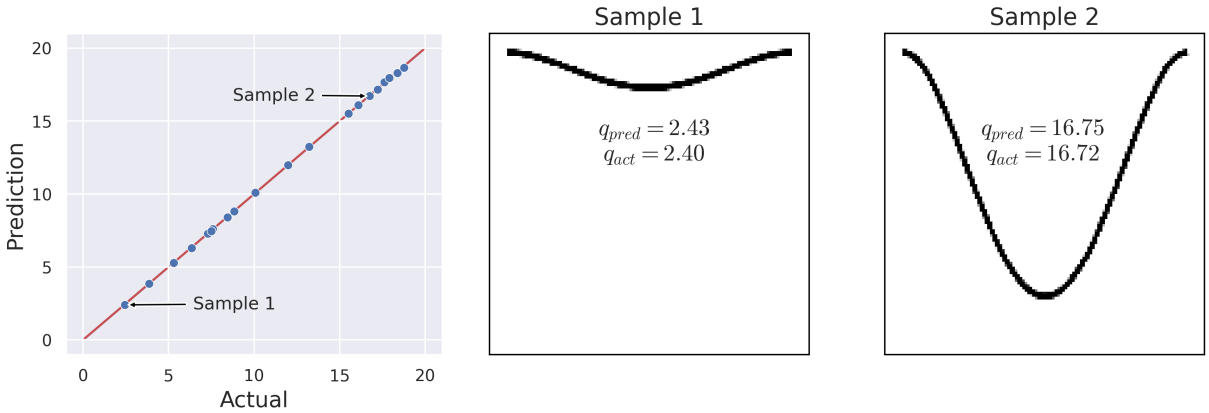


Figure 3.45.: Comparison of the actual load (q_{act}) and the CNN predictions (q_{pred}).

4. Application of Machine Learning Techniques to Rough Surface Contact Problems

4.1. Data-driven problem: learning a surrogate model

Despite the huge steps forward in numerical solution schemes as explained in Section 2.4.1, and even though the available computational resources continuously increased, the high-fidelity numerical solution of a rough contact problem can still be cumbersome. That is indeed the case, exemplified by parameter studies where numerous simulations have to be run, causing the computational cost to grow rapidly and eventually become prohibitive.

In this regard, a possible solution is offered by the huge advancement and development experienced in the last few years by efficient surrogate modeling techniques. Among them, a promising approach, namely data-driven surrogate modeling, is based on experimental or synthetically generated data, thus deliberately excluding physical knowledge of the system; instead, it extracts underlying patterns in provided data by deploying machine learning and deep learning algorithms. Many application cases of data-driven surrogate modeling techniques specifically crafted for the realms of science and engineering are available in the literature. [27, 87, 109, 160] These techniques aim to achieve high accuracy while requiring minimal computational or prediction time.

Surrogate modeling aims to reduce computational costs by approximating complex, computationally expensive models with more efficient alternatives. Often, this reduction in computational cost comes at the expense of model accuracy and generality. Surrogate models can take various forms, such as analytical models [45], physics-informed machine learning models [146], or purely data-driven approaches based on machine learning techniques [58]. The choice of surrogate model depends on the desired balance between model accuracy, generality, and computational efficiency.

Mathematically, the data-driven surrogate modeling task can be formulated as a regression task. Given a dataset (or database) $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1,\dots,N}$ of N pairs of inputs $\boldsymbol{x}_i$ and corresponding outputs $y_i = f(\boldsymbol{x}_i)$, the goal is to learn a function $\hat{f}_{\boldsymbol{w}} : \mathbb{R}^d \to \mathbb{R}; \boldsymbol{x} \mapsto \hat{y} = \hat{f}_{\boldsymbol{w}}(\boldsymbol{x})$ that approximates the true model $f : \mathbb{R}^d \to \mathbb{R}; \boldsymbol{x} \mapsto y = f(\boldsymbol{x})$ such that

$$\hat{f}_{\boldsymbol{w}} \approx f, \tag{4.1}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the d-dimensional vector of input parameters, $y \in \mathbb{R}$ is the scalar quantity of interest (or target quantity), $\hat{y} \in \mathbb{R}$ is the approximation of the quantity of interest by the surrogate model, and $\boldsymbol{w}$ are the free parameters of the surrogate model. Learning the surrogate model $\hat{f}_{\boldsymbol{w}}$ typically involves minimizing a loss function

$$\mathcal{L}(\mathcal{D}, \hat{f}_{\boldsymbol{w}}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{4.2}$$

with respect to the free parameters $\boldsymbol{w}$, where $\hat{y}_i = \hat{f}_{\boldsymbol{w}}(\boldsymbol{x}_i)$ denotes the surrogate prediction at sample point $\boldsymbol{x}_i$.

Data-driven surrogate modeling techniques can be developed for various purposes such as prediction, parameter identification, classification, and anomaly detection. When focusing on rough surfaces, these techniques find application across a diverse range of scenarios in fluid mechanics, tribology, and contact mechanics. Sanhueza et al.[151] deployed machine learning as a parameter identifier to predict the behavior of turbulent flows past rough surfaces. They applied convolutional neural networks on the height map of a rough surface to identify the corresponding skin friction values and Nusselt numbers. Elangovan et al.[50] employed multiple linear regression to identify surface roughness in turning operations by considering machining parameters such as speed, depth of cut and tool wear. The resulting database comprises statistical features derived from vibration signals, and, in addition, principal component analysis is applied to reduce the dimensionality of the input space. In the study by Prajapati et al.,[130] the researchers constructed a surrogate model utilizing a regression-based Multi-Layer Perceptron (MLP) to predict elastohydrodynamic lubrication parameters, specifically, the traction coefficient and asperity load ratio. These predictions were customized for different surface topographies encountered in rough contact problems. Similarly, the authors in[20] applied the machine learning radial basis function method to identify the coefficient of friction on textured and porous surfaces. In addressing rough surface problems, there are numerous additional examples where machine learning techniques have been applied to identify parameters associated with surface roughness, lubrication and wear. This is well-documented in the reviews by Syam[71] and Batu[12]. However, only a small number of studies have delved into predicting the mechanical responses associated with contact problems on rough surfaces, such as the prediction of effective contact area and contact force. In the work led by Rapetto et al.,[137] the authors explored the ability of neural networks to predict the relation between roughness parameters and the effective contact area. The generated dataset comprises statistical features derived solely from 2D profiles, with the authors noting that the error tends to be higher for small values of the effective contact area. Similarly, Kalliorinne et al. [86] explored the ability of multitask neural networks to predict the mechanical response of contact, including the effective contact area and contact pressure, across surface topographies generated using distinct methods based on a given height probability distribution and power spectrum. In research conducted by Jiang et al.,[82] a new method was proposed to predict the effective contact area of fractal rough surfaces. This method employs a regression model based on least-square support vector machines (SVMs). As the fractal surfaces are generated using a deterministic Weierstrass-Mandelbrot (WM) model, the resulting database does not include the statistical features but comprises solely the roughness parameters inherent to the WM model and the external load. Furthermore, the generated database has a very limited range due to costly FEM simulations. Additionally, they do not report generalization errors on the unseen data since only a single sample is left for testing. Another approach for generating a surrogate model, namely Physics-Informed Neural Networks (PINNs), incorporates the physical laws of a governing problem and the additional data into the loss function of the neural networks.[134] For instance, in

the context of contact mechanics, Sahin et al.[144] deployed PINNs as a fast-to-evaluate surrogate model to solve forward and inverse contact mechanics problems on macro-scale.

4.2. Surrogate model for prediction of effective contact area

Given this mathematical framework, we propose a data-driven surrogate modeling workflow to reduce the cost of evaluating the effective contact area in rough surface contact problems, which depend on the far-field displacement Δ and the surface height field $\boldsymbol{z}$, as defined in Section 2.4.2. In this context, the system is described by the function f introduced in Eq. (4.4) and illustrated in Fig. 4.1a. The input vector is defined as $\boldsymbol{x} = [\Delta, \boldsymbol{z}]$, and the target quantity is $y = \mathcal{A}_\mathrm{e}$. We aim to approximate f using a surrogate model $\hat{f}_{\boldsymbol{w}}$, as depicted in Fig. 4.1a.

A central challenge in this problem is the high dimensionality d of the input vector, primarily due to the discretized height field $\boldsymbol{z}$. To address this, we adopt a dimensionality reduction approach: instead of using $\boldsymbol{z}$ directly, we represent it through a set of summary statistics or statistical descriptors $\boldsymbol{\vartheta}$ (cf. Section 2.4.4). This significantly reduces the input dimensionality, as $\boldsymbol{\vartheta}$ captures the essential characteristics of $\boldsymbol{z}$ with far fewer parameters.

Moreover, while the dimensionality of $\boldsymbol{z}$ increases with the grid size (e.g., 1024 for a 32×32 grid and 16384 for a 128×128 grid), the number of statistical parameters in $\boldsymbol{\vartheta}$ remains fixed. In this study, we compute 22 such statistical parameters, enabling efficient and scalable surrogate modeling.

Ultimately, the surrogate model $\hat{f}_{\boldsymbol{w}}$ can be equivalently described by the following set of equations:

$$\hat{\mathcal{A}}_\mathrm{e} = \hat{f}_{\boldsymbol{w}}(\Delta, \boldsymbol{z}) \iff \begin{cases} \boldsymbol{\vartheta} & = g(\boldsymbol{z}), \\ \hat{\mathcal{A}}_\mathrm{e} & = \hat{h}_{\boldsymbol{w}}(\Delta, \boldsymbol{\vartheta}), \end{cases} \tag{4.3}$$

Here, g is a known deterministic function that maps the height field $\boldsymbol{z}$ to a vector of statistical descriptors $\boldsymbol{\vartheta}$ (as detailed in Section 2.4.4), and $\hat{h}_{\boldsymbol{w}}$ is a regression model trained from data. This model is learned using the dataset $\mathcal{D} = \{([\Delta_i, \boldsymbol{\vartheta}_i], \mathcal{A}_\mathrm{e}^i)\}_{i=1,\dots,N}$, where each $\boldsymbol{\vartheta}_i = g(\boldsymbol{z}_i)$.

A graphical representation of Eq. (4.3) is shown in Fig. 4.1a. Once the model $\hat{h}_{\boldsymbol{w}}$ is trained, evaluating the surrogate on a new, unseen rough surface involves computing its statistical descriptors via g and applying the learned regression model as outlined in Eq. (4.3). While the original model f could also be used to predict the target quantity, doing so would incur significant computational cost. In contrast, evaluating the surrogate model is substantially faster, making it particularly well-suited for multi-query analysis scenarios (cf. previous section), where thousands of such model evaluations may be required.

In summary, the key parameters governing the computation of the effective contact area A_e are the imposed load Δ and the surface height field $\boldsymbol{z}$. Thus, in abstract terms, solving the problem means evaluating the following function:

$$A_e = f(\Delta, \boldsymbol{z}). \tag{4.4}$$

This problem setup is illustrated in Fig. 4.1a. We know that such a functional relationship exists, and that approximate well-known general closed-form expressions for f are available from many

different classical analytical models. However, a lack of overall general agreement in terms of outcome is experienced, with different theories leading to slightly but non-negligible differences in their findings [188]. Furthermore, the derivation of closed form expressions comes with the cost of strong starting assumptions, e.g., in terms of the topography that must be considered. Instead, evaluating the function f without a priori simplifications requires the numerical solution of a boundary integral equation that models the whole contact mechanics problem. As described in Section 2.4.1, various numerical schemes can be employed to solve this problem. In this work, we use the Boundary Element Method (BEM) for which has already been provided in Section 2.4.5.

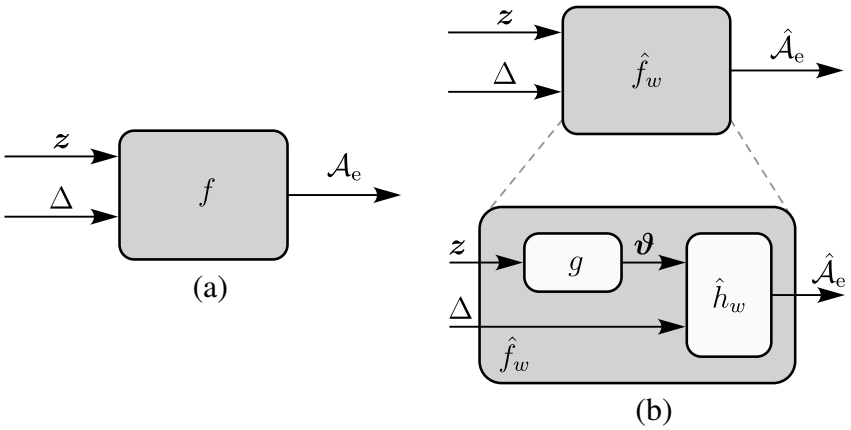


Figure 4.1.: Surrogate modeling framework for the effective contact area in rough surface contact: (a) The reference model f computes $\mathcal{A}_\mathrm{e}$ as a function of far-field displacement Δ and surface height field $\boldsymbol{z}$, requiring costly numerical simulations; (b) the surrogate model $\hat{f}_{\boldsymbol{w}}$ approximates this using a fast-to-evaluate regression model $\hat{h}_{\boldsymbol{w}}(\Delta, \boldsymbol{\vartheta})$, where $\boldsymbol{\vartheta} = g(\boldsymbol{z})$ is a vector of statistical parameters. The function g is a known deterministic mapping that enables dimensionality reduction of the input vector.

In summary, the following sections present a surrogate modeling framework for predicting the effective contact area resulting from the interaction between a linear elastic half-space and a rigid rough surface. Rough topographies are generated utilizing the RMD algorithm to obtain self-affine fractal surfaces. The generated database includes a wide range of parameters, viz. model parameters, material parameters and roughness parameters obtained through statistical characterization of topographies. These statistical roughness parameters are calculated based on the height field, summits (maxima in 3D) and peaks (maxima in 2D) of fractal surfaces. In contrast to earlier investigations in the literature, various machine learning algorithms are deployed as fast-to-evaluate models utilizing a database generated with Boundary Element Method simulations. As mentioned earlier, BEM stands out as an efficient approach, providing highly accurate results with significantly reduced computational time compared to other numerical schemes. Thus, we couple state-of-the-art data-driven methodologies with enriched simulation data based on BEM. These methodologies include Decision Tree, Random Forest, Support Vector Machine (SVM) Regressor, Adaptive Boosting (AdaBoost), Gaussian Process (GP) Regressor, Gradient Boost, Kernel Ridge, eXtreme Gradient Boosting (XGBoost), and Multi-Layer Perceptron (MLP) Regressor. Hyperparameter optimization is performed to identify the best configuration for each

model type. This optimization is carried out through grid search combined with cross-validation to identify the optimal model parameters. For quantitative comparisons among surrogate models, several accuracy metrics are computed. These metrics include the normalized mean square error (nMSE), normalized mean absolute error (nMAE), normalized max-error (nMaxE) and the R^2 score. Selecting an optimal surrogate model requires balancing accuracy and prediction performance, depending on the specific requirements of the application task. Our results indicate that kernel-based regressors generally perform very well in the given context of rough surface contact simulations. Among them, the Kernel Ridge model offers the best trade-off between accuracy and computational efficiency. It achieves high predictive accuracy, low prediction time, and minimal training overhead, making it a strong candidate for general-purpose surrogate modeling. Therefore, the Kernel Ridge model is eventually compared to the BEM model in terms of overall computational time. This assessment encompasses not only the prediction time of the trained model but also considers the training, tuning, and data generation time. Going one step further, to assess the model's generalization capability, the optimal model is tested in a new simulation scenario, which is not seen during the training process. The findings reveal that the developed surrogate model performs well in this previously unseen scenario. Specific scientific contributions of the paper can be outlined as follows:

- Development of an efficient fast-to-evaluate surrogate modeling workflow to predict the effective contact area for rough surface contact problems.
- Establishment of an informative database encompassing a diverse set of parameters combined with BEM simulations.
- Generalization of the machine learning model by testing on the unseen simulation scenario.

The remainder of this chapter is structured as follows. In Section 4.3, we elucidate a methodology for surrogate modeling as an alternative to conventional techniques. This includes the development of a surrogate modeling workflow, the generation of a database exploiting BEM simulations, and the introduction of fast-to-evaluate surrogate models as baseline models and tuned models via hyperparameter optimization. Furthermore, several accuracy metrics are introduced to validate the accuracy of surrogate models. Section 4.4 covers an evaluation of baseline and tuned models comparing them in terms of accuracy and performance. Among the investigated machine learning approaches, the so-called optimal model is compared with the BEM model in terms of performance. Section 4.5 concludes the manuscript by summarizing the main findings and providing insights into future research directions.

4.3. Methodolody

4.3.1. Surrogate modeling workflow

After mathematically defining the problem setup in the previous section, we describe the full modeling workflow from a practical perspective. The proposed modeling workflow is illustrated in Fig. 4.2. First, we define the input parameter space and sample from it. The input parameter space consists of the statistical parameters of a rough surface and the far-field displacement. We use controlled sampling [63] to explore this space. While the far-field displacement can be

sampled directly, the surface parameters are sampled indirectly: a random rough surface is first generated using the RMD algorithm (see Section 2.4.3), and its statistical parameters are then computed as described in Section 2.4.4.

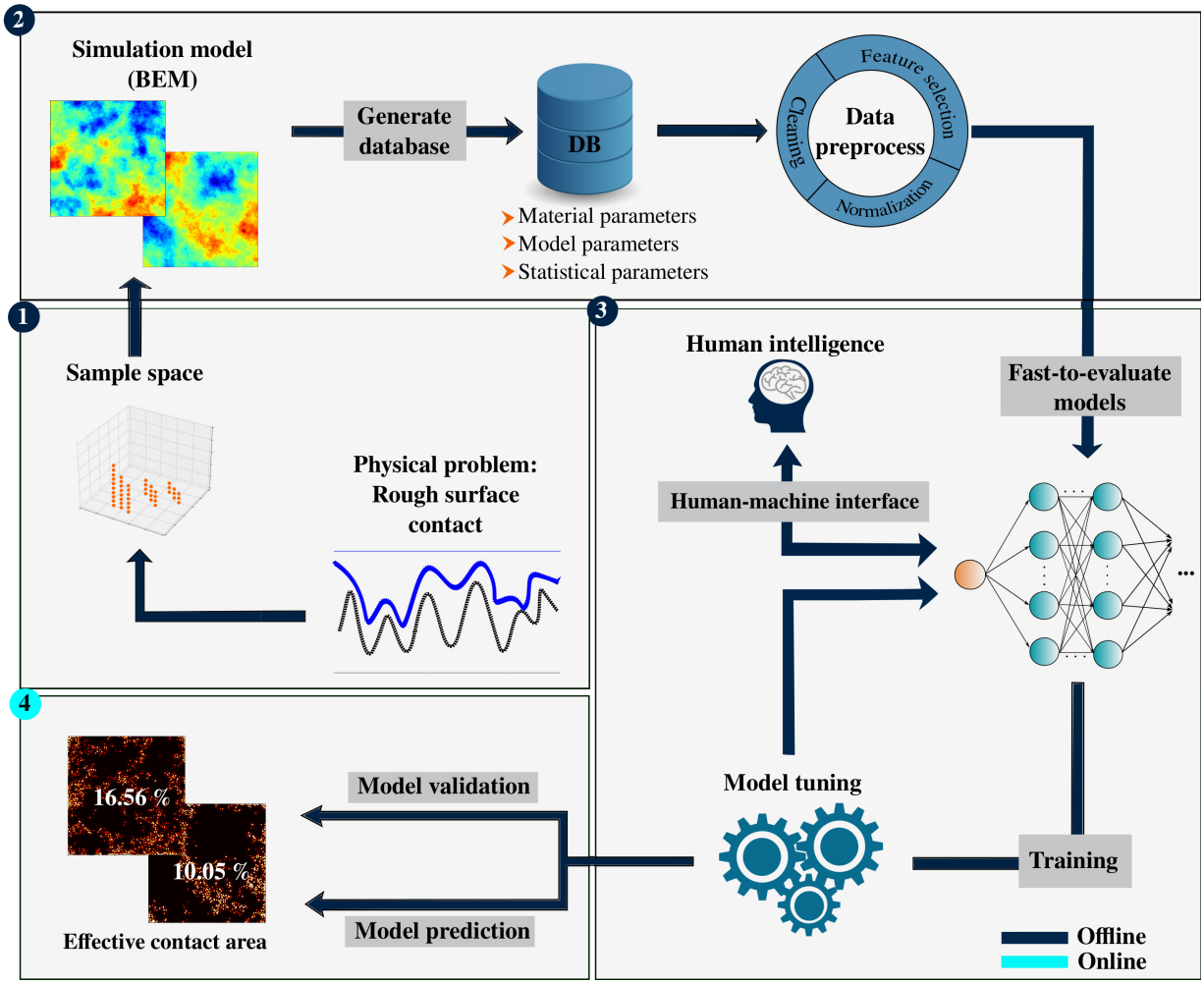


Figure 4.2.: Data-driven surrogate modeling workflow to solve rough surface contact simulations.

Second, we run BEM simulations for all input samples (pairs of rough surfaces and far-field displacements) to compute the corresponding effective contact areas. In this way, we construct a database $\mathcal{D}$ consisting of input-output samples. An important practical aspect of data-driven surrogate modeling is data preprocessing, which follows database generation. Not all parameters may be relevant, and some data points might be incomplete, e.g., due to failed BEM simulations. Thus, methods such as feature selection and data cleaning are applied to obtain the final form of the database [59]. Feature selection is the process of identifying and selecting a subset of the most relevant input parameters (features) from the dataset that contribute the most to the model's predictive performance. The goal is to improve model accuracy, reduce overfitting, and decrease training time by removing irrelevant, redundant, or noisy features. Data cleaning is used to remove missing values resulting from failed simulations. Moreover, data normalization [59, 156] is applied to scale the problem and improve model performance and stability.

Third, various machine learning regression models are trained as fast-to-evaluate surrogate models, and hyperparameter optimization is performed to tune them further. Due to the black-box nature of purely data-driven methods, human oversight is necessary throughout the training process. This includes defining the hyperparameter search space, managing the workflow, making critical decisions, and selecting the optimal model. In this work, the performance of the trained surrogate models is evaluated on unseen but known test data (a subset of the database) using accuracy metrics, such as the mean square error (see Section 4.3.4).

Finally, in the fourth step, the surrogate model can be used to predict the quantity of interest for any new input point, for example during a multi-query analysis such as a parameter study or uncertainty quantification. This stage is referred to as the online phase. The first three steps (cf. Fig. 4.2) take place in the offline phase, meaning they are performed only once. It is worth noting that model validation is also carried out in the online phase, as it is separate from the training process.

The workflow is implemented in our in-house package called *DeepSim* (`https://github.com/imcs-compsim/deep_sim`), which is developed to automate numerical simulations, and machine learning frameworks.

4.3.2. Generation of database

To generate the database, we first sample from the input parameter space and then run BEM simulations for each sample. The input space includes the statistical parameters of rough surfaces and the far-field displacement. As described in Section 2.4.2, the material properties do not influence the contact pattern. Therefore, parameters such as Young's modulus and Poisson's ratio are held constant throughout this study. Additionally, more samples are placed in regions of greater interest through controlled sampling. The far-field displacement is sampled directly from the range $5\,\mu\mathrm{m}$ to $45\,\mu\mathrm{m}$, with a higher concentration of samples for $\Delta \leq 25\,\mu\mathrm{m}$ (see Fig. 4.3a and Fig. 4.3b), since larger far-field displacements lead to prohibitive computational cost (cf. Fig. 2.20). Statistical surface parameters are sampled indirectly: rough surfaces with a 128×128 grid are generated using the RMD algorithm (see Section 2.4.3), and their statistical properties are computed as described in Section 2.4.4. BEM simulations are performed for each input sample to compute the corresponding effective contact area.

The resulting input-output pairs form the desired database. In total, we run 15,878 BEM simulations, with an average simulation time of $190\,\mathrm{s}$ per model evaluation. A comprehensive list of all parameters used in this study is provided in Table 4.1. The table includes fixed parameters (e.g., material and model parameters) used to generate the database, along with the observed ranges for all input and target quantities. The input features for the data-driven surrogate models consist of the far-field displacement and 22 statistical parameters describing the rough surfaces, as listed in Table 4.1. The database includes one quantity of interest, the effective contact area, which serves as the model output.

Note that the table presents raw data, before preprocessing (see Section 4.3.1). In particular, an L_2-normalization is applied to ensure each feature vector has unit length:

$$X_i = \frac{x_i}{\sqrt{\sum_{i=1}^{n_f} x_i^2}}, \tag{4.5}$$

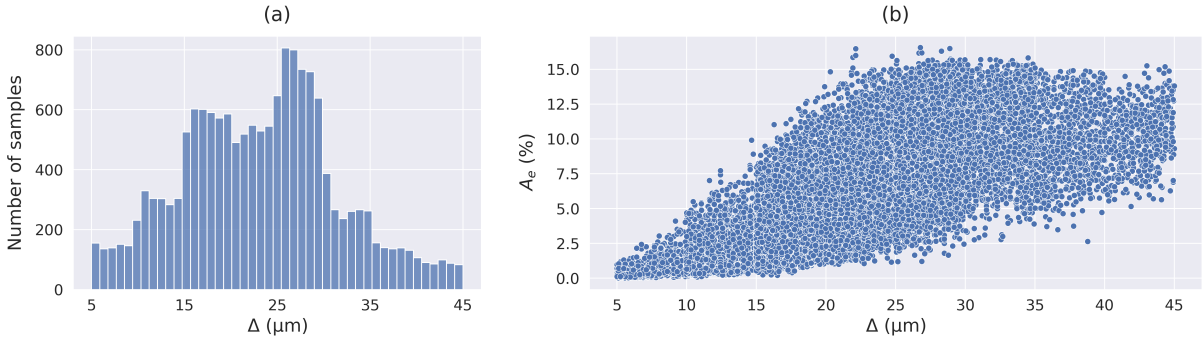


Figure 4.3.: Visualization of the database. (a) distribution of far-field displacement, and (b) scatter plot showing the target effective contact area against the feature far-field displacement. Each point represents a triplet consisting of a rough surface, a far-field displacement, and the corresponding effective contact area. Variations in effective contact area at a fixed far-field displacement are due to differences in the characteristics of the rough surfaces.

where x_i is the non-normalized feature, n_f is the number of features, and X_i is the normalized feature.

After preprocessing, the database is used to train fast-to-evaluate surrogate models. We adopt an 80:20 train-test split, resulting in a training set with 12,703 samples and a test set with 3,175 samples. Each sample has 23 features and one scalar target.

4.3.3. Fast-to-evaluate modeling with hyperparameter optimization

To find the optimal data-driven model, we compare different regression models, including Decision Tree [181], Random Forest [33], Support Vector Machine (SVM) Regressor [169], Adaptive Boosting (AdaBoost) [55], Gaussian Process (GP) Regressor [138], Gradient Boost [56], Kernel Ridge [171], eXtreme Gradient Boosting (XGBoost) [40], and Multi-Layer Perceptron (MLP) Regressor [62].

Figure 4.4 illustrates the classification of all the deployed machine learning algorithms based on their characteristics, categorized into three main groups: tree-based models, kernel-based models, and neural networks.

Tree-based models are further divided into single and ensemble methods, depending on whether the model is a standalone decision tree or it is a combination of multiple decision trees. Boosting algorithms, on the other hand, sequentially construct ensemble algorithms to enhance prediction accuracy.

Kernel-based regression methods are non-parametric techniques that use kernel functions to map input data into high-dimensional feature spaces, allowing them to capture complex, nonlinear relationships. They leverage the *kernel trick* to compute inner products in the transformed space without explicitly performing the transformation [110]. Among the kernel-based models used, Kernel Ridge and SVM Regressors are deterministic, while the GP Regressor is probabilistic. When using the same kernel and hyperparameters, the mean prediction of a GP Regressor is equivalent to that of Kernel Ridge Regression (see Appendix A). The SVM Regressor aims to

Type	Name	Symbol	Range
Model parameters	Scan length (µm)	L	1,000
	Hurst exponent (-)	H	$[0.5, 0.8]$
	Far-field displacement (µm)	Δ	$[5, 45]$
Statistical parameters	Mean of peaks (µm)	$\bar{z}_{\mathrm{p}}$	$[8.20, 47.51]$
	Root mean square of peaks (µm)	RMS$[z_{\mathrm{p}}]$	$[2.54, 15.49]$
	Kurtosis of peaks (-)	K$[z_{\mathrm{p}}]$	$[1.56, 7.13]$
	Skewness of peaks (-)	Sk$[z_{\mathrm{p}}]$	$[-1.34, 1.36]$
	Density of peaks (-)	ρ_{p}	$[0.19, 0.27]$
	Mean of curvature of peaks (µm^{-1})	$\bar{\kappa}_{\mathrm{p}}$	$[0.015, 0.066]$
	Kurtosis of curvature of peaks (-)	K$[\kappa_{\mathrm{p}}]$	$[-0.07, 5.35]$
	Skewness of curvature of peaks (-)	Sk$[\kappa_{\mathrm{p}}]$	$[0.55, 1.15]$
	Bandwidth parameter in x direction (-)	α_x	$[4.06, 46.19]$
	Bandwidth parameter in y direction (-)	α_y	$[4.19, 48.52]$
	Mean of asperities (µm)	$\bar{z}_{\mathrm{a}}$	$[8.54, 48.18]$
	Root mean square of asperities (µm)	RMS$[z_{\mathrm{a}}]$	$[2.56, 15.60]$
	Kurtosis of asperities (-)	K$[z_{\mathrm{a}}]$	$[-1.45, 4.26]$
	Skewness of asperities (-)	Sk$[z_{\mathrm{a}}]$	$[-1.34, 1.48]$
	Density of asperities (-)	ρ_{a}	$[0.08, 0.13]$
	Mean of curvature of asperities (µm^{-1})	$\bar{\kappa}_{\mathrm{a}}$	$[0.017, 0.072]$
	Root mean square of curvature of asperities (µm^{-1})	RMS$[\kappa_{\mathrm{a}}]$	$[0.007, 0.031]$
	Kurtosis of curvature of asperities (-)	K$[\kappa_{\mathrm{a}}]$	$[2.81, 15.14]$
	Skewness of curvature of asperities (-)	Sk$[\kappa_{\mathrm{a}}]$	$[0.41, 1.6]$
	Mean of surface height (µm)	$\bar{z}$	$[7.71, 45.36]$
	Max. of surface height (µm)	$z^{\max}$	$[16.36, 79.25]$
	Root mean square of surface height (µm)	RMS$[z]$	$[2.62, 15.55]$
Target parameter	Effective contact area (%)	A_e	$[0.01, 16.56]$

Table 4.1.: The list of parameters of the generated raw database including material parameters, model parameters, statistical parameters, and the target quantity. Each row contains a parameter with corresponding symbol and the range of variation.

find a function (or hyperplane) that deviates from the actual target values by at most a specified margin (epsilon) for as many data points as possible, while also maintaining model simplicity (i.e., minimizing the norm of the weights).

The MLP Regressor, in contrast, is a neural network-based method capable of modeling nonlinear relationships through learned weights and activations. Note that the MLP Regressor with regularization and a single-layer architecture using a linear activation function and the Kernel Ridge algorithm employing a linear kernel exhibit similar characteristics.

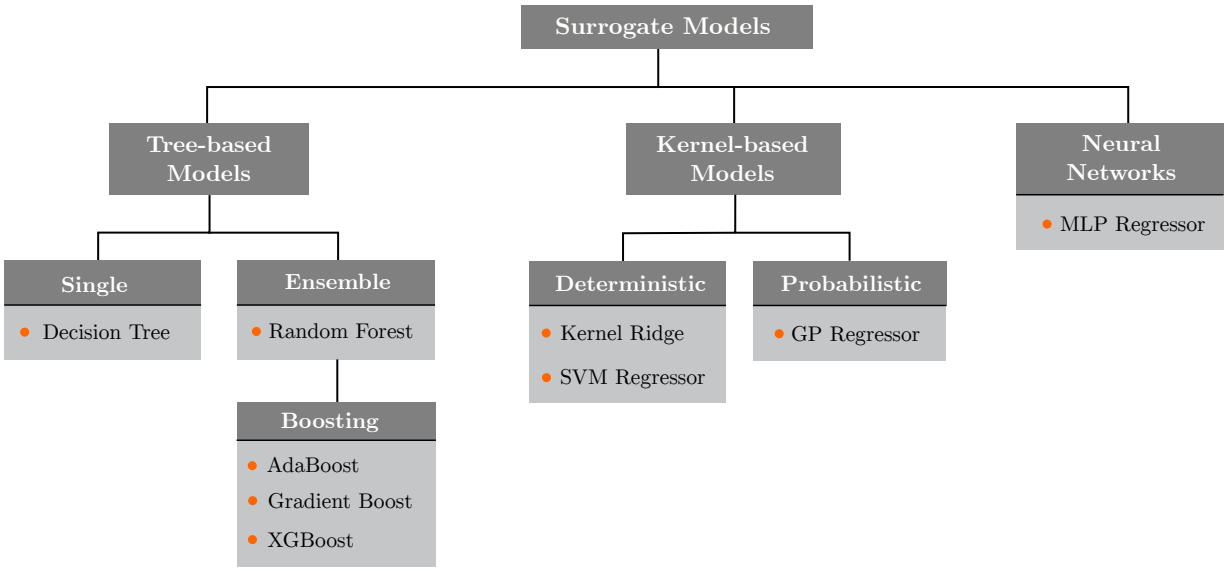


Figure 4.4.: Classification of surrogate modeling techniques based on their characteristics.

Many machine learning models have hyperparameters. Hyperparameters are configuration settings that are set before training begins. They control the learning process (e.g., learning rate, batch size, number of layers) and significantly influence model performance but are not learned from the data. Since the choice of hyperparameters directly affects model accuracy and performance [186], we perform a hyperparameter optimization using grid search with cross-validation to identify the best-performing configuration of a model [51]. By comparing the best-performing configuration of each model, we ensure a fair comparison across models.

In a grid search, the model is trained on all combinations of hyperparameters on a predefined grid: First, a search space is specified for each hyperparameter. Second, the search space is sampled on a fixed number of grid points. Grid-based hyperparameter optimization suffers from the curse of dimensionality and can quickly become computationally very expensive. Despite its computational cost, we demonstrate that the effort is justified, as it leads to substantially improved model performance (cf. Section 4.4.2). To balance computational cost, we focus on tuning key hyperparameters only [186]. For example, in tree-based algorithms such as XGBoost, important hyperparameters include the learning rate, maximum tree depth, and the number of estimators, corresponding to the maximum number of boosting trees [64]. A full list of the investigated hyperparameters is provided in Table 4.2. Detailed descriptions of the machine learning models and their corresponding hyperparameters can be found in Pedregosa et al. [118].

To avoid setting aside a portion of the training data exclusively for validation (note that we already split the database into 80% training and 20% test data, cf. Section 4.3.2), we use k-fold

cross-validation during hyperparameter optimization. This method splits the training data into k equal parts, or folds. The model is then trained k times, each time using $k-1$ folds for training and the remaining fold for validation. This ensures that every data point is used for both training and validation, providing a more reliable estimate of model performance. The final performance metric is obtained by averaging the results over all k runs. For more details on k-fold cross-validation, we refer readers to the original literature [92, 193]. In this study, we specifically apply 5-fold cross-validation.

Table 4.3 compares the computation times required for hyperparameter optimization across the models. Due to the curse of dimensionality, models with more hyperparameters generally require much longer tuning times. For instance, the Random Forest model involves evaluating $19{,}008$ hyperparameter combinations ($9{\times}4{\times}11{\times}4{\times}3{\times}4{=}19{,}008$), as detailed in Table 4.2. In contrast, the Decision Tree and MLP Regressor models involve fewer combinations–600 and 756, respectively. However, the MLP Regressor still requires significantly more tuning time due to the higher training cost per configuration. Unlike Decision Trees, MLPs must learn additional internal parameters, such as weights and biases, for each hyperparameter setting [62]. In total, hyperparameter tuning for the MLP Regressor takes over $12\,\mathrm{h}$, whereas for the Decision Tree model it takes only about $16\,\mathrm{s}$, a negligible amount by comparison.

4.3.4. Accuracy metrics

To evaluate the performance of the models, four metrics are introduced: normalized mean square error (nMSE), normalized mean absolute error (nMAE), normalized max-error (nMaxE), and the R^2 score [26]. Accuracy metrics without normalization are scale-dependent [78]. In other words, scale-dependent measures always require information about the scale of the target to assess the model's accuracy effectively. For the following expressions of normalized metrics, A_e denotes the ground truth, $\hat{A}_e$ is the prediction, $\bar{A}_e$ is the mean ground truth of the test data, $\max(|A_e|)$ is the max value of ground truth in the test data, and n is the number of test points.

The normalized mean square error (nMSE) is defined as:

$$\text{nMSE} = 100\,\frac{\frac{1}{n}\sum_{i=1}^{n}\left(A_e^i - \hat{A}_e^i\right)^2}{\bar{A}_e}\%. \tag{4.6}$$

The nMSE metric quantifies the average squared differences between predictions and ground truths, normalized by the mean of the ground truth. Since it penalizes larger errors more heavily, it is particularly sensitive to outliers. A lower nMSE indicates a better model fit.

The normalized mean absolute error (nMAE) is formulated as:

$$\text{nMAE} = 100\,\frac{\frac{1}{n}\sum_{i=1}^{n}\left|A_e^i - \hat{A}_e^i\right|}{\bar{A}_e}\%. \tag{4.7}$$

The nMAE metric measures the average absolute differences between predictions and ground truths, providing an interpretable error metric. Unlike nMSE, nMAE treats all errors equally and is more robust to outliers. Lower nMAE values indicate higher prediction accuracy.

Model	Hyperparams.	Search space	Grid points	Optimum
Decision Tree	max depth	1 - 30	5	30
	max features	auto, sqrt, log2, none	4	none
	min samples leaf	1 - 15	6	7
	min samples split	2 - 10	5	10
Random Forest	max depth	5 - 100	9	20
	max features	sqrt, log2, 0.3, 1	4	1
	# estimators	20 - 2,000	11	1,000
	criterion	sq. err., abs. err., Friedman, Poisson	4	sq. err.
	min samples split	2 - 10	3	2
	min samples leaf	1 - 15	4	1
AdaBoost	max depth	3 - 35	7	18
	loss	linear, square, exponential	3	square
	# estimators	10 - 1500	11	1,500
	learning rate	1×10^{-2} - 2.0	9	1.0
Gradient Boost	learning rate	1×10^{-3} - 2.0	8	0.05
	loss	sq. err., abs. err., huber, quantile	4	sq. err.
	max depth	5 - 100	8	5
	# estimators	50 - 1,600	10	1,600
	criterion	Friedman, sq. err.	2	Friedman
	max features	auto, sqrt, log2	3	auto
XGBoost	learning rate	1×10^{-2} - 2.0	8	0.1
	max depth	3 - 15	5	3
	# estimators	10 - 1,500	14	1,500
	gamma	0 - 2	8	0.0
SVM Regressor	C	0.1 - 1,400	12	1,400
	degree	3 - 7	3	7
	kernel	linear, rbf, poly, sigmoid	4	poly
	gamma	scale, auto	2	scale
	cache size	200 - 5000	6	200
GP Regressor	alpha	1×10^{-3} - 1.0	110	0.0374
Kernel Ridge	alpha	1×10^{-5} - 1	8	1×10^{-5}
	gamma	0.01 - 150	10	5
	kernel	rbf, linear	2	rbf
MLP Regressor	activation	logistic, relu, tanh	3	tanh
	alpha	1×10^{-4} - 5×10^{-3}	7	1×10^{-3}
	hidden layer sizes	nodes: 50 - 120, 2-4 layers	4	$[50, 100, 200, 100, 50]$
	learning rate	constant, invscaling, adaptive	3	adaptive
	solver	sgd, adam, lbfgs	3	lbfgs

Table 4.2.: Investigated surrogate model hyperparameter optimization, including the model type, selected hyperparameters, the defined parameter search space, and the optimal parameter value.

	Decision Tree	Random Forest	AdaBoost	Gradient Boost	XGBoost	SVM Regressor	GP Regressor	Kernel Ridge	MLP Regressor
Tuning time (s)	16	1,628,304	42,814	900,946	120,845	25,120	61,494	2,599	44,325

Table 4.3.: Assessment of time required for performing hyperparameter optimizations.

The normalized max-error (nMaxE) is expressed as:

$$\text{nMaxE} = 100\,\frac{\max\Big(\big|\boldsymbol{A_e} - \hat{\boldsymbol{A}}_{\boldsymbol{e}}\big|\Big)}{\max\Big(\big|\boldsymbol{A_e}\big|\Big)}\%. \tag{4.8}$$

The nMaxE metric captures the largest absolute error relative to the maximum observed value in the test data. It highlights the worst-case error scenario, making it useful for applications where large deviations are critical. A lower nMaxE suggests that the model avoids large prediction errors.

The R^2 metric, also referred to as the coefficient of determination, quantifies how well the predictions match with actual values. Usually, the R^2 value ranges from 0 to 1, or from 0% to 100%. The accuracy metric R^2 is defined as

$$R^2 = 100\left[1 - \frac{\sum_{i=1}^{n}\big(A_e^i - \hat{A}_e^i\big)^2}{\sum_{i=1}^{n}\big(A_e^i - \bar{A}_e\big)^2}\right]\%. \tag{4.9}$$

R^2 measures the proportion of variance in the actual values that is explained by the model predictions. A value close to 100% indicates a strong predictive model, while lower values suggest poor explanatory power.

Accuracy metrics are utilized for quantifying the investigated surrogate models based on the predicted and actual effective contact area. Also, note that all metrics are unitless and defined as percentages.

4.4. Results and discussion

4.4.1. Evaluation of baseline models

The evaluation of the so-called baseline models aims to compare the investigated machine learning algorithms without hyperparameter optimization in terms of accuracy metrics, fit, and prediction time on unseen test data.

Fig. 4.5 shows the qualitative comparison of the investigated baseline models to predict the effective contact area. Each figure has a red line, also referred to as the perfection line, which is used to qualify the model prediction. The qualitative accuracy of the model can be determined by how closely its predictions align with the red curve. A simple tree-based method such as Decision Tree shows a linear trend that is consistent with the perfection line. However, it exhibits a high

variance, which causes a bad generalization. On the other hand, ensemble methods that rely on multiple decision trees, such as Random Forest, Gradient Boost, and XGBoost, exhibit reduced variance, thus leading to improved accuracy. While the majority of ensemble methods perform well, AdaBoost stands out as a negative exception. The AdaBoost algorithm struggles to grasp the underlying patterns in the data, often delivering the same predictions although the actual values are different, thus indicating potential underfitting.

The SVM Regressor deviates slightly from the perfection line with an increasing variance for larger values of the effective contact area. In cases where the target values are small, the SVM Regressor tends to produce some negative, physically unrealistic predictions (the effective contact area can not take negative values). Similarly, the Kernel Ridge algorithm excessively deviate from the perfection line, predicting large negative values of the effective contact area even though the actual values are meant to be small yet positive. These noticeable deviations around small effective contact area values can be related to the dominating effect of the loss function contributions of data points with large effective contact area values during training. Since the Kernel Ridge algorithm is not yet tuned and has a simple model complexity, it tends to prioritize minimizing errors for larger values, which is why we observe significant deviations around the small effective contact values. On the other hand, the MLP Regressor with a single hidden layer using the rectified linear unit (relu) activation function demonstrates a behavior similar to that of the SVM Regressor, In particular, it tends to predict comparably large negative values of the effective contact area around the origin. However, in contrast to the SVM Regressor, exhibits significantly lower variance, especially for larger effective contact areas. The GP Regressor model outperforms all other baseline models in terms of accuracy and exhibits comparably reduced variance around the perfection line.

Fig. 4.6a provides a quantitative comparison of accuracy metrics, namely nMSE (%), nMAE (%), nMaxE (%), and the R^2 score (%) for the investigated baseline models. Since the Decision Tree algorithm by definition mimics the training data, it yields 0% nMSE, nMAE, nMaxE along with 100% R^2 in the training data set. However, it suffers from a relatively high nMaxE value in the test data, e.g., $\mathrm{nMaxE} = 24.62\%$. The corresponding numbers for nMSE, nMAE, nMaxE and R^2 are provided in Table 4.4. The large difference between train and test metrics means that the baseline model based on the Decision Tree can not properly generalize the problem. Yet, the Decision Tree model has the fastest prediction and fit times as depicted in Fig. 4.6b. The Random Forest exhibits smaller test errors in terms of nMSE and nMAE compared to nMaxE and it does not indicate any sign of overfitting. However, it has the largest fit time of $t_f = 19.99\,\mathrm{s}$. Similarly, the SVM Regressor exhibits larger nMaxE values when compared to other accuracy metrics and it has the largest prediction time with $t_p = 3.408\,\mathrm{s}$. The Kernel Ridge model produces the largest test errors in the test data with $\mathrm{nMSE} = 19.11\%$, $\mathrm{nMAE} = 13.02\%$, and $\mathrm{nMaxE} = 34.45\%$. Moreover, it has a poor performance in terms of R^2. The ensemble method AdaBoost also produces large test errors and has the poorest coefficient of determination of $R^2 = 90.71\%$. Other ensemble methods such as Gradient Boost and XGBoost demonstrate better results in general. Additionally, XGBoost has the second-fastest fit and prediction time, following closely after the Decision Tree. As a neural network model, MLP Regressor produces smaller test errors in terms of nMSE and nMAE, but it has the largest fit time with $t_f = 12.88\,\mathrm{s}$. Among all the investigated methods, the GP Regressor model stands out for its exceptional performance when considering various accuracy metrics in test data, e.g., $\mathrm{nMSE} = 1.15\%$, $\mathrm{nMAE} = 3.05\%$, and $\mathrm{nMaxE} = 11.76\%$ along with $R^2 = 99.53\%$.

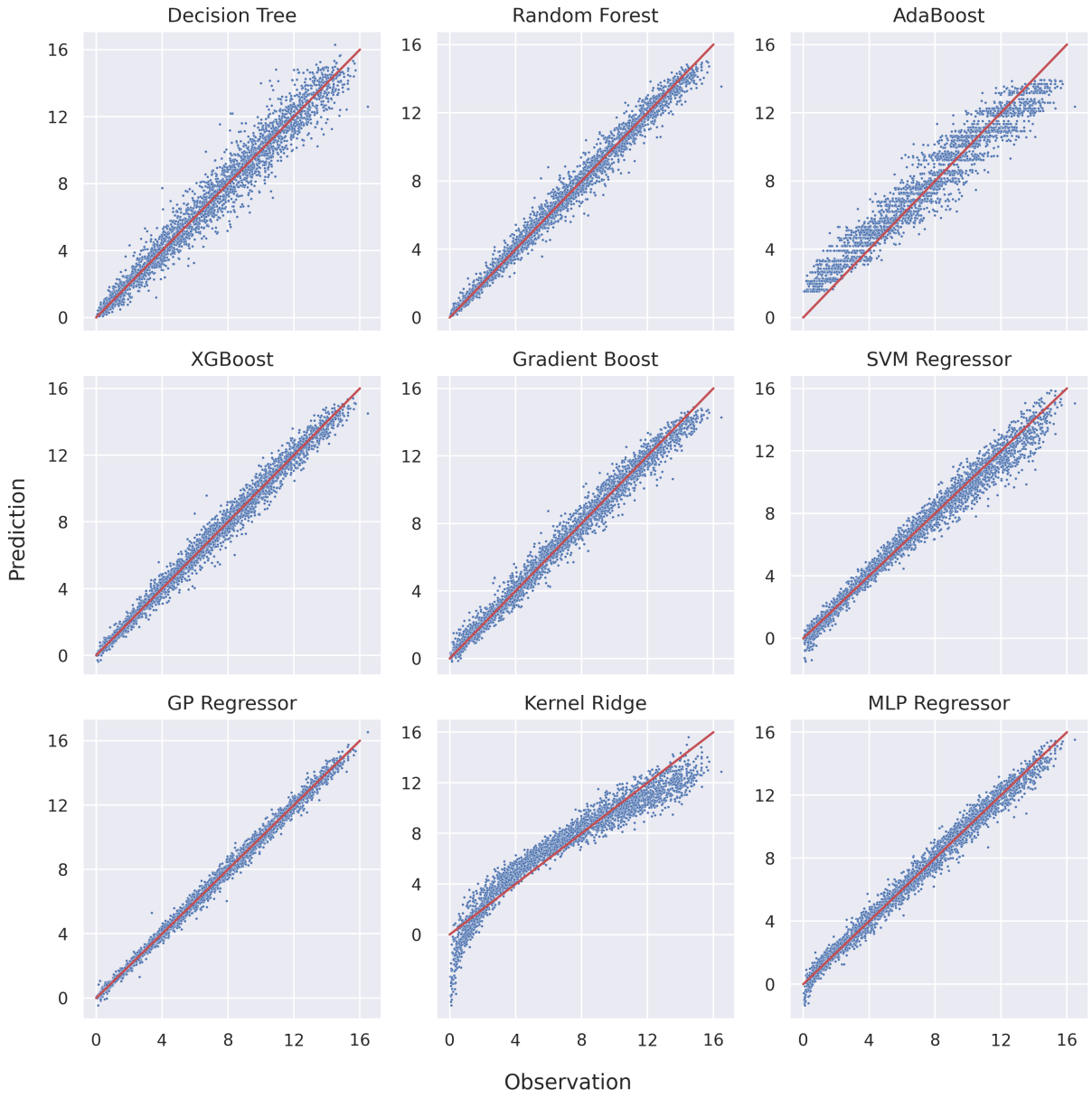


Figure 4.5.: Comparison of the investigated baseline models to predict the effective contact area (%). Horizontal axes are the observations or actual values, while vertical axes denote the predictions. The red lines represent a perfect fit, which is used to qualify the model's accuracy.

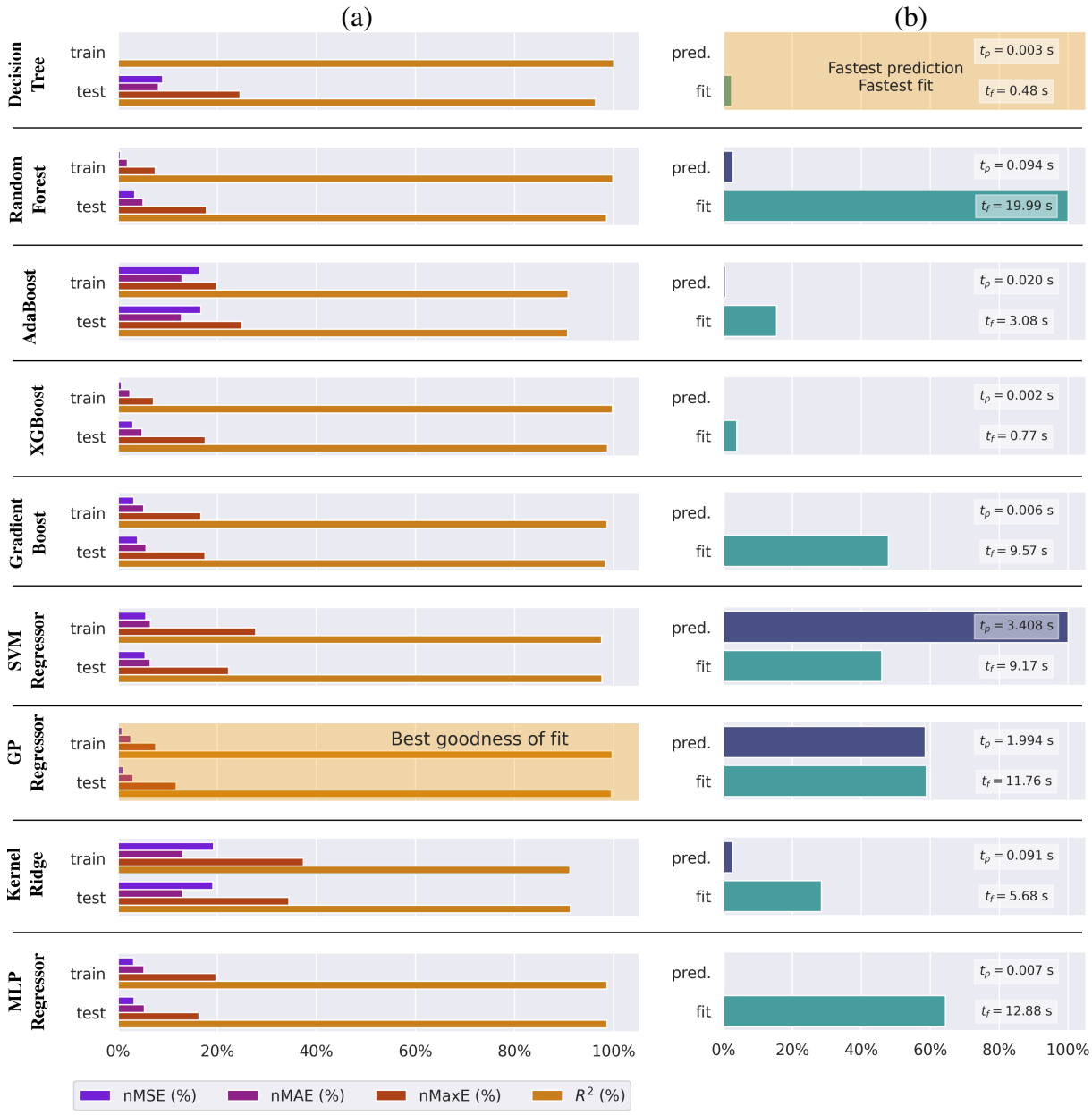


Figure 4.6.: Comparison of the investigated data-driven baseline models to predict the effective contact area for rough surface contact: (a) Accuracy metrics (%), (b) times for fit and prediction in % are given including the corresponding actual values in s. Note that each accuracy metric is scaled relative to the worst accuracy across all models to enable better comparisons. Similarly, each prediction and fit time measures are scaled relative to the maximum prediction and fit time across all models.

4.4.2. Evaluation of tuned models

As mentioned earlier, hyperparameter optimization or tuning is a crucial and necessary step to improve model performance and the ability to generalize from seen to unseen data. Also, tuning can reduce prediction and fit times. Fig. 4.7 qualitatively illustrates the comparison of baseline models and tuned models with their respective predictions of the effective contact area. For all investigated models, there is a drastic improvement in accuracy. Particularly for AdaBoost, SVM Regressor, Kernel Ridge, and MLP Regressor models, predictions now closely align with the perfection line in contrast to the baseline models. Furthermore, the tuned models are quite effective in avoiding unphysical negative values for the effective contact area, particularly for the Kernel Ridge, the SVM Regressor and the MLP Regressor.

Table 4.4 provides quantitative comparisons of baseline and tuned models in terms of accuracy metrics and computational time for model fitting and prediction. Hyperparameter optimization yields varying improvements across models, often at the expense of increased training time. For both the Decision Tree and Random Forest models, tuning yields slight improvements in most accuracy metrics, except for nMaxE. Specifically, for the Decision Tree, the prediction time remains almost the same, while the fit time decreases slightly from $0.48\,\mathrm{s}$ to $0.42\,\mathrm{s}$. Conversely, the Random Forest model shows a drastic increase in both training and prediction time, nearly 15 times larger. The fit time increases dramatically from $19.99\,\mathrm{s}$ to $298.58\,\mathrm{s}$ and the prediction time rises from $0.094\,\mathrm{s}$ to $1.364\,\mathrm{s}$. The tuned SVM Regressor demonstrates remarkable performance improvements, with nMSE decreasing from from 5.47% to 1.20%, nMAE from 6.50% to 3.09%, and nMaxE from 22.30% to 10.62%, while the R^2 value increases from 97.61% to 99.50%. However, these gains come with a steep computational cost, as the fit time increases from $9.17\,\mathrm{s}$ to $2{,}550\,\mathrm{s}$. Yet, the tuned SVM Regressor model exhibits a reasonable reduction in prediction time. Tuned ensemble methods, e.g., AdaBoost, Gradient Boost, and XGBoost, exhibit great improvements in accuracy metrics with an average improvement factor of approximately 40%. Among the ensemble approaches, the tuned XGBoost model demonstrates outstanding computing performance, delivering a fit time of $t_f = 4.13\,\mathrm{s}$ and a prediction time of $t_p = 0.005\,\mathrm{s}$.

The tuned MLP Regressor model exhibits better goodness of fit than the tree-based models across all accuracy metrics, with $\mathrm{nMSE} = 1.19\%$, $\mathrm{nMAE} = 3.09\%$, $\mathrm{nMaxE} = 9.95\%$, and $R^2 = 99.51\%$. With these values, it ranks as the third-best model overall in terms of accuracy. However, training the tuned MLP Regressor requires approximately $3{,}435\,\mathrm{s}$, making it the most time-consuming training process among all models. The fit time is approximately 266 times longer than that of the baseline model, primarily due to a significantly higher number of training epochs and increased model complexity. Specifically, the tuned model is trained over $20{,}000\,\mathrm{epochs}$, compared to just 200 epochs for the baseline. In addition, the tuned MLP employs a deeper architecture with more neurons and layers, resulting in a substantially larger number of internal parameters to optimize. With a prediction time of $0.049\,\mathrm{s}$, the MLP Regressor lies in the middle range. It is an order of magnitude slower than the Decision Tree, yet two orders of magnitude faster than the slowest models, such as the GP Regressor and Random Forest.

For the GP Regressor model, tuning leads to slight improvements in nMSE and nMAE, and a substantial reduction in nMaxE—from 11.76% to 8.98%. As a result, the GP Regressor achieves the best overall accuracy across all evaluated metrics, making it the most effectively tuned model in this study. However, this gain in accuracy comes at a considerable computational cost: the fit time increases sharply from $11.76\,\mathrm{s}$ to $853.23\,\mathrm{s}$, representing more than a 70-fold increase,

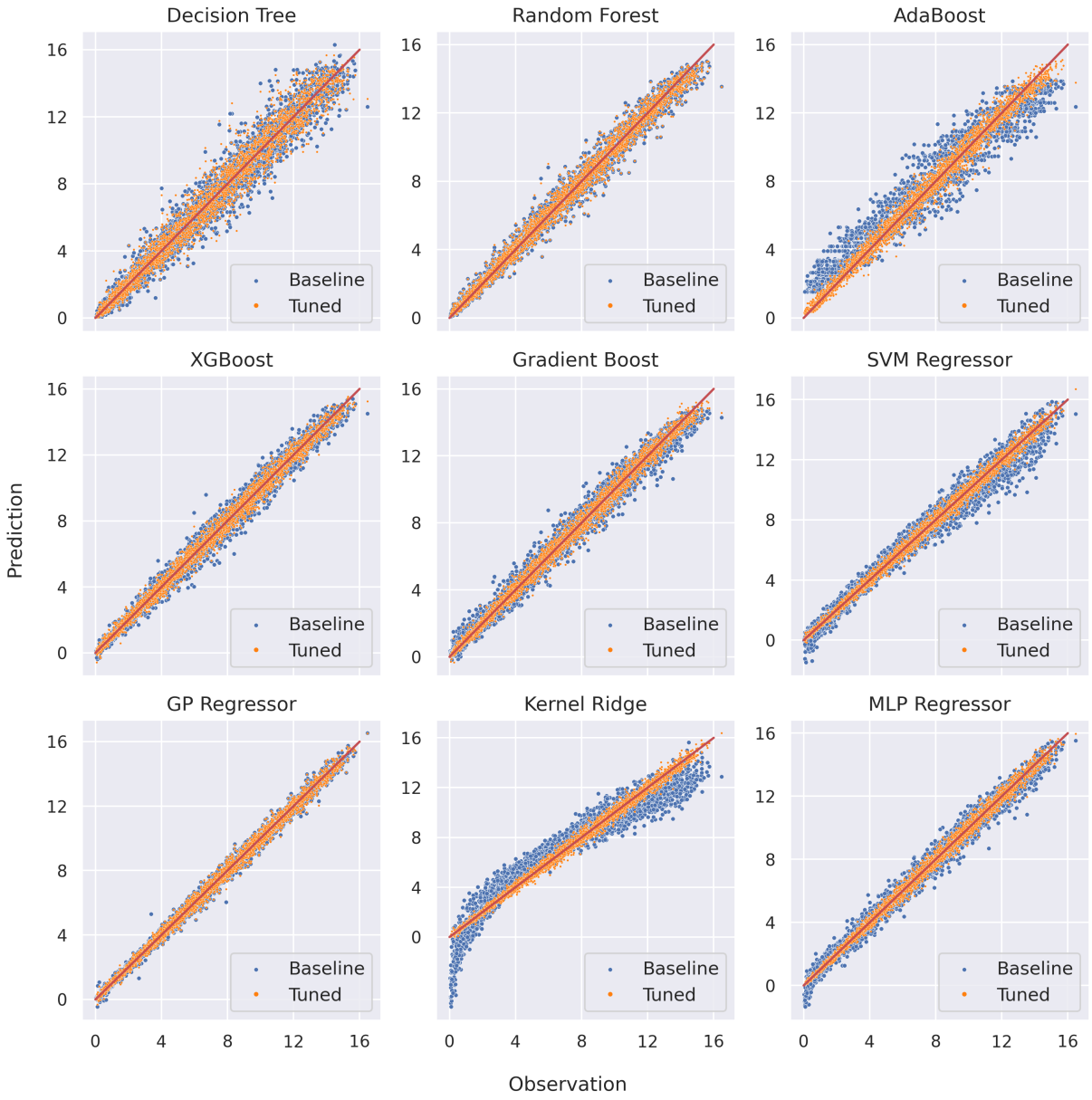


Figure 4.7.: Comparison of the baseline models with the tuned models. Horizontal axes represent the observations or actual values, while vertical axes denote the predictions. The red lines represent a perfect fit, which is used to qualify the model's accuracy.

while the prediction time remains nearly unchanged. A significant portion of this fit time is spent on computing the maximum likelihood estimation of the kernel parameters. In contrast, the tuned Kernel Ridge model delivers accuracy metrics comparable to those of the GP Regressor, but with much lower computational demands-requiring only $9.18\,\mathrm{s}$ for training and $0.827\,\mathrm{s}$ for prediction. Overall, we consider the tuned Kernel Ridge model to be the most balanced surrogate, as discussed in Section 4.4.4. For this reason, we focus on this model in greater detail in the following sections.

4.4.3. Comparative cost analysis of reference and surrogate models

The motivation behind building surrogate models is to construct fast-to-evaluate approximations of the BEM reference model, thereby making multi-query scenarios, requiring numerous model evaluations, computationally tractable. While surrogate models reduce evaluation time compared to the reference model, they also introduce additional upfront costs associated with model construction.

Ultimately, our goal is to determine the break-even point, the number of model evaluations at which the computational savings from using the surrogate outweigh the cost of building it. To analyze this, we conduct a comparative cost analysis of the overall computational cost of using the Kernel Ridge model versus the reference numerical method (BEM) for an increasing number of simulations. For the BEM model, the total cost t_{BEM} is obtained by summing the time required for each individual simulation. As shown in Fig. 2.20, the simulation time depends on both surface properties and the applied far-field displacement. Therefore, the exact cost of a specific BEM simulation is not known in advance. To approximate the total cost, we use the average simulation time from the database generation process, which is $189.79\,\mathrm{s}$. This average is assumed to be a reliable estimate of expected cost, especially when considering a large number of simulations.

In contrast, computing the total cost for the surrogate model is more intricate. To ensure a fair comparison, all components of the surrogate modeling process must be considered. Specifically, we account for four time contributions involved in building and using the surrogate model. The first contribution is the model evaluation time, which refers to the total cost t_p of making predictions using the surrogate model across all input samples. To accurately estimate the prediction time for a single sample, we compute the average prediction time based on 2,000 randomly selected input samples, drawn uniformly from the feature ranges specified in Table 4.1. The second contribution is the fit time t_f, i.e., the time required to train the surrogate model on the available training data. The third contribution is the cost of hyperparameter optimization t_h, which, as shown in Table 4.3, can be substantial. Finally, we include the cost of generating the database t_d, as this is a necessary prerequisite for training any surrogate model. This includes both the generation of input samples (e.g., synthetic surfaces and far-field displacements) and the corresponding evaluations using the reference BEM model to obtain the output quantities. The total cost of the Kernel Ridge Regressor can finally be computed as $t_{\text{KRR}} = t_p + t_f + t_h + t_d$.

A visualization of the comparative computational cost analysis is provided in Fig. 4.8. The overall cost of using the surrogate model is dominated by the database generation time t_d, which is several orders of magnitude greater than the other cost components. The model training time is negligible: on average, a single BEM evaluation is more expensive than the combined cost of training and evaluating the Kernel Ridge model. Even the hyperparameter optimization step, while computationally intensive within the surrogate modeling workflow, contributes only

		Performance metric (%)				Time (s)	
		nMSE	nMAE	nMaxE	R^2	fit (t_f)	pred. (t_p)
Decision Tree	Baseline	8.99	8.17	24.62	96.31	0.48	0.003
	Tuned	8.51	8.03	27.66	96.51	0.42	0.003
	Improvement	5.34	1.71	-12.35	0.21		
Random Forest	Baseline	3.39	5.05	17.81	98.56	19.99	0.094
	Tuned	3.34	5.01	18.58	98.58	298.58	1.364
	Improvement	1.47	0.79	-4.32	0.02		
AdaBoost	Baseline	16.71	12.80	25.03	90.71	3.08	0.020
	Tuned	2.92	4.77	16.34	98.76	230.11	1.174
	Improvement	82.53	62.73	37.72	8.87		
XGBoost	Baseline	3.01	4.86	17.59	98.75	0.77	0.002
	Tuned	1.95	4.03	10.34	99.19	4.13	0.005
	Improvement	35.22	17.08	41.25	0.45		
Gradient Boost	Baseline	3.95	5.68	17.55	98.30	9.57	0.006
	Tuned	2.03	4.03	11.55	99.15	197.10	0.067
	Improvement	48.61	29.05	34.19	0.86		
SVM Regressor	Baseline	5.47	6.50	22.30	97.61	9.17	3.408
	Tuned	1.20	3.09	10.62	99.50	2550.27	0.624
	Improvement	78.06	52.46	52.38	1.94		
GP Regressor	Baseline	1.15	3.05	11.76	99.52	11.76	1.994
	Tuned	1.12	3.02	8.98	99.54	853.23	1.965
	Improvement	2.61	0.98	23.64	0.02		
Kernel Ridge	Baseline	19.11	13.02	34.45	91.24	5.68	0.091
	Tuned	1.16	3.07	9.84	99.52	9.18	0.827
	Improvement	93.93	76.44	71.44	9.07		
MLP Regressor	Baseline	3.24	5.32	16.34	98.64	12.88	0.007
	Tuned	1.19	3.09	9.95	99.51	3435.54	0.049
	Improvement	63.27	41.92	39.11	0.88		

Table 4.4.: Comparison of the baseline models with the tuned models. Accuracy metrics nMSE, nMAE, nMaxE, R^2 are evaluated on the test data. Improvement is calculated based on $100(m_t - m_b)/m_b$ where m_b is the evaluated performance metric of the baseline model and m_t is the evaluated performance metric of the tuned model. The cells with blue background show the model accuracy of the best baseline model, the cells with orange background denote those of the best-tuned model, while the cells with green background show the model accuracy and performance of the model considered "optimal" overall. The model accuracy is determined based on accuracy metrics, and the model performance is assessed by examining fit and prediction time. The fit (or training) time refers to the time required to train the models on the training dataset, whereas the prediction time is measured on the test dataset and represents the total time needed to make predictions for all test data points.

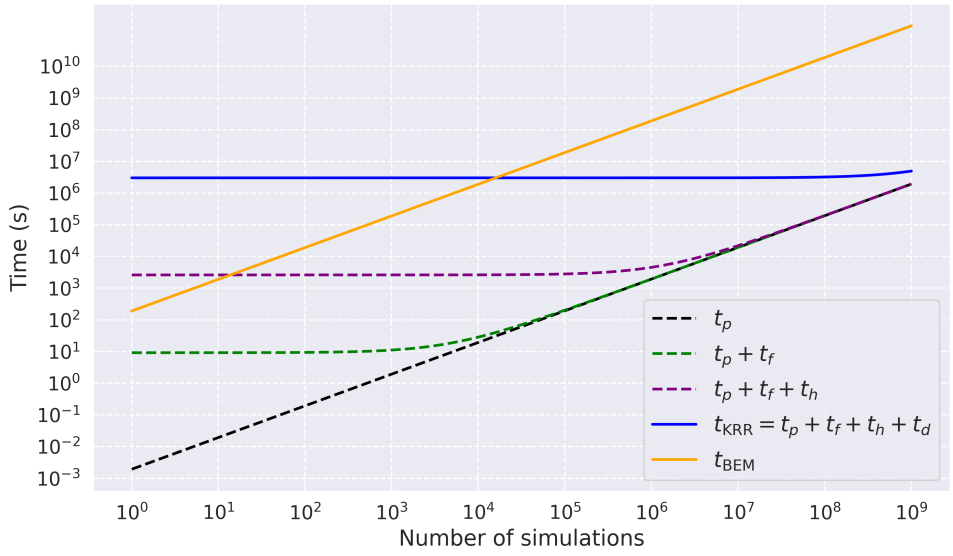


Figure 4.8.: Comparison of the total computational cost for the BEM model and the Kernel Ridge surrogate model. For BEM, the cost is estimated using the average simulation time ($189.79\,\mathrm{s}$) per sample. The surrogate model cost includes four components: prediction time (t_p), training time (t_f), hyperparameter optimization time (t_h), and database generation time (t_d), with total cost $t_{\mathrm{KRR}} = t_p + t_f + t_h + t_d$. The surrogate model's overall cost is dominated by database generation (t_d), while training and hyperparameter optimization contribute marginally. The surrogate becomes more cost-efficient than BEM after approximately 15,893 simulations.

marginally when compared to the cost of evaluating the reference model. Its cost breaks even after just 14 model evaluations, including both training and prediction. A comparison of the total costs, t_{BEM} and t_{KRR}, shows that the break-even point is reached at approximately 15,893 simulations. In other words, the surrogate model becomes the more cost-efficient choice only when more than this number of simulations is required. It is noteworthy that the database used to train the surrogate model consists of 15,878 samples, further underscoring the dominant cost associated with generating training data. Essentially, all other surrogate-related costs, including evaluation, training, and tuning times, are negligible in comparison. Although the break-even may initially seem like a high threshold, it is important to consider that applications such as uncertainty quantification (UQ) and Bayesian parameter calibration often involve several tens of thousands of model evaluations [30]. In such scenarios, the break-even point can be reached relatively quickly, making surrogate modeling a practical and cost-effective solution.

4.4.4. Discussion

In Section 4.4.1 and Section 4.4.2, the investigated surrogate models were compared in terms of accuracy-using standard metrics-and in terms of performance, based on model fit and prediction times. Our findings suggest that selecting an optimal surrogate model requires balancing accuracy with prediction performance, depending on the specific requirements of the application task. While the optimal choice is task-dependent, several general trends can be observed.

Tree-based models, such as Decision Trees and Random Forests, typically perform worse in terms of accuracy compared to kernel-based methods. However, they offer significantly faster prediction times. This makes them suitable candidates in scenarios where rapid evaluations are critical and minor accuracy trade-offs are acceptable-such as in real-time applications. In contrast, kernel-based models achieve superior accuracy. Although their prediction times are generally higher than tree-based models, they are still relatively low in absolute terms (on the order of seconds for 3,175 test points), making them viable for most practical applications. Among the kernel-based models, the GP Regressor yields the best accuracy metrics, closely followed by the Kernel Ridge model. The differences in accuracy between these two models are negligible for most practical purposes. However, the GP Regressor has approximately twice the prediction time of the Kernel Ridge model. This is expected, as the GP Regressor also provides a predictive variance alongside the mean estimate—an added value that explains the increased computational cost. If uncertainty quantification is not required, the Kernel Ridge model provides nearly identical predictions at half of the computational cost. This is not surprising, as both methods are closely related. In fact, the mean prediction of the GP Regressor is mathematically equivalent to that of the Kernel Ridge model, provided the same hyperparameters are used—namely, the kernel type, kernel parameters, and regularization parameter. The MLP Regressor, while slightly less accurate than the Kernel Ridge model, achieves lower prediction times—around one order of magnitude faster. However, this comes at the expense of a significantly longer training time and increased model complexity.

Overall, the tuned Kernel Ridge model offers the best trade-off between accuracy and computational efficiency. It delivers high prediction accuracy with low prediction times and minimal training overhead, making it a strong candidate for general-purpose surrogate modeling. Therefore, while the GP Regressor achieves the highest accuracy, it is not considered the optimal model

due to its substantially longer fit time and higher prediction cost. The tuned Kernel Ridge model is identified as the most balanced surrogate for a wide range of application scenarios.

In Section 4.4.3, we compared the overall costs of using a surrogate model with those of the reference model. Our results highlight that a fair comparison between data-driven surrogate models and reference models can only be made by considering the total cost—including both offline and online phases. The reasoning behind this is as follows: A general machine learning regression task aims to learn an unknown functional relationship from input-output data. It is common practice to distinguish between the offline and online phases of model development, as they typically involve different tasks and cost structures [59]. The offline phase includes data generation (collection, preprocessing, and feature selection) and model training. The online phase refers to the cost of evaluating the trained model. This separation is particularly relevant because online applications often involve a large number of model evaluations, while offline processes are performed only once. As a result, the substantial cost of the offline phase is usually justified as a one-time investment to obtain a usable model in the first place.

However, unlike standard machine learning scenarios where no prior model exists, surrogate modeling assumes the existence of a reliable reference model-typically one that is accurate but computationally expensive. The surrogate's purpose is to approximate this reference model efficiently, enabling its use in applications such as optimization, uncertainty quantification (UQ), or parameter identification, where many evaluations are required. In these contexts, the total cost of using a surrogate model-including both offline and online phases-must be considered to fairly assess its efficiency in solving the application task. Solving the task with the reference model involves direct evaluation across many configurations, often resulting in high computational cost. In contrast, the surrogate model approach requires an initial investment: building the surrogate entails data generation, model training, and hyperparameter tuning. Once constructed, the surrogate is then evaluated across the same configurations as the reference model, but at significantly lower cost per evaluation. Ultimately, the initial cost of building a surrogate is incurred specifically for the application task at hand. Therefore, the offline cost must be included in the overall assessment of the surrogate model's efficiency. Our results demonstrate that ignoring the offline costs leads to a distorted assessment, which falsely overstates the advantages of data-driven modeling. Comparing only the evaluation times of the models, as it is often done in the literature, is insufficient, since it neglects the largest cost component: database generation. However, the encouraging finding is that there is no practical need to rely on such an incomplete or unfair comparison. Even though the break-even point may initially appear high, our analysis shows that the proposed surrogate modeling approach remains highly worthwhile for many application tasks that involve large numbers of model evaluations.

4.5. Conclusion

In this study, we developed a surrogate modeling framework to predict the effective contact area in rough surface contact problems involving a linear elastic half-space and a rigid rough surface. A comprehensive database was constructed using BEM simulations, containing far-field displacements, 22 statistical parameters derived from synthetic surface height fields, and the target output-the effective contact area. Several machine learning regression models were evaluated as surrogate modeling techniques. Optimal model configurations were determined

through hyperparameter optimization using grid search with cross-validation. All models were assessed using standard accuracy metrics, as well as training and prediction times to evaluate their computational performance.

Our findings indicate that selecting an optimal surrogate model requires balancing predictive accuracy and computational performance, depending on the specific requirements of the application task. Each model exhibits distinct characteristics, meaning that the ideal choice is context-dependent-for instance, a model prioritizing speed may be preferred in time-critical applications, whereas a more accurate model might be favored in precision-sensitive tasks. Overall, our results show that the Kernel Ridge model offers the most favorable trade-off between accuracy and efficiency across a broad range of scenarios. It achieves high accuracy, low prediction time, and minimal training overhead, making it a strong candidate for general-purpose surrogate modeling. The Gaussian Process Regressor presents a compelling alternative when uncertainty quantification is required, although it incurs a higher computational cost due to variance estimation. To assess the robustness of the surrogate modeling approach, the generalization capability of the Kernel Ridge was evaluated on previously unseen simulation scenarios. When evaluating the effectiveness of the surrogate modeling approach, the break-even point—where the overhead of building the surrogate is offset by its reduced evaluation cost—is of particular interest. We emphasize the importance of accounting for all offline costs, especially database generation. Neglecting these costs, as is sometimes done in the literature, results in a skewed evaluation that overstates the benefits of data-driven modeling. Nonetheless, we conclude that such an incomplete assessment is unnecessary in practice. Even though the break-even point may initially appear high, the proposed surrogate modeling framework proves to be highly beneficial for many application scenarios that involve large numbers of model evaluations.

To conclude, the identified Kernel Ridge surrogate model predicts the effective contact area with acceptable accuracy and robustness while significantly reducing evaluation time compared to state-of-the-art numerical methods such as the Boundary Element Method (BEM). As such, it serves as an attractive and efficient surrogate for use in multi-query analyses, such as uncertainty quantification, parameter identification, and sensitivity analysis.

This work reveals several opportunities for further investigations. Apart from statistical parameters, only the far-field displacement has been considered as input for the surrogate modeling strategy. However, the input space could be extended to other properties, e.g., geometrical parameters such as the scan length. Moreover, not only the effective contact area but also the overall contact force could be considered as relevant model output to formulate and solve the problem as a multi-target regression task. As an alternative approach, the contact force field can be predicted using a neural network approach consisting of both fully connected and convolutional neural networks, by taking contour plots of the topographies as image-based input and far-field displacement values as numerical input. Eventually, the proposed surrogate modeling framework could be formulated as a hybrid model that combines data-driven components and model-based approaches that include the physics of the underlying problem.

5. Combining Neural Networks with Physics: Physics-Informed Neural Networks

Machine learning and deep learning approaches typically require large amounts of simulation or experimental data. However, obtaining such data is not always feasible due to the complexity of simulations and the high cost of experiments. The limited data availability often leads to reduced accuracy in data-driven approaches, as these methods rely solely on available datasets. Consequently, they are prone to making incorrect predictions when data-driven models are trained on datasets that do not accurately represent real-world observations (such as sensor or experimental data) or contain mislabeled samples. These errors can result in irreversible failures, as there is no physics-based validation mechanism to ensure the reliability of predictions.

To address the challenges posed by data scarcity and the black-box nature of data-driven models, Raissi et al. [134] introduced Physics-Informed Neural Networks (PINNs), an approach that integrates data-driven methods with the underlying physics of the problem. By incorporating governing equations into the learning process, PINNs enable neural networks to achieve higher accuracy even with limited data, improving both model interpretability and generalization.

As illustrated in Fig. 5.1, there is an inherent trade-off between data availability and the incorporation of physics in model development. Acquiring more data increases costs due to the need for additional simulations or experiments, whereas incorporating more physics introduces greater complexity to the problem. Striking a balance between these two aspects is essential for building efficient and accurate models.

This trade-off underscores the importance of PINNs, which provide a middle ground by leveraging data and physics. PINNs integrate governing equations into the learning process, allowing the model to learn patterns effectively even with limited data. By embedding physics-based constraints, PINNs enhance model generalization, reduce dependency on large datasets, and improve interpretability, making them particularly valuable for scientific and engineering applications.

The following key characteristics highlight the fundamental aspects of PINNs:

- **Mesh-free framework:** PINNs do not rely on predefined mesh structures, allowing them to handle unstructured or irregular domains, making them applicable to a wide range of complex geometries.

- **Versatility in forward and inverse problems:** PINNs can be applied to both forward and inverse problems. In forward problems, they approximate the unknown solution when the problem parameters are known. In inverse problems, they infer unknown parameters using observed data, a process also referred to as data-driven identification.

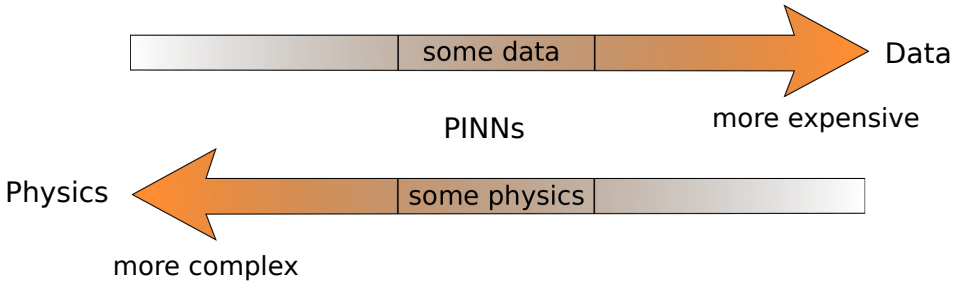


Figure 5.1.: Illustration of the trade-off between data and physics.

- **Data efficiency:** Since PINNs integrate governing equations into the learning process, they require fewer labeled data points than purely data-driven models, making them particularly useful in scenarios with limited or expensive-to-acquire data.

- **Automatic differentiation for efficient computation:** PINNs leverage automatic differentiation, implemented in deep learning frameworks such as TensorFlow and PyTorch to efficiently compute derivatives of the predicted solution with respect to spatial and temporal variables.

- **Regularization through physics constraints:** Unlike conventional neural networks, PINNs incorporate physics-based constraints directly into the loss function, helping to prevent overfitting and ensuring physically consistent predictions even in the presence of noisy data.

- **Higher interpretability and robustness:** By embedding physical laws into the learning process, PINNs offer better interpretability compared to traditional black-box machine learning models. This makes them more robust in extrapolating beyond the training data and in handling uncertainties in real-world applications.

- **Compatibility with hybrid modeling:** PINNs can be integrated with traditional numerical solvers, such as finite element or finite volume methods, to enhance computational efficiency and accuracy.

- **Scalability and computational challenges:** While PINNs offer many advantages, their training can be computationally intensive, especially for high-dimensional problems or complex physical systems. Strategies such as adaptive weighting of loss terms, domain decomposition, and multi-fidelity learning can help improve scalability and convergence.

- **Broad applicability across scientific domains:** PINNs can be used in various fields, including physics, engineering, computational fluid dynamics, biomedical imaging (e.g., clinical MRI reconstruction), and material science. Their flexibility allows them to be applied to problems governed by differential equations, such as continuum mechanics, electromagnetism, and biological systems.

5.1. Generic PINNs with output transformation

A generalized formulation for partial differential equations can be expressed in residual form with accompanying boundary conditions as

$$\mathcal{D}\big[\boldsymbol{u}(\boldsymbol{x})\big] = \mathbf{0}, \quad \text{on } \Omega, \tag{5.1}$$

$$\mathcal{B}\big[\boldsymbol{u}(\boldsymbol{x})\big] - g(\boldsymbol{x}) = \mathbf{0}, \quad \text{on } \partial\Omega. \tag{5.2}$$

Here, $\mathcal{D}[\cdot]$ denotes a differential operator acting on an unknown solution $\boldsymbol{u}$, $\mathcal{B}[\cdot]$ is the boundary operator, $g(\boldsymbol{x})$ represents the prescribed boundary condition, $\boldsymbol{x}$ are the spatial coordinates that span the domain Ω and the boundary $\partial\Omega$.

Consider a fully-connected L-layers neural network to construct an approximated solution $\tilde{\boldsymbol{u}}$ to the boundary value problem (BVP) as follows [93, 96]

$$\boldsymbol{u} \approx \tilde{\boldsymbol{u}} := \big(\mathcal{N}^L(\boldsymbol{z};\boldsymbol{\theta})\big)', \quad \mathcal{N}^L(\boldsymbol{z};\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}^n. \tag{5.3}$$

$\mathcal{N}^L(\boldsymbol{z};\boldsymbol{\theta})$ represents the network output in the output layer L, trainable network parameters, namely weights and biases, are denoted as $\boldsymbol{\theta}$, $(.)'$ represents a user-defined output transformation [102], and $\boldsymbol{z}$ is the network input such that $\boldsymbol{x} \subset \boldsymbol{z}$. Note that $\boldsymbol{z}$ can consist of spatial coordinates, time, and other additional input parameters. The network output is calculated using recursive $L-1$ element-wise operations between the input layer and hidden layers as

$$\begin{aligned} \text{input layer} &\rightarrow \mathcal{N}^1(\boldsymbol{z};\boldsymbol{\theta}) = \boldsymbol{z}, \\ \text{hidden layers} &\rightarrow \mathcal{N}^l(\boldsymbol{z};\boldsymbol{\theta}) = \psi(\boldsymbol{\theta}^l \cdot \mathcal{N}^{l-1}(\boldsymbol{z};\boldsymbol{\theta})), \text{ for } 2 \leqslant l \leqslant L-1, \\ \text{output layer} &\rightarrow \mathcal{N}^L(\boldsymbol{z};\boldsymbol{\theta}) = \boldsymbol{\theta}^L \cdot \mathcal{N}^{L-1}(\boldsymbol{z};\boldsymbol{\theta}), \end{aligned} \tag{5.4}$$

where ψ denotes the activation function that adds non-linearity to the layer output.

Output transformation enables the neural network to enforce boundary conditions explicitly. A user-defined output transformation can be obtained with suitable helper functions as

$$\tilde{u}_i(\boldsymbol{z};\boldsymbol{\theta}) = g(\boldsymbol{z}) + s \cdot h(\boldsymbol{z}) \cdot \mathcal{N}_{u_i}(\boldsymbol{z};\boldsymbol{\theta}), \tag{5.5}$$

where g is the prescribed boundary condition, s is a scaling parameter and h is a distance-to-boundary function fulfilling the following conditions:

$$\begin{aligned} h(\boldsymbol{z}) &= 0, \quad \text{on } \partial\Omega, \\ h(\boldsymbol{z}) &> 0, \quad \text{in } \Omega \setminus \partial\Omega. \end{aligned} \tag{5.6}$$

For simple boundaries $\partial\Omega$, it is relatively easy to define an appropriate distance-to-boundary function. On the other hand, it can become a quite challenging task in the case of arbitrary boundaries. One method to find a generalized distance-to-boundary function is to use NURBS parametrizations [147]. Moreover, the scaling parameter can help the optimizer avoid getting stuck in local minima by balancing target governing equations.

To ensure that $\tilde{\boldsymbol{u}}$ is a reasonable approximation of $\boldsymbol{u}$, the network parameters must be determined accordingly to the BVP. As shown in Fig. 5.2, the overall loss $\mathcal{L}(\boldsymbol{\theta})$ consists of partial differential equation (PDE) losses $\mathcal{L}_{\text{PDE}}$, boundary condition losses $\mathcal{L}_{\text{BCs}}$ and experimental data losses $\mathcal{L}_{\text{EXPs}}$.

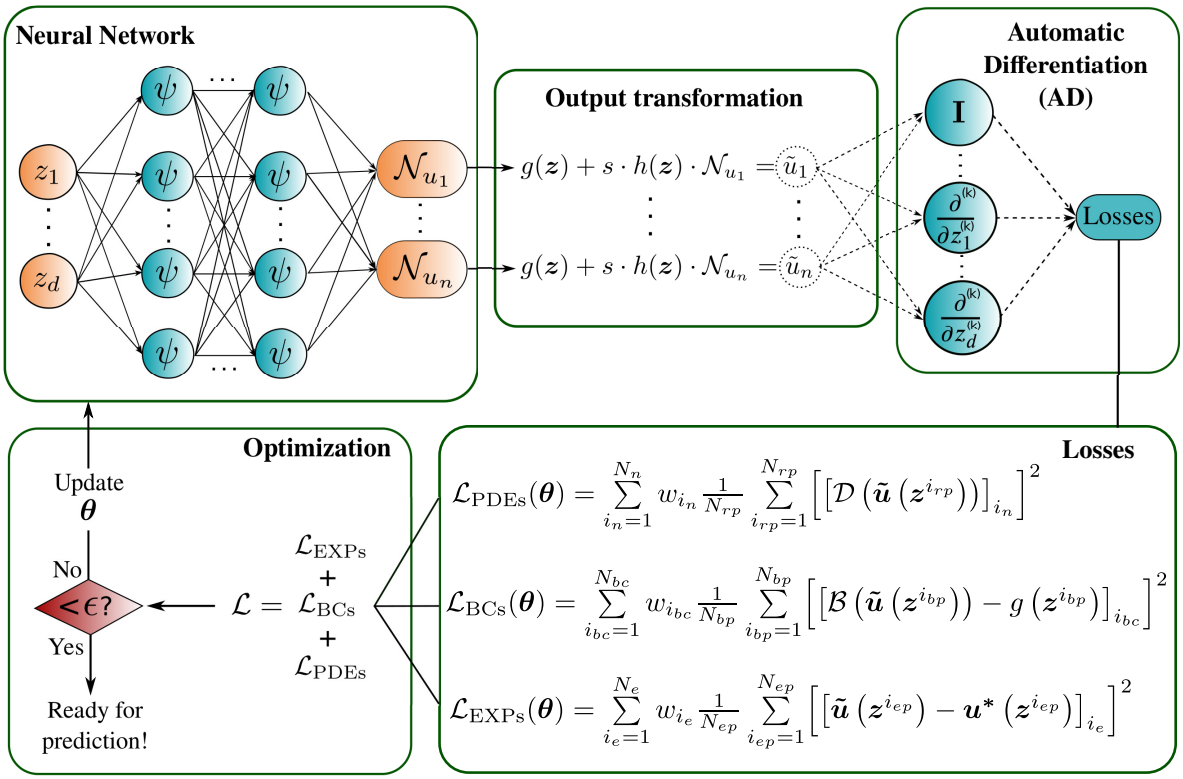


Figure 5.2.: The general representation of a physics-informed neural network for a BVP.

Note that PDE derivatives are calculated using automatic differentiation (AD) [13]. To minimize the overall loss, the optimization process continues until a prescribed tolerance ϵ is reached so that optimal network parameters $\boldsymbol{\theta}^*$ are calculated

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{5.7}$$

For each loss term, the inner loop sums up the *mean squared error* contributions of data points collected inside the domain or on the boundary. Specifically, $\{\boldsymbol{z}^{i_{rp}}\}_{i_{rp}=1}^{N_{rp}}$ denotes the collocation points in the domain, $\{\boldsymbol{z}^{i_{bp}}\}_{i_{bp}=1}^{N_{bp}}$ are the boundary points corresponding to the prescribed boundary conditions, and $\{\boldsymbol{z}^{i_{ep}}\}_{i_{ep}=1}^{N_{ep}}$ represents the points on which measurement data $\boldsymbol{u}^*$ is available. As the PDE residual $\mathcal{D}$ might have multiple terms and usually more than one boundary condition is defined, we use a lower index to explicitly point out that the inner summation is done for a given specific component, i.e., $\mathcal{D}_{i_n=1}$. Consequently, the outer loop sums up the weighted contributions coming from individual components of the loss function. The terms $\{w_{i_n}\}_{i_n=1}^{N_n}$, $\{w_{i_{bc}}\}_{i_{bc}=1}^{N_{bc}}$ and $\{w_{i_e}\}_{i_e=1}^{N_e}$ denote the loss weights for the individual components of $\mathcal{L}_{\text{PDE}}$, $\mathcal{L}_{\text{BCs}}$, $\mathcal{L}_{\text{EXPs}}$, respectively. We observe that the weighting of loss terms can be quite crucial for the convergence of the overall loss since it avoids the optimizer expanding greater efforts on loss components that have a larger order of magnitude compared to others. It should be noted that the identical collocation points $\{\boldsymbol{z}^{i_{rp}}\}_{i_{rp}=1}^{N_{rp}}$ are employed in the calculation of every component of $\mathcal{L}_{\text{PDE}}$. However, the boundary and experimental points may vary in each $\mathcal{L}_{\text{BCs}}$ and $\mathcal{L}_{\text{EXPs}}$ contribution.

5.1.1. PINNs for inverse problems

As mentioned earlier, in PINNs, one of the key advantages is the ability to solve inverse problems efficiently. In such cases, the goal is to determine unknown physical parameters when the solution is known. For example, PINNs can identify material parameters such as Young's modulus using experimental or artificially generated data.

There are three main approaches to implementing PINNs for inverse problems. Before diving into them, let us clarify the terminology for better understanding. Suppose x represents a given parameter, such as spatial coordinates, u denotes the target solution of the governing equations, and λ is the unknown parameter we wish to identify.

5.1.1.1. Network Inversion

In the first approach, referred to as *network inversion*, the target solution, which is typically modeled as the neural network's output in forward problems, now serves as the input to the network. The goal of the network is to predict the unknown parameter λ, as illustrated in Fig. 5.3a. One of the main advantages of this approach is that it does not require loss formulations due to the boundary conditions of the physical problem to create the overall loss. For instance, if the unknown parameter is a material parameter, it is enough to formulate the PDE loss. However, this method has several limitations:

- Since λ is treated as a function of x, this approach is effective for identifying parameters that act across the entire domain, but it may struggle when the unknown parameter is localized, such as a boundary load that applies only to a portion of the domain.
- Additionally, this approach requires the behavior of the target solution u across the entire domain, not just in a subset of it.
- Another constraint is that u is a function of x, which can impose practical limitations regarding implementation. In particular, for our framework hosted at `https://github.com/imcs-compsim/pinns_for_comp_mech`, this approach is not feasible, as it does not support derivatives between the network inputs.

5.1.1.2. Data-driven discovery

The second approach, referred to as *data-driven discovery*, is the most widely used method for solving inverse problems with PINNs. The core idea is straightforward: the inverse problem is reformulated as a forward problem, where the unknown parameter λ is treated as one of the network's outputs (Fig. 5.3b). While solving the forward problem, the network simultaneously learns and discovers the unknown parameter field.

Unlike the network inversion approach, data-driven discovery does not require data across the entire domain. Instead, a limited amount of data, either experimental or synthetic, on the target solution u is sufficient. This characteristic makes data-driven discovery a more practical approach, particularly for applications where full-domain data is unavailable or difficult to obtain.

However, this approach does not guarantee that the identified unknown parameter is the unique solution, even if the network accurately approximates the target solution. In other words, the

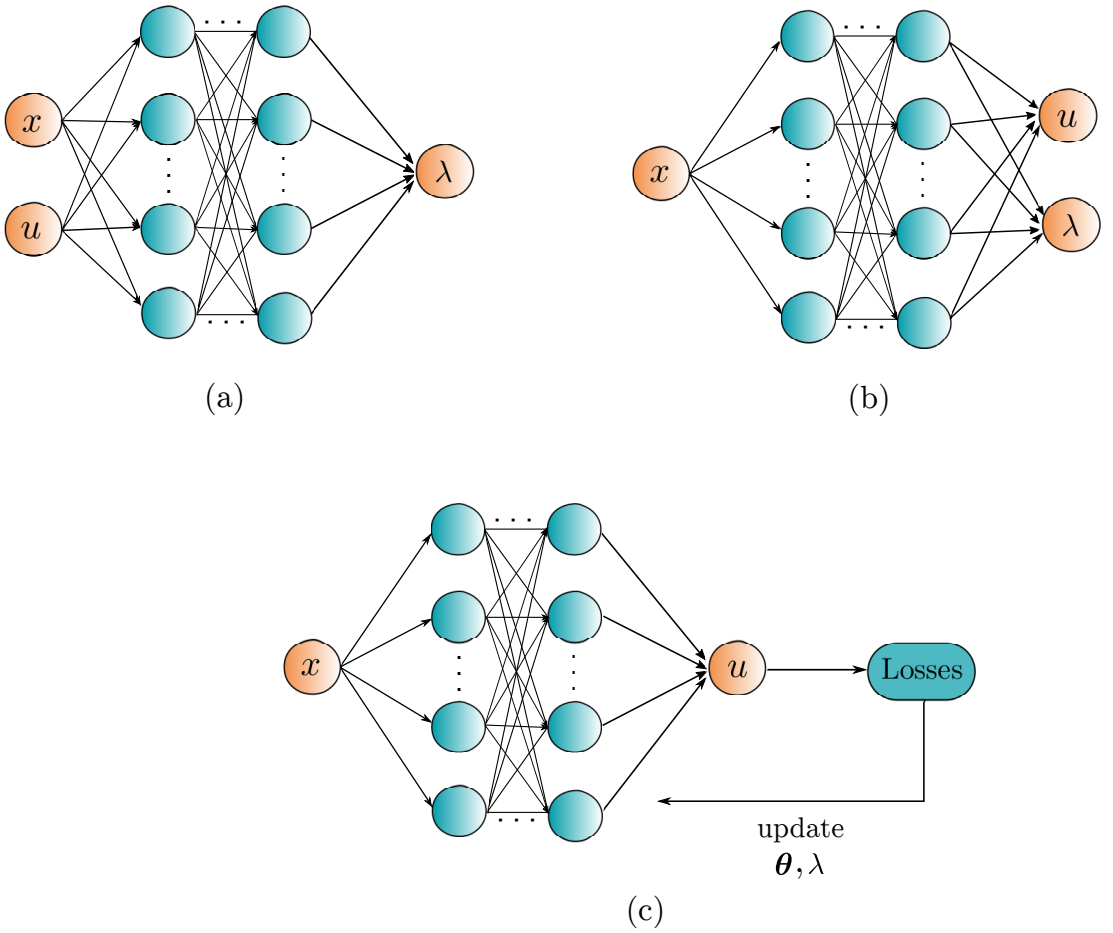


Figure 5.3.: Approaches for solving inverse problems using PINNs: (a) Network inversion, (b) Data-driven discovery, (c) Learnable parameter optimization.

network may capture one of multiple possible solutions for the unknown parameter, leading to non-uniqueness in parameter identification.

5.1.1.3. Learnable parameter via optimization

The third approach is similar to the second but differs in a key aspect. Instead of including λ as a network output, it is treated as a trainable parameter in the network, included as part of the set of trainable parameters $\boldsymbol{\theta}$ (Fig. 5.3c). The parameter λ is identified by optimizing an extended loss function that includes terms based on the difference between predictions and observations, which can be formalized as

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}), \quad \text{where} \quad \boldsymbol{\Theta} = (\lambda, \boldsymbol{\theta}). \tag{5.8}$$

This method has several advantages over the other approaches:

- λ does not need to be a function of x, allowing it to represent a parameter that applies only to a specific region of the domain, e.g., identification of a boundary load.
- This approach is particularly effective when the unknown parameter is a constant.
- Compared to the previous methods, this method does not require additional loss formulation for the unknown parameter as it is treated as a trainable parameter in the network.

Each approach has pros and cons depending on the nature of the target inverse problem. *Network inversion* is suitable for domain-wide parameters but requires the target solution across the entire domain *data-driven discovery* offers a practical way to uncover unknown parameters across the domain using limited data, but still, the unknown parameter to be identified acts on the entire domain. Finally, treating λ as a learnable parameter in the network provides flexibility, especially for constant or localized unknowns, and often leads to more straightforward implementations. However, this method can increase the computational cost for a large set of λ values. To conclude, choosing the appropriate method depends heavily on the target problem and the amount of additional data available.

Remark 5.1.1: Loss Formulation in Inverse Problems

In inverse problems, the formulation of the loss function must be tailored to the specific unknown parameter being identified. The unknown parameter may influence the governing PDEs, the BCs, or potentially both, depending on the nature of the problem. Therefore, a generalized loss formulation for inverse problems is not feasible. Instead, the loss formulation must be carefully adapted based on the occurrence and influence of the unknown parameter within the problem domain.

5.2. Element-based PINN formulations

Element-based PINN formulations extend classical PINNs by incorporating domain decomposition and numerical integration strategies inspired by the finite element method. Instead of

enforcing residuals at scattered collocation points, these approaches evaluate physics-informed quantities–such as weak residuals or energy densities–over discretized elements of the computational domain. This allows for better accuracy, improved numerical stability, and compatibility with existing finite element workflows. In this section, we detail two key element-based formulations: Variational PINNs (VPINNs), which are grounded in the weak form of the governing equations, and element-based energy PINNs (EBE-PINNs), which minimize the total potential energy of the system.

5.2.1. Variational PINNs

Variational PINNs (VPINNs) are formulated based on the variational or weak formulation of the PDE as their foundation. To obtain a generalized weak form, the introduced PDE in Eq. 5.1 is multiplied with a test function and integrated over the computational domain Ω, such as

$$\int_{\Omega} \mathcal{D}\big[\boldsymbol{u}(\boldsymbol{z})\big]\ \boldsymbol{\phi}(\boldsymbol{z}) d\boldsymbol{z} = \mathbf{0}. \tag{5.9}$$

Here, $\boldsymbol{\phi}(\boldsymbol{z})$ denotes the test function defined across the entire domain. A discrete test space with finite dimensions can be formed by employing N_k sets of admissible test functions such that

$$\boldsymbol{\phi}(\boldsymbol{z}) = \big[\phi_1(\boldsymbol{z}), \phi_2(\boldsymbol{z}), \dots, \phi_{N_k}(\boldsymbol{z})\big]. \tag{5.10}$$

Instead of evaluating an integral over the whole domain, we decompose Ω into nonoverlapping subdomains Ω^e to calculate subdomain integrals. Thus, the element-based residual for a specific test function set k, denoted as $\mathcal{R}_k^e$, takes the following form

$$\mathcal{R}_k^e = \int_{\Omega^e} \mathcal{D}^e\big[\boldsymbol{u}(\boldsymbol{z})\big]\ \phi_k^e(\boldsymbol{z}) d\boldsymbol{z}. \tag{5.11}$$

Element-based approach necessitates localized and nonoverlapping test functions denoted as ϕ_k^e. This approach is also referred to as *hp*-VPINN [89].

To compute the integral in each subdomain, a mapping is introduced from natural coordinates to each subdomain such as $\bar{\boldsymbol{z}}(\boldsymbol{\xi}) : \big[-1, 1\big]^d \rightarrow \Omega^e$, which allows the evaluation of the domain integral in natural coordinates such that

$$\mathcal{R}_k^e = \int_{\bar{\Omega}^e} \mathcal{D}^e\big[\boldsymbol{u}(\bar{\boldsymbol{z}}(\boldsymbol{\xi}))\big]\ \phi_k^e(\boldsymbol{\xi})\ \det(\mathbf{J}) d\boldsymbol{\xi}, \tag{5.12}$$

where $\bar{\Omega}^e = \big[-1, 1\big]^d$ represents the integral bounds for a dimension d of each subdomain in the natural coordinates. The Jacobian matrix $\mathbf{J}$ is defined by $\partial\bar{\boldsymbol{z}}/\partial\boldsymbol{\xi}$. The mapping can be done using linear interpolation, such as

$$\bar{\boldsymbol{z}}(\boldsymbol{\xi}) = \sum_{i=1}^{n_{\text{nodes}}} \boldsymbol{z}^i N^i(\boldsymbol{\xi}), \tag{5.13}$$

where N denotes linear shape functions and n_{nodes} represents the number of nodes per element.

Since the integral is evaluated in the bounded natural domain, numerical integration is used to approximate the integral in Eq. 5.12 such that

$$\mathcal{R}_k^e \approx \sum_{i_1=1}^{N_{x_1}^{gp}} \cdots \sum_{i_d=1}^{N_{x_d}^{gp}} c_{i_1} \cdots c_{i_d} \mathcal{D}^e \left[\boldsymbol{u}(\bar{\boldsymbol{z}}(\xi_{x_1}, \ldots, \xi_{x_d})) \right] \phi_k^e(\xi_{x_1}, \ldots, \xi_{x_d}) \det(\mathbf{J}), \tag{5.14}$$

where $N_{x_i}^{gp}$ denotes the number of Gauss points along the axis ξ_{x_i}, with corresponding coefficients or Gauss weights c_i. A detailed explanation of the numerical integration procedure is provided in Section 5.2.3.

By evaluating the integral using N_k number of test functions of each subdomain and N^e number of subdomains, the overall variational loss function of PDE is calculated as

$$\mathcal{L}_{\text{PDE}}^{\phi} = \sum_{e=1}^{N^e} \frac{1}{N_k^e} \sum_{k=1}^{N_k^e} (\mathcal{R}_k^e)^2. \tag{5.15}$$

Similar to PINNs, boundary condition losses $\mathcal{L}_{\text{BCs}}$ and experimental data losses $\mathcal{L}_{\text{EXPs}}$ are enforced as additional loss terms as shown in Fig. 5.2.

The overall process for VPINNs is summarized in Fig. 5.4. Like PINNs, a neural network takes $\boldsymbol{z}$ as input and calculates the intermediate quantity $\mathcal{N}_{u_i}$. Via output transformation, quantities of interest of physics $\tilde{\boldsymbol{u}}$ are obtained. Automatic differentiation is applied to calculate high-order derivatives of the PDE. The weak formulation is derived by integrating the strong form of the PDE over the entire domain, involving a suitable set of test functions $\boldsymbol{\phi}$. Instead of evaluating the integral over the entire domain, the domain Ω is partitioned into non-overlapping subdomains Ω^e, called domain decomposition. A numerical integration scheme is performed based on Gauss integration with coefficients c_i and a mapping from physical to natural coordinates to integrate individual element-based integrals. In PINNs, the residual is assessed at collocation points, whereas in VPINNs, the residual is evaluated at quadrature points. The element-wise variational loss is computed for each subdomain, employing a set of k test functions, and the variational PDE loss over the entire domain $\mathcal{L}_{\text{PDE}}^{\phi}$ is determined by aggregating the individual loss contributions across all elements. The overall loss is minimized after involving the contributions of boundary condition losses and experimental data losses until a prescribed tolerance is reached. Furthermore, the procedure shown in Fig. 5.4 is structured using the Petrov-Galerkin method, in which solution and test functions are chosen independently. The solution is expressed by a nonlinear approximation of a neural network and the test functions are chosen from linear function spaces.

A test function is chosen based on the target problem and desired properties of the solution. There are several ways to choose the test functions. At the core, the selection of test functions can be divided into two groups: global test functions and local test functions. Since the formulation is based on elements, the appropriate selection would involve localized test functions. The possible choices for nonoverlapping polynomials include modified Legendre polynomials, Legendre shape functions and Lagrange shape functions. Modified Legendre polynomials have the following form

$$\phi_k(\boldsymbol{\xi}) = P_{k+1}(\boldsymbol{\xi}) - P_{k-1}(\boldsymbol{\xi}), \tag{5.16}$$

which is generated by a combination of Legendre polynomials P. The key characteristic of the modified Legendre polynomial test functions is their property to vanish at the boundaries of the

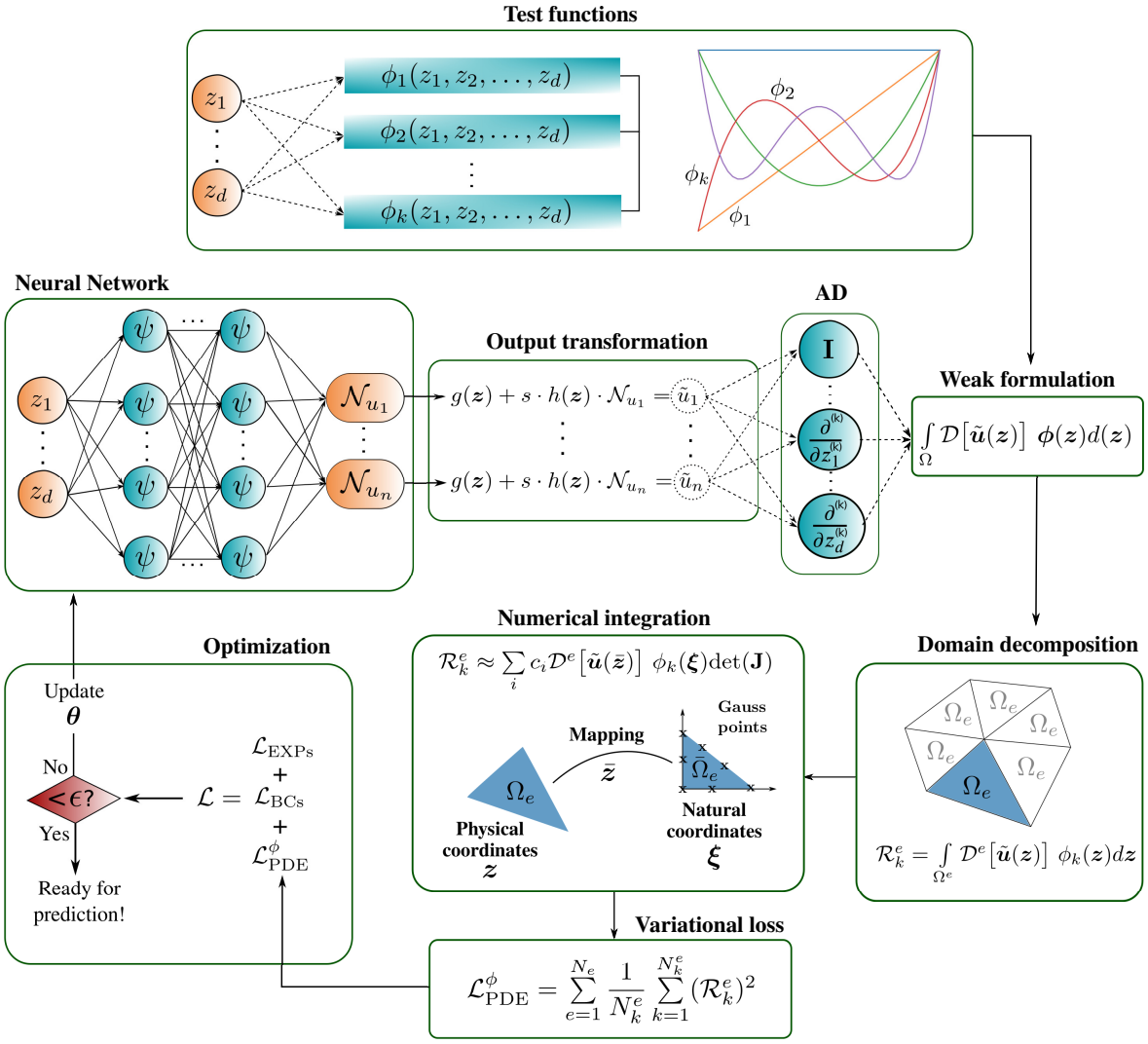


Figure 5.4.: The general representation of variational physics-informed neural networks.

integration domains. Note that the k-th test function is formed by the multiplication of the d number of test functions for higher dimensions as

$$\phi_k(\boldsymbol{\xi}) = \phi_k(\xi_{x_1})\phi_k(\xi_{x_2})\dots\phi_k(\xi_{x_d}). \tag{5.17}$$

Another possible choice is the Legendre shape functions defined as

$$\phi_k(\boldsymbol{\xi}) = \begin{cases} \dfrac{1-\boldsymbol{\xi}}{2}, & \text{for } k=1, \\ \dfrac{1+\boldsymbol{\xi}}{2}, & \text{for } k=2, \\ \dfrac{P_{k-1}(\boldsymbol{\xi}) - P_{k-3}(\boldsymbol{\xi})}{\sqrt{2(2k-3)}}, & \text{for } k \geq 3. \end{cases} \tag{5.18}$$

Legendre shape functions form a hierarchical basis, implying that advancing from k to $k+1$ does not necessitate recomputing basis functions. Consequently, lower-polynomial orders, such as linear basis functions remain the same during the computation of higher-order functions. Compared to modified Legendre functions, Legendre shape functions incorporate linear basis functions that do not vanish on the boundary. This characteristic could be highly advantageous for certain problem scenarios.

Last but not least, another potential choice for a test function is the Lagrange shape functions, defined as follows:

$$\phi_k(\boldsymbol{\xi}) = \prod_{i=0, i\neq k}^{k} \frac{\boldsymbol{\xi} - \boldsymbol{\xi}^i}{\boldsymbol{\xi}^k - \boldsymbol{\xi}^i}. \tag{5.19}$$

where i denotes the number of interpolation nodes. Compared to Legendre-based formulations, Lagrange shape functions are generated based on the interpolation on nodes. As an example, six equidistant nodes are generated in the interval [-1,1] to generate Lagrange polynomials as shown in Fig. 5.5c. Similar to Legendre shape functions, Lagrange shape functions do not completely vanish on the boundary due to the interpolation property. This means shape functions $\phi_k(\boldsymbol{\xi}^i)$ take the value of 1 at the node i and zero at the remaining nodes, such as

$$\phi_k(\boldsymbol{\xi}^i) = \begin{cases} 1 & \text{if} \quad i = k, \\ 0 & \text{if} \quad i \neq k. \end{cases} \tag{5.20}$$

Moreover, Fig. 5.5 illustrates the first-order derivatives of modified Legendre polynomials, Legendre shape functions and Lagrange shape functions. Note that in the Lagrange-based formulation, higher-order test functions may exhibit significantly elevated derivative values near the end nodes, potentially leading to larger loss values.

5.2.2. Element-based energy PINNs

Element-based energy physics-informed neural networks (EBE-PINNs) are formulated based on the principle of energy minimization. Instead of enforcing a strong or weak form of the governing

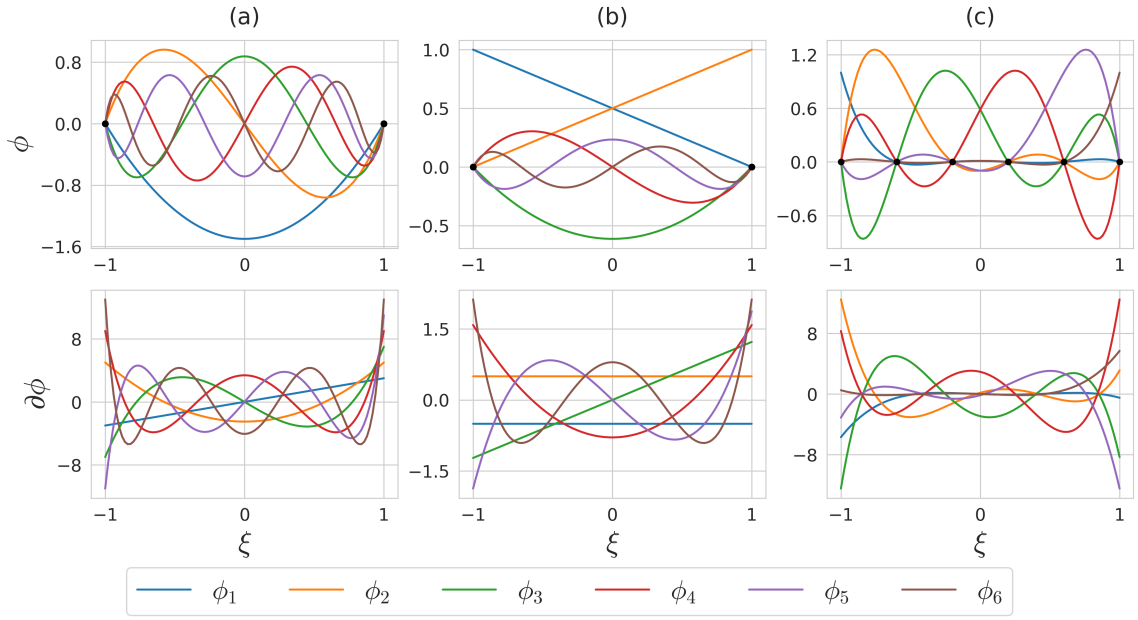


Figure 5.5.: An illustration of introduced test functions ϕ_k and their first-order derivatives $\partial\phi_k$ up to $k = 6$: (a) modified Legendre polynomials, (b) Legendre shape functions, and (c) Lagrange shape functions.

PDE, the model is trained by minimizing a scalar energy functional defined over the domain. The energy functional is expressed in general form as

$$\mathbb{E}[\boldsymbol{u}] = \int_{\Omega} \Pi\big[\boldsymbol{u}(\boldsymbol{z})\big]\, d\boldsymbol{z}, \tag{5.21}$$

where $\Pi[\cdot]$ denotes a generalized energy density functional, $\boldsymbol{u}$ is the neural network approximation of the field variable of interest, and $\mathbb{E}$ represents the total potential energy of the system.

To facilitate localized computation and numerical integration, the domain Ω is decomposed into a collection of non-overlapping subdomains Ω^e. The total energy is then represented as the sum of element-wise energy contributions such as

$$\mathbb{E} = \sum_{e=1}^{N^e} \mathbb{E}^e, \quad \text{where} \quad \mathbb{E}^e = \int_{\Omega^e} \Pi^e\big[\boldsymbol{u}(\boldsymbol{z})\big]\, d\boldsymbol{z}. \tag{5.22}$$

Here, N^e denotes the total number of elements. Each element-wise integral is mapped from physical coordinates to natural coordinates $\boldsymbol{\xi} \in [-1, 1]^d$ using a transformation defined by linear shape functions:

$$\bar{\boldsymbol{z}}(\boldsymbol{\xi}) = \sum_{i=1}^{n_{\text{nodes}}} \boldsymbol{z}^i N^i(\boldsymbol{\xi}), \tag{5.23}$$

where n_{nodes} is the number of element nodes and N^i denotes the corresponding shape functions.

The energy integral within each element is then evaluated in natural coordinates as

$$\mathbb{E}^e = \int_{\bar{\Omega}^e} \Pi^e\big[\boldsymbol{u}(\bar{\boldsymbol{z}}(\boldsymbol{\xi}))\big]\ \det(\mathbf{J})\, d\boldsymbol{\xi}, \tag{5.24}$$

where $\bar{\Omega}^e = [-1, 1]^d$ and $\mathbf{J} = \partial \boldsymbol{z} / \partial \boldsymbol{\xi}$ is the Jacobian matrix of the transformation for element e.

To approximate the element-wise integral, Gaussian quadrature is used in the natural coordinate system such that

$$\mathbb{E}^e \approx \sum_{i_1=1}^{N_{x_1}^{gp}} \dots \sum_{i_d=1}^{N_{x_d}^{gp}} c_{i_1} \cdots c_{i_d} \, \Pi^e \left[\boldsymbol{u}(\bar{\boldsymbol{z}}(\xi_{x_1}, \dots, \xi_{x_d})) \right] \, \det(\mathbf{J}), \tag{5.25}$$

where c_i are Gauss quadrature weights and $\bar{\boldsymbol{z}}(\xi_{x_1}, \dots, \xi_{x_d})$ is the mapped quadrature point. A detailed explanation of the numerical integration procedure is provided in Section 5.2.3.

The overall energy-based loss function, which is equal to the overall potential energy, is assembled by summing the element-wise energy contributions as

$$\mathcal{L}_{\mathbb{E}} = \mathbb{E} = \sum_{e=1}^{N^e} \mathbb{E}^e. \tag{5.26}$$

Additional loss terms can be appended for Dirichlet boundary conditions $\mathcal{L}_{\text{DBCs}}$ and experimental data $\mathcal{L}_{\text{EXPs}}$, forming the total loss function:

$$\mathcal{L} = \mathcal{L}_{\mathbb{E}} + \mathcal{L}_{\text{DBCs}} + \mathcal{L}_{\text{EXPs}}. \tag{5.27}$$

The complete EBE-PINN pipeline is illustrated in Fig. 5.6. A neural network takes spatial coordinates $\boldsymbol{z}$ as input and outputs intermediate representations $\mathcal{N}_{u_i}$, which are transformed into physical quantities of interest $\boldsymbol{u}$. Automatic differentiation is applied as needed to evaluate derivatives inside the energy functional. The domain is decomposed into elements, and energy integrals are evaluated locally using Gaussian quadrature over mapped natural coordinates. The total energy, along with additional loss terms, is minimized using gradient-based optimization.

Unlike traditional PINNs that evaluate residuals at collocation points, EBE-PINNs directly minimize the total energy functional, offering a variationally consistent framework grounded in energy principles without the explicit use of test functions.

5.2.3. Numerical integration

In element-based PINN formulations, numerical integration plays a central role in computing domain-specific quantities such as residuals, weak forms, or energy functionals. Unlike pointwise evaluation in classical PINNs, these methods require evaluating integrals over the spatial domain, which is typically decomposed into a finite number of non-overlapping elements.

To perform these integrals efficiently and accurately, a mapping is established between a reference (natural) coordinate system and the physical coordinates of each element. This mapping is constructed using shape functions, which interpolate both geometry and field quantities over the element. The transformation requires the Jacobian matrix $\mathbf{J}$, whose determinant scales the integration measure to account for geometric deformation of the element in physical space.

In this thesis, we focus on linear shape functions corresponding to standard finite elements. For 2D problems, triangular and quadrilateral elements are used, while for 3D cases, tetrahedral and hexahedral elements are adopted. These choices strike a balance between geometric flexibility and computational efficiency.

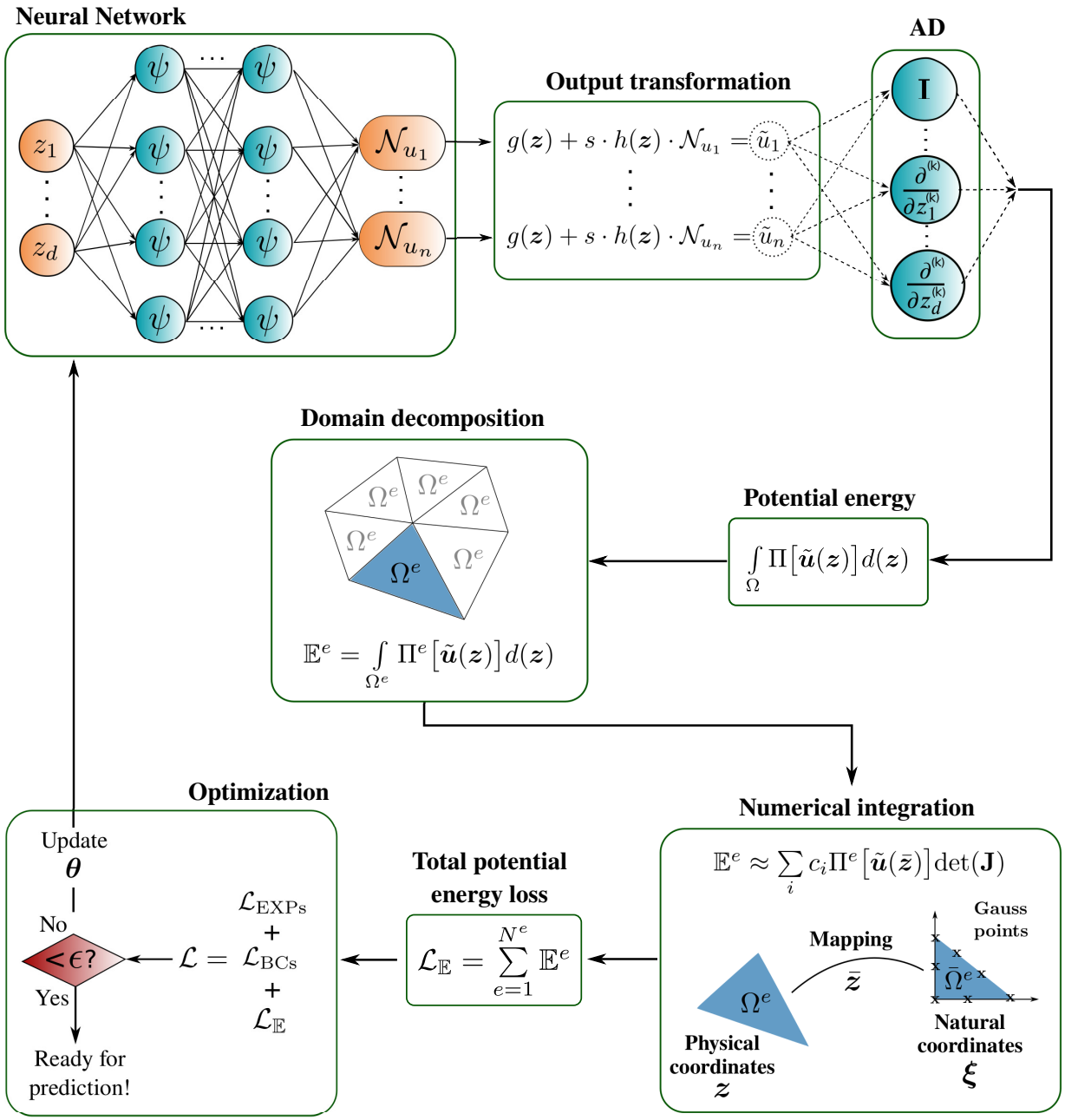


Figure 5.6.: The general representation of element-based energy physics-informed neural networks (EBE-PINNs).

Numerical integration is then carried out in the natural coordinate space using Gaussian quadrature rules. For each element, the integrand is evaluated at a set of quadrature points, and the total integral is approximated as a weighted sum over these points:

$$\int_{\Omega^e} f(\boldsymbol{z})\, d\boldsymbol{z} \approx \sum_{i=1}^{n_{\mathrm{gp}}} c_i\, f(\bar{\boldsymbol{z}}(\boldsymbol{\xi}))\, \det(\mathbf{J}), \tag{5.28}$$

where f is the integrand (e.g., energy density or weak form expression), $\bar{\boldsymbol{z}}(\boldsymbol{\xi})$ are the mapped coordinates of quadrature points, c_i are the corresponding weights, and $\det(\mathbf{J})$ is the determinant of the Jacobian.

In general, the mapping from natural coordinates $\boldsymbol{\xi}$ to physical coordinates $\bar{\boldsymbol{x}}(\boldsymbol{\xi})$ is expressed as

$$\bar{\boldsymbol{z}}(\boldsymbol{\xi}) = \sum_{i=1}^{n_{\mathrm{nodes}}} \boldsymbol{z}^i N^i(\boldsymbol{\xi}), \tag{5.29}$$

where N^i are the shape functions associated with node i, and $\boldsymbol{z}^i$ are the coordinates of node i in the physical domain. Similarly, the Jacobian matrix is defined as

$$\mathbf{J} = \partial \bar{\boldsymbol{z}}(\boldsymbol{\xi}) / \partial \boldsymbol{\xi}. \tag{5.30}$$

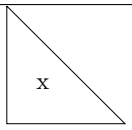
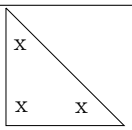
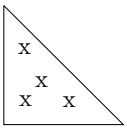
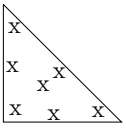
n	$f(\xi,\eta)$	ξ_i	η_j	c_i
1		$\xi_1 = 1/3$	$\eta_1 = 1/3$	$c_1 = 1$
3		$\xi_1 = 1/6$ $\xi_2 = 2/3$ $\xi_3 = 1/6$	$\eta_1 = 1/6$ $\eta_2 = 1/6$ $\eta_3 = 2/3$	$c_1 = 1/3$ $c_2 = 1/3$ $c_3 = 1/3$
4		$\xi_1 = 1/3$ $\xi_2 = 3/5$ $\xi_3 = 1/5$ $\xi_4 = 1/5$	$\eta_1 = 1/3$ $\eta_2 = 1/5$ $\eta_3 = 3/5$ $\eta_4 = 1/5$	$c_1 = -27/48$ $c_2 = 25/48$ $c_3 = 25/48$ $c_4 = 25/48$
7		$\xi_1 = 0.33333$ $\xi_2 = 0.10128$ $\xi_3 = 0.79742$ $\xi_4 = \eta_2$ $\xi_5 = 0.47014$ $\xi_6 = \eta_5$ $\xi_7 = 0.05971$	$\eta_1 = \xi_1$ $\eta_2 = \xi_2$ $\eta_3 = \xi_2$ $\eta_4 = \xi_3$ $\eta_5 = \xi_5$ $\eta_6 = \xi_2$ $\eta_7 = \eta_5$	$c_1 = 0.22500$ $c_2 = 0.12593$ $c_3 = c_2$ $c_4 = c_2$ $c_5 = 0.13239$ $c_6 = c_5$ $c_7 = c_5$

Table 5.1.: Gauss points ξ_i, η_j and weighting factors c_i of Gauss-Legendre quadrature for triangular elements.

Gauss points ξ_i, η_j and weighting factors c_i for Gauss-Legendre quadrature over triangular elements are listed in Table 5.1. For a detailed treatment of quadrature rules for other element types, we refer the reader to Zienkiewicz et al. [198]. Note that Eq.(5.28) should include a scaling factor of $1/2$ for triangular elements and $1/6$ for tetrahedral elements, accounting for the area or volume of the reference element.

1D elements

For a 1D element, there are two nodes, and the natural coordinate is defined in the interval $\xi \in [-1, 1]$. The linear shape functions are defined as

$$\begin{aligned} N^1(\xi) &= \frac{1}{2}(1-\xi), \\ N^2(\xi) &= \frac{1}{2}(1+\xi), \end{aligned} \tag{5.31}$$

with the corresponding derivatives

$$N^1_{,\xi}(\xi) = -\frac{1}{2}, \tag{5.32}$$

$$N^2_{,\xi}(\xi) = \frac{1}{2}. \tag{5.33}$$

The mapped coordinate $\bar{\boldsymbol{z}} = \bar{x}$ of a point within the element is obtained by interpolating the nodal coordinates as

$$\bar{x}(\xi) = \sum_{i=1}^{2} x^i N^i(\xi), \tag{5.34}$$

where x^i are the physical coordinates of the i^{th} node.

The Jacobian $\mathbf{J}$ of the 1D mapping reduces to a scalar quantity representing the derivative of $\bar{x}$ with respect to ξ as follows

$$\mathbf{J} = \frac{\partial \bar{x}}{\partial \xi} = \sum_{i=1}^{2} x^i \frac{dN^i}{d\xi} = \frac{1}{2}(x^2 - x^1). \tag{5.35}$$

Note that this formulation for computing the Jacobian assumes that the element's physical axis aligns with the natural coordinate axis. When the 1D element is rotated in space, the Jacobian must be computed as

$$\mathbf{J} = \frac{1}{2}\left|\boldsymbol{x}^2 - \boldsymbol{x}^1\right| = \frac{1}{2}\sqrt{(x^2-x^1)^2 + (y^2-y^1)^2 + (z^2-z^1)^2}. \tag{5.36}$$

Triangular elements

For a linear triangular element, there are three nodes, and the natural coordinates are usually defined in terms of the reference triangle with $(\xi, \eta) \in [0, 1]$. The linear shape functions are

$$\begin{aligned} N^1(\xi,\eta) &= 1 - \xi - \eta, \\ N^2(\xi,\eta) &= \xi, \\ N^3(\xi,\eta) &= \eta, \end{aligned} \tag{5.37}$$

with the corresponding derivatives

$$N^1_{,\xi}(\xi,\eta) = -1, \qquad N^1_{,\eta}(\xi,\eta) = -1, \tag{5.38}$$

$$N^2_{,\xi}(\xi,\eta) = 1, \qquad N^2_{,\eta}(\xi,\eta) = 0, \tag{5.39}$$

$$N^3_{,\xi}(\xi,\eta) = 0, \qquad N^3_{,\eta}(\xi,\eta) = 1. \tag{5.40}$$

The mapped coordinates $\bar{\boldsymbol{z}} = (\bar{x}, \bar{y})$ of a point within the element are obtained by interpolating the nodal coordinates using the following equations

$$\begin{aligned} \bar{x}(\xi,\eta) &= \sum_{i=1}^{3} x^i N^i(\xi,\eta), \\ \bar{y}(\xi,\eta) &= \sum_{i=1}^{3} y^i N^i(\xi,\eta), \end{aligned} \tag{5.41}$$

where (x^i, y^i) are the physical coordinates of the i^{th} node.

The Jacobian matrix $\mathbf{J}$ of the mapping is given by:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \bar{x}}{\partial \xi} & \dfrac{\partial \bar{x}}{\partial \eta} \\ \dfrac{\partial \bar{y}}{\partial \xi} & \dfrac{\partial \bar{y}}{\partial \eta} \end{bmatrix}. \tag{5.42}$$

The same form of the Jacobian matrix shown in Eq. 5.42 is also used for quadrilateral elements, as both triangular and quadrilateral elements involve mappings from 2D natural coordinates (ξ, η) to 2D physical coordinates $(\bar{x}, \bar{y})$.

Quadrilateral elements

For a linear quadrilateral element, there are four nodes, and the natural coordinates are defined within the reference square $(\xi, \eta) \in [-1, 1]$. The bilinear shape functions are defined as

$$\begin{aligned} N^1(\xi,\eta) &= \frac{1}{4}(1-\xi)(1-\eta), \\ N^2(\xi,\eta) &= \frac{1}{4}(1+\xi)(1-\eta), \\ N^3(\xi,\eta) &= \frac{1}{4}(1+\xi)(1+\eta), \\ N^4(\xi,\eta) &= \frac{1}{4}(1-\xi)(1+\eta). \end{aligned} \tag{5.43}$$

The derivatives of the shape functions with respect to the natural coordinates ξ and η are

$$N^1_{,\xi} = -\frac{1}{4}(1-\eta), \qquad N^1_{,\eta} = -\frac{1}{4}(1-\xi), \tag{5.44}$$

$$N^2_{,\xi} = \frac{1}{4}(1-\eta), \qquad N^2_{,\eta} = -\frac{1}{4}(1+\xi), \tag{5.45}$$

$$N^3_{,\xi} = \frac{1}{4}(1+\eta), \qquad N^3_{,\eta} = \frac{1}{4}(1+\xi), \tag{5.46}$$

$$N^4_{,\xi} = -\frac{1}{4}(1+\eta), \qquad N^4_{,\eta} = \frac{1}{4}(1-\xi). \tag{5.47}$$

The mapped coordinates $(\bar{x}, \bar{y})$ of a point within the element are obtained by interpolating the nodal coordinates

$$\begin{aligned} \bar{x}(\xi, \eta) &= \sum_{i=1}^{4} x^i N^i(\xi, \eta), \\ \bar{y}(\xi, \eta) &= \sum_{i=1}^{4} y^i N^i(\xi, \eta). \end{aligned} \tag{5.48}$$

Tetrahedral elements

A linear tetrahedral element has four nodes, and the natural coordinates are given in terms of barycentric coordinates (ξ, η, ζ), with the constraint $\xi + \eta + \zeta \leq 1$ and $\xi, \eta, \zeta \geq 0$. The shape functions are defined as

$$\begin{aligned} N^1(\xi, \eta, \zeta) &= 1 - \xi - \eta - \zeta, \\ N^2(\xi, \eta, \zeta) &= \xi, \\ N^3(\xi, \eta, \zeta) &= \eta, \\ N^4(\xi, \eta, \zeta) &= \zeta. \end{aligned} \tag{5.49}$$

The derivatives of the shape functions with respect to the natural coordinates are constant and obtained as follows

$$N^1_{,\xi}(\xi, \eta, \zeta) = -1, \qquad N^1_{,\eta}(\xi, \eta, \zeta) = -1, \qquad N^1_{,\zeta}(\xi, \eta, \zeta) = -1, \tag{5.50}$$

$$N^2_{,\xi}(\xi, \eta, \zeta) = 1, \qquad N^2_{,\eta}(\xi, \eta, \zeta) = 0, \qquad N^1_{,\zeta}(\xi, \eta, \zeta) = 0, \tag{5.51}$$

$$N^3_{,\xi}(\xi, \eta, \zeta) = 0, \qquad N^3_{,\eta}(\xi, \eta, \zeta) = 1 \qquad N^1_{,\zeta}(\xi, \eta, \zeta) = 0, \tag{5.52}$$

$$N^3_{,\xi}(\xi, \eta, \zeta) = 0, \qquad N^3_{,\eta}(\xi, \eta, \zeta) = 0 \qquad N^1_{,\zeta}(\xi, \eta, \zeta) = 1. \tag{5.53}$$

The mapped coordinates $(\bar{x}, \bar{y}, \bar{z})$ are obtained by:

$$\begin{aligned} \bar{x} &= \sum_{i=1}^{4} x^i N^i(\xi, \eta, \zeta), \\ \bar{y} &= \sum_{i=1}^{4} y^i N^i(\xi, \eta, \zeta), \\ \bar{z} &= \sum_{i=1}^{4} z^i N^i(\xi, \eta, \zeta). \end{aligned} \tag{5.54}$$

The Jacobian matrix $\mathbf{J}$ is given as:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \bar{x}}{\partial \xi} & \dfrac{\partial \bar{x}}{\partial \eta} & \dfrac{\partial \bar{x}}{\partial \zeta} \\ \dfrac{\partial \bar{y}}{\partial \xi} & \dfrac{\partial \bar{y}}{\partial \eta} & \dfrac{\partial \bar{y}}{\partial \zeta} \\ \dfrac{\partial \bar{z}}{\partial \xi} & \dfrac{\partial \bar{z}}{\partial \eta} & \dfrac{\partial \bar{z}}{\partial \zeta} \end{bmatrix}. \tag{5.55}$$

Similarly, the same form of the Jacobian matrix shown in Eq. 5.55 is also used for hexahedral elements, as both tetrahedral and hexahedral elements involve mappings from 3D natural coordinates (ξ, η, ζ) to 3D physical coordinates $(\bar{x}, \bar{y}, \bar{z})$.

Hexahedral elements

A linear hexahedral element has eight nodes, and the natural coordinates $(\xi, \eta, \zeta) \in [-1, 1]^3$ define the reference cube. The shape functions are trilinear and defined as:

$$N^i(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi\xi^i)(1 + \eta\eta^i)(1 + \zeta\zeta^i), \tag{5.56}$$

where $\xi^i, \eta^i, \zeta^i \in \{-1, 1\}$ are the signs associated with the natural coordinates of the i^{th} node in the reference element.

The mapped coordinates $(\bar{x}, \bar{y}, \bar{z})$ are given by:

$$\begin{aligned} \bar{x} &= \sum_{i=1}^{8} x^i N^i(\xi, \eta, \zeta), \\ \bar{y} &= \sum_{i=1}^{8} y^i N^i(\xi, \eta, \zeta), \\ \bar{z} &= \sum_{i=1}^{8} z^i N^i(\xi, \eta, \zeta), \end{aligned} \tag{5.57}$$

where (x^i, y^i, z^i) are the physical coordinates of the i^{th} node.

The derivatives of the shape functions with respect to ξ, η, ζ are computed using the product rule, based on:

$$\frac{\partial N^i}{\partial \xi} = \frac{1}{8}\xi^i(1 + \eta\eta^i)(1 + \zeta\zeta^i), \tag{5.58}$$

and similarly for $\partial N^i/\partial\eta$ and $\partial N^i/\partial\zeta$.

5.3. Algorithmic challenges

Although PINNs demonstrate that they can introduce powerful methods, they face algorithmic challenges that impact model performance and training efficiency. Unlike traditional neural networks, which are often considered black-box models, PINNs incorporate underlying physical laws, making them much more interpretable. This inclusion of physics transforms PINNs into ”gray-box” models, offering more profound insights into the problem being solved. However, on the negative side, this transformation also introduces new challenges that are not typically encountered in traditional neural networks. On the positive side, integrating physics into the model not only increases interpretability but also enables having more control to construct methods to overcome those introduced challenges.

5.3.1. Vanishing gradients: Choosing a proper activation function

As explained in Section 3.4.3.2, there are many types of activation functions, and the choice of activation function directly affects model accuracy.

Recalling the update rule in backpropagation

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)} \boldsymbol{a}^{(l-1)^\top}$$
$$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)},$$

where

$$\delta^{(l)} = \boldsymbol{W}^{(l+1)} \delta^{(l+1)} \odot \frac{\partial \psi(\boldsymbol{z}^{(l)})}{\partial \boldsymbol{z}^{(l)}}. \tag{5.59}$$

When the arbitrary layer l is the output layer, then the error term δ becomes

$$\delta^{(L)} = (\hat{y} - y) \odot \frac{\partial \psi(\boldsymbol{z}^{(L)})}{\partial \boldsymbol{z}^{(L)}}. \tag{5.60}$$

In traditional neural networks, the most commonly used activation function is the ReLU activation, which is a piece-wise linear function. As provided in Eqs. 5.59 and 5.60, choosing ReLU activation does not create any problems for traditional neural networks, since derivatives of the activation function with respect to the output on a previous layer are not zero. In other words, error terms do not vanish. Even using a linear activation function will not create any problems. However, this is not the case for PINNs. The loss function of PINNs already includes derivatives. It is not the difference between the predicted and actual target quantities anymore, but it is a function of at least first-order derivatives with respect to the input. Thus, the error terms would include at least second-order derivatives due to backpropagation and the given problem's nature. If the second-order derivatives of the chosen activation function are zero, then the error terms will vanish, causing the network not to learn at all.

Let us verify this statement using a simple example. Consider the following ordinary differential equation (ODE) with initial condition

$$\frac{dy}{dt} = \sin(t)$$
$$y(0) = 1,$$

where $t = [0, 2\pi]$. The analytical solution can be easily derived using the initial condition as follows

$$y = -\cos(t) + 2.$$

We present a PINN architecture designed to solve the given ODE. The architecture consists of one input layer for the input variable t, three hidden layers with 50 neurons each, and a single output layer for the variable y_{pred}. To emphasize the significance of the activation function in the learning process, we investigate two distinct activation functions: the ReLU and tanh activation functions. For both models, *Adam* optimization is used to train models for $10{,}000$ epochs.

The key distinction between these activation functions is their behavior with respect to second derivatives. The second derivative of the ReLU activation function is zero, leading to the so-called

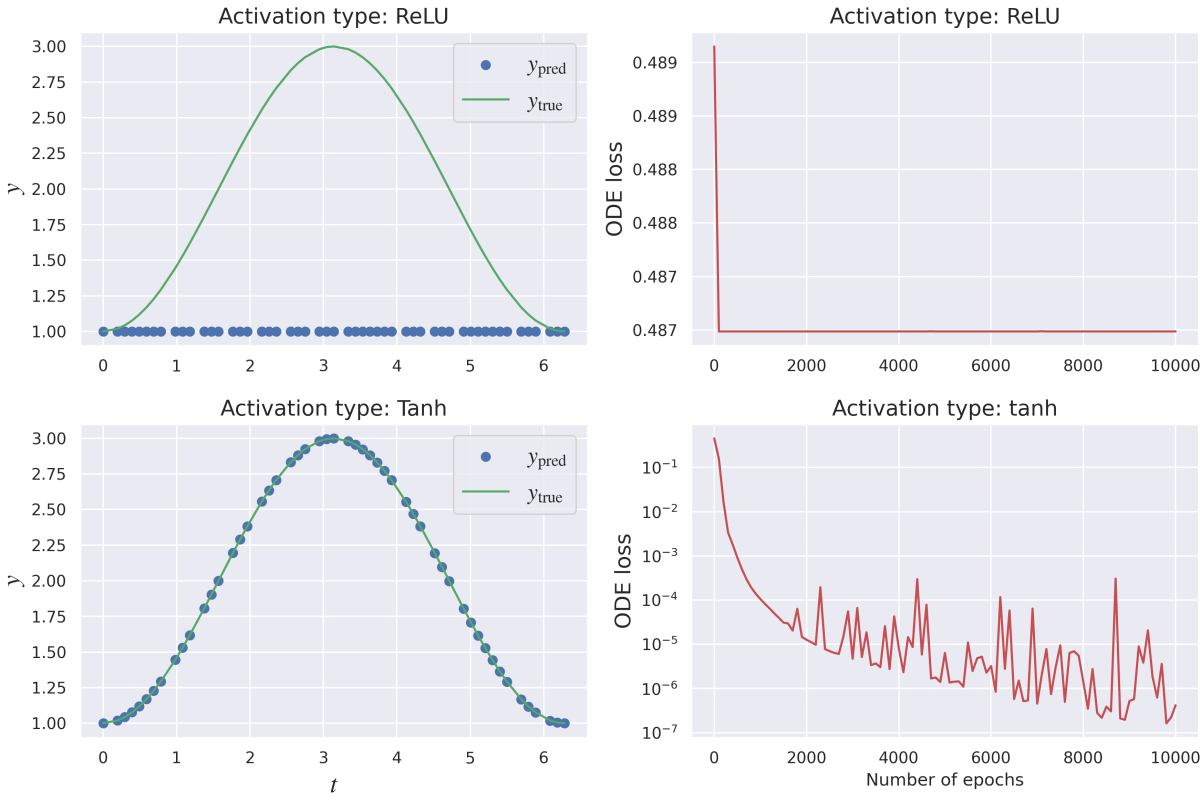


Figure 5.7.: Comparison of ReLU and tanh activation functions for solving an ODE system. Plots include the comparison of predicted solutions with true or analytical solutions and corresponding ODE loss evolutions through epochs.

vanishing gradient problem during training. In contrast, the tanh activation function maintains non-zero second derivatives, enabling more effective gradient propagation through the neural network.

Fig. 5.7 illustrates the predicted values y_{pred} versus the true values y_{true} for the solution of the exemplary ODE, comparing the ReLU and tanh activation functions. The first row displays the predictions and the corresponding ODE loss evolution over epochs for the ReLU activation function, while the second row presents the same for the tanh activation function.

The network utilizing the tanh activation function successfully approximates the true solution, demonstrating that the model effectively learns the underlying dynamics of the target ODE. In contrast, the results obtained with the ReLU activation function exhibit significant deviations from the true solution. The accompanying ODE loss plots further support this observation, showing that the model with the ReLU activation function struggles to learn the model parameters effectively due to the vanishing gradient problem. The loss function remains constant after the first iteration at $\mathcal{L}_{\text{ODE}} = 0.487$. On the other hand, the ODE loss for the model with the tanh activation function converges toward zero, demonstrating effective learning. However, some oscillations are observed due to the nature of the chosen optimizer.

5.3.2. Stiffness of PDEs

The stiffness of a PDE in a neural network arises when terms within the same loss function have different magnitudes at the initial stages of training, often due to the natural characteristics of the target PDE and the random initialization of network parameters. Particularly for the mixed-variable formulations (e.g. Section 2.2.2.3), where the network predicts multiple variables (e.g., stress and displacement or other field variables), random initialization can lead to initial outputs with different scales depending on the physical properties and relations between these variables.

Standard initialization methods, such as Glorot initialization, generate random estimates for the network parameters following a normal distribution with a mean of zero and a standard deviation of one [59]. However, this approach can cause convergence issues, as network outputs may begin with magnitudes that do not reflect the actual scales of each variable in the PDE. When network predictions for coupled variables have similar magnitudes, terms involving large constants can dominate the loss function in the first iteration. As a result, the optimizer faces the challenge of balancing a large-scale loss contribution, leading to potentially unstable variable adjustments.

Such significant adjustments in the optimization process are problematic, as they can lead to vanishing gradients, especially for standard activation functions whose gradients approach zero with extensive function arguments [76]. This makes it difficult for the optimizer to reduce the loss and achieve convergence efficiently.

A common approach to addressing this challenge is scaling specific network outputs using output scaling. This, indeed, necessitates knowledge regarding the physical target problem. Output scaling helps the optimizer maintain stable gradients and ensures that the loss terms contribute more equally to the overall optimization process, facilitating convergence.

5.3.3. Balancing of loss terms

As explained in Section 5.1, target problems often involve multiple loss terms due to the nature and dimensionality of the problem. In such cases, it is crucial to balance the contributions of these loss terms, which can be done by setting appropriate weights for each component. Certain loss terms might dominate others during training without proper balancing, causing the neural network to ignore important aspects associated with the so-called less influential components. For example, the predicted solution may satisfy the governing equations but fail to fit the data. Hence, balancing the loss weights is a critical step in the training process. A simple approach is to check the training loss of each component at the first iteration, as this can provide insight into which loss terms are dominating and guide how the loss weights for the individual components should be adjusted accordingly.

5.3.4. Non-uniqueness in inverse problems

Another problem one might face is that PINNs might struggle to find a unique solution, particularly for inverse problems, which are used to identify the problem parameters. This can be particularly the case where multiple problem parameter sets might fulfill governing equations, including boundary conditions. However, these identified parameters do not match the actual values. This can be avoided by incorporating more experimental data and additional physical constraints. For

instance, if prior knowledge or relations exist within unknown target parameters, this can be enforced by introducing an additional loss component.

5.3.5. Gradient calculation

PINNs rely on automatic differentiation to calculate the gradients of the PDE residuals and the gradients used in boundary conditions. While this collocation-based approach is reasonably practical, the automatic differentiation can become computationally expensive, particularly for PDEs, including higher-order partial derivatives. As seen in Section 2.1.2, the governing equations of beam mechanics include fourth-order derivatives.

Mixed-variable approaches can be developed to alleviate computational costs due to higher-order derivatives. In such approaches, higher-order derivatives are avoided by introducing additional coupling operators that transform the original higher-order derivatives into a system of first-order or lower-order equations. This method, indeed, increases the number of loss terms to be minimized during the optimization process; however, it significantly reduces the computational cost caused by higher-order derivatives.

5.3.6. Selection of an appropriate optimizer

As discussed in Section 3.4.2.1, there are various optimization techniques, such as *Adam*, *SGD*, and *L-BFGS*. Each optimizer offers distinct advantages and disadvantages. Through experimentation, we observe that sequentially applying two different algorithms significantly improves the results: first training the model using the *Adam* optimizer, followed by switching to the *L-BFGS* algorithm, a strategy also highlighted in the work of [105]. While *Adam* tends to be slower compared to *L-BFGS*, using *L-BFGS* from the start often leads to the convergence of the solution to a local minimum. In contrast, starting with *Adam* helps the model avoid local minima, and then switching to *L-BFGS* enables faster convergence to the global minimum.

6. Application of Physics-Informed Neural Networks to Computational Mechanics

6.1. Use Case 1: Beam mechanics

6.1.1. PINN formulation for beam mechanics

6.1.1.1. Dynamic beam equation

For one-dimensional Euler-Bernoulli beam problems, PINNs are deployed using output transformation to predict the vertical displacement field u. The network takes x-coordinates and t as the neural network input (i.e. $\boldsymbol{z} = [x, t]$). A fully-connected neural network (FNN) approximates the vertical displacement $\tilde{u}$ as

$$\tilde{u} := \left(\mathcal{N}_u(x; t; \boldsymbol{\theta})\right)' \tag{6.1}$$

Combining the information provided in Fig. 5.2 for losses of general PINNs with the governing equations of beam mechanics (Section 2.1.2), we obtain the total loss $\mathcal{L}_{\mathrm{B}}$ for one-dimensional beam mechanics, including experimental data loss as

$$\mathcal{L}_{\mathrm{B}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{NBCs}} + \mathcal{L}_{\mathrm{EXPs}} + \mathcal{L}_{\mathrm{ICs}}, \tag{6.2}$$

where

$$\begin{aligned}
\mathcal{L}_{\mathrm{PDE}} &= w_n \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\rho A \frac{\partial^2 \tilde{u}(x^{i_{rp}}, t)}{\partial t^2} + \frac{\partial^2}{\partial x^2} \left(EI \frac{\partial^2 \tilde{u}(x^{i_{rp}}, t)}{\partial x^2} \right) - q(x^{i_{rp}}, t) \right]^2, \\
\mathcal{L}_{\mathrm{DBCs}} &= w_{bc,D} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \left[\left[\tilde{u}(x^{i_{bp,D}}, t) - \hat{u}(x^{i_{bp,D}}, t) \right] \right]^2, \\
\mathcal{L}_{\mathrm{NBCs}} &= \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \left[\left[\frac{\partial^k \tilde{u}(x^{i_{bp,N}}, t)}{\partial x^k} - \hat{t}(x^{i_{bp,N}}, t) \right]_{i_{bc,N}} \right]^2, \\
\mathcal{L}_{\mathrm{EXPs}} &= w_e \frac{1}{N_{ep}} \sum_{i_{ep}=1}^{N_{ep}} \left[\left[\tilde{u}(x^{i_{ep}}, t) - u^*(x^{i_{ep}}, t) \right] \right]^2.
\end{aligned} \tag{6.3}$$

Here, $\hat{u}$ denotes the prescribed displacements, and $\hat{t}$ represents the prescribed contributions for the NBCs. Moreover, u^* denotes the additional measurement data. For details on problem-related

parameters, we refer the reader to Section 2.1. The loss formulation due to initial conditions $\mathcal{L}_{\text{ICs}}$ has two main contributions that are derived from the displacement and the velocity fields. The term $\mathcal{L}_{\text{ICs}}$ can be expressed as

$$\begin{aligned}\mathcal{L}_{\text{ICs}} = w_{ic,D}\frac{1}{N_{ip}}\sum_{i_{bp,D}=1}^{N_{ip}}\Big[\big[\tilde{u}(x^{i_{ip}},0)-\hat{u}_0(x^{i_{ip}},0)\big]\Big]^2+\\ w_{ic,V}\frac{1}{N_{ip}}\sum_{i_{bp,D}=1}^{N_{ip}}\Big[\big[\frac{\partial\tilde{u}(x^{i_{ip}},0)}{\partial t}-\hat{\dot{u}}_0(x^{i_{ip}},0)\big]\Big]^2.\end{aligned} \tag{6.4}$$

Here, $\hat{u}_0$ is the prescribed initial displacement and $\hat{\dot{u}}_0$ denotes the prescribed initial velocity, which is the time derivative of the displacement. In this formulation, the terms $\mathcal{L}_{\text{PDE}}$, $\mathcal{L}_{\text{DBCs}}$, $\mathcal{L}_{\text{NBCs}}$, $\mathcal{L}_{\text{ICs}}$, and $\mathcal{L}_{\text{EXPs}}$ represent the losses corresponding to the governing PDE, Dirichlet boundary conditions, Neumann boundary conditions, initial conditions, and experimental data, respectively. Each of these losses is computed based on specific input points:

- $\{x^{i_{rp}},t\}_{i_{rp}=1}^{N_{rp}}$ refers to the collocation points within the domain,
- $\{x^{i_{bp,D}},t\}_{i_{bp,D}=1}^{N_{bp,D}}$ denotes the boundary points for Dirichlet BCs,
- $\{x^{i_{bp,N}},t\}_{i_{bp,N}=1}^{N_{bp,N}}$ represents the boundary points for Neumann BCs,
- $\{x^{i_{ep}},t\}_{i_{ep}=1}^{N_{ep}}$ corresponds to additional points based on experimental data,
- $\{x^{i_{ip}},0\}_{i_{ip}=1}^{N_{ip}}$ denotes the points related to initial conditions.

The weights for the individual loss components are denoted as w_n, $w_{\text{bc,D}}$, $\{w_{i_{\text{bc,N}}}\}_{i_{\text{bc,N}}=1}^{N_{\text{bc,N}}}$, w_e, and $\{w_{\text{ic,D}}, w_{\text{ic,V}}\}$ for $\mathcal{L}_{\text{PDE}}$, $\mathcal{L}_{\text{DBCs}}$, $\mathcal{L}_{\text{NBCs}}$, $\mathcal{L}_{\text{EXPs}}$, and $\mathcal{L}_{\text{ICs}}$, respectively. In the case of Neumann BCs, multiple loss weights are required due to the higher-order derivatives present in beam mechanics, as these derivatives may vary in magnitude or scale. Therefore, $\mathcal{L}_{\text{NBCs}}$ is formulated to accommodate multiple loss terms. Moreover, an illustration of the application of PINNs for dynamic one-dimensional Euler-Bernoulli beams is depicted in Fig. 6.1.

6.1.1.2. Static beam equation

In static models, we exclude time from the input space, so the input for the network is $\boldsymbol{z} = x$. Thus, an FNN approximates the vertical displacement $\tilde{u}$ as

$$\tilde{u} := (\mathcal{N}_u(x;\boldsymbol{\theta}))'. \tag{6.5}$$

An illustration of the PINNs for static one-dimensional Euler-Bernoulli beams is depicted in Fig. 6.2. We can derive the loss formulation for a static 1D beam problem using the general loss formulation of beam mechanics (Eq. 6.3) as follows

$$\mathcal{L}_{\text{B}} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{DBCs}} + \mathcal{L}_{\text{NBCs}} + \mathcal{L}_{\text{EXPs}}, \tag{6.6}$$

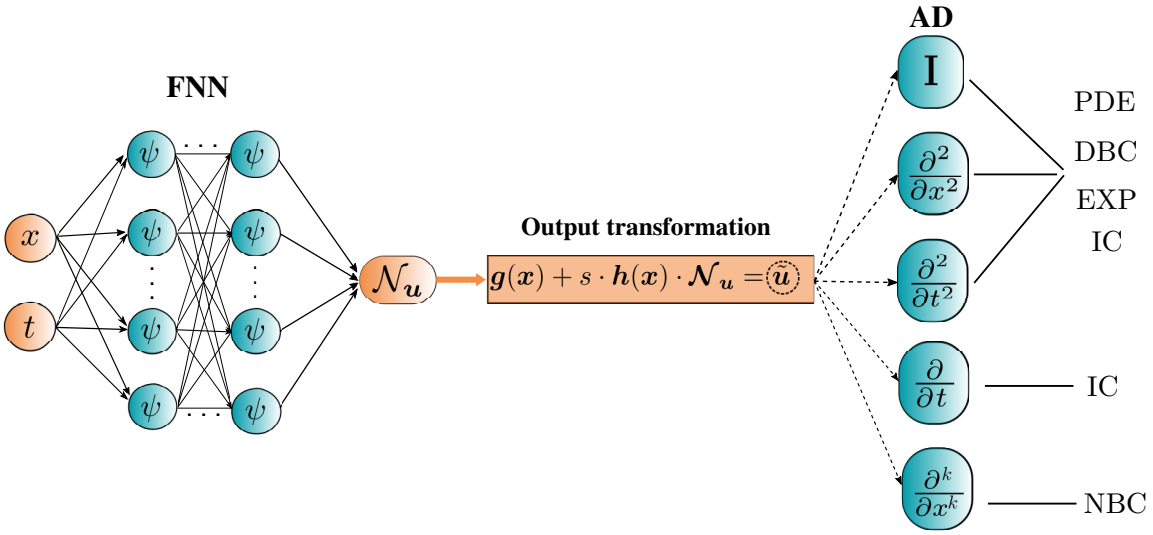


Figure 6.1.: An illustration of the application of PINNs for dynamic one-dimensional Euler-Bernoulli beams.

where

$$
\begin{aligned}
\mathcal{L}_{\text{PDE}} &= w_n \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\frac{\partial^2}{\partial x^2} \left(EI \frac{\partial^2 \tilde{u}(x^{i_{rp}})}{\partial x^2} \right) - q(x^{i_{rp}}) \right]^2, \\
\mathcal{L}_{\text{DBCs}} &= w_{bc,D} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \left[\left[\tilde{u}(x^{i_{bp,D}}) - \hat{u}(x^{i_{bp,D}}) \right] \right]^2, \\
\mathcal{L}_{\text{NBCs}} &= \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \left[\left[\frac{\partial^k \tilde{u}(x^{i_{bp,N}})}{\partial x^k} - \hat{t}(x^{i_{bp,N}}) \right]_{i_{bc,N}} \right]^2 \\
\mathcal{L}_{\text{EXPs}} &= w_e \frac{1}{N_{ep}} \sum_{i_{ep}=1}^{N_{ep}} \left[\left[\tilde{u}(x^{i_{ep}}) - u^*(x^{i_{ep}}) \right] \right]^2.
\end{aligned}
\tag{6.7}
$$

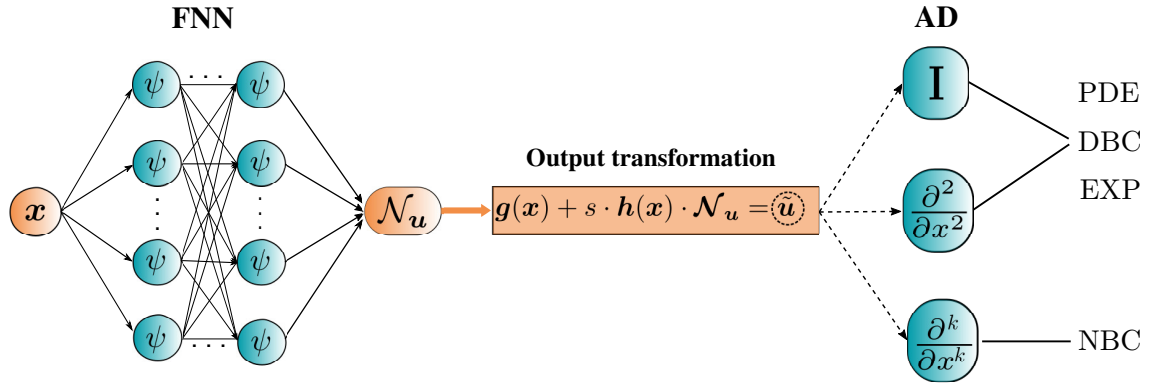


Figure 6.2.: An illustration of the application of PINNs for static one-dimensional Euler-Bernoulli beams.

6.1.2. Numerical examples

In the following sections, we will explore a range of beam problems under both static and dynamic models. The aim is here to increase the problem complexity through the examples to show the capability of PINNs.

We begin with static models, examining classic beam configurations such as clamped, cantilever, and simply supported beams. We use constant material properties and relatively simple distributed loads for these cases. In the following example, we increase the complexity of the problem by introducing a more sophisticated load distribution and a non-uniform cross-section for the beam, challenging the model with more realistic scenarios. Then, we employ PINNs as an inverse solver to identify the distributed load of the beam using the provided so-called experimental data. This step-by-step approach highlights the power of PINNs in tackling both forward and inverse problems. Finally, we address time-dependent beam problems, where dynamic effects are considered.

As a common error metric, we report the vector-based relative L_2 errors for the displacement $E^u_{L_2}$ as follows

$$E^u_{L_2} = \frac{\sqrt{\sum_{j=1}^{N_{\text{test}}} \left(\tilde{u}(x^j) - u(x^j)\right)^2}}{\sqrt{\sum_{j=1}^{N_{\text{test}}} \left(u(x^j)\right)^2}}. \tag{6.8}$$

Similarly, for the inverse problems, we report the vector-based relative L_2 errors for the distributed load $E^q_{L_2}$ as follows

$$E^q_{L_2} = \frac{\sqrt{\sum_{j=1}^{N_{\text{test}}} \left(\tilde{q}(x^j) - q(x^j)\right)^2}}{\sqrt{\sum_{j=1}^{N_{\text{test}}} \left(q(x^j)\right)^2}}. \tag{6.9}$$

The number of test points for both error calculations is set to $N_{\text{test}} = 20$.

The following PINN architecture is employed through the examples:

- The network architecture consists of one input layer, one output layer, and three hidden layers, each containing 30 neurons.
- The hyperbolic tangent activation function (tanh) is used to introduce non-linearity in the model.
- Weights are initialized using the *Glorot uniform* initializer.
- The optimization process utilizes the *Adam* optimizer for $1{,}000$ iterations, with a learning rate set to 0.001, followed by the *L-BFGS-B* optimizer.
- The number of collocation points is set to $N_f = 25$, and these points are sampled using Sobol sampling to ensure a non-uniform, well-distributed set of points across the domain.
- Except for the inverse problems, no experimental or artificially generated data is utilized. Specifically for inverse problems, 20 additional artificially generated data points are incorporated, i.e., $N_e = 20$.
- The loss weights are constant and set to 1 for all investigated examples.

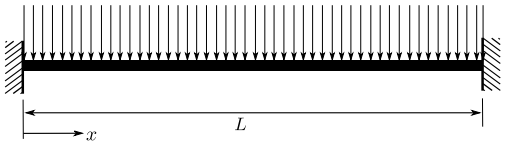
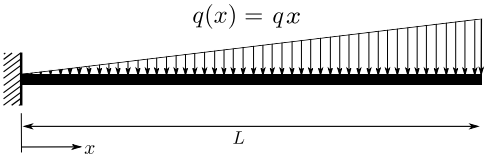
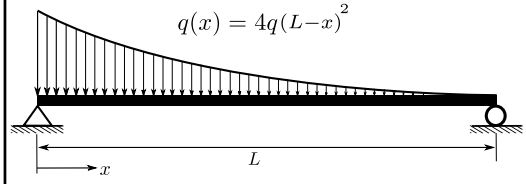
Beam model	**Description**
Case 1 $q(x) = q$  $u(x=0)=0$ $\quad u(x=L)=0$ $u_x(x=0)=0$ $\quad u_x(x=L)=0$	Clamped beam under a uniformly distributed load q(x) with the magnitude q. The analytical solution: $$u(x) = -\frac{qx^2}{24EI}(x-L)^2$$
Case 2 $q(x) = qx$  $u(x=0)=0$ $\quad u_{xx}(x=L)=0$ $u_x(x=0)=0$ $\quad u_{xxx}(x=L)=0$	Cantilever beam under a triangular distributed load q(x) with the magnitude q at x=L. The analytical solution: $$u(x) = -\frac{qx^2}{120EI}\left(20L^3 - 10L^2x + x^3\right)$$
Case 3 $q(x) = 4q(L-x)^2$  $u(x=0)=0$ $\quad u(x=L)=0$ $u_{xx}(x=0)=0$ $\quad u_{xx}(x=L)=0$	Simply supported beam under a quadratically distributed load q(x) with the magnitude q at x=L/2. The analytical solution: $$u(x) = -\frac{qx}{90EI}\left(x^5 - 6x^4L + 15x^3L^2 - 15x^2L^3 + 5L^5\right)$$

Figure 6.3.: Three investigated beam models with their descriptions and analytical solutions.

6.1.2.1. Static models

Three investigated models, along with their corresponding descriptions, are presented in Fig. 6.3. For these examples, the product of Young's modulus and the moment of inertia is set to $AE = 1$, while the beam length and the magnitude of the distributed load are set to $L = 1$ and $q = 1$, respectively.

In the first case, a clamped beam is subjected to a uniformly distributed load. In the second case, the complexity of the applied load is slightly increased, resulting in a cantilever beam under a triangularly distributed load. Finally, in the third case, a quadratic load is applied to a simply supported beam.

The following output transformation is applied to enforce displacement BCs as hard constraints

$$\tilde{u} = x(L-x)\mathcal{N}_u \quad \text{for Case 1,} \tag{6.10}$$

$$\tilde{u} = x\mathcal{N}_u \quad \text{for Case 2,} \tag{6.11}$$

$$\tilde{u} = x(L-x)\mathcal{N}_u \quad \text{for Case 3.} \tag{6.12}$$

As seen in Fig. 6.3, the constructed output transformations fulfill the prescribed displacements. For instance, in Case 1, the augmented displacement $\tilde{u}$ is equal to zero at $x = 0$ and $x = L$, which allows hard enforcement of DBCs. Since only the displacement is considered as the network output, we can enforce only the displacement boundary conditions as hard constraints.

The results of the analyzed cases are presented in Fig. 6.4. The figures on the left demonstrate that the predicted displacements closely match the analytical solutions, as confirmed by the computed relative L_2 errors of $6.66 \times 10^{-4}\%$, $5.12 \times 10^{-2}\%$, and $8.05 \times 10^{-3}\%$ for cases 1, 2, and 3, respectively.

During training with the *Adam* optimizer, the convergence rate slows as the number of epochs increases, as illustrated in the figures on the right (Fig. 6.4). Switching to the *L-BFGS-B* optimizer after *Adam* improves the convergence rate and significantly reduces the PDE and boundary condition losses. For the PDE loss, the mean squared error reaches approximately 2.03×10^{-9}, 3.97×10^{-6}, and 7.86×10^{-6}, while the average MSE for the boundary condition losses is around 6.99×10^{-14}, 5.69×10^{-11}, and 4.9×10^{-12} for Cases 1, 2, and 3, respectively.

The sequential use of *Adam* followed by *L-BFGS-B* plays a crucial role in achieving high accuracy. *Adam* helps prevent premature convergence to a local minimum, a benefit also emphasized in [105].

A complex consideration

So far, simple beam structures with uniform material and geometrical properties under non-complex loadings have been investigated. Now, we increase the complexity. Consider a simply supported beam having the following non-uniform inertia function

$$I(x) = \left(\frac{x}{L}\right)^2, \tag{6.13}$$

and under the following distributed loading

$$q(x) = \frac{8\pi^2 \left((2\pi^2 x^2 - L^2)\sin(\frac{2\pi x}{L}) - 4\pi x \cos(\frac{2\pi x}{L})\right)}{L^4}. \tag{6.14}$$

An illustration of the loading and the representative inertia distribution is depicted in Fig. 6.5. The analytical solution can be easily derived as

$$u(x) = \sin(\frac{2\pi x}{L}). \tag{6.15}$$

For simplicity, Young's modulus is set to $E = 1$, and the length of the beam is set to $L = 1$.

To enforce the DBCs, we employ the following output transformation

$$\tilde{u} = x(L - x)\mathcal{N}_u, \tag{6.16}$$

which fulfills the DBCs at $x = 0$ and $x = L$ for a simply supported beam (Case 3), as depicted in Fig. 6.3.

The results for the complex model are presented in Fig. 6.6. As with the previously analyzed simpler models, the figure on the left indicates that the predicted displacements closely match the actual solution. However, a notable difference between the PDE loss and BC loss is observed,

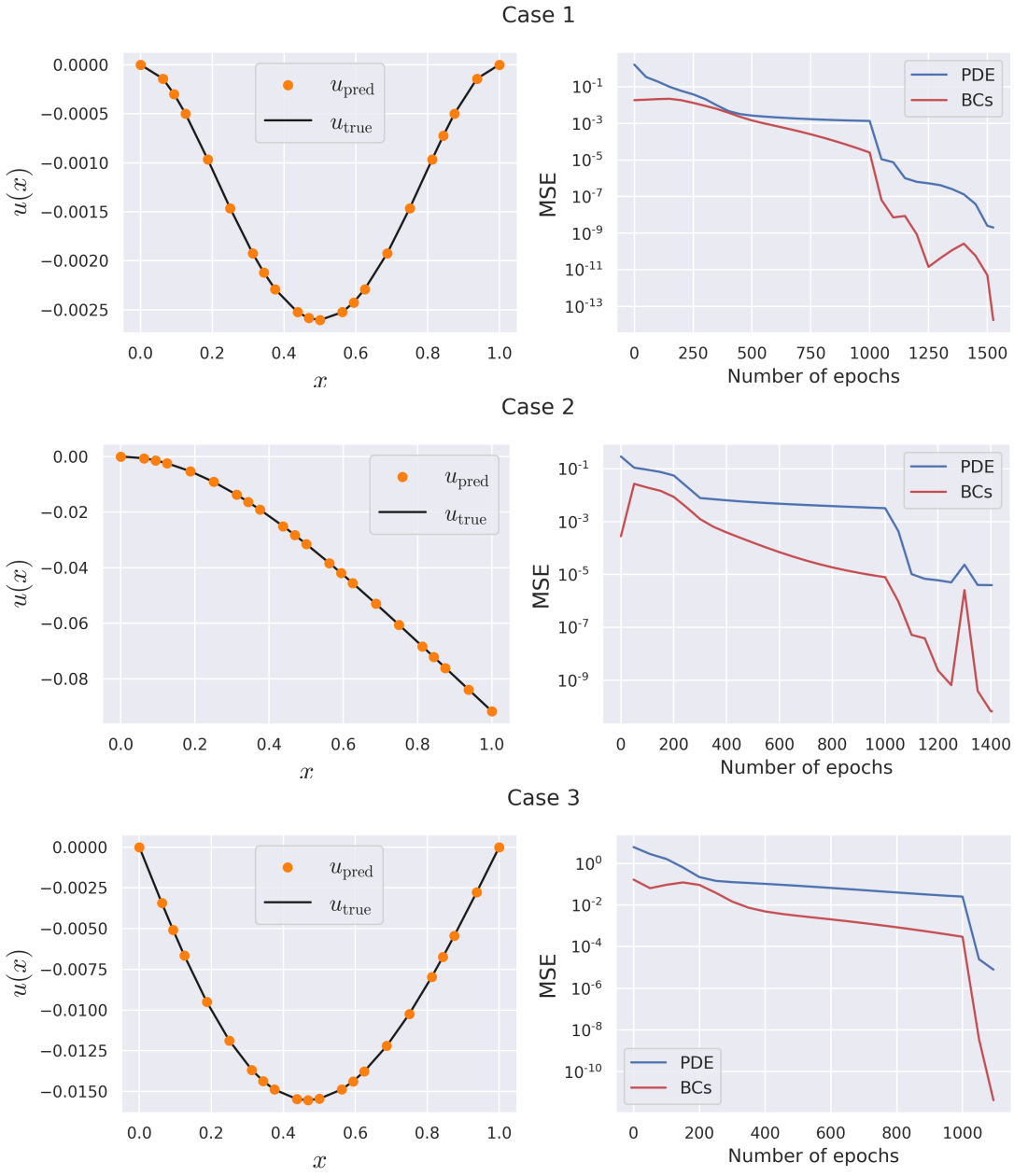


Figure 6.4.: On the figures left, the predicted and true deflections through the coordinate x are compared for three different cases. On the figures right, the loss history for PDE and BCs is given through epochs.

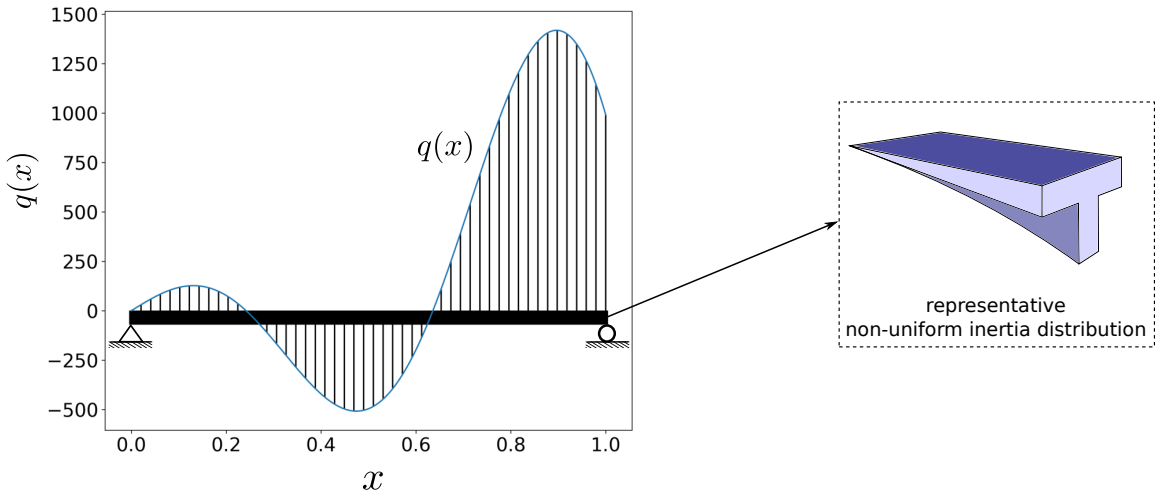


Figure 6.5.: An illustration of a simple supported beam having a non-uniformly distributed cross-section under an arbitrary load

which is larger than the simpler models (see 6.4). This difference arises due to the high magnitude of the applied load.

Because the applied load is significantly larger relative to the predicted displacements, the initial PDE loss is substantially higher than the BC loss. During network training, the PDE loss decreases while the BC loss increases, reducing the gap between them. The relative L_2 error is calculated as 0.3%, with an MSE of 5.1×10^{-3} for the PDE loss and 1.0×10^{-4} for the BC loss. Although these MSE values are considerably larger compared to the simpler models (6.4), the key factor is achieving convergence. Notably, the initial PDE loss was on the order of 1e7, demonstrating significant convergence throughout the training process.

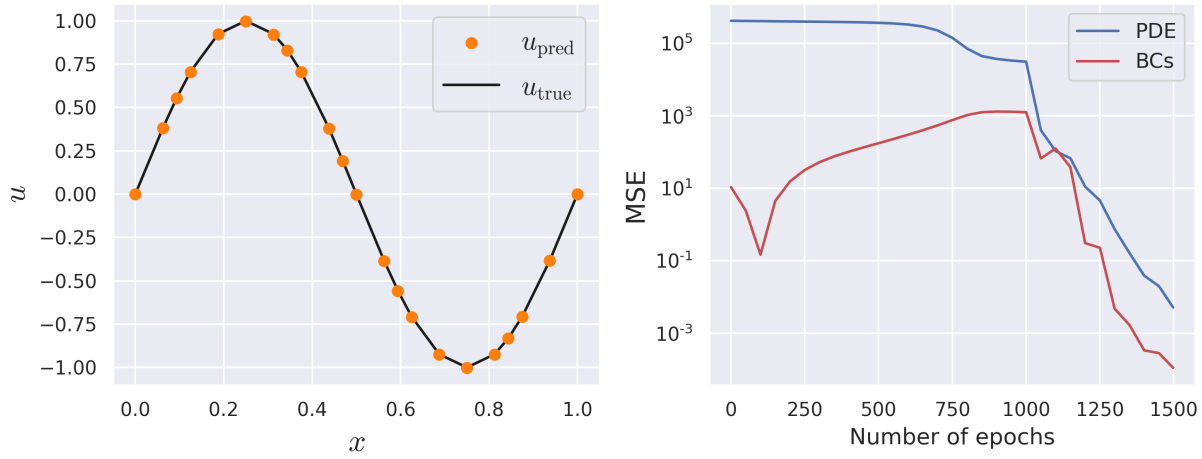


Figure 6.6.: Results for the complex beam example. On the left is the comparison of displacements obtained from the analytical solution and the predicted values, on the right, the evolution of the MSE for the PDE loss and BCs loss.

6.1.2.2. Inverse problem: Load discovery

In the context of the inverse problem, PINNs can be employed to estimate the distributed load on a cantilever beam, as described in Case 2 of Section 6.1.2.1.

As outlined in Section 5.1.1, three principal approaches exist for constructing PINN models for inverse problems. Given that the unknown parameter in this case is a distributed quantity, we adopt the data-driven discovery method (Approach 2), as detailed in Section 5.1.1.2.

Compared to the PINN network illustrated in Fig. 6.2, the network for this inverse problem is configured such that a fully connected neural network approximates both the vertical displacement, denoted as $\tilde{u}$, and the distributed load, $\tilde{q}$, as follows:

$$\tilde{u} := (\mathcal{N}_u(x;\boldsymbol{\theta}))', \quad \tilde{q} := (\mathcal{N}_q(x;\boldsymbol{\theta}))'. \tag{6.17}$$

As discussed in Remark 5.1.1.3, a generalized loss formulation for inverse problems is not feasible, as the loss function depends on the characteristics of the problem and the specific parameter to be identified. Since the distributed load appears only in the governing PDE, we define the loss function solely using the $\mathcal{L}_{\text{PDE}}$ term, adapted from the static model loss function (Eq. 6.7) as follows:

$$\mathcal{L}_{\text{PDE}} = w_n \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \Big[\frac{\partial^2}{\partial x^2} \left(EI \frac{\partial^2 \tilde{u}(x^{i_{rp}})}{\partial x^2} \right) - \tilde{q}(x^{i_{rp}}) \Big]^2. \tag{6.18}$$

The results of the inverse problem are presented in Fig. 6.7. As shown in Fig. 6.7(a), the predicted and actual displacements are well-aligned, with a relative L_2 error of $E^u_{L_2} = 0.72\%$. Additionally, the MSE values for the PDE, boundary condition (BC), and experimental data (EXP) losses are computed as 2.5×10^{-8}, 8.76×10^{-10}, and 8.75×10^{-12}, respectively.

However, as illustrated in Fig. 6.7(b), the predicted distributed load does not match the true load well, with an L_2 relative error of $E^q_{L_2} = 47\%$. While the network accurately predicts the target solution u, it struggles to correctly identify the unknown load distribution q.

As discussed in Section 5.3 on algorithmic challenges, this issue is an example of the *non-uniqueness in inverse problems* phenomenon. Although the identified load satisfies all governing equations, it does not necessarily correspond to the actual load we seek to recover.

To mitigate this issue, we incorporate prior knowledge of the load into the experimental loss term $\mathcal{L}_{\text{EXPs}}$. Assuming we have the following prior information about the unknown load:

$$q(x = 0) = 0, \tag{6.19}$$

$$q(x = 1) = q. \tag{6.20}$$

These equations imply that the values of the unknown load at the boundaries are known and provided. Thus, we can incorporate this prior knowledge into the loss function by adding these measurements to the experimental loss term, resulting in an updated $\mathcal{L}_{\text{EXPs}}$:

$$\mathcal{L}_{\text{EXPs}} = \frac{1}{N_{ep}} \sum_{i_{ep}=1}^{N_{ep}} \Big[\big[\tilde{u}(x^{i_{ep}}) - u^*(x^{i_{ep}}) \big] \Big]^2 + \frac{1}{2} \sum_{i_{ep}=1}^{2} \Big[\big[\tilde{q}(x^{i_{qp}}) - q^*(x^{i_{qp}}) \big] \Big]^2, \tag{6.21}$$

where $x^{i_{qp}} = \{0, 1\}$ and $q^* = \{0, q = -1\}$ as defined in Eqs. 6.19 and 6.20.

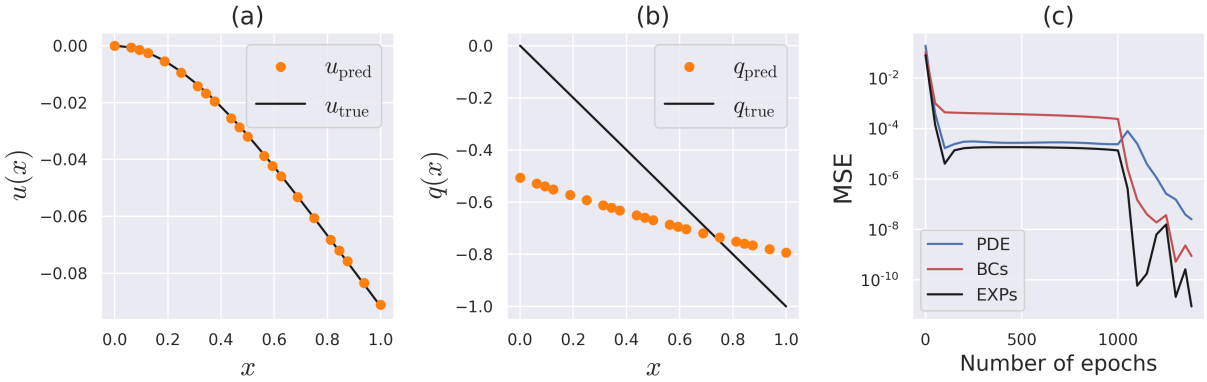


Figure 6.7.: Results for the inverse problem. (a) Comparison of displacements obtained from the analytical solution and the predicted values, (b) comparison of the analytical and predicted loads, (c) evolution of the MSE for the PDE, BCs and the experimental data (EXPs).

The results of the inverse problem with prior knowledge of the load distribution are shown in Fig. 6.8. As seen in Fig. 6.8(a), the predicted and actual displacements align closely, yielding a relative L_2 error of $E^u_{L_2} = 0.14\%$. Compared to the previous results without prior knowledge, the identified distributed load now accurately matches the true value, with an L_2 relative error of $E^q_{L_2} = 1.3\%$. Furthermore, the loss convergence plots show a smoother convergence pattern (Fig. 6.8c). By incorporating prior knowledge about the distributed load, the network effectively solves the inverse problem, accurately identifying both the target solution u and the unknown load parameter q.

A key insight from addressing inverse problems is the inherent risk of non-uniqueness in the identified parameters, which may arise from the underlying characteristics of the problem. To mitigate this, incorporating additional constraints or prior knowledge can significantly improve the optimization process, helping to accurately determine the unknown parameter. Indeed, an alternative approach to address non-uniqueness is *network inversion* (see Section 5.1.1.1), where the network is specifically designed to approximate only the unknown parameter field, potentially enhancing stability.

6.1.2.3. Dynamic models

We now extend our analysis to time-dependent beam problems, where dynamic effects are incorporated. Unlike static models, the neural network takes both the spatial coordinate x and the temporal variable t as inputs.

Consider a simply supported beam subjected to the following time-dependent distributed load:

$$q(x,t) = -\sin(\pi x)e^{-t}\left(\pi^4(t+1) + t - 1\right). \tag{6.22}$$

The boundary conditions are defined as:

$$\begin{aligned} u(x=0,t) &= 0 \quad \text{and} \quad u_x(x=0,t) = 0, \\ u(x=L,t) &= 0 \quad \text{and} \quad u_x(x=L,t) = 0. \end{aligned} \tag{6.23}$$

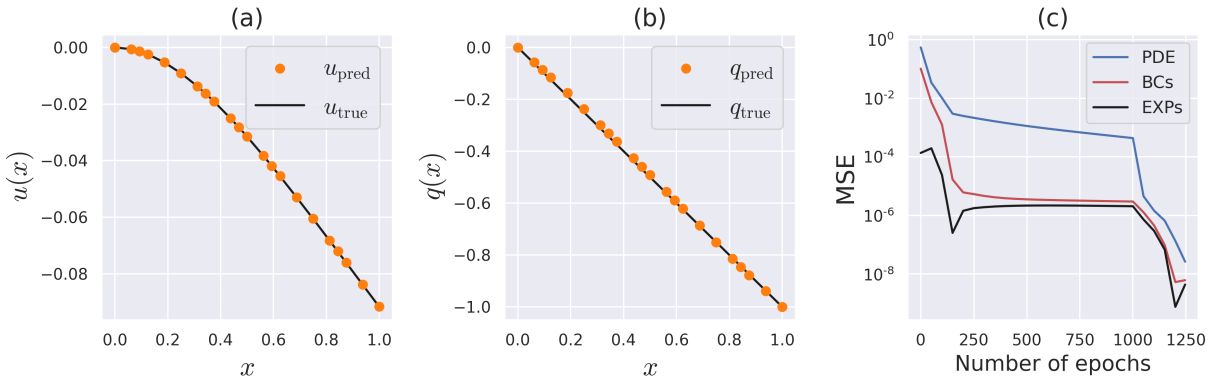


Figure 6.8.: Results for the inverse problem with a prior knowledge of the distributed load. (a) Comparison of displacements obtained from the analytical solution and the predicted values, (b) comparison of the analytical and predicted loads, (c) evolution of the MSE for the PDE, BCs and the experimental data (EXPs)

The initial conditions are given by:

$$u(x,0) = -\sin(\pi x) \quad \text{and} \quad u_t(x,0) = 0. \tag{6.24}$$

This results in the following analytical solution:

$$u(x,t) = -\sin(\pi x)(t+1)e^{-t}. \tag{6.25}$$

For this analysis, the moment of inertia is set to $AE = 1$, while the density and cross-sectional area terms are set to $\rho A = 1$.

The results of the time-dependent beam problem are presented in Fig. 6.9. A comprehensive comparison between the predicted and true solutions at various spatial and temporal locations demonstrates that the PINN model provides accurate predictions. This agreement is further supported by the computed relative L_2 error, which is $E^u_{L_2} = 0.12\%$.

The top two contour plots illustrate the true and predicted solutions across the entire spatial and temporal domain. The close visual similarity between these plots confirms that the PINN model effectively captures the underlying dynamics of the beam's response.

The lower section of Fig. 6.9 provides a more detailed validation of the model's accuracy by comparing predicted and true solutions at specific spatial and temporal cross-sections

- The three plots in the second row show displacement profiles at fixed time instances ($t = 0.00$, $t = 0.50$, and $t = 1.00$).
- The three plots in the third row display displacement evolution over time at fixed spatial locations ($x = 0.25$, $x = 0.50$, and $x = 0.75$).

In all cases, the predicted values closely align with the true values, highlighting the model's good accuracy in predicting beam displacements under dynamic loading conditions.

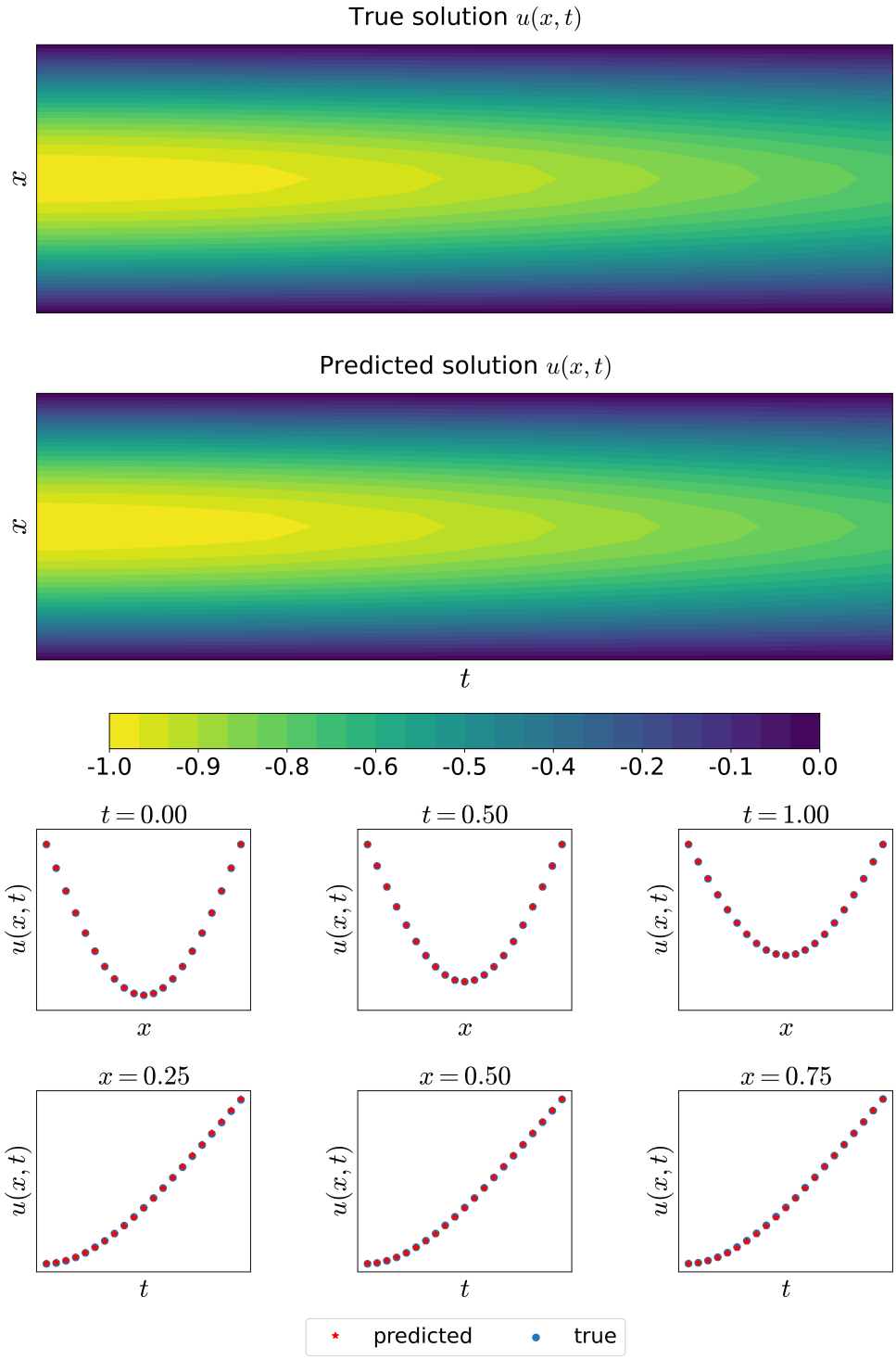


Figure 6.9.: Results for the time-dependent beam example. The figure above shows the predicted solution, while the figures below compare the predicted solution to the analytical solution when one input dimension is kept constant.

6.2. Use case 2: Linear elasticity

In this section, we develop comprehensive methodologies for constructing PINN frameworks to solve the governing equations in solid mechanics. We begin with PINN formulations based on a displacement-based approach (detailed in Section 2.2.2.2) and extend to formulations incorporating the mixed-variable approach (outlined in Section 2.2.2.3). These frameworks are specifically designed to address both elastodynamics (time-dependent) and elastostatics (quasi-static) problems. Furthermore, we adapt the developed methods to the 3D Cartesian coordinate system for three-dimensional solid models while also providing reduced formulations for two-dimensional cases, such as plane strain and plane stress conditions. Additionally, we discuss the critical importance of output scaling in the mixed-variable approach to ensure a stable optimization.

In the numerical examples, we present a diverse range of examples, including the Lamé elasticity problem (2D elastostatics), a hollow sphere subjected to internal pressure (3D elastostatics), a 2D block under time-dependent loading (elastodynamics), a hollow sphere under time-dependent internal pressure (3D elastodynamics), and an inverse problem for identifying material parameters.

6.2.1. PINNs with displacement-based formulation

6.2.1.1. Elastodynamics

In the displacement-based approach for elastodynamics problems without additional network input parameters, the network takes spatial coordinates, denoted as $\boldsymbol{x}$, and time, denoted as t, as the neural network input such that $\boldsymbol{z} = [\boldsymbol{x}, t]$. An FNN maps the given network input $\boldsymbol{z}$ to the approximated displacement $\tilde{\boldsymbol{u}}$ (see Fig. 6.10) such that

$$\tilde{\boldsymbol{u}} := (\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x}; t; \boldsymbol{\theta}))'. \tag{6.26}$$

An illustration of the displacement-based (single-variable) PINN approach for solving elastodynamics problems is presented in Fig. 6.10. In this formulation, an FNN is employed to approximate the displacement field by taking spatial coordinates, $\boldsymbol{x}$, and time, t, as input variables. The network learns a continuous representation of the displacement field, denoted by $\tilde{\boldsymbol{u}}$, which serves as the primary output of the model. To enforce the governing equations of elastodynamics, the method leverages automatic differentiation, allowing efficient computation of both first-order and second-order derivatives of $\tilde{\boldsymbol{u}}$ with respect to $\boldsymbol{x}$ and t. This capability is crucial for deriving all necessary intermediate physical quantities, including the predicted displacement gradient, $\boldsymbol{\nabla}\tilde{\boldsymbol{u}}$, and the predicted acceleration field, $\ddot{\tilde{\boldsymbol{u}}}$. These derivatives are then used to enforce equilibrium conditions, boundary constraints, and initial conditions, ensuring that the identified solution remains consistent with the underlying physics. For instance, the governing Navier equations do not explicitly include stress terms. Therefore, these intermediate variables can be directly incorporated into the PDE loss term. However, for NBCs, stress terms are required, which are derived using the kinematic equation and Hooke's law.

As depicted in Fig. 6.10, a key limitation of the displacement-based PINN approach lies in its dependence on second-order derivatives of the predicted displacement field with respect to both spatial coordinates and time. This requirement stems from the governing equations of elastodynamics, which inherently involve acceleration terms and spatial gradients of stress

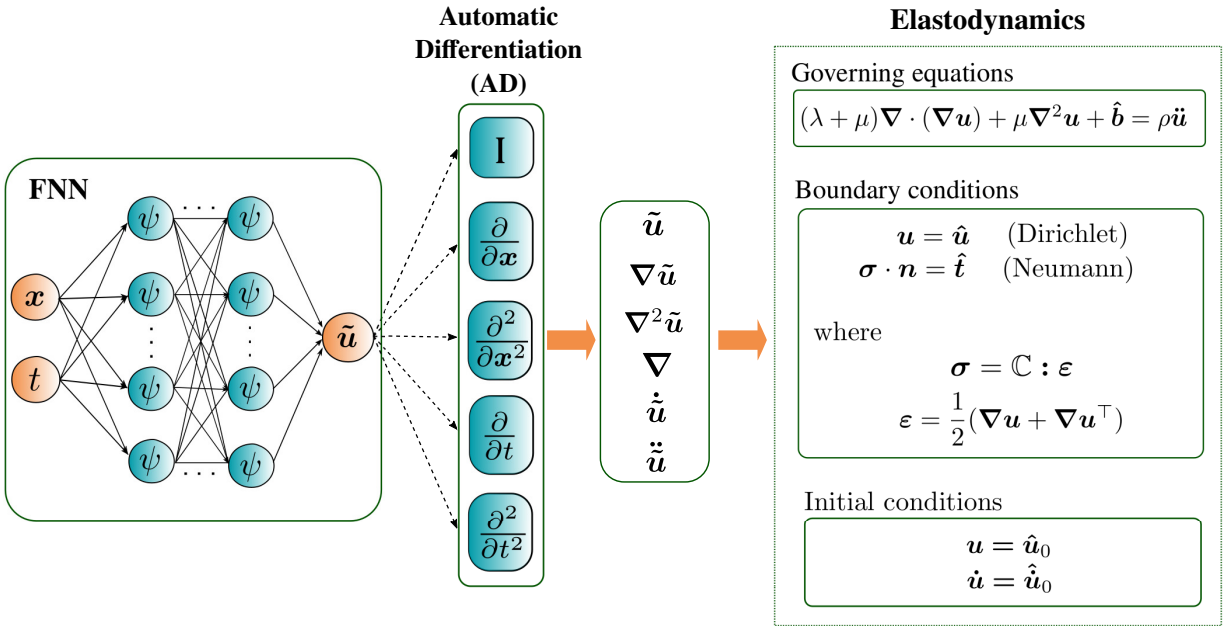


Figure 6.10.: Physics-informed neural networks using the displacement-based formulation to solve elastodynamics problems.

and strain. The necessity for computing these higher-order derivatives introduce a significant computational burden, as second-order derivatives are considerably more expensive to evaluate compared to first-order derivatives. This computational overhead becomes particularly significant in large-scale or high-dimensional problems, such as three-dimensional models subjected to time-dependent loads, where the increased complexity amplifies training inefficiencies.

Beyond computational cost, another limitation of the displacement-based formulation is its restrictive handling of boundary conditions. Since only the displacement field is explicitly represented as the neural network output, the method inherently favors directly enforcing Dirichlet boundary conditions through output transformations, ensuring strict adherence to prescribed displacements. While initial conditions related to the displacement field can similarly be imposed as hard constraints via output transformations, enforcing Neumann or mixed boundary conditions, such as traction or stress constraints, becomes more challenging. These conditions must typically be incorporated in a soft-constraint manner through penalty terms in the loss function, which may lead to approximation errors or convergence difficulties.

Combining the information provided in Fig. 5.2 for losses of general PINNs with the governing equations of time-dependent solid mechanics in the case of the displacement-based approach as explained in Section 2.2.2.2, we obtain the total loss $\mathcal{L}_{\mathrm{E}}$ for the displacement-based formulation including additional experimental data as

$$\mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{NBCs}} + \mathcal{L}_{\mathrm{ICs}} + \mathcal{L}_{\mathrm{EXPs}}, \tag{6.27}$$

where

$$\mathcal{L}_{\text{PDE}} = \sum_{i_n=1}^{N_n} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \Big[\big[\boldsymbol{\nabla} \cdot \big(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t) \big) + \mu \boldsymbol{\nabla}^2 \tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{rp}}, t \right) + \hat{\boldsymbol{b}} \left(\boldsymbol{x}^{i_{rp}}, t \right) - \rho \ddot{\tilde{\boldsymbol{u}}} \left(\boldsymbol{x}^{i_{rp}}, t \right) \big]_{i_n} \Big]^2,$$

$$\mathcal{L}_{\text{DBCs}} = \sum_{i_{bc,D}=1}^{N_{bc,D}} w_{i_{bc,D}} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \Big[\big[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}}, t \right) - \hat{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}}, t \right) \big]_{i_{bc,D}} \Big]^2,$$

$$\mathcal{L}_{\text{NBCs}} = \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \Big[\big[\boldsymbol{\sigma}^{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,N}}, t \right) \cdot \boldsymbol{n} - \hat{\mathbf{t}} \left(\boldsymbol{x}^{i_{bp,N}}, t \right) \big]_{i_{bc,N}} \Big]^2 \tag{6.28}$$

$$\mathcal{L}_{\text{ICs}} = \sum_{i_{ic,\boldsymbol{u}}=1}^{N_{ic,\boldsymbol{u}}} w_{i_{ic,\boldsymbol{u}}} \frac{1}{N_{ip,\boldsymbol{u}}} \sum_{i_{ip,\boldsymbol{u}}=1}^{N_{ip,\boldsymbol{u}}} \Big[\big[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ip,\boldsymbol{u}}}, 0 \right) - \hat{\boldsymbol{u}}_0 \left(\boldsymbol{x}^{i_{ip,\boldsymbol{u}}}, 0 \right) \big]_{i_{e,\boldsymbol{u}}} \Big]^2 + \sum_{i_{ic,\dot{\boldsymbol{u}}}=1}^{N_{ic,\dot{\boldsymbol{u}}}} w_{i_{ic,\dot{\boldsymbol{u}}}} \frac{1}{N_{ip,\dot{\boldsymbol{u}}}} \sum_{i_{ip,\dot{\boldsymbol{u}}}=1}^{N_{ip,\dot{\boldsymbol{u}}}} \Big[\big[\dot{\tilde{\boldsymbol{u}}} \left(\boldsymbol{x}^{i_{ip,\dot{\boldsymbol{u}}}}, 0 \right) - \hat{\dot{\boldsymbol{u}}}_0 \left(\boldsymbol{x}^{i_{ip,\dot{\boldsymbol{u}}}}, 0 \right) \big]_{i_{e,\dot{\boldsymbol{u}}}} \Big]^2,$$

$$\mathcal{L}_{\text{EXPs}} = \sum_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}} w_{i_{e,\boldsymbol{u}}} \frac{1}{N_{ep,\boldsymbol{u}}} \sum_{i_{ep,\boldsymbol{u}}=1}^{N_{ep,\boldsymbol{u}}} \Big[\big[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}}, t \right) - \boldsymbol{u}^* \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}}, t \right) \big]_{i_{e,\boldsymbol{u}}} \Big]^2,$$

including

$$\boldsymbol{\sigma}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) = \lambda \operatorname{tr}\Big(\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) \Big) \boldsymbol{I} + 2\mu \boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t), \tag{6.29}$$

$$\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) = \frac{1}{2} \left(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) + \boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t)^\top \right). \tag{6.30}$$

Since the $\boldsymbol{\sigma}^{\boldsymbol{u}}$ term is only present in the NBCs loss, $\mathcal{L}_{\text{NBCs}}$, the constitutive equation and kinematic relation are evaluated specifically at the boundary points on the Neumann boundaries. The first loss term, denoted as $\mathcal{L}_{\text{PDE}}$, represents the loss associated with the governing PDE. In the displacement-based formulation, the $\mathcal{L}_{\text{PDE}}$ term is constructed to fulfill only the equilibrium equation, so-called the Navier equation (NE) (Eq. 2.83). The loss weights for the PDE loss are denoted as $\{w_{i_n}\}_{i_n=1}^{N_n}$. Note that $\ddot{\tilde{\boldsymbol{u}}}$ represents the second derivative of the predicted displacement with respect to time. Moreover, Tonti's diagram in Fig. 2.9, can be examined to identify the primary variables, secondary variables, and prescribed quantities.

The terms $\mathcal{L}_{\text{DBCs}}$ and $\mathcal{L}_{\text{NBCs}}$ denote losses for Dirichlet and Neumann BCs, respectively. They include the prescribed displacement fields, denoted as $\hat{\boldsymbol{u}}$, and traction fields, denoted as $\hat{\dot{\boldsymbol{u}}}$. Since the neural network does not directly predict the stress field, it is computed based on Hooke's law (Eq. 6.29) and the kinematic equation (Eq. 6.30). The loss weights for the DBCs are represented as $\{w_{i_{bc,D}}\}_{i_{bc,D}=1}^{N_{bc,D}}$, while the loss weights for the NBCs are $\{w_{i_{bc,N}}\}_{i_{bc,N}=1}^{N_{bc,N}}$.

The loss term associated with the initial conditions is represented as $\mathcal{L}_{\text{ICs}}$. It mainly contains two components. The first loss component enforces the initial conditions due to displacements, where $\hat{\boldsymbol{u}}_0$ defines the prescribed initial displacement at $t = 0$. On the other hand, the second component enforces the initial conditions due to velocity fields, where $\hat{\dot{\boldsymbol{u}}}_0$ denotes the prescribed initial velocity fields. The loss weights associated with initial conditions are given as $\{w_{i_{ic,\boldsymbol{u}}}\}_{i_{ic,\boldsymbol{u}}=1}^{N_{ic,\boldsymbol{u}}}$ and $\{w_{i_{ic,\dot{\boldsymbol{u}}}}\}_{i_{ic,\dot{\boldsymbol{u}}}=1}^{N_{ic,\dot{\boldsymbol{u}}}}$, respectively for the displacement and velocity components. Note that $\dot{\tilde{\boldsymbol{u}}}$ denotes the first derivative of the predicted displacement with respect to time.

The last loss term denotes the experimental loss due to the provided additional displacement fields, denoted as $\boldsymbol{u}^*$. Similarly, the loss weights for the experimental loss components due to displacements are represented as $\{w_{i_{e,\boldsymbol{u}}}\}_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}}$.

Each defined loss component is computed using specific sets of input points, as outlined below:

- The $\left\{\boldsymbol{x}^{i_{rp}}, t\right\}_{i_{rp}=1}^{N_{rp}}$ term refers to the collocation points within the entire domain, where N_{rp} represents the corresponding total number of collocation points.

- The $\left\{\boldsymbol{x}^{i_{bp,D}}, t\right\}_{i_{bp,D}=1}^{N_{bp,D}}$ term denotes the boundary points for Dirichlet BCs, where $N_{bp,D}$ represents the corresponding total number of boundary points.

- The $\left\{\boldsymbol{x}^{i_{bp,N}}, t\right\}_{i_{bp,N}=1}^{N_{bp,N}}$ term represents the boundary points for Neumann BCs, where $N_{bp,N}$ represents the corresponding total number of boundary points.

- The $\left\{\boldsymbol{x}^{i_{ip,\boldsymbol{u}}}, 0\right\}_{i_{ip,\boldsymbol{u}}=1}^{N_{ip,\boldsymbol{u}}}$ term denotes the points related to displacement initial conditions, where $N_{ip,\boldsymbol{u}}$ represents the corresponding total number of initial points.

- The $\left\{\boldsymbol{x}^{i_{ip,\dot{\boldsymbol{u}}}}, 0\right\}_{i_{ip,\dot{\boldsymbol{u}}}=1}^{N_{ip,\dot{\boldsymbol{u}}}}$ term refers to initial points associated with the velocity fields, where $N_{ip,\dot{\boldsymbol{u}}}$ represents the corresponding total number of initial points.

- The $\left\{\boldsymbol{x}^{i_{ep,\boldsymbol{u}}}, t\right\}_{i_{ep,\boldsymbol{u}}=1}^{N_{ep}}$ term corresponds to additional data points based on experimental displacement data, where $N_{ep,\boldsymbol{u}}$ represents the total number of additional points.

The total loss formulation (Eq. 6.27) in a 3D Cartesian system using Lamé constants can be expressed as

$$\mathcal{L}_{\text{E}} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{DBCs}} + \mathcal{L}_{\text{NBCs}} + \mathcal{L}_{\text{ICs}} + \mathcal{L}_{\text{EXPs}}. \tag{6.31}$$

Each loss term is defined as follows:

$$\begin{aligned}
\mathcal{L}_{\text{PDE}} = w_1^{(\text{pde})}\Bigg| & c_1\frac{\partial^2\tilde{u}_x}{\partial x^2} + c_2\left(\frac{\partial^2\tilde{u}_x}{\partial y^2} + \frac{\partial^2\tilde{u}_x}{\partial z^2}\right) + \\
& c_3\left(\frac{\partial^2\tilde{u}_y}{\partial x\partial y} + \frac{\partial^2\tilde{u}_z}{\partial x\partial z}\right) + \hat{b}_x - \rho\frac{\partial^2\tilde{u}_x}{\partial t^2}\Bigg|_\Omega + \\
w_2^{(\text{pde})}\Bigg| & c_1\frac{\partial^2\tilde{u}_y}{\partial y^2} + c_2\left(\frac{\partial^2\tilde{u}_y}{\partial x^2} + \frac{\partial^2\tilde{u}_y}{\partial z^2}\right) + \\
& c_3\left(\frac{\partial^2\tilde{u}_x}{\partial x\partial y} + \frac{\partial^2\tilde{u}_z}{\partial y\partial z}\right) + \hat{b}_y - \rho\frac{\partial^2\tilde{u}_y}{\partial t^2}\Bigg|_\Omega + \\
w_3^{(\text{pde})}\Bigg| & c_1\frac{\partial^2\tilde{u}_z}{\partial z^2} + c_2\left(\frac{\partial^2\tilde{u}_z}{\partial x^2} + \frac{\partial^2\tilde{u}_z}{\partial y^2}\right) + \\
& c_3\left(\frac{\partial^2\tilde{u}_x}{\partial x\partial z} + \frac{\partial^2\tilde{u}_y}{\partial y\partial z}\right) + \hat{b}_z - \rho\frac{\partial^2\tilde{u}_z}{\partial t^2}\Bigg|_\Omega,
\end{aligned} \tag{6.32}$$

$$\mathcal{L}_{\text{DBCs}} = w_1^{(\text{dbc})}\big|\tilde{u}_x - \hat{u}_x\big|_{\partial\Omega_D} + w_2^{(\text{dbc})}\big|\tilde{u}_y - \hat{u}_y\big|_{\partial\Omega_D} + w_3^{(\text{dbc})}\big|\tilde{u}_z - \hat{u}_z\big|_{\partial\Omega_D},$$

$$\begin{aligned}
\mathcal{L}_{\text{NBCs}} = & w_1^{(\text{nbc})}\big|\sigma^u_{xx}n_x + \sigma^u_{xy}n_y + \sigma^u_{xz}n_z - \hat{t}_x\big|_{\partial\Omega_N} + \\
& w_2^{(\text{nbc})}\big|\sigma^u_{yx}n_x + \sigma^u_{yy}n_y + \sigma^u_{yz}n_z - \hat{t}_y\big|_{\partial\Omega_N} + \\
& w_3^{(\text{nbc})}\big|\sigma^u_{zx}n_x + \sigma^u_{zy}n_y + \sigma^u_{zz}n_z - \hat{t}_z\big|_{\partial\Omega_N},
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}_{\text{ICs}} = & w_1^{(\text{ic})}\big|\tilde{u}_x - \hat{u}_{x0}\big|_{\Omega_0} + w_2^{(\text{ic})}\big|\tilde{u}_y - \hat{u}_{y0}\big|_{\Omega_0} + w_3^{(\text{ic})}\big|\tilde{u}_z - \hat{u}_{z0}\big|_{\Omega_0} + \\
& w_4^{(\text{ic})}\big|\dot{\tilde{u}}_x - \dot{\hat{u}}_{x0}\big|_{\Omega_0} + w_5^{(\text{ic})}\big|\dot{\tilde{u}}_y - \dot{\hat{u}}_{y0}\big|_{\Omega_0} + w_6^{(\text{ic})}\big|\dot{\tilde{u}}_z - \dot{\hat{u}}_{z0}\big|_{\Omega_0},
\end{aligned}$$

$$\mathcal{L}_{\text{EXPs}} = w_1^{(\text{EXPs})}\big|\tilde{u}_x - u^*_x\big|_{\Omega_e} + w_2^{(\text{EXPs})}\big|\tilde{u}_y - u^*_y\big|_{\Omega_e} + w_3^{(\text{exps})}\big|\tilde{u}_z - u^*_z\big|_{\Omega_e},$$

where $c_1 = \lambda + 2\mu$, $c_2 = \mu$ and $c_3 = \lambda + \mu$. Also, the stress components can be computed using the constitutive equations, Eqs. 2.29-2.34, such that

$$\begin{aligned}
\sigma^u_{xx} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{xx}, \\
\sigma^u_{yy} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{yy}, \\
\sigma^u_{zz} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{zz}, \\
\sigma^u_{xy} = 2\mu\varepsilon_{xy}, &\quad \sigma^u_{xz} = 2\mu\varepsilon_{xz}, \quad \sigma^u_{yz} = 2\mu\varepsilon_{yz},
\end{aligned} \tag{6.33}$$

with the following set of kinematic equations (Eq. 2.23-2.28)

$$\begin{aligned}
\varepsilon^u_{xx} = \tilde{u}_{x,x}, &\quad \varepsilon^u_{yy} = \tilde{u}_{y,y}, \quad \varepsilon^u_{zz} = \tilde{u}_{z,z}, \\
\varepsilon^u_{xy} = \frac{1}{2}(\tilde{u}_{x,y} + \tilde{u}_{y,x}), &\quad \varepsilon^u_{yz} = \frac{1}{2}(\tilde{u}_{y,z} + \tilde{u}_{z,y}), \quad \varepsilon^u_{xz} = \frac{1}{2}(\tilde{u}_{x,z} + \tilde{u}_{z,x}).
\end{aligned} \tag{6.34}$$

Similarly, the network predictions are computed as follows

$$\tilde{u}_x \approx (\mathcal{N}_{u_x}(x,y,z,t))', \quad \tilde{u}_y \approx (\mathcal{N}_{u_y}(x,y,z,t))', \quad \tilde{u}_z \approx (\mathcal{N}_{u_z}(x,y,z,t))'. \tag{6.35}$$

For simplicity, we denote the *mean squared error* (MSE) as $|\cdot|$, which is defined as $\frac{1}{n}\sum_{i=1}^{n}(\cdot)^2$. This detailed, component-wise rewriting of the total loss function is essential for implementing PINNs, not only in the displacement-based formulation but also across various types of problems. By explicitly defining each term in a 3D Cartesian system–such as the governing PDE, boundary conditions, initial conditions, and experimental data terms–we ensure that the constructed network adheres to physical laws, boundary constraints, and available data. This clarity enables a precise and practical implementation. Furthermore, this breakdown highlights the computational load imposed on the PINN architecture when minimizing the various loss terms. For instance, in a 3D elastodynamics problem, the network must optimize 3 PDE loss components, 3 DBC losses, 3 NBC losses, 6 IC losses, and 3 experimental data losses. Altogether, the network must optimize a total of 18 distinct loss terms, underscoring the complexity and computational workload required for accurate modeling.

In the case of plane strain, each loss component (Eq. 6.32) reduces to

$$\begin{aligned}
\mathcal{L}_{\text{PDE}} &= w_1^{(\text{pde})}\left|c_1\frac{\partial^2\tilde{u}_x}{\partial x^2}+c_2\frac{\partial^2\tilde{u}_x}{\partial y^2}+c_3\frac{\partial^2\tilde{u}_y}{\partial x\partial y}+\hat{b}_x-\rho\frac{\partial^2\tilde{u}_x}{\partial t^2}\right|_{\Omega}+\\
&\quad w_2^{(\text{pde})}\left|c_1\frac{\partial^2\tilde{u}_y}{\partial y^2}+c_2\frac{\partial^2\tilde{u}_y}{\partial x^2}+c_3\frac{\partial^2\tilde{u}_x}{\partial x\partial y}+\hat{b}_y-\rho\frac{\partial^2\tilde{u}_y}{\partial t^2}\right|_{\Omega},\\
\mathcal{L}_{\text{DBCs}} &= w_1^{(\text{dbc})}\left|\tilde{u}_x-\hat{u}_x\right|_{\partial\Omega_D}+w_2^{(\text{dbc})}\left|\tilde{u}_y-\hat{u}_y\right|_{\partial\Omega_D},\\
\mathcal{L}_{\text{NBCs}} &= w_1^{(\text{nbc})}\left|\sigma^u_{xx}n_x+\sigma^u_{xy}n_y-\hat{t}_x\right|_{\partial\Omega_N}+\\
&\quad w_2^{(\text{nbc})}\left|\sigma^u_{yx}n_x+\sigma^u_{yy}n_y-\hat{t}_y\right|_{\partial\Omega_N},\\
\mathcal{L}_{\text{ICs}} &= w_1^{(\text{ic})}\left|\tilde{u}_x-\hat{u}_{x0}\right|_{\Omega_0}+w_2^{(\text{ic})}\left|\tilde{u}_y-\hat{u}_{y0}\right|_{\Omega_0}\\
&\quad w_3^{(\text{ic})}\left|\dot{\tilde{u}}_x-\dot{\hat{u}}_{x0}\right|_{\Omega_0}+w_4^{(\text{ic})}\left|\dot{\tilde{u}}_y-\dot{\hat{u}}_{y0}\right|_{\Omega_0},\\
\mathcal{L}_{\text{EXPs}} &= w_1^{(\text{EXPs})}\left|\tilde{u}_x-u^*_x\right|_{\Omega_e}+w_2^{(\text{EXPs})}\left|\tilde{u}_y-u^*_y\right|_{\Omega_e}.
\end{aligned}\tag{6.36}$$

Similarly, the stress components simplify to

$$\begin{aligned}
\sigma^u_{xx} &= \lambda\left(\varepsilon^u_{xx}+\varepsilon^u_{yy}\right)+2\mu\varepsilon^u_{xx},\\
\sigma^u_{yy} &= \lambda\left(\varepsilon^u_{xx}+\varepsilon^u_{yy}\right)+2\mu\varepsilon^u_{yy},\\
\sigma^u_{xy} &= 2\mu\varepsilon^u_{xy},
\end{aligned}\tag{6.37}$$

where

$$\varepsilon^u_{xx}=\tilde{u}_{x,x},\quad \varepsilon^u_{yy}=\tilde{u}_{y,y},\quad \varepsilon^u_{xy}=\frac{1}{2}(\tilde{u}_{x,y}+\tilde{u}_{y,x}).\tag{6.38}$$

This necessitates the following network outputs:

$$\tilde{u}_x\approx\left(\mathcal{N}_{u_x}(x,y,t)\right)',\quad \tilde{u}_y\approx\left(\mathcal{N}_{u_y}(x,y,t)\right)'.\tag{6.39}$$

In the case of the plane strain condition, the designed network is required to minimize 2 PDE loss components, 2 DBC losses, 2 NBC losses, 4 IC losses, and 2 experimental data losses. Altogether,

the network must optimize a total of 12 distinct loss terms. This statement also holds true for the plane stress condition.

For the plane stress condition, the only necessary adjustment is in the set of constitutive equations, i.e., Eq. 6.37. In the case of plane stress, these equations are given by

$$\begin{aligned}\sigma^u_{xx} &= \frac{E}{1-\nu^2}(\varepsilon^u_{xx} + \nu\varepsilon^u_{yy}),\\ \sigma^u_{yy} &= \frac{E}{1-\nu^2}(\nu\varepsilon^u_{xx} + \varepsilon^u_{yy}),\\ \sigma^u_{xy} &= \frac{E}{1+\nu}\varepsilon^u_{xy}.\end{aligned} \tag{6.40}$$

6.2.1.2. Elastostatics

For elastostatics or quasi-static problems, an FNN, constructed based on the displacement-based formulation, maps the given spatial coordinates $\boldsymbol{x}$ to the approximated displacement $\tilde{\boldsymbol{u}}$ (see Fig. 6.11) such that

$$\tilde{\boldsymbol{u}} := (\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}))'. \tag{6.41}$$

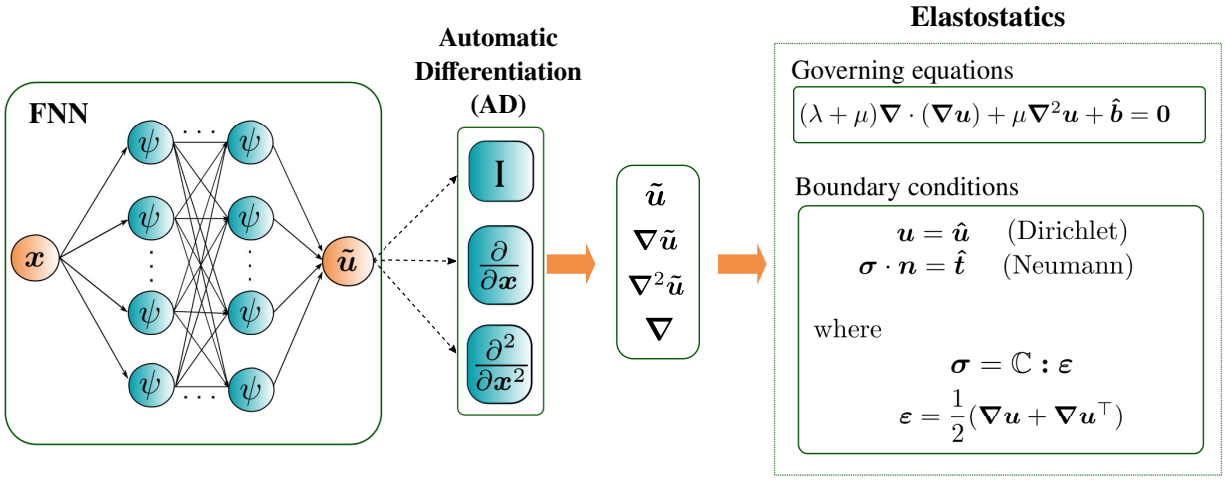


Figure 6.11.: Physics-informed neural networks in the displacement-based approach for elastostatics problems.

To obtain the governing equations of quasi-static solid mechanics for the displacement-based formulation, we omit the terms associated with time in Eq. 6.28. By doing so, we obtain the total loss $\mathcal{L}_{\mathrm{E}}$, including the losses due to additional experimental data as

$$\mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{NBCs}} + \mathcal{L}_{\mathrm{EXPs}}, \tag{6.42}$$

where

$$\begin{aligned}
\mathcal{L}_{\text{PDE}} &= \sum_{i_n=1}^{N_n} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \Big[\big[\boldsymbol{\nabla} \cdot \big(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}) \big) + \mu \boldsymbol{\nabla}^2 \tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{rp}} \right) + \hat{\boldsymbol{b}} \left(\boldsymbol{x}^{i_{rp}} \right) \big]_{i_n} \Big]^2, \\
\mathcal{L}_{\text{DBCs}} &= \sum_{i_{bc,D}=1}^{N_{bc,D}} w_{i_{bc,D}} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \Big[\big[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) - \hat{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) \big]_{i_{bc,D}} \Big]^2, \\
\mathcal{L}_{\text{NBCs}} &= \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \Big[\big[\boldsymbol{\sigma}^{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,N}} \right) \cdot \boldsymbol{n} - \hat{\mathbf{t}} \left(\boldsymbol{x}^{i_{bp,N}} \right) \big]_{i_{bc,N}} \Big]^2, \\
\mathcal{L}_{\text{EXPs}} &= \sum_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}} w_{i_{e,\boldsymbol{u}}} \frac{1}{N_{ep,\boldsymbol{u}}} \sum_{i_{ep,\boldsymbol{u}}=1}^{N_{ep,\boldsymbol{u}}} \Big[\big[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) - \boldsymbol{u}^{*} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) \big]_{i_{e,\boldsymbol{u}}} \Big]^2,
\end{aligned} \tag{6.43}$$

with the following compatibility and constitutive conditions

$$\boldsymbol{\sigma}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}) = \lambda \operatorname{tr}\Big(\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}) \Big) \boldsymbol{I} + 2\mu \boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}), \tag{6.44}$$

$$\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}) = \frac{1}{2} \left(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}) + \boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}})^{\top} \right). \tag{6.45}$$

Unlike elastodynamic problems, the quasi-static approach primarily differs in its total loss formulation by excluding the initial condition loss term ($\mathcal{L}_{\text{ICs}}$). This is because, in the absence of time-dependent inertia effects, initial conditions are irrelevant in purely quasi-static analyses. Another key distinction lies in the PDE loss formulation, where the acceleration term is omitted. Since quasi-static problems assume equilibrium at all times, the governing equations reduce to a balance of internal and external forces, without inertial effects. As a result, in a 3D Cartesian coordinate system with Lamé parameters, the PDE loss function simplifies to:

$$\begin{aligned}
\mathcal{L}_{\text{PDE}} = w_1^{(\text{pde})} \Bigg| & c_1 \frac{\partial^2 \tilde{u}_x}{\partial x^2} + c_2 \left(\frac{\partial^2 \tilde{u}_x}{\partial y^2} + \frac{\partial^2 \tilde{u}_x}{\partial z^2} \right) + c_3 \left(\frac{\partial^2 \tilde{u}_y}{\partial x \partial y} + \frac{\partial^2 \tilde{u}_z}{\partial x \partial z} \right) + \hat{b}_x \Bigg|_{\Omega} + \\
w_2^{(\text{pde})} \Bigg| & c_1 \frac{\partial^2 \tilde{u}_y}{\partial y^2} + c_2 \left(\frac{\partial^2 \tilde{u}_y}{\partial x^2} + \frac{\partial^2 \tilde{u}_y}{\partial z^2} \right) + c_3 \left(\frac{\partial^2 \tilde{u}_x}{\partial x \partial y} + \frac{\partial^2 \tilde{u}_z}{\partial y \partial z} \right) + \hat{b}_y \Bigg|_{\Omega} + \\
w_3^{(\text{pde})} \Bigg| & c_1 \frac{\partial^2 \tilde{u}_z}{\partial z^2} + c_2 \left(\frac{\partial^2 \tilde{u}_z}{\partial x^2} + \frac{\partial^2 \tilde{u}_z}{\partial y^2} \right) + c_3 \left(\frac{\partial^2 \tilde{u}_x}{\partial x \partial z} + \frac{\partial^2 \tilde{u}_y}{\partial y \partial z} \right) + \hat{b}_z \Bigg|_{\Omega}.
\end{aligned} \tag{6.46}$$

Here, the constants are defined as $c_1 = \lambda + 2\mu$, $c_2 = \mu$, and $c_3 = \lambda + \mu$. The remaining loss formulations are already given in Eq. 6.32. Also, the constitutive equations and kinematic relations remain the same as with those provided in Eq. 6.33 and Eq. 6.34, respectively. However, as time is no longer an input to the network, the network outputs are defined as

$$\begin{aligned}
\tilde{u}_x &\approx \left(\mathcal{N}_{u_x}(x, y, z) \right)', \\
\tilde{u}_y &\approx \left(\mathcal{N}_{u_y}(x, y, z) \right)', \\
\tilde{u}_z &\approx \left(\mathcal{N}_{u_z}(x, y, z) \right)'.
\end{aligned} \tag{6.47}$$

Concerning the plane strain and plane stress formulations, the same simplification is valid, meaning that time-dependent terms and the $\mathcal{L}_{\text{ICs}}$ term are omitted. Thus, the PDE loss for both plane strain and plane stress conditions is defined as

$$\begin{aligned}\mathcal{L}_{\text{PDE}} = w_1^{(\text{pde})} \left| c_1 \frac{\partial^2 \tilde{u}_x}{\partial x^2} + c_2 \frac{\partial^2 \tilde{u}_x}{\partial y^2} + c_3 \frac{\partial^2 \tilde{u}_y}{\partial x \partial y} + \hat{b}_x - \rho \frac{\partial^2 \tilde{u}_x}{\partial t^2} \right|_\Omega + \\ w_2^{(\text{pde})} \left| c_1 \frac{\partial^2 \tilde{u}_y}{\partial y^2} + c_2 \frac{\partial^2 \tilde{u}_y}{\partial x^2} + c_3 \frac{\partial^2 \tilde{u}_x}{\partial x \partial y} + \hat{b}_y - \rho \frac{\partial^2 \tilde{u}_y}{\partial t^2} \right|_\Omega .\end{aligned} \tag{6.48}$$

Indeed, the appropriate constitutive equations must be selected based on whether the problem is governed by a plane strain or plane stress condition to compute the stress field accurately, σ^u. Specifically, for plane strain problems, the material behavior follows Eq. 6.37, whereas for plane stress conditions, it adheres to Eq. 6.40. This distinction is crucial, as plane strain assumptions apply to scenarios where out-of-plane deformations are negligible, such as deep structural elements, while plane stress conditions are typically used for thin structures where out-of-plane stresses can be ignored. The network outputs for plane strain and plane stress conditions are

$$\tilde{u}_x \approx \left(\mathcal{N}_{u_x}(x, y, z)\right)', \quad \tilde{u}_y \approx \left(\mathcal{N}_{u_y}(x, y, z)\right)'. \tag{6.49}$$

6.2.2. PINNs with mixed-variable formulation

6.2.2.1. Elastodynamics

In the mixed-variable formulation for elastodynamics problems without additional network input parameters, the network takes spatial coordinates, denoted as $\boldsymbol{x}$, and time, denoted as t, as the neural network input such that $\boldsymbol{z} = [\boldsymbol{x}, t]$. Thus, an FNN maps the given network input $\boldsymbol{z}$ to the predicted displacement $\tilde{\boldsymbol{u}}$ and the predicted stress $\tilde{\boldsymbol{\sigma}}$ such as

$$\tilde{\boldsymbol{u}} := \left(\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x}; t; \boldsymbol{\theta})\right)' \quad \text{and} \quad \tilde{\boldsymbol{\sigma}} := \left(\mathcal{N}_{\boldsymbol{\sigma}}(\boldsymbol{x}; t; \boldsymbol{\theta})\right)'. \tag{6.50}$$

By integrating the general PINN loss formulations depicted in Fig. 5.2 with the governing equations for time-dependent solid mechanics, as outlined in Sections 2.2.2.1 and 2.2.2.3, we derive the total loss $\mathcal{L}_{\text{E}}$ for linear elasticity in the mixed-variable formulation, incorporating additional experimental data, as follows

$$\mathcal{L}_{\text{E}} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{DBCs}} + \mathcal{L}_{\text{NBCs}} + \mathcal{L}_{\text{ICs}} + \mathcal{L}_{\text{EXPs}}, \tag{6.51}$$

where

$$
\begin{aligned}
\mathcal{L}_{\text{PDE}} &= \sum_{i_n=1}^{N_m} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\left[\boldsymbol{\nabla} \cdot \tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{rp}}, t \right) + \hat{\boldsymbol{b}} \left(\boldsymbol{x}^{i_{rp}}, t \right) - \rho \ddot{\tilde{\boldsymbol{u}}} \left(\boldsymbol{x}^{i_{rp}}, t \right) \right]_{i_n} \right]^2 + \\
&\quad \sum_{i_n=N_m+1}^{N_n} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{rp}}, t \right) - \boldsymbol{\sigma}^{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{rp}}, t \right) \right]_{i_n} \right]^2, \\
\mathcal{L}_{\text{DBCs}} &= \sum_{i_{bc,D}=1}^{N_{bc,D}} w_{i_{bc,D}} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}}, t \right) - \hat{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}}, t \right) \right]_{i_{bc,D}} \right]^2, \\
\mathcal{L}_{\text{NBCs}} &= \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{bp,N}}, t \right) \cdot \boldsymbol{n} - \hat{\mathbf{t}} \left(\boldsymbol{x}^{i_{bp,N}}, t \right) \right]_{i_{bc,N}} \right]^2 \\
\mathcal{L}_{\text{ICs}} &= \sum_{i_{ic,\boldsymbol{u}}=1}^{N_{ic,\boldsymbol{u}}} w_{i_{ic,\boldsymbol{u}}} \frac{1}{N_{ip,\boldsymbol{u}}} \sum_{i_{ip,\boldsymbol{u}}=1}^{N_{ip,\boldsymbol{u}}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ip,\boldsymbol{u}}}, 0 \right) - \hat{\boldsymbol{u}}_0 \left(\boldsymbol{x}^{i_{ip,\boldsymbol{u}}}, 0 \right) \right]_{i_{e,\boldsymbol{u}}} \right]^2 + \\
&\quad \sum_{i_{ic,\dot{\boldsymbol{u}}}=1}^{N_{ic,\dot{\boldsymbol{u}}}} w_{i_{ic,\dot{\boldsymbol{u}}}} \frac{1}{N_{ip,\dot{\boldsymbol{u}}}} \sum_{i_{ip,\dot{\boldsymbol{u}}}=1}^{N_{ip,\dot{\boldsymbol{u}}}} \left[\left[\dot{\tilde{\boldsymbol{u}}} \left(\boldsymbol{x}^{i_{ip,\dot{\boldsymbol{u}}}}, 0 \right) - \dot{\hat{\boldsymbol{u}}}_0 \left(\boldsymbol{x}^{i_{ip,\dot{\boldsymbol{u}}}}, 0 \right) \right]_{i_{e,\dot{\boldsymbol{u}}}} \right]^2, \\
\mathcal{L}_{\text{EXPs}} &= \sum_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}} w_{i_{e,\boldsymbol{u}}} \frac{1}{N_{ep,\boldsymbol{u}}} \sum_{i_{ep,\boldsymbol{u}}=1}^{N_{ep,\boldsymbol{u}}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}}, t \right) - \boldsymbol{u}^* \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}}, t \right) \right]_{i_{e,\boldsymbol{u}}} \right]^2 + \\
&\quad \sum_{i_{e,\boldsymbol{\sigma}}=1}^{N_{e,\boldsymbol{\sigma}}} w_{i_{e,\boldsymbol{\sigma}}} \frac{1}{N_{ep,\boldsymbol{\sigma}}} \sum_{i_{ep,\boldsymbol{\sigma}}=1}^{N_{ep,\boldsymbol{\sigma}}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{\sigma}}}, t \right) - \boldsymbol{\sigma}^* \left(\boldsymbol{x}^{i_{ep,\boldsymbol{\sigma}}}, t \right) \right]_{i_{e,\boldsymbol{\sigma}}} \right]^2,
\end{aligned}
\tag{6.52}
$$

with

$$
\boldsymbol{\sigma}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t) = \mathbb{C} : \boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t), \tag{6.53}
$$

$$
\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t) = \frac{1}{2} \left(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t) + \boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}, t)^\top \right). \tag{6.54}
$$

Since the $\boldsymbol{\sigma}^{\boldsymbol{u}}$ term appears only in the PDE loss, $\mathcal{L}_{\text{PDE}}$, the constitutive equation and kinematic relation are evaluated specifically at the domain collocation points. In the mixed-variable formulation, the $\mathcal{L}_{\text{PDE}}$ term is constructed in a composite form to fulfill both the balance equation (BE) (Eq. 2.8) and the stress-to-stress coupling (SS) (Eq. 2.87) as depicted in Fig. 2.10. The index N_m is used to distinguish the loss weights related to BE and SS. The loss weights for the PDE loss are denoted as $\{w_{i_n}\}_{i_n=1}^{N_n}$. Note that $\ddot{\tilde{\boldsymbol{u}}}$ represents the second derivative of the predicted displacement with respect to time.

Refer to Section 6.2.1.1 for the definitions of the loss terms $\mathcal{L}_{\text{DBCs}}$, $\mathcal{L}_{\text{NBCs}}$, and $\mathcal{L}_{\text{ICs}}$. Compared to the displacement-based approach, we can now directly utilize the predicted stress to enforce the NBCs.

The last loss term denotes the experimental loss due to the provided additional displacement fields, denoted as $\boldsymbol{u}^*$, and the additional stress field, denoted as $\boldsymbol{\sigma}^*$. Similarly, the loss weights for the experimental loss components due to displacements are represented as $\{w_{i_{e,\boldsymbol{u}}}\}_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}}$, while the

experimental data loss weights due to stresses are provided as $\{w_{i_{e,\boldsymbol{\sigma}}}\}_{i_{e,\boldsymbol{\sigma}}=1}^{N_{e,\boldsymbol{\sigma}}}$. Note that $\dot{\tilde{\boldsymbol{u}}}$ denotes the first derivative of the predicted displacement with respect to time.

Each of the defined loss components in Eq. 6.52 is computed based on specific input points as described in the displacement-based formulation for time-dependent problems (see Section 6.2.1.1). The only difference in the mixed-variable formulation is that the experimental data loss, $\mathcal{L}_{\text{EXPs}}$, now also includes data points related to the stress field. Specifically,

- $\left\{\boldsymbol{x}^{i_{ep,\boldsymbol{\sigma}}}, t\right\}_{i_{ep,\boldsymbol{\sigma}}=1}^{N_{ep}}$ refers to additional points corresponding to experimental stress data, where $N_{ep,\boldsymbol{\sigma}}$ represents the total number of additional stress data points.

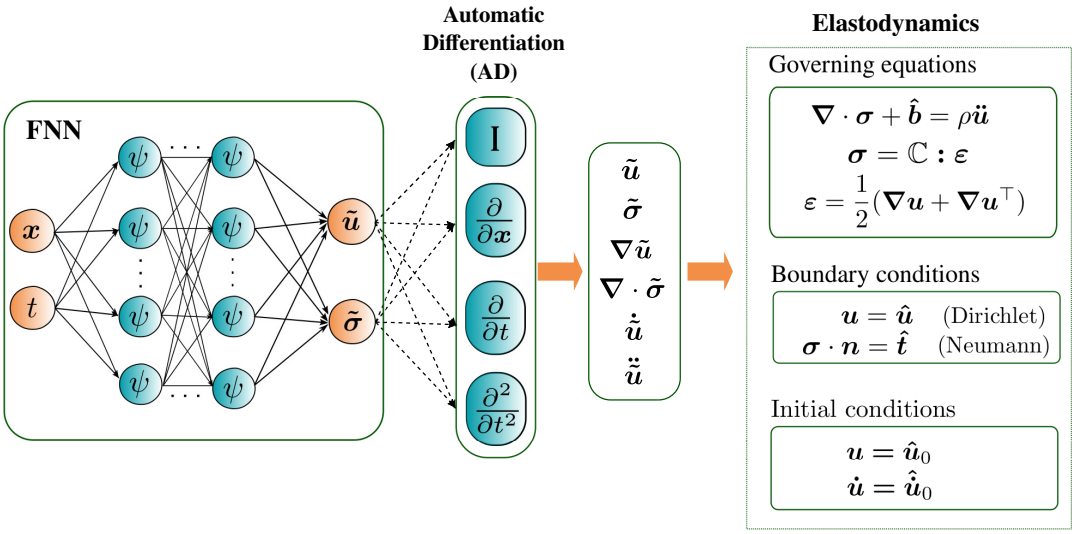


Figure 6.12.: Physics-informed neural networks in the mixed-variable formulation to solve elastodynamics problems.

An illustration of the mixed-variable PINN approach for solving elastodynamics problems is shown in Fig. 6.12. In this framework, an FNN takes the spatial coordinates, $\boldsymbol{x}$, and time, t, as inputs to predict both the displacement field, denoted by $\tilde{\boldsymbol{u}}$, and the stress field, denoted by $\tilde{\boldsymbol{\sigma}}$.

Using automatic differentiation, we can efficiently compute the first-order derivatives of the network outputs with respect to spatial coordinates, $\boldsymbol{x}$, as well as both first- and second-order derivatives with respect to time, t. This enables the derivation of all intermediate variables required for formulating the governing equations of the problem, including boundary and initial conditions.

However, the mixed-variable formulation still involves second-order time derivatives of displacement, which are computationally expensive. The velocity field can be introduced as an additional network output [140] to design a network that requires only first-order derivatives in both spatial and temporal domains. This approach, however, necessitates additional coupling operators between the displacement, velocity, and stress fields.

Furthermore, displacement and stress fields are directly represented as network outputs in the mixed-variable formulation, allowing Dirichlet and Neumann boundary conditions to be strictly enforced as hard constraints through output transformations. Similarly, initial conditions associated with the displacement field can be imposed as hard constraints using output transformations.

Conversely, since the velocity field is not treated as a network input, its initial conditions must be enforced using soft constraints.

The total loss formulation (Eq. 6.52) in 3D Cartesian system using Lamé constants can be expressed as

$$\mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{NBCs}} + \mathcal{L}_{\mathrm{ICs}} + \mathcal{L}_{\mathrm{EXPs}}. \tag{6.55}$$

Each loss term is defined as follows:

$$\begin{aligned}
\mathcal{L}_{\mathrm{PDE}} = {} & w_1^{(\mathrm{pde})}\big|\tilde{\sigma}_{xx,x} + \tilde{\sigma}_{xy,y} + \tilde{\sigma}_{xz,z} + \hat{b}_x - \rho\ddot{\tilde{u}}_x\big|_{\Omega} + \\
& w_2^{(\mathrm{pde})}\big|\tilde{\sigma}_{yx,x} + \tilde{\sigma}_{yy,y} + \tilde{\sigma}_{yz,z} + \hat{b}_y - \rho\ddot{\tilde{u}}_y\big|_{\Omega} + \\
& w_3^{(\mathrm{pde})}\big|\tilde{\sigma}_{zx,x} + \tilde{\sigma}_{zy,y} + \tilde{\sigma}_{zz,z} + \hat{b}_z - \rho\ddot{\tilde{u}}_z\big|_{\Omega} + \\
& w_4^{(\mathrm{pde})}\big|\tilde{\sigma}_{xx} - \sigma^u_{xx}\big|_{\Omega} + w_5^{(\mathrm{pde})}\big|\tilde{\sigma}_{yy} - \sigma^u_{yy}\big|_{\Omega} + w_6^{(\mathrm{pde})}\big|\tilde{\sigma}_{zz} - \sigma^u_{zz}\big|_{\Omega} + \\
& w_7^{(\mathrm{pde})}\big|\tilde{\sigma}_{xy} - \sigma^u_{xy}\big|_{\Omega} + w_8^{(\mathrm{pde})}\big|\tilde{\sigma}_{xz} - \sigma^u_{xz}\big|_{\Omega} + w_9^{(\mathrm{pde})}\big|\tilde{\sigma}_{yz} - \sigma^u_{yz}\big|_{\Omega},
\end{aligned}$$

$$\mathcal{L}_{\mathrm{DBCs}} = w_1^{(\mathrm{dbc})}\big|\tilde{u}_x - \hat{u}_x\big|_{\partial\Omega_D} + w_2^{(\mathrm{dbc})}\big|\tilde{u}_y - \hat{u}_y\big|_{\partial\Omega_D} + w_3^{(\mathrm{dbc})}\big|\tilde{u}_z - \hat{u}_z\big|_{\partial\Omega_D},$$

$$\begin{aligned}
\mathcal{L}_{\mathrm{NBCs}} = {} & w_1^{(\mathrm{nbc})}\big|\tilde{\sigma}_{xx}n_x + \tilde{\sigma}_{xy}n_y + \tilde{\sigma}_{xz}n_z - \hat{t}_x\big|_{\partial\Omega_N} + \\
& w_2^{(\mathrm{nbc})}\big|\tilde{\sigma}_{yx}n_x + \tilde{\sigma}_{yy}n_y + \tilde{\sigma}_{yz}n_z - \hat{t}_y\big|_{\partial\Omega_N} + \\
& w_3^{(\mathrm{nbc})}\big|\tilde{\sigma}_{zx}n_x + \tilde{\sigma}_{zy}n_y + \tilde{\sigma}_{zz}n_z - \hat{t}_z\big|_{\partial\Omega_N},
\end{aligned} \tag{6.56}$$

$$\begin{aligned}
\mathcal{L}_{\mathrm{ICs}} = {} & w_1^{(\mathrm{ic})}\big|\tilde{u}_x - \hat{u}_{x0}\big|_{\Omega_0} + w_2^{(\mathrm{ic})}\big|\tilde{u}_y - \hat{u}_{y0}\big|_{\Omega_0} + w_3^{(\mathrm{ic})}\big|\tilde{u}_z - \hat{u}_{z0}\big|_{\Omega_0} + \\
& w_4^{(\mathrm{ic})}\big|\dot{\tilde{u}}_x - \dot{\hat{u}}_{x0}\big|_{\Omega_0} + w_5^{(\mathrm{ic})}\big|\dot{\tilde{u}}_y - \dot{\hat{u}}_{y0}\big|_{\Omega_0} + w_6^{(\mathrm{ic})}\big|\dot{\tilde{u}}_z - \dot{\hat{u}}_{z0}\big|_{\Omega_0},
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}_{\mathrm{EXPs}} = {} & w_1^{(\mathrm{EXPs})}\big|\tilde{u}_x - u^*_x\big|_{\Omega_e} + w_2^{(\mathrm{EXPs})}\big|\tilde{u}_y - u^*_y\big|_{\Omega_e} + w_3^{(\mathrm{exps})}\big|\tilde{u}_z - u^*_z\big|_{\Omega_e} + \\
& w_4^{(\mathrm{exps})}\big|\tilde{\sigma}_{xx} - \sigma^*_{xx}\big|_{\Omega_e} + w_5^{(\mathrm{exps})}\big|\tilde{\sigma}_{yy} - \sigma^*_{yy}\big|_{\Omega_e} + w_6^{(\mathrm{exps})}\big|\tilde{\sigma}_{zz} - \sigma^*_{zz}\big|_{\Omega_e} \\
& w_7^{(\mathrm{exps})}\big|\tilde{\sigma}_{xy} - \sigma^*_{xy}\big|_{\Omega_e} + w_8^{(\mathrm{exps})}\big|\tilde{\sigma}_{yz} - \sigma^*_{yz}\big|_{\Omega_e} + w_9^{(\mathrm{exps})}\big|\tilde{\sigma}_{xz} - \sigma^*_{xz}\big|_{\Omega_e}
\end{aligned}$$

Also, the stress components (secondary variables) can be computed using the constitutive equations, Eqs. 2.29-2.34, such that

$$\begin{aligned}
\sigma^u_{xx} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{xx}, \\
\sigma^u_{yy} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{yy}, \\
\sigma^u_{zz} &= \lambda\big(\varepsilon^u_{xx} + \varepsilon^u_{yy} + \varepsilon^u_{zz}\big) + 2\mu\varepsilon^u_{zz}, \\
\sigma^u_{xy} = 2\mu\varepsilon_{xy}, &\quad \sigma^u_{xz} = 2\mu\varepsilon_{xz}, \quad \sigma^u_{yz} = 2\mu\varepsilon_{yz},
\end{aligned} \tag{6.57}$$

with the following set of kinematic equations (Eq. 2.23-2.28)

$$\begin{aligned}
\varepsilon^u_{xx} = \tilde{u}_{x,x}, &\quad \varepsilon^u_{yy} = \tilde{u}_{y,y}, \quad \varepsilon^u_{zz} = \tilde{u}_{z,z}, \\
\varepsilon^u_{xy} = \frac{1}{2}(\tilde{u}_{x,y} + \tilde{u}_{y,x}), &\quad \varepsilon^u_{yz} = \frac{1}{2}(\tilde{u}_{y,z} + \tilde{u}_{z,y}), \quad \varepsilon^u_{xz} = \frac{1}{2}(\tilde{u}_{x,z} + \tilde{u}_{z,x}).
\end{aligned} \tag{6.58}$$

Similarly, the network outputs are expressed as follows

$$\begin{gathered}\tilde{u}_x \approx (\mathcal{N}_{u_x}(x,y,z,t))', \quad \tilde{u}_y \approx (\mathcal{N}_{u_y}(x,y,z,t))', \quad \tilde{u}_z \approx (\mathcal{N}_{u_z}(x,y,z,t))', \\ \tilde{\sigma}_{xx} \approx (\mathcal{N}_{\sigma_{xx}}(x,y,z,t))', \quad \tilde{\sigma}_{yy} \approx (\mathcal{N}_{\sigma_{yy}}(x,y,z,t))', \quad \tilde{\sigma}_{zz} \approx (\mathcal{N}_{\sigma_{zz}}(x,y,z,t))', \\ \{\tilde{\sigma}_{xy} = \tilde{\sigma}_{yx}\} \approx (\mathcal{N}_{\sigma_{xy}}(x,y,z,t))', \\ \{\tilde{\sigma}_{yz} = \tilde{\sigma}_{zy}\} \approx (\mathcal{N}_{\sigma_{yz}}(x,y,z,t))', \\ \{\tilde{\sigma}_{xz} = \tilde{\sigma}_{zx}\} \approx (\mathcal{N}_{\sigma_{xz}}(x,y,z,t))'.\end{gathered} \tag{6.59}$$

Similar to the displacement-based formulation, this detailed, component-wise breakdown of the total loss function is essential for implementing PINNs in elastodynamics using a mixed-variable approach. This decomposition highlights the computational demands on the PINN architecture in minimizing the various loss terms. For example, in a 3D elastodynamics problem, the network must minimize 9 PDE loss components, 3 DBC losses, 3 NBC losses, 6 IC losses, and 9 experimental data losses. Altogether, the network is tasked to optimize a total of 30 distinct loss terms.

In the case of plane strain, each loss component (Eq. 6.55) simplifies to

$$\begin{aligned}\mathcal{L}_{\text{PDE}} &= w_1^{(\text{pde})}\left|\tilde{\sigma}_{xx,x} + \tilde{\sigma}_{xy,y} + \hat{b}_x - \rho\ddot{\tilde{u}}_x\right|_\Omega + w_2^{(\text{pde})}\left|\tilde{\sigma}_{yx,x} + \tilde{\sigma}_{yy,y} + \hat{b}_y - \rho\ddot{\tilde{u}}_y\right|_\Omega + \\ &\quad w_3^{(\text{pde})}\left|\tilde{\sigma}_{xx} - \sigma^u_{xx}\right|_\Omega + w_4^{(\text{pde})}\left|\tilde{\sigma}_{yy} - \sigma^u_{yy}\right|_\Omega + w_5^{(\text{pde})}\left|\tilde{\sigma}_{xy} - \sigma^u_{xy}\right|_\Omega, \\ \mathcal{L}_{\text{DBCs}} &= w_1^{(\text{dbc})}\left|\tilde{u}_x - \hat{u}_x\right|_{\partial\Omega_D} + w_2^{(\text{dbc})}\left|\tilde{u}_y - \hat{u}_y\right|_{\partial\Omega_D}, \\ \mathcal{L}_{\text{NBCs}} &= w_1^{(\text{nbc})}\left|\tilde{\sigma}_{xx}n_x + \tilde{\sigma}_{xy}n_y - \hat{t}_x\right|_{\partial\Omega_N} + w_2^{(\text{nbc})}\left|\tilde{\sigma}_{yx}n_x + \tilde{\sigma}_{yy}n_y - \hat{t}_y\right|_{\partial\Omega_N}, \\ \mathcal{L}_{\text{ICs}} &= w_1^{(\text{ic})}\left|\tilde{u}_x - \hat{u}_{x0}\right|_{\Omega_0} + w_2^{(\text{ic})}\left|\tilde{u}_y - \hat{u}_{y0}\right|_{\Omega_0} + \\ &\quad w_3^{(\text{ic})}\left|\dot{\tilde{u}}_x - \dot{\hat{u}}_{x0}\right|_{\Omega_0} + w_4^{(\text{ic})}\left|\dot{\tilde{u}}_y - \dot{\hat{u}}_{y0}\right|_{\Omega_0}, \\ \mathcal{L}_{\text{EXPs}} &= w_1^{(\text{exps})}\left|\tilde{u}_x - u^*_x\right|_{\Omega_e} + w_2^{(\text{exps})}\left|\tilde{u}_y - u^*_y\right|_{\Omega_e} + \\ &\quad w_3^{(\text{exps})}\left|\tilde{\sigma}_{xx} - \sigma^*_{xx}\right|_{\Omega_e} + w_4^{(\text{exps})}\left|\tilde{\sigma}_{yy} - \sigma^*_{yy}\right|_{\Omega_e} + w_5^{(\text{exps})}\left|\tilde{\sigma}_{xy} - \sigma^*_{xy}\right|_{\Omega_e}.\end{aligned} \tag{6.60}$$

Similarly, the set of constitutive equations (Eq. 6.57) can be written as

$$\begin{aligned}\sigma^u_{xx} &= \lambda\left(\varepsilon^u_{xx} + \varepsilon^u_{yy}\right) + 2\mu\varepsilon^u_{xx}, \\ \sigma^u_{yy} &= \lambda\left(\varepsilon^u_{xx} + \varepsilon^u_{yy}\right) + 2\mu\varepsilon^u_{yy}, \\ \sigma^u_{xy} &= 2\mu\varepsilon_{xy},\end{aligned} \tag{6.61}$$

with the corresponding set of kinematic equations (Eq. 6.58)

$$\varepsilon^u_{xx} = \tilde{u}_{x,x}, \quad \varepsilon^u_{yy} = \tilde{u}_{y,y}, \quad \varepsilon^u_{xy} = \frac{1}{2}(\tilde{u}_{x,y} + \tilde{u}_{y,x}). \tag{6.62}$$

The approximated solutions are expressed as

$$\begin{gathered}\tilde{u}_x \approx (\mathcal{N}_{u_x}(x,y,t))', \quad \tilde{u}_y \approx (\mathcal{N}_{u_y}(x,y,t))', \\ \tilde{\sigma}_{xx} \approx (\mathcal{N}_{\sigma_{xx}}(x,y,t))', \quad \tilde{\sigma}_{yy} \approx (\mathcal{N}_{\sigma_{yy}}(x,y,t))', \quad \{\tilde{\sigma}_{xy} = \tilde{\sigma}_{yx}\} \approx (\mathcal{N}_{\sigma_{xy}}(x,y,t))'.\end{gathered} \tag{6.63}$$

Under the plane strain condition, the designed network is tasked to minimize 5 PDE loss components, 2 DBC losses, 2 NBC losses, 4 IC losses, and 5 experimental data losses. Altogether, the network faces a total of 18 distinct loss terms to optimize. This statement is also valid for the plane stress condition.

For the plane stress condition, the only adjustment needed lies in the set of constitutive equations, i.e. Eq. 6.61. It is expressed as

$$\begin{aligned}\sigma^u_{xx} &= \frac{E}{1-\nu^2}(\varepsilon^u_{xx} + \nu\varepsilon^u_{yy}) \\ \sigma^u_{yy} &= \frac{E}{1-\nu^2}(\nu\varepsilon^u_{xx} + \varepsilon^u_{yy}) \\ \sigma^u_{xy} &= \frac{E}{1+\nu}\varepsilon^u_{xy}.\end{aligned} \tag{6.64}$$

6.2.2.2. Elastostatics

In the elastostatics (quasi-static) problems without additional network input parameters, the network takes spatial coordinates, denoted as $\boldsymbol{x}$, as the neural network input such that $\boldsymbol{z} = [\boldsymbol{x}]$. As depicted in Fig. 6.11, an FNN maps the given spatial coordinates $\boldsymbol{x}$ to the displacement vector $\tilde{\boldsymbol{u}}$ and stress tensor $\tilde{\boldsymbol{\sigma}}$ such that

$$\tilde{\boldsymbol{u}} := (\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}))' \quad \text{and} \quad \tilde{\boldsymbol{\sigma}} := (\mathcal{N}_{\boldsymbol{\sigma}}(\boldsymbol{x};\boldsymbol{\theta}))'. \tag{6.65}$$

To obtain the governing equations of quasi-static solid mechanics, we omit the terms associated with time as explained in Section 2.2.2.1. By doing so, we obtain the total loss $\mathcal{L}_{\mathrm{E}}$ for linear elasticity in the mixed-variable formulation with additional experimental data as

$$\mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{NBCs}} + \mathcal{L}_{\mathrm{EXPs}}, \tag{6.66}$$

where

$$
\begin{aligned}
\mathcal{L}_{\text{PDE}} &= \sum_{i_n=1}^{N_m} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\left[\boldsymbol{\nabla} \cdot \tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{rp}} \right) + \hat{\boldsymbol{b}} \left(\boldsymbol{x}^{i_{rp}} \right) \right]_{i_n} \right]^2 + \\
&\quad \sum_{i_n=N_m+1}^{N_n} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{rp}} \right) - \boldsymbol{\sigma}^{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{rp}} \right) \right]_{i_n} \right]^2, \\
\mathcal{L}_{\text{DBCs}} &= \sum_{i_{bc,D}=1}^{N_{bc,D}} w_{i_{bc,D}} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) - \hat{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) \right]_{i_{bc,D}} \right]^2, \\
\mathcal{L}_{\text{NBCs}} &= \sum_{i_{bc,N}=1}^{N_{bc,N}} w_{i_{bc,N}} \frac{1}{N_{bp,N}} \sum_{i_{bp,N}=1}^{N_{bp,N}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{bp,N}} \right) \cdot \boldsymbol{n} - \hat{\mathbf{t}} \left(\boldsymbol{x}^{i_{bp,N}} \right) \right]_{i_{bc,N}} \right]^2 \\
\mathcal{L}_{\text{EXPs}} &= \sum_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}} w_{i_{e,\boldsymbol{u}}} \frac{1}{N_{ep,\boldsymbol{u}}} \sum_{i_{ep,\boldsymbol{u}}=1}^{N_{ep,\boldsymbol{u}}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) - \boldsymbol{u}^{*} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) \right]_{i_{e,\boldsymbol{u}}} \right]^2 + \\
&\quad \sum_{i_{e,\boldsymbol{\sigma}}=1}^{N_{e,\boldsymbol{\sigma}}} w_{i_{e,\boldsymbol{\sigma}}} \frac{1}{N_{ep,\boldsymbol{\sigma}}} \sum_{i_{ep,\boldsymbol{\sigma}}=1}^{N_{ep,\boldsymbol{\sigma}}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{\sigma}}} \right) - \boldsymbol{\sigma}^{*} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{\sigma}}} \right) \right]_{i_{e,\boldsymbol{\sigma}}} \right]^2,
\end{aligned}
\tag{6.67}
$$

with

$$
\boldsymbol{\sigma}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}) = \mathbb{C} : \boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}), \tag{6.68}
$$

$$
\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}) = \frac{1}{2} \left(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}}) + \boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{rp}})^{\top} \right). \tag{6.69}
$$

An illustration of the mixed-variable PINN approach for solving quasi-static problems is shown in Fig. 6.13. In this approach, an FNN takes spatial coordinates, $\boldsymbol{x}$, as inputs to predict displacement and stress fields. Using automatic differentiation, we can efficiently compute the first-order derivatives of the network outputs with respect to spatial coordinates. This allows us to obtain all the necessary intermediate variables required for formulating the governing equations and boundary conditions of the target problem. Compared to time-dependent approach (Section 6.2.2.1), the mixed-variable formulation for quasi-static problems offers a huge computational advantage due to involving only first-order derivatives.

Unlike elastodynamics, the quasi-static total loss formulation does not include the losses associated with the initial conditions ($\mathcal{L}_{\text{ICs}}$). The second difference lies in the PDE loss, which does not include the acceleration term. Therefore, in 3D Cartesian system with Lamé parameters, the PDE loss can be expressed as follows

$$
\begin{aligned}
\mathcal{L}_{\text{PDE}} = {} & w_1^{(\text{pde})} \left| \tilde{\sigma}_{xx,x} + \tilde{\sigma}_{xy,y} + \tilde{\sigma}_{xz,z} + \hat{b}_x \right|_{\Omega} + \\
& w_2^{(\text{pde})} \left| \tilde{\sigma}_{yx,x} + \tilde{\sigma}_{yy,y} + \tilde{\sigma}_{yz,z} + \hat{b}_y \right|_{\Omega} + \\
& w_3^{(\text{pde})} \left| \tilde{\sigma}_{zx,x} + \tilde{\sigma}_{zy,y} + \tilde{\sigma}_{zz,z} + \hat{b}_z \right|_{\Omega} + \\
& w_4^{(\text{pde})} \left| \tilde{\sigma}_{xx} - \sigma^u_{xx} \right|_{\Omega} + w_5^{(\text{pde})} \left| \tilde{\sigma}_{yy} - \sigma^u_{yy} \right|_{\Omega} + w_6^{(\text{pde})} \left| \tilde{\sigma}_{zz} - \sigma^u_{zz} \right|_{\Omega} + \\
& w_7^{(\text{pde})} \left| \tilde{\sigma}_{xy} - \sigma^u_{xy} \right|_{\Omega} + w_8^{(\text{pde})} \left| \tilde{\sigma}_{xz} - \sigma^u_{xz} \right|_{\Omega} + w_9^{(\text{pde})} \left| \tilde{\sigma}_{yz} - \sigma^u_{yz} \right|_{\Omega}
\end{aligned}
\tag{6.70}
$$

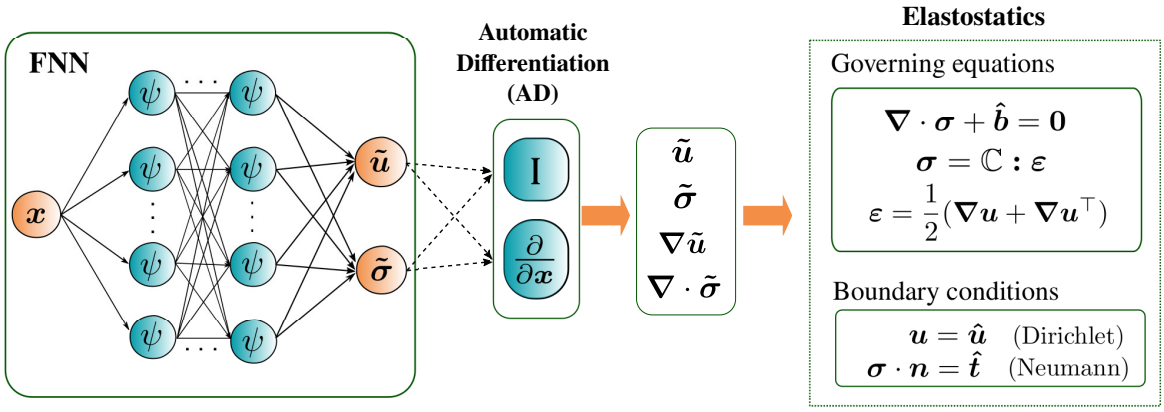


Figure 6.13.: Physics-informed neural networks in the mixed-variable form to solve elastostatics problems.

The remaining loss formulations are already given in Eq. 6.56. Also, the constitutive equations and kinematic relations remain the same as those provided in Eq. 6.57 and Eq. 6.58, respectively. However, as time is no longer an input to the network, the network outputs are defined as

$$\begin{aligned} \tilde{u}_x \approx (\mathcal{N}_{u_x}(x,y,z))', \quad \tilde{u}_y \approx (\mathcal{N}_{u_y}(x,y,z))', \quad \tilde{u}_z \approx (\mathcal{N}_{u_z}(x,y,z))', \\ \tilde{\sigma}_{xx} \approx (\mathcal{N}_{\sigma_{xx}}(x,y,z))', \quad \tilde{\sigma}_{yy} \approx (\mathcal{N}_{\sigma_{yy}}(x,y,z))', \quad \tilde{\sigma}_{zz} \approx (\mathcal{N}_{\sigma_{zz}}(x,y,z))', \\ \{\tilde{\sigma}_{xy} = \tilde{\sigma}_{yx}\} \approx (\mathcal{N}_{\sigma_{xy}}(x,y,z))', \\ \{\tilde{\sigma}_{yz} = \tilde{\sigma}_{zy}\} \approx (\mathcal{N}_{\sigma_{yz}}(x,y,z))', \\ \{\tilde{\sigma}_{xz} = \tilde{\sigma}_{zx}\} \approx (\mathcal{N}_{\sigma_{xz}}(x,y,z))'. \end{aligned} \tag{6.71}$$

Concerning the plane strain and plane stress formulations, the same simplification applies, meaning that time-dependent terms and the $\mathcal{L}_{\text{ICs}}$ term are omitted. Thus, the PDE loss for both plane strain and plane stress conditions can be defined as follows

$$\begin{aligned} &= w_1^{(\text{pde})} \left| \tilde{\sigma}_{xx,x} + \tilde{\sigma}_{xy,y} + \hat{b}_x \right|_\Omega + \\ &\quad w_2^{(\text{pde})} \left| \tilde{\sigma}_{yx,x} + \tilde{\sigma}_{yy,y} + \hat{b}_y \right|_\Omega + \\ &\quad w_3^{(\text{pde})} \left| \tilde{\sigma}_{xx} - \sigma^u_{xx} \right|_\Omega + w_4^{(\text{pde})} \left| \tilde{\sigma}_{yy} - \sigma^u_{yy} \right|_\Omega + \quad w_5^{(\text{pde})} \left| \tilde{\sigma}_{xy} - \sigma^u_{xy} \right|_\Omega + \end{aligned} \tag{6.72}$$

Depending on whether the plane strain or plane stress condition is considered, the appropriate constitutive equations must be chosen to compute the stress field, σ^u. Specifically, Eq. 6.61 applies to plane strain, while Eq. 6.64 applies to plane stress.

Moreover, the components of the network output for both plane strain and plane stress conditions are expressed as

$$\begin{aligned} \tilde{u}_x \approx (\mathcal{N}_{u_x}(x,y))', \quad \tilde{u}_y \approx (\mathcal{N}_{u_y}(x,y))', \\ \tilde{\sigma}_{xx} \approx (\mathcal{N}_{\sigma_{xx}}(x,y))', \quad \tilde{\sigma}_{yy} \approx (\mathcal{N}_{\sigma_{yy}}(x,y))', \quad \{\tilde{\sigma}_{xy} = \tilde{\sigma}_{yx}\} \approx (\mathcal{N}_{\sigma_{xy}}(x,y))'. \end{aligned} \tag{6.73}$$

6.2.2.3. Importance of output scaling

As outlined in Section 5.3.2, output scaling, a form of output transformation, helps prevent the optimizer from becoming trapped in local minima. In solid mechanics applications, PINNs predict

the displacement field, $\boldsymbol{u}$, and the stress field, $\boldsymbol{\sigma}$, as network outputs. With random initialization and no transfer learning, these outputs are initially estimated from a random parameter initialization. While effective, this approach can lead to convergence challenges if output variables have similar initial magnitudes. This issue can be explained through the example of the SS loss term (see Eq. 6.52) which is expressed as

$$\mathcal{L}_{\mathrm{SS}} = \sum_{i_n=N_m+1}^{N_n} w_{i_n} \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\left[\tilde{\boldsymbol{\sigma}} \left(\boldsymbol{x}^{i_{rp}}, t \right) - \boldsymbol{\sigma}^{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{rp}}, t \right) \right]_{i_n} \right]^2, \tag{6.74}$$

where

$$\boldsymbol{\sigma}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) = \lambda \operatorname{tr}\Big(\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) \Big) \boldsymbol{I} + 2\mu \boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) \tag{6.75}$$

$$\boldsymbol{\varepsilon}^{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) = \frac{1}{2} \left(\boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t) + \boldsymbol{\nabla} \tilde{\boldsymbol{u}}(\boldsymbol{x}^{i_{bp,N}}, t)^{\top} \right). \tag{6.76}$$

Minimizing $\mathcal{L}_{\mathrm{SS}}$ requires satisfying $\boldsymbol{\sigma} = \boldsymbol{\sigma}^{\boldsymbol{u}}$. When large-scale material property values, such as large λ and μ (e.g., $\mathcal{O}(3)$), are used, $\boldsymbol{\sigma}^{\boldsymbol{u}}$ can initially dominate $\boldsymbol{\sigma}$, resulting in a significant difference in magnitudes within the loss term $\mathcal{L}_{\mathrm{SS}}$. Why is this the case? This occurs because, at the first forward pass of the neural network, $\tilde{\boldsymbol{\sigma}}$ and $\tilde{\boldsymbol{u}}$ are initialized to similar magnitudes, around $\mathcal{O}(1)$, due to the use of Glorot initialization (see Sections 3.4.3.3 and 5.3.2). Assuming the derivatives of $\tilde{\boldsymbol{\sigma}}$ remain on the order of $\mathcal{O}(1)$, $\boldsymbol{\varepsilon}^{\boldsymbol{u}}$ will also initially have this same order of magnitude. However, when we compute $\boldsymbol{\sigma}^{\boldsymbol{u}}$ using the small-scale $\boldsymbol{\varepsilon}^{\boldsymbol{u}}$ (of order $\mathcal{O}(1)$) and the large material parameters (of order $\mathcal{O}(3)$), the result is a large-scale $\boldsymbol{\sigma}^{\boldsymbol{u}}$, around $\mathcal{O}(3)$. Now, we expect from optimizer to fulfill the requirement $\boldsymbol{\sigma} = \boldsymbol{\sigma}^{\boldsymbol{u}}$, despite this large difference in magnitudes. This requires either a substantial increase in $\boldsymbol{\sigma}$ or an extensive decrease in $\boldsymbol{\sigma}^{\boldsymbol{u}}$, leading to issues with large adjustments during optimization. Such adjustments can be problematic because the gradient of the commonly used tanh activation function tends toward zero for large input values, resulting in the vanishing gradient problem.

To improve optimization, the displacement output, $\boldsymbol{u}$, can be scaled by the inverse of the Lamé's first parameter $\frac{1}{\lambda}$, as it is typically larger than the shear modulus μ. Alternatively, this scaling can also be done using Young's modulus, i.e., $\frac{1}{E}$. Such output scaling ensures that the initial magnitudes of both terms are of a similar order of magnitude in the SS condition, effectively balancing the loss contributions and enhancing convergence stability.

6.2.3. Comparison of displacement-based and mixed-variable approaches

Table 6.1 compares the displacement-based and mixed-variable PINN approaches for solving elastodynamics and elastostatics problems in both 2D and 3D. The table highlights key differences regarding the maximum order of time and spatial derivatives, hard enforcement capabilities, and the number of loss terms and weights required for each approach. Both methods require a maximum time-derivative order of 2. However, the displacement-based approach requires second-order spatial derivatives, while the mixed-variable approach only requires first-order spatial derivatives, providing a computational advantage.

Dirichlet boundary conditions (DBCs) and initial conditions (ICs) can be enforced strictly as hard constraints using both approaches. However, Neumann boundary conditions (NBCs) can be enforced only in the mixed-variable approach. The total loss terms and weights depend on the target problem and the PINN formulation. For example, in 3D elastodynamics, the displacement-based method minimizes 18 loss terms, whereas the mixed-variable approach minimizes 30, indicating more computational intensity. Similarly, the displacement-based method requires 18 distinct loss weights, while the mixed-variable approach requires 30. These loss weights are hyperparameters and are typically chosen manually. Overall, this table provides a clear overview of the requirements and capabilities of each approach, highlighting the main advantages and trade-offs for specific elastodynamics and elastostatics applications.

		Displacement-based approach	**Mixed-variable approach**
	time-derivative order (max)	2	2
	spatial-derivative order (max)	2	1
Hard enforcement	DBC	yes	yes
	NBC	no	yes
	IC	yes	yes
# loss terms	elastodynamics (3D)	18	30
	elastostatics (3D)	12	24
	elastodynamics (2D)	12	18
	elastostatics (2D)	8	14
# loss weights	elastodynamics (3D)	18	30
	elastostatics (3D)	12	24
	elastodynamics (2D)	12	18
	elastostatics (2D)	8	14

Table 6.1.: Comparison of displacement-based and mixed-variable PINN approaches for elastodynamics and elastostatics in 2D and 3D, showing the maximum order of time and spatial derivatives, hard enforcement capabilities, and the number of loss terms and weights.

6.2.4. Numerical examples

In this section, we investigate five numerical examples.

The first example is the classical Lamé elasticity problem, treated as a quasi-static problem. This example is divided into two parts. In the first part, we construct a displacement-based PINN, both with and without output transformation, to emphasize the significance of output scaling. In the second part, we compare the displacement-based PINN (with output transformation) to the mixed-variable approach, analyzing differences in computational cost and accuracy. As we will demonstrate, the mixed-variable formulation offers several advantages. Hence, the following examples deploy PINNs using the mixed-variable formulation.

The second example focuses on an inverse problem using PINNs, where we identify Young's modulus of the Lamé elasticity problem. We also emphasize the importance of a "good" initial guess to reduce the computational time needed to train the model.

In the third example, we increase the problem complexity by applying PINNs to solve a 3D elastostatics problem involving a hollow sphere subjected to internal pressure.

The fourth example shifts the focus to time-dependent problems. We predict displacement and stress fields for a 2D block under time-dependent loading, including a body force. This example demonstrates that the mixed-variable formulation facilitates the enforcement of initial conditions.

Finally, in the last example, we examine a 4D problem involving a hollow sphere subjected to time-dependent internal pressure. We show that while PINNs can mitigate the curse of dimensionality, they come at the cost of increased computational resources required for training the models.

Note that training points are generated by *GMSH*, since it has strong mesh generation and visualization capabilities and provides boundary normals at arbitrary query points [60]. When not explicitly specified, 2D examples are modeled using the plane strain condition, and body forces are neglected.

Moreover, unless otherwise specified, all numerical examples have the following common settings. Networks are initialized using the *Glorot uniform* initializer, and the tanh function is chosen as activation function. Models are first trained using a stochastic gradient descent optimizer, so-called *Adam* [91], with a learning rate of $lr = 0.001$, for $2{,}000$ epochs, and then we switch to the limited memory BFGS algorithm including box constraints (*L-BFGS-B*) [195] until one of the stopping criteria is met [36, 105]. We refer to the *DeepXDE*[101] documentation for default *L-BFGS-B* options.

As a common error metric, we report the vector-based relative L_2 errors for displacement $E^{\boldsymbol{u}}_{L_2}$ and stress fields $E^{\boldsymbol{\sigma}}_{L_2}$ as follows

$$
\begin{aligned}
E^{\boldsymbol{u}}_{L_2} &= \frac{\sqrt{\sum_i \sum_{j=1}^{N_{\text{test}}} \left(\tilde{u}_i(\boldsymbol{x}^j) - u_i(\boldsymbol{x}^j)\right)^2}}{\sqrt{\sum_i \sum_{j=1}^{N_{\text{test}}} \left(u_i(\boldsymbol{x}^j)\right)^2}} \quad \text{for } i = (x, y, z), \\
E^{\boldsymbol{\sigma}}_{L_2} &= \frac{\sqrt{\sum_k \sum_{j=1}^{N_{\text{test}}} \left(\tilde{\sigma}_k(\boldsymbol{x}^j) - \sigma_k(\boldsymbol{x}^j)\right)^2}}{\sqrt{\sum_k \sum_{j=1}^{N_{\text{test}}} \left(\sigma_k(\boldsymbol{x}^j)\right)^2}} \quad \text{for } k = (xx, yy, yy, xy, yz, xz).
\end{aligned}
\tag{6.77}
$$

In the case of 2D, $i = (x, y)$ and $k = (xx, yy, xy)$. Here, $\tilde{u}_i$ and $\tilde{\sigma}_i$ represent the PINN solutions, while u_i and σ_i denote the reference solutions, which are obtained either analytically

or numerically. Additionally, the index $j = (1, \ldots, N_{\text{test}})$ iterates over the test points generated using structured meshes.

6.2.4.1. Lamé problem of elasticity

The first example involves the well-known Lamé problem of a cylinder subjected to internal pressure p (see Fig. 6.14a). Due to the axisymmetric geometry and uniformly applied internal pressure on the inner boundary, only a quarter of the annulus needs to be modeled. The analytical solution for the stress and displacement fields can be derived in polar coordinates $\{r, \alpha\}$ as given by [164]:

$$\begin{aligned} \sigma_{rr} &= \frac{R_i^2 p}{R_o^2 - R_i^2}\left(1 - \frac{R_o^2}{r^2}\right), \\ \sigma_{\alpha\alpha} &= \frac{R_i^2 p}{R_o^2 - R_i^2}\left(1 + \frac{R_o^2}{r^2}\right), \\ \sigma_{r\alpha} &= 0, \\ u_r &= \frac{(1+\nu) p R_i^2 R_o^2}{E(R_o^2 - R_i^2)}\left(1 + \frac{r}{R_o^2}\right). \end{aligned} \tag{6.78}$$

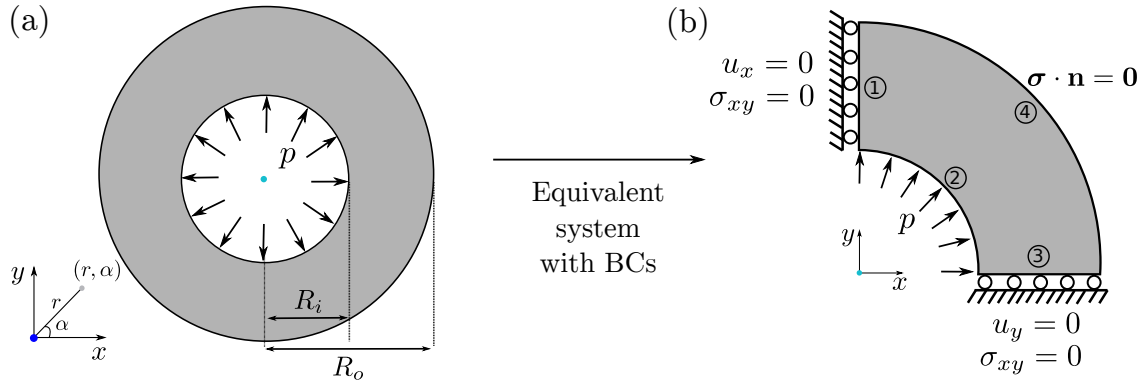


Figure 6.14.: The Lam'e elasticity problem. (a) The full cylinder under internal pressure, and (b) the equivalent reduced model due to axisymmetry in geometry and loading, along with boundary conditions.

In Eq. 6.78, R_i and R_o are the inner and outer radii of the annulus, p is the internal pressure applied to the inner radius, E denotes Young's modulus, and ν represents Poisson's ratio. In this specific setup, values are set to $R_i = 1$, $R_o = 2$, $p = 1$, $E = 2000$, and $\nu = 0.3$ It is important to note that our formulation uses Cartesian coordinates; therefore, a polar transformation is performed to compare results.

In the first case, we deploy a displacement-based PINN (see Section 6.2.1.2) both with and without output transformation to show the effect of the output transformation. For a 2D quasi-static problem, the constructed PINN in the displacement-based approach maps the spatial coordinates x, y to $\mathcal{N}_{u_x}, \mathcal{N}_{u_y}$. The architecture is a fully connected neural network of 3 hidden layers of 50 neurons. We train the networks using 330 training points, of which 262 are located within the domain, and the remaining 68 points are on the boundary.

For the model with output transformation, the following transformation is applied

$$\tilde{u}_x = \frac{x}{E}\mathcal{N}_{u_x}, \quad \tilde{u}_y = \frac{y}{E}\mathcal{N}_{u_y}. \tag{6.79}$$

As shown in Fig. 6.14, the constructed output transformation fulfills the displacement boundary conditions at $x = 0$ and $y = 0$. Also, the displacement outputs are scaled by $1/E$ to ease the optimization process (see Section 6.2.2.3).

	MSE (%)						
	PDE	NBC	DBC	training time (s)	prediction time (s)	$E^{\boldsymbol{u}}_{L_2}$ (%)	$E^{\boldsymbol{\sigma}}_{L_2}$ (%)
No output transformation	0.4	1.8	0.2	15.2	0.009	6,100	84
With output transformation	4.1×10^{-5}	6.3×10^{-5}	-	25.3	0.013	0.5	0.2

Table 6.2.: Comparison of PINN results with and without the defined output transformation for the Lamé elasticity problem.

Fig. 6.15 compares results with and without the defined output transformation. For both cases, normalized stress comparisons from the analytical solution and predicted values are presented in Figs. 6.15(a,d), normalized displacement comparisons are shown in Figs. 6.15(b,e), and MSE evolutions are displayed in Figs. 6.15(c,f).

Without output transformation, the predicted stress and displacement results deviate significantly from the analytical solutions, even though the losses decrease to the order of 10^{-1} from 10^5. Relative L_2 errors for displacements and stresses, detailed in Table 6.2, further confirm that the results without transformation poorly match the analytical solution.

In contrast, with output transformation, the results align closely with the analytical solutions, as shown in Figs. 6.15(d,e). This alignment is also supported by the relative L_2 errors in Table 6.2.

Training time for the model without output transformation is $15.2\,\mathrm{s}$, while training with output transformation takes slightly longer at $25.3\,\mathrm{s}$, primarily due to the *L-BFGS-B* optimizer. As illustrated in Figs 6.15(c,f), after $2{,}000$ epochs, we switch to *L-BFGS-B* from *Adam*. Since *L-BFGS-B* has additional stopping criteria beyond the maximum number of epochs, it requires more epochs to converge in the case of output transformation, resulting in a longer training time.

To conclude, output transformation improves the quality of the results by easing the optimization process as explained in Section 6.2.2.3.

In the second case, we compare the displacement-based PINN (with output transformation) against the mixed-variable approach. Compared to the displacement-based PINN, the PINN architecture for mixed-variable approach involves both displacement and stress fields as network output. Consequently, the constructed PINN for the mixed-variable approach maps the spatial coordinates x, y to $\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{\sigma_{xx}}, \mathcal{N}_{\sigma_{yy}}, \mathcal{N}_{\sigma_{xy}}$, yielding a total of 5 outputs. Since the stress field is included as a network output, traction BCs can also be enforced as hard constraints. Therefore, the following output transformation is applied to enforce displacement and traction BCs as hard

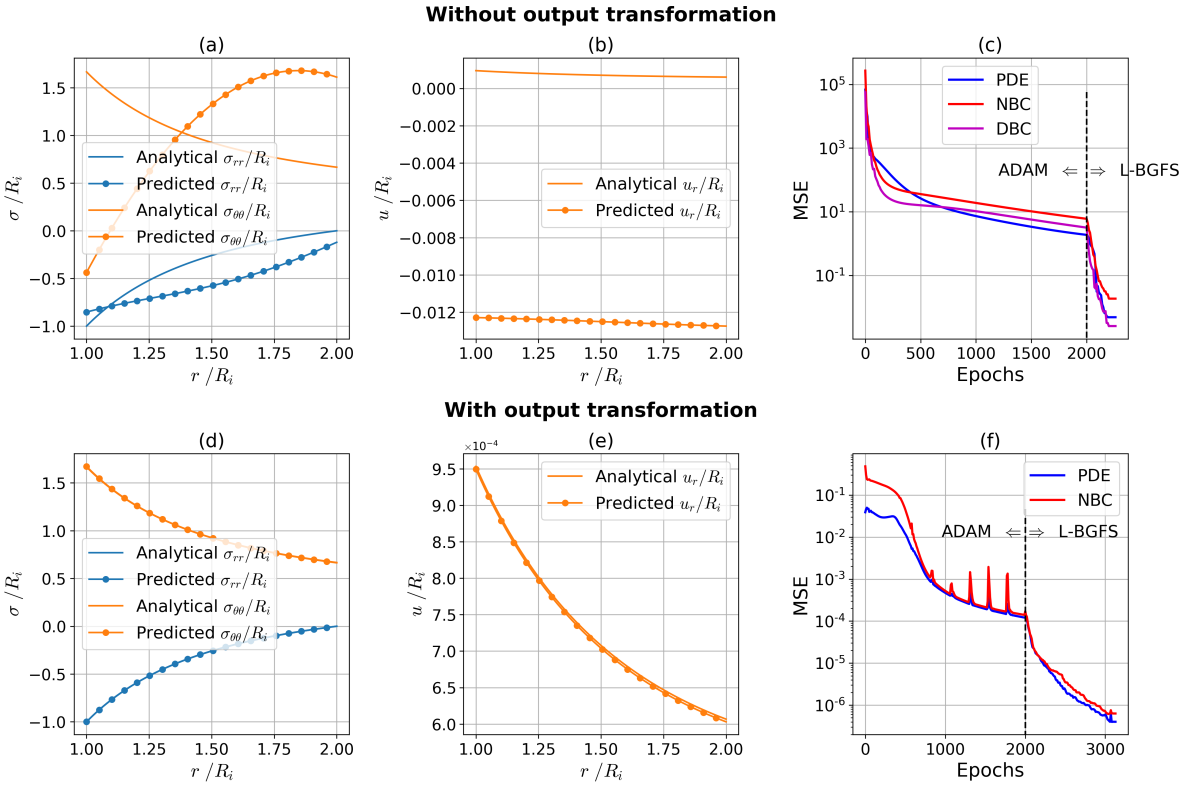


Figure 6.15.: Results for the Lamé problem of elasticity with and without output transformation. (a,d) Comparison of normalized stresses obtained from the analytical solution and the predicted values, (b,e) comparison of normalized displacements, and (c,f) evolution of the MSE for the cumulative PDE loss and NBC loss.

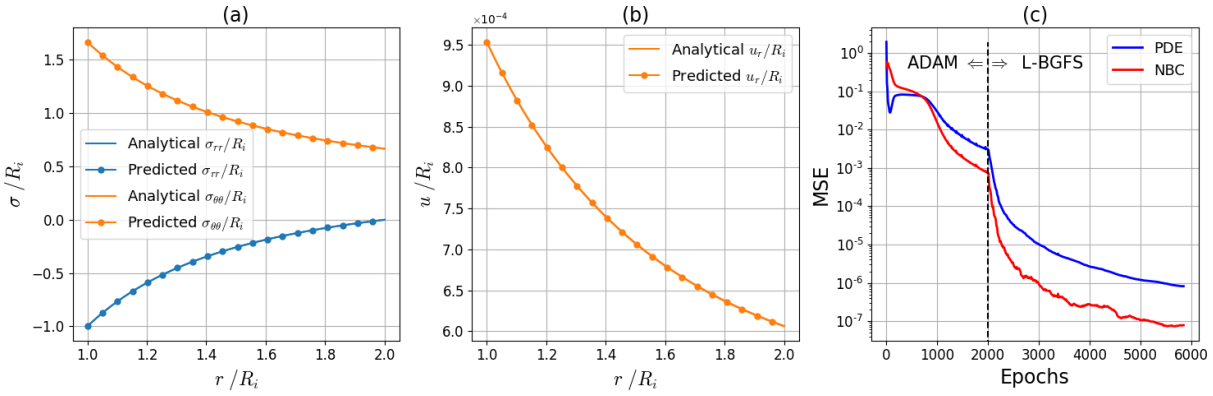


Figure 6.16.: Results of the Lamé problem of linear elasticity for the mixed-variable formulation. (a) Comparison of normalized stresses obtained from the analytical solution and the predicted values, (b) comparison of normalized displacements, and (c) evolution of the MSE for the cumulative PDE loss and NBC loss.

constraints on the edges numbered 1 and 3:

$$\tilde{u}_x = \frac{x}{E}\mathcal{N}_{u_x}, \quad \tilde{u}_y = \frac{y}{E}\mathcal{N}_{u_y}, \quad \tilde{\sigma}_{xy} = xy\mathcal{N}_{\sigma_{xy}}. \tag{6.80}$$

Similarly, the constructed output transformation fulfills the displacement boundary conditions at $x = 0$ and $y = 0$ (Fig. 6.14). Also, the displacement outputs are scaled by $1/E$. Traction boundary conditions are enforced as hard constraints for edges numbered 1 and 3, which lets us represent zero shear stresses there. Since traction boundary conditions on edges 2 and 4 contain a coupling of normal and shear stresses, we enforce them as soft constraints.

Fig. 6.16 shows the results of the Lamé problem of linear elasticity for the mixed-variable formulation. In Fig. 6.16a, a comparison of the normalized stress and displacement solutions in radial direction is given. Relative L_2 errors for displacements and stresses are calculated as 0.082% and 0.084%, respectively. These results show a significant improvement in accuracy compared to the displacement-based approach with output transformation, as provided in Table 6.2.

Similarly, switching to the *L-BFGS-B* optimizer after *Adam* enhances convergence rates, leading to further reductions in PDE and NBC losses. The cumulative MSE for PDE loss reaches about 8.17e-5%, while the average MSE for NBC loss is around 7.85e-6%. Training time is $20.41\,\mathrm{s}$, and prediction takes $0.013\,\mathrm{s}$. Although the mixed-variable formulation includes more PDE loss terms, it achieves greater efficiency in both accuracy and computational time than the displacement-based approach. Additionally, for more complex problems, such as 3D time-dependent cases, this efficiency gap between the mixed-variable and displacement-based formulations may become even more substantial. Thus, in the following examples, we deploy PINNs with output transformation in the mixed-variable formulation.

6.2.4.2. Inverse problem: Identifying material parameters

In this example, we utilize PINNs in the mixed-variable approach to solve an inverse problem for identifying the material parameters in the Lamé elasticity problem. The target parameter to be identified is chosen as Young's modulus.

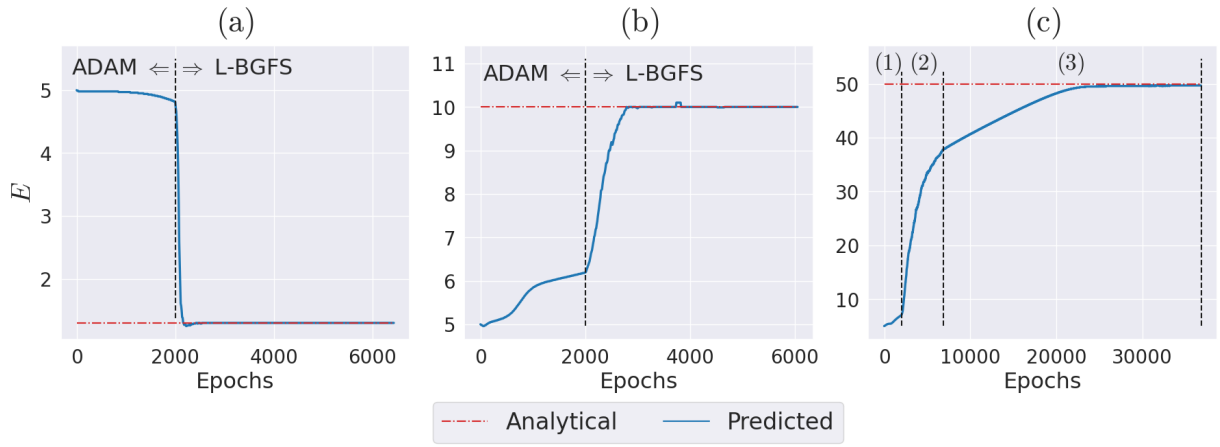


Figure 6.17.: Identification of Young's modulus for the Lamé problem of elasticity.

As outlined in Section 5.1.1, there are three main approaches for solving inverse problems using PINNs. Assuming that Young's modulus is a constant variable, we employ the third approach (see Section 5.1.1.3) to identify the unknown material parameter by incorporating it into the set of network trainable parameters, $\boldsymbol{\theta}$, as follows:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \quad \text{where,} \quad \boldsymbol{\theta} = (E, \boldsymbol{\theta}). \tag{6.81}$$

Indeed, solving inverse problems with PINNs necessitate additional measurement data. To achieve this, 100 points are randomly selected based on the previous forward model (see Fig.6.16). At these randomly selected points, displacements in the x-direction and stresses in the yy-direction are computed and incorporated into the additional data loss components (Eq. 6.67).

Fig. 6.17 illustrates the convergence behavior of the identified Young's modulus, denoted as $\tilde{E}$, compared to the distinct actual Young's modulus values E through epochs. The initial estimate for Young's modulus, represented as $\tilde{E}_o$, is set to 5 for all cases. As shown in Figs. 6.17a and 6.17b, convergence is achieved around 6000 epochs for both inverse models. In other words, the identified and actual Young's modulus values closely match, as also confirmed in Table 6.3. The relative absolute error, denoted as $E_{\text{rel,abs}}$, measures the difference between the identified and actual values, yielding errors of 0.015% and 0% for $E = 1.33$ and $E = 10$, respectively. It is important to note that these predictions were obtained with a favorable initial guess of $\tilde{E}_o = 5$.

Additionally, there is a noticeable increase in the convergence rate after $2{,}000$ epochs, which can be explained by switching to the *L-BFGS-B* optimizer. As discussed in Section 5.3.6, we employ *Adam* for the first $2{,}000$ epochs to mitigate the risk of getting trapped in local minima.

For the case of $E = 50$, which is considerably different from the initial guess, the predicted Young's modulus still converges well, with a value of $\tilde{E} = 50.0397$ and a relative absolute error of $E_{\text{rel,abs}} = 0.079\%$. However, as seen in Fig. 6.17c, this case requires more epochs for convergence compared to the others. Initially, (1) the model is trained using *Adam* for $2{,}000$ epochs, and then (2) is switched to the *L-BFGS-B* optimizer until a stopping criterion is met, which occurs at $6{,}859$ epochs, as indicated by the second dashed line in Fig. 6.17c. At this point, the predicted Young's modulus is still not sufficiently accurate; thus, the pretrained model is further trained using *Adam* for an additional $30{,}000$ epochs. As a result, the identified Young's modulus aligns well with the

	Actual $\tilde{E}$	Identified $\tilde{E}_f$	$E_{\text{rel,abs}}(\%)$	Training time (s)	# epochs
Fig 6.17(a)	1.33	1.3298	0.015	22.68	5,768
Fig 6.17(b)	10	10.0000	0	26.45	6,016
Fig 6.17(c)	50	50.0397	0.079	83.33	36,117

Table 6.3.: Comparison of relative errors, training time and number of epochs for the inverse Lamé problem of elasticity problem used to identify Young's modulus. The relative absolute error is defined as $E^*_{\text{rel,abs}} = \text{abs}\left((\tilde{*} - *)/*\right)$.

analytical value. Since, in the last case, the initial guess is quite different from the actual Young's modulus, more training steps are required.

6.2.4.3. 3D hollow sphere subjected to internal pressure

In the second example, we consider a hollow sphere subjected to internal pressure as depicted in Fig. 6.18. Due to the axisymmetric geometry and the uniformly applied internal pressure on the inner boundary, only one-eighth of the hollow sphere needs to be modeled. The analytical solution is expressed in spherical coordinates $\{r, \phi, \theta\}$ as given in [153]

$$\begin{aligned} \sigma_{rr} &= -\frac{p}{\left(\frac{R_\text{o}}{R_\text{i}}\right)^3 - 1}\left[\left(\frac{R_\text{o}}{r}\right)^3 - 1\right], \\ \sigma_{\phi\phi} = \sigma_{\theta\theta} &= \frac{p}{\left(\frac{R_\text{o}}{R_\text{i}}\right)^3 - 1}\left[\frac{1}{2}\left(\frac{R_\text{o}}{r}\right)^3 + 1\right], \\ u_r &= \frac{r}{E}\left[(1-\nu)\sigma_{\theta\theta} - \nu\sigma_{rr}\right]. \end{aligned} \tag{6.82}$$

Similarly, in this specific setup, the parameter values are set as $R_i = 1$, $R_o = 2$, $p = 1$, $E = 2000$, and $\nu = 0.3$.

As this is a 3D quasi-static problem, the constructed PINN (see Section 6.2.2.2) in the mixed-variable approach maps the spatial coordinates x, y, z to $\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_z}, \mathcal{N}_{\sigma_{xx}}, \mathcal{N}_{\sigma_{yy}}$, $\mathcal{N}_{\sigma_{zz}}, \mathcal{N}_{\sigma_{xy}}, \mathcal{N}_{\sigma_{yz}}, \mathcal{N}_{\sigma_{xz}}$, yielding a total of 9 outputs. The architecture is a fully connected neural network consisting of 5 hidden layers of 50 neurons each. We train the networks using 1000 training points, of which 356 are located within the domain, and the remaining 644 points are on the boundary. Similarly, the PINN architecture in the mixed-variable form enables us to enforce both Dirichlet BCs and Neumann BCs as hard constraints. Consequently, the following output transformation is applied

$$\begin{aligned} \tilde{u}_x = \frac{x}{E}\mathcal{N}_{u_x}, \quad \tilde{u}_y = \frac{y}{E}\mathcal{N}_{u_y}, \quad \tilde{u}_z = \frac{z}{E}\mathcal{N}_{u_z}, \\ \tilde{\sigma}_{xy} = xy\mathcal{N}_{\sigma_{xy}}, \quad \tilde{\sigma}_{yz} = yz\mathcal{N}_{\sigma_{yz}}, \quad \tilde{\sigma}_{xz} = xz\mathcal{N}_{\sigma_{xz}}. \end{aligned} \tag{6.83}$$

As depicted in Fig. 6.18c, the defined output transformation fulfills the Dirichlet BCs at $x = 0$, $y = 0$ and $z = 0$. For traction BCs, only those on flat surfaces are enforced as hard constraints, as

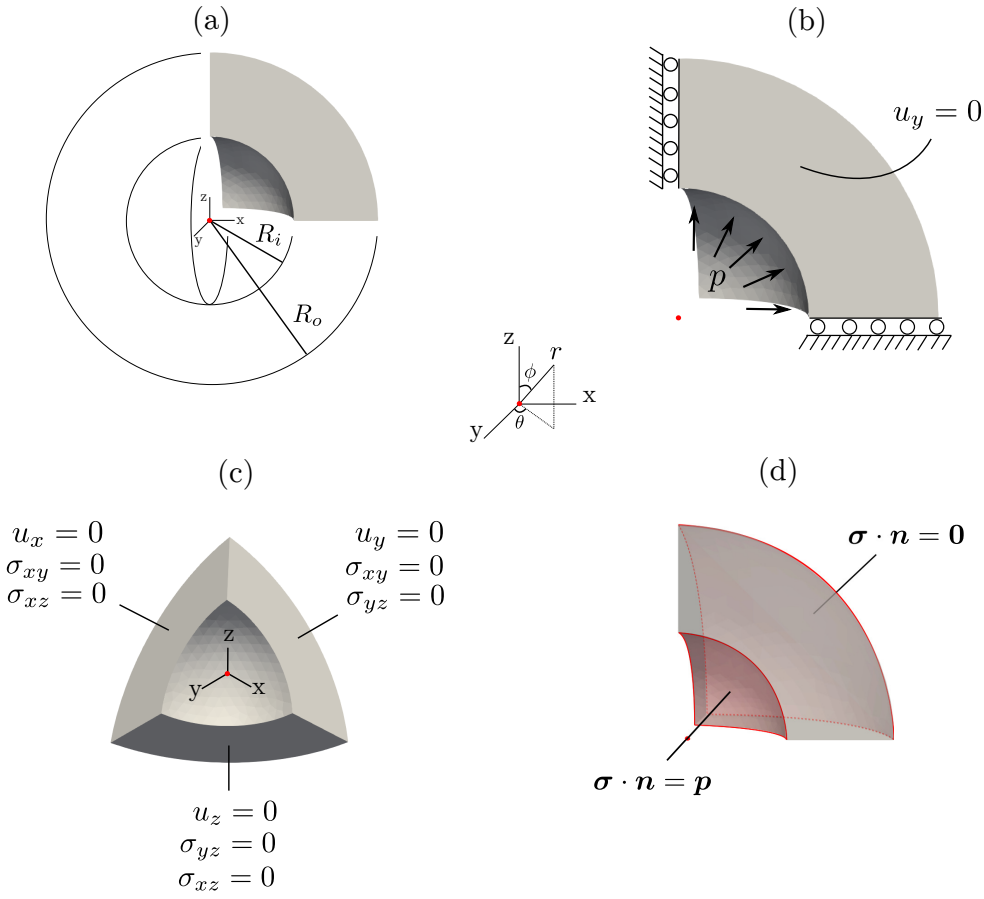


Figure 6.18.: A hollow sphere subjected to internal pressure. (a) The complete geometry, including radial dimensions. (b) One-eighth of the hollow sphere under internal pressure and Dirichlet boundary conditions due to the axisymmetric nature of both the geometry and the loading. (c) Dirichlet and Neumann boundary conditions to be enforced as hard constraints. (d) Neumann boundary conditions to be enforced as soft constraints.

the stress components are uncoupled. For further understanding, let us consider the flat surface at $y = 0$, where the Dirichlet BC $u_y = 0$ is applied. Recall the Neumann BCs on this specific plane at $y = 0$ (Eqs. 2.60-2.62) as

$$\sigma_{xy} - \hat{t}_x(y = 0) = 0 \tag{6.84}$$

$$\sigma_{yy} - \hat{t}_y(y = 0) = 0 \tag{6.85}$$

$$\sigma_{yz} - \hat{t}_z(y = 0) = 0 \tag{6.86}$$

Since there is a defined Dirichlet BC in y-direction, we cannot enforce σ_{yy} as a hard constraint. On the other hand, $\hat{t}_x = 0$ and $\hat{t}_z = 0$ at $y = 0$, since the surface is subjected to zero traction in the x- and and z-directions. Consequently, these zero tractions can be enforced as hard constraints by multiplying the network outputs $\mathcal{N}_{\sigma_{xy}}$ and $\mathcal{N}_{\sigma_{yz}}$ with y, as provided in Eq. 6.83. Similarly, traction components in other planes, i.e, $x = 0$ and $z = 0$, can be obtained using the same approach. Note that traction BCs on the inner and outer curved surfaces of the hollow sphere are enforced as soft constraints as depicted in Fig. 6.18d, as normal and shear stress components are coupled.

The results for the hollow sphere subjected to internal pressure are presented in Fig. 6.19. These results include the displacement field in the x-, y-, and z-directions (Cartesian coordinates); the normal stress components in the xx-, yy-, and zz-directions (Cartesian coordinates); the shear stress components in the xy-, yz-, and xz-directions (Cartesian coordinates); the normal stress components in the rr-, $\theta\theta$-, and $\phi\phi$-directions (polar coordinates); and the shear stress components in the $r\theta$-, $\theta\phi$-, and $r\phi$-directions (polar coordinates).

From the displacement field, we observe that the results exhibit perfect symmetry, with zero displacements at the surfaces where Dirichlet BCs are applied as hard constraints. Similarly, the normal stress components range from 7.15e-1 to -1 across all directions. Since shear stress components are enforced as hard constraints, the corresponding surfaces exhibit strictly zero shear stress. Additionally, the normal stresses in the $\theta\theta$- and $\phi\phi$-directions (polar coordinates) show almost identical distributions, as expected. The normal stress component in the rr-direction reaches a value of -1 on the inner curvature, which matches the magnitude of the applied pressure $p = 1$.

Furthermore, a comparison between the predicted values and analytical solutions for the hollow sphere under internal pressure is shown in Fig. 6.20. We observe a strong alignment between the predicted and analytical values across all stress and displacement components. This agreement is further validated by the relative L_2 errors, with $E^{\boldsymbol{u}}_{L_2} = 0.1\%$ and $E^{\boldsymbol{\sigma}}_{L_2} = 0.17\%$ for the displacement and stress fields, respectively. Additionally, the total training time is $188.51\,\mathrm{s}$.

6.2.4.4. 2D block under time-dependent loading

In the third example, we examine a 2D block subjected to time-dependent loading, as depicted in Fig. 6.21. The 2D block is constrained along its left edge in the x-direction and its bottom edge in the y-direction. Additionally, a time-dependent body force $\hat{\boldsymbol{b}} = [\hat{b}_x, \hat{b}_y]$ is applied throughout the domain, defined as

$$\hat{b}_x = \frac{\lambda x}{4\mu(\lambda + \mu)} 4\pi^2 \sin(2\pi t),$$

$$\hat{b}_y = \frac{(\lambda + 2\mu)y}{4\mu(\lambda + \mu)} 4\pi^2 \sin(2\pi t).$$

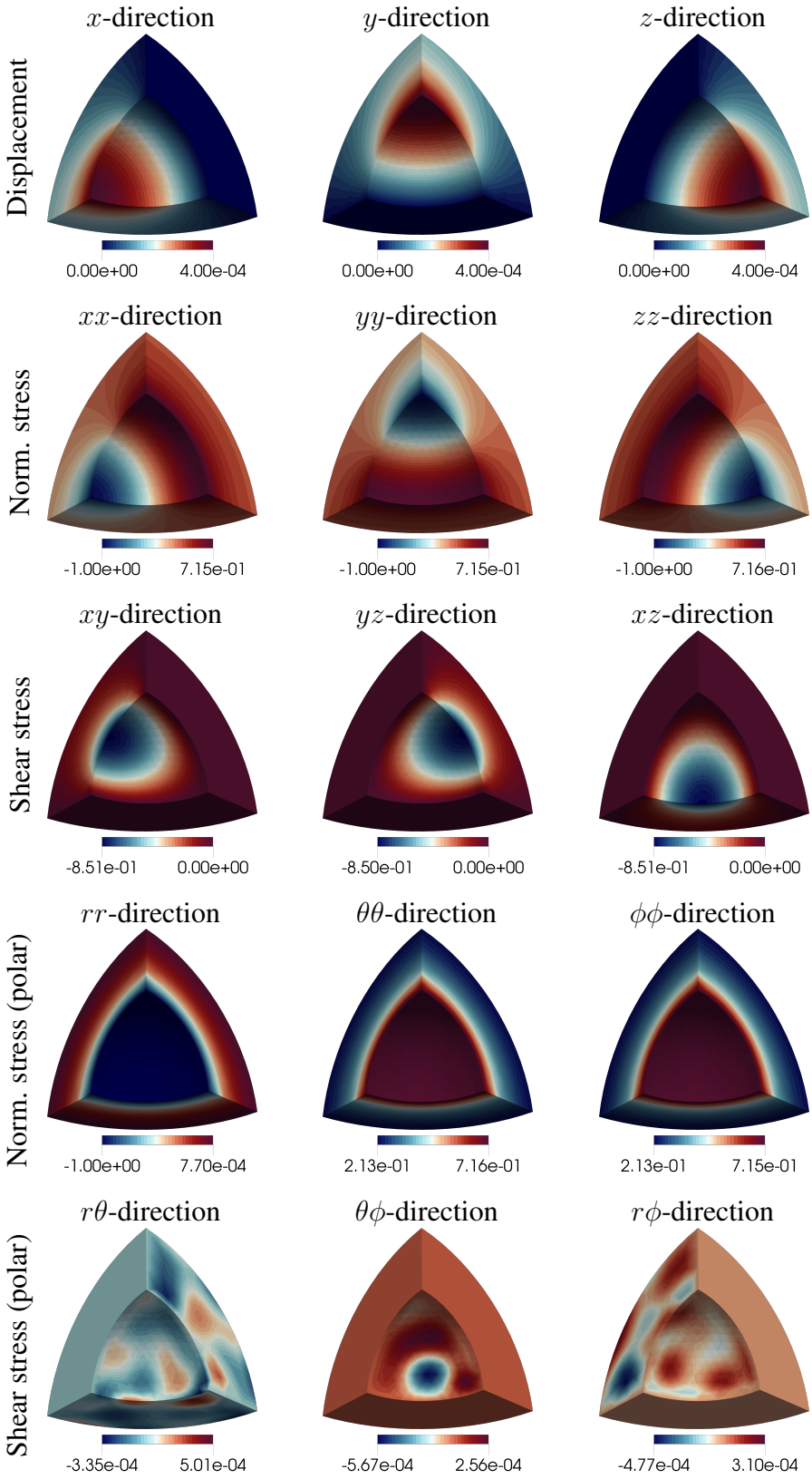


Figure 6.19.: Results for the hollow sphere subjected to internal pressure.

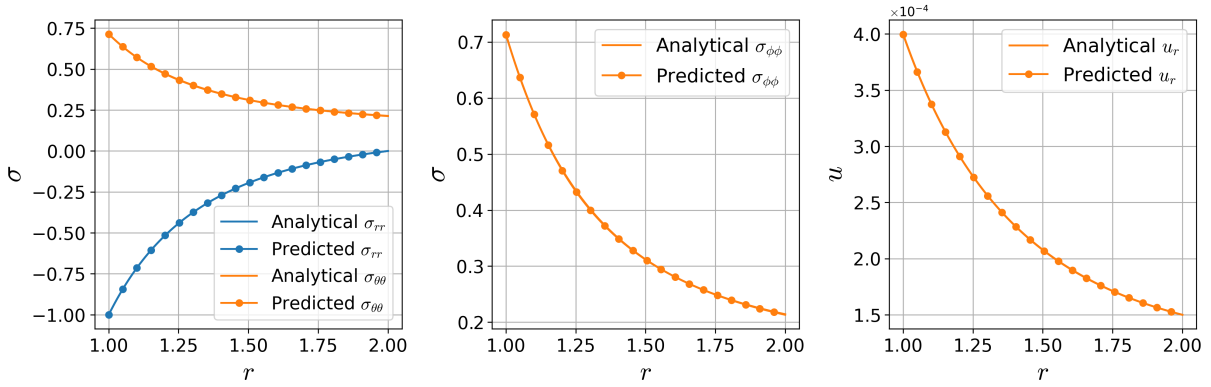


Figure 6.20.: Comparison of the predicted values and analytical solutions for the hollow sphere subjected to internal pressure. (a) Comparison of stresses in the rr- and $\theta\theta$-directions, (b) comparison of stresses in the $\phi\phi$-direction, and (c) comparison of the radial displacements u_r.

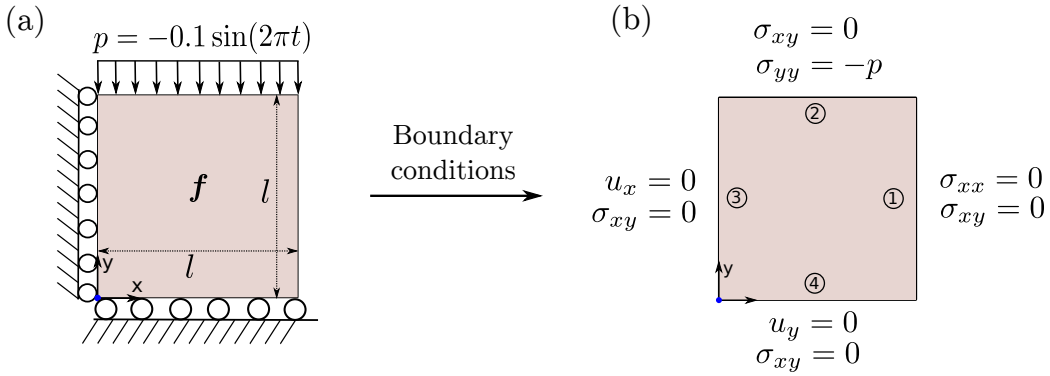


Figure 6.21.: A 2D block under time-dependent loading. (a) geometry, supports, loading, (b) all relevant boundary conditions.

Since this is a time-dependent problem, initial conditions at $t = 0$ are specified as follows:

$$u_x(t=0) = 0, \tag{6.87}$$

$$u_y(t=0) = 0, \tag{6.88}$$

$$\dot{u}_x(t=0) = -p\frac{\lambda}{4\mu(\lambda+\mu)}x, \tag{6.89}$$

$$\dot{u}_y(t=0) = p\frac{\lambda+2\mu}{4\mu(\lambda+\mu)}y. \tag{6.90}$$

The analytical solution can be derived as

$$u_x = -p\frac{\lambda}{4\mu(\lambda+\mu)}xt, \quad u_y = p\frac{\lambda+2\mu}{4\mu(\lambda+\mu)}yt,$$

$$\sigma_{xx} = 0, \quad \sigma_{yy} = -p, \quad \sigma_{xy} = 0.$$

Here, λ denotes the Lamé's first parameter and μ is the shear modulus. We set $\lambda = 1.5$ and $\mu = 0.5$ for our specific setup. The length of the 2D block is set to $l = 1$. Additionally, the time-dependent load is applied for $1\,\mathrm{s}$, thus $t = [0, 1]$.

In this example, we deploy a PINN in the mixed-variable approach for the time-dependent problem (see Section 6.2.2.1). Since this is a 2D problem, the constructed PINN maps the spatial coordinates x, y, and time t to $\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_{xx}}, \mathcal{N}_{u_{yy}}, \mathcal{N}_{u_{xy}}$. The architecture is a fully connected neural network consisting of 5 hidden layers of 50 neurons each. We train the network using 1000 training points, of which 512 are located within the domain, 388 points are on the boundary, and 100 points are for initial conditions.

We train the network using 1000 training points, of which 512 are located within the domain, and the remaining 488 points are on the boundary. Additionally, the following output transformation is applied to enforce Dirichlet and Neumann BCs (Fig 6.21b) including ICs due the displacement field (Eqs. 6.87-6.88) as

$$\tilde{u}_x = xt\mathcal{N}_{u_x}, \quad \tilde{u}_y = yt\mathcal{N}_{u_x},$$

$$\tilde{\sigma}_{xx} = (l - x)\mathcal{N}_{u_{xx}}, \quad \tilde{\sigma}_{yy} = -\sin(2\pi t) + (l - y)\mathcal{N}_{u_{yy}}, \quad \tilde{\sigma}_{xy} = xy(l - x)(l - y)\mathcal{N}_{u_{xy}}.$$

This constructed output transformation strictly fulfills the Dirichlet BCs and ICs for displacements, namely $u_x(x = 0, t) = 0$ and $u_x(x, t = 0) = 0$, as well as $u_y(y = 0, t) = 0$ and $u_y(y, t = 0) = 0$. Similarly, all traction BCs are enforced as hard constraints. Since the velocity field is not included as a network output, ICs for the velocity fields (Eqs. 6.89-6.90) are imposed as soft constraints.

The results for the 2D block under time-dependent loading are shown in Fig. 6.22, with the time step chosen as $t = 0.2\,\mathrm{s}$. These results include the predicted values, analytical or reference solutions, and the point-wise absolute errors. A linear distribution is observed for both displacement components, which is consistent with the analytical solutions. The maximum point-wise absolute errors are reduced to $E^{u_x}_{abs,max} =$ 6.76e-4 and $E^{u_y}_{abs,max} =$ 4.38e-4 for the displacement components u_x and u_y, respectively. Since the Dirichlet BCs are enforced as hard constraints on the left and bottom edges, and the minimum point-wise absolute errors are strictly zero. Similarly, the normal stress in the xx-direction, σ_{xx}, and the shear stress in the xy-direction, σ_{xy}, are close to zero, consistent with the analytical solutions. On the edges where traction BCs are applied as hard constraints, the stress components are also strictly zero, as defined in the output transformation. However, the minimum point-wise absolute error for σ_{yy} is not strictly zero ($E^{u_x}_{abs,min} =$ 2.37e-8) due to round-off error. Additionally, σ_{yy} is computed as -0.0951, which aligns with the applied pressure at $t = 0.2$, given by $-\sin(2\pi \cdot 0.2) = -0.0951$.

Additionally, Fig. 6.23 illustrates comparisons between the predicted values and analytical solutions at various time steps. These comparisons are computed along a horizontal line at $y = 0.5$ across the domain. As seen in Fig. 6.23a, predicted displacements in the x-direction exhibit a linear variation and are correctly scaled according to each time step, aligning well with the analytical solutions. In contrast, the predicted displacements in the y-direction remain constant as expected. This behavior is similarly observed for the predicted normal stress in the yy-direction. Since u_{xx} and u_{xy} are zero, they are omitted from the visualization. Quantitative comparisons using relative L_2 errors further validate these alignments between predicted and analytical values, as summarized in Table 6.4. At $t = 0.5$, the applied load is zero, resulting in zero displacements and stresses. Consequently, relative errors are undefined at $t = 0.5$. Additionally, network training takes approximately $96\,\mathrm{s}$.

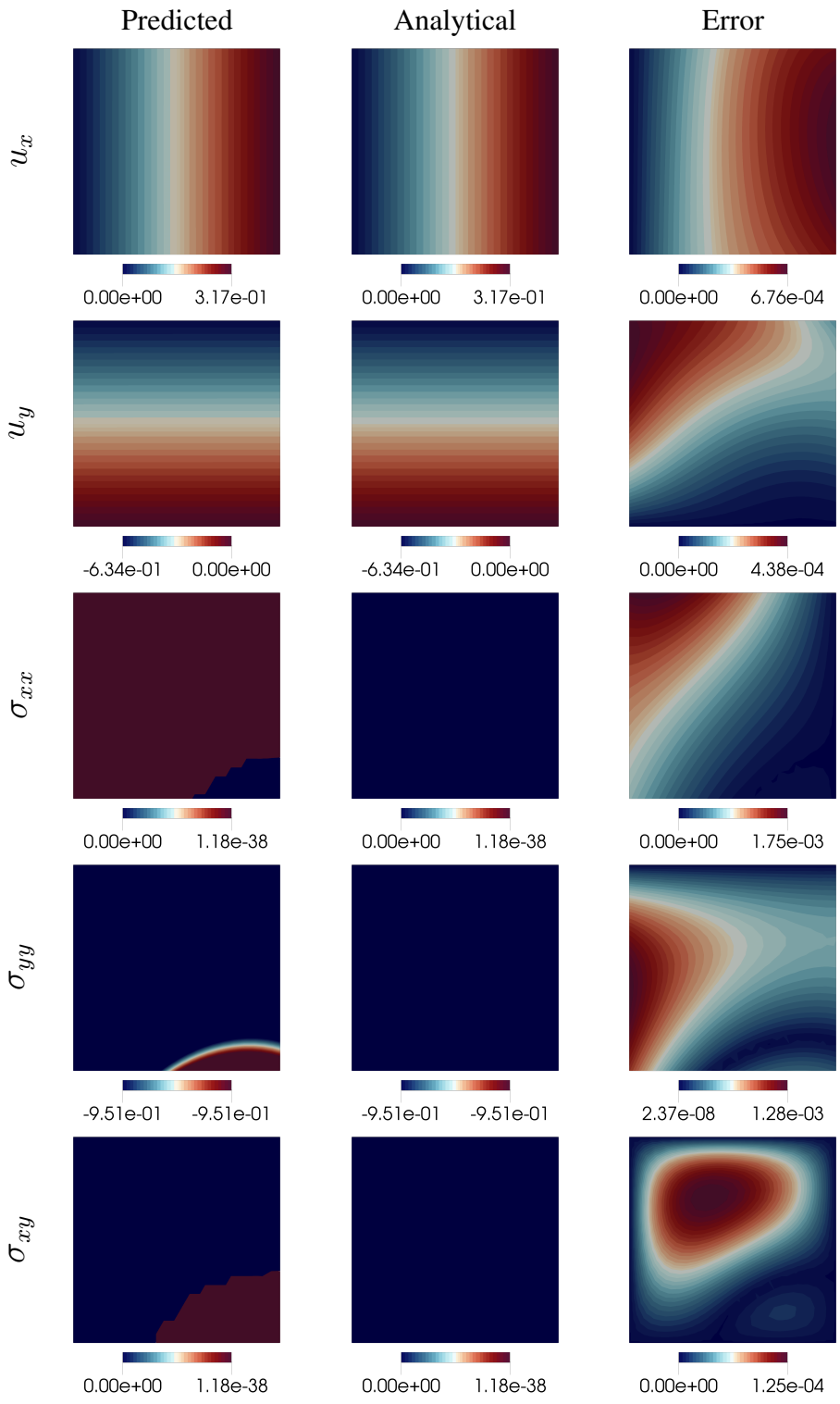


Figure 6.22.: Results for the 2D block under time-dependent loading at $t = 0.2$. Predicted and analytical solutions for the displacement (u_x, u_y) and stress ($\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$) components. Additionally, corresponding point-wise absolute errors are provided, which are defined as $E^*_{abs} = \text{abs}(\tilde{*} - *)$

	$t = 0.125$	$t = 0.25$	$t = 0.5$	$t = 0.75$	$t = 0.875$
$E^{\boldsymbol{u}}_{L_2}$ (%)	0.07	0.05	NaN	0.09	0.1
$E^{\boldsymbol{\sigma}}_{L_2}$ (%)	0.16	0.11	NaN	0.13	0.2

Table 6.4.: Relative L_2 errors for displacement ($E^{\boldsymbol{u}}_{L_2}$) and stress ($E^{\boldsymbol{\sigma}}_{L_2}$) fields at different time steps for the 2D block under time-dependent loading. These errors are computed based on Fig. 6.23.

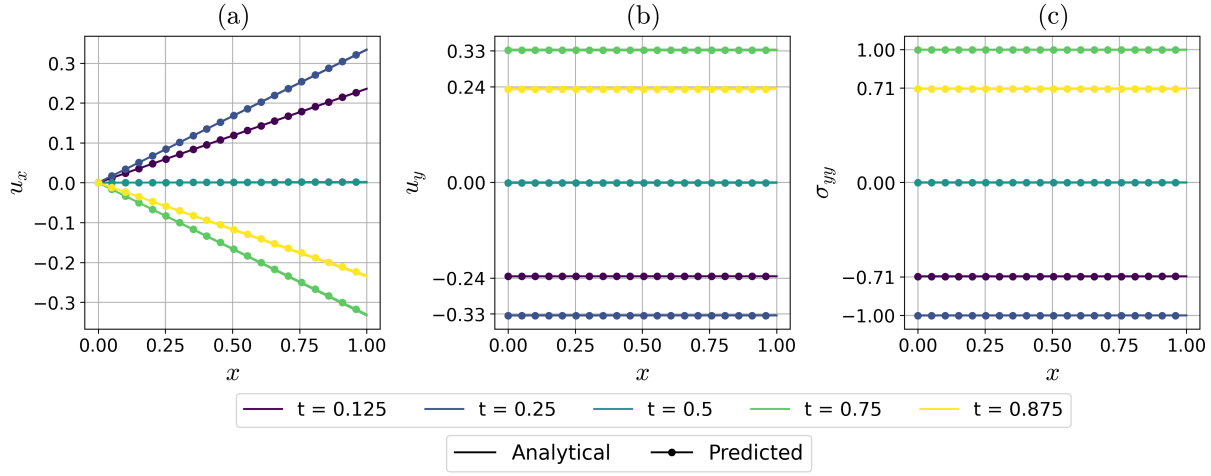


Figure 6.23.: Comparisons between the predicted values and analytical solutions of the 2D block under time-dependent loading for different time steps. (a) Comparison of the displacement component u_x, (b) comparison of the displacement component u_y, and (c) comparison of the normal stress component σ_{yy}.

6.2.4.5. 4D problem: 3D hollow sphere under time-dependent loading

In the final example for deploying PINNs to solve a problem, we revisit the hollow sphere scenario (Section 6.2.4.3), this time applying a time-dependent loading, $p = \sin(2\pi t)$. This makes the problem 4D: three spatial dimensions and time.

Under the given initial conditions

$$\boldsymbol{u}(t=0) = \mathbf{0}, \tag{6.91}$$
$$\dot{\boldsymbol{u}}(t=0) = \mathbf{0}, \tag{6.92}$$

the analytical solution can be derived as

$$\begin{aligned}
\sigma_{rr} &= -\frac{\sin(2\pi t)}{\left(\frac{R_\mathrm{o}}{R_\mathrm{i}}\right)^3 - 1}\left[\left(\frac{R_\mathrm{o}}{r}\right)^3 - 1\right], \\
\sigma_{\phi\phi} = \sigma_{\theta\theta} &= \frac{\sin(2\pi t)}{\left(\frac{R_\mathrm{o}}{R_\mathrm{i}}\right)^3 - 1}\left[\frac{1}{2}\left(\frac{R_\mathrm{o}}{r}\right)^3 + 1\right], \\
u_r &= \mathrm{sign}\Big(\sin(2\pi t)\Big)\frac{r}{E}\left[(1-\nu)\sigma_{\theta\theta} - \nu\sigma_{rr}\right].
\end{aligned} \tag{6.93}$$

Here, sign denotes the *sign* operation to determine the direction of the applied load. Similar to quasi-static case (Section 6.2.4.3), the following parameters are set $R_i = 1$, $R_o = 2$, $p = 1$, $E = 2000$, and $\nu = 0.3$. Additionally, the time-dependent load is applied for $1\,\mathrm{s}$, resulting in $t = [0, 1]$.

In this example, we deploy a PINN in the mixed-variable approach for the time-dependent problem (see Section 6.2.2.1). For a 3D problem, the constructed PINN in the mixed formulation maps the spatial coordinates x, y, z, and time t to $\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_z}, \mathcal{N}_{\sigma_{xx}}, \mathcal{N}_{\sigma_{yy}}, \mathcal{N}_{\sigma_{zz}}, \mathcal{N}_{\sigma_{xy}}, \mathcal{N}_{\sigma_{yz}}, \mathcal{N}_{\sigma_{xz}}$. The architecture is a fully connected neural network consisting of 8 hidden layers of 70 neurons each. Due to the complexity of the problem, a wider and deeper network is generated compared to the previous examples. We train the network using 1100 training points, of which 356 are located within the domain, 644 points are on the boundary, and 100 points are for initial conditions.

The following output transformation is employed to enforce the Dirichlet and Neumann BCs, including ICs

$$\begin{aligned}
\tilde{u}_x &= \frac{xt}{E}\mathcal{N}_{u_x}, \\
\tilde{u}_y &= \frac{yt}{E}\mathcal{N}_{u_y}, \\
\tilde{u}_z &= \frac{zt}{E}\mathcal{N}_{u_z}, \\
\tilde{\sigma}_{xy} = xy\mathcal{N}_{\sigma_{xy}}, &\quad \tilde{\sigma}_{yz} = yz\mathcal{N}_{\sigma_{yz}}, \quad \tilde{\sigma}_{xz} = xz\mathcal{N}_{\sigma_{xz}}.
\end{aligned} \tag{6.94}$$

The predicted solutions for the hollow sphere under time-dependent loading are shown in Fig. 6.24. The predictions include only $\tilde{u}_x$, $\tilde{\sigma}_{xx}$, $\tilde{\sigma}_{xy}$, $\tilde{\sigma}_{rr}$, and $\tilde{\sigma}_{r\theta}$. Results are presented for three specific time steps: $t = 0.25\,\mathrm{s}$, $t = 0.5\,\mathrm{s}$, and $t = 0.75\,\mathrm{s}$. At $t = 0.5\,\mathrm{s}$, the applied load is zero, resulting in nearly zero values across all predicted fields. In contrast, at $t = 0.25\,\mathrm{s}$ and $t = 0.75\,\mathrm{s}$,

	$t = 0.125$	$t = 0.25$	$t = 0.5$	$t = 0.75$	$t = 0.875$
$E^{\boldsymbol{u}}_{L_2}$ (%)	0.07	0.05	NaN	0.09	0.1
$E^{\boldsymbol{\sigma}}_{L_2}$ (%)	0.16	0.11	NaN	0.13	0.2

Table 6.5.: Relative L_2 errors for displacement ($E^{\boldsymbol{u}}_{L_2}$) and stress ($E^{\boldsymbol{\sigma}}_{L_2}$) fields at different time steps for the hollow sphere under a time-dependent loading. These errors are computed based on Fig. 6.25.

the applied loads are $p = -1$ and $p = 1$, respectively. As a result, the predicted stress field in the rr-direction ($\tilde{\sigma}_{rr}$) at the inner surface of the sphere matches the applied loading. Since the magnitudes of the applied loads at $t = 0.25\,\mathrm{s}$ and $t = 0.75\,\mathrm{s}$ are equal, the predictions exhibit almost a mirrored pattern. For example, the minimum and maximum values of $\tilde{\sigma}_{xx}$ at $t = 0.25\,\mathrm{s}$ are -9.98e-1 and 7.25e-1, respectively, while at $t = 0.75\,\mathrm{s}$, these values are -7.28e-1 and 9.98e-1, indicating a reversal in sign.

Additionally, Fig. 6.25 illustrates comparisons between the predicted values and analytical solutions at various time steps. These comparisons are computed along a horizontal line at $y = 0.5$ across the domain. We can see a good match between predicted and analytical values for all fields, which is confirmed by provided relative L_2 errors in Table 6.5. Additionally, training of the network takes approximately $2{,}070\,\mathrm{s}$, which is significantly longer compared to previous examples due to the complexity of the problem as it is a 4D problem.

Fig. 6.25 compares the predicted values and analytical solutions at different time steps. These comparisons are made along a random horizontal line in the radial direction. The predicted values strongly agree with the analytical solutions for all fields, as confirmed by the relative L_2 errors listed in Table 6.5. Additionally, the total training time for the network is approximately $2{,}070\,\mathrm{s}$, which is significantly longer than in previous examples due to the increased complexity of the 4D problem.

6.3. Use Case 3: Contact mechanics under linear elasticity

6.3.1. PINNs with mixed-variable formulation

In the context of contact mechanics under small deformation, we employ PINNs with output transformation within the framework of the mixed-variable formulation (refer to Section 2.3.2.2). As previously demonstrated, the mixed-variable PINN formulation provides several key advantages over the displacement-based approach, such as enforcing traction boundary conditions as hard constraints and avoiding higher-order derivatives. Consequently, we utilize only the mixed-variable formulation to implement PINNs to solve contact mechanics problems.

For time-dependent contact problems, using the mixed-variable formulation without additional network input parameters, an FNN maps the spatial coordinates $\boldsymbol{x}$ and time t to the approximated displacement $\tilde{\boldsymbol{u}}$ and stress $\tilde{\boldsymbol{\sigma}}$ as follows

$$\tilde{\boldsymbol{u}} := \left(\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x}; t; \boldsymbol{\theta})\right)' \quad \text{and} \quad \tilde{\boldsymbol{\sigma}} := \left(\mathcal{N}_{\boldsymbol{\sigma}}(\boldsymbol{x}; t; \boldsymbol{\theta})\right)'. \tag{6.95}$$

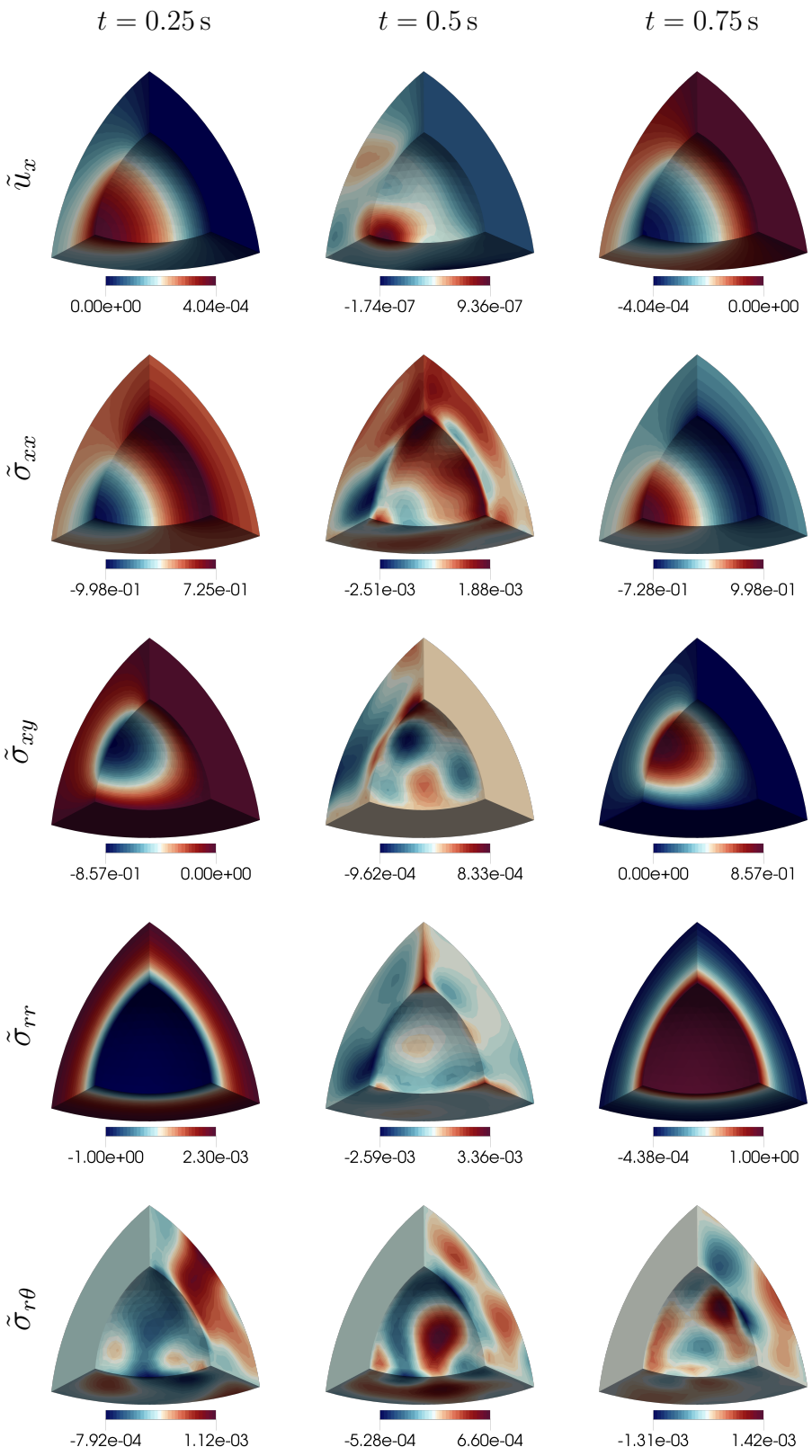


Figure 6.24.: Predicted solutions of the hollow sphere subjected to time-dependent loading for three steps: $t = 0.25\,\mathrm{s}$, $t = 0.5\,\mathrm{s}$ and $t = 0.75\,\mathrm{s}$.

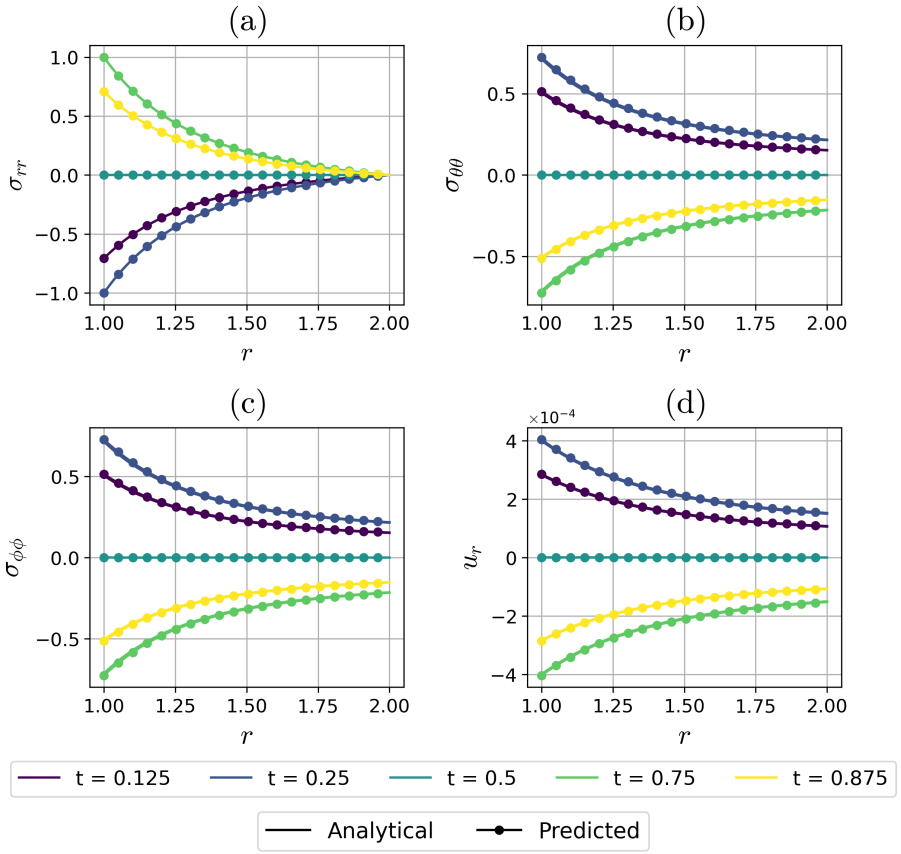


Figure 6.25.: Comparison between analytical and predicted solutions for the hollow sphere subjected to time-dependent loading at different time steps. (a) Radial stress σ_{rr}, (b) circumferential stress $\sigma_{\theta\theta}$, (c) circumferential stress $\sigma_{\phi\phi}$, and (d) radial displacement u_r.

In the quasi-static case, the neural network input involves only the spatial coordinates $\boldsymbol{x}$. Fig. 6.26 presents the PINN frameworks for solving normal contact problems in the mixed-variable form for both time-dependent and quasi-static cases.

For contact problems under small deformation and frictionless sliding assumptions, the total loss function $\mathcal{L}_{\mathrm{C}}$ is defined as

$$\mathcal{L}_{\mathrm{C}} = \mathcal{L}_{\mathrm{E}} + \mathcal{L}_{\mathrm{FSC}} + \mathcal{L}_{\mathrm{KKT}}, \tag{6.96}$$

where:

- $\mathcal{L}_{\mathrm{E}}$ represents the losses associated with linear elasticity, which have already been detailed in Eq. 6.52 and Eq. 6.67 for time-dependent and quasi-static problems, respectively (see Section 6.2.2.1 and Section 6.2.2.2 for details.).
- $\mathcal{L}_{\mathrm{FSC}}$ denotes the losses due to frictionless sliding conditions, defined as:

$$\mathcal{L}_{\mathrm{FSC}} = w^{(\mathrm{fs},\xi)}|t_\tau^\xi|_{\partial\Omega_c} + w^{(\mathrm{fs},\eta)}|t_\tau^\eta|_{\partial\Omega_c}, \tag{6.97}$$

  where $w^{(\mathrm{fs},\xi)}$ and $w^{(\mathrm{fs},\eta)}$ are the weights associated with the computed tangential traction components. This term enforces the frictionless sliding condition within the contact zone $\partial\Omega_c$. Note that for 2D contact problems, only one tangential traction loss component is introduced.

- $\mathcal{L}_{\mathrm{KKT}}$ incorporates the losses associated with the Karush-Kuhn-Tucker (KKT) conditions, ensuring the enforcement of non-penetration, admissibility of contact pressure, and complementarity. This loss term will be explained in the following section.

While the computation of the normal gap g_n remains a widely debated topic in discretization techniques, such as the FEM [178, 187], a simplified approach is sufficient here. This simplification arises from the fact that we only focus on contact interactions between an elastic body and a rigid flat surface. The normal gap g_n is consistently defined by evaluating the orthogonal projection of the elastic body onto the rigid flat surface.

6.3.2. Enforcing Karush-Kuhn-Tucker constraints

There are various methods available for enforcing inequality conditions in general. One common approach is to formulate loss functions that represent the inequalities and impose them as soft constraints with fixed loss weights [69, 70]. However, this direct method has its limitations: using large loss weights can lead to ill-conditioned optimization problems [102], while small loss weights may result in solutions that fail to satisfy the inequality constraints.

To address these challenges, authors in [102] propose using penalty and augmented Lagrangian methods, which are well-established techniques from constrained optimization. These methods construct loss formulations with adaptive loss weights, dynamically balancing the trade-off between constraint enforcement and numerical stability.

In the following, we explore three distinct methods for enforcing the Karush-Kuhn-Tucker (KKT) conditions for normal contact problems using soft constraint formulations. Each method is analyzed for its effectiveness and applicability in handling these inequality constraints.

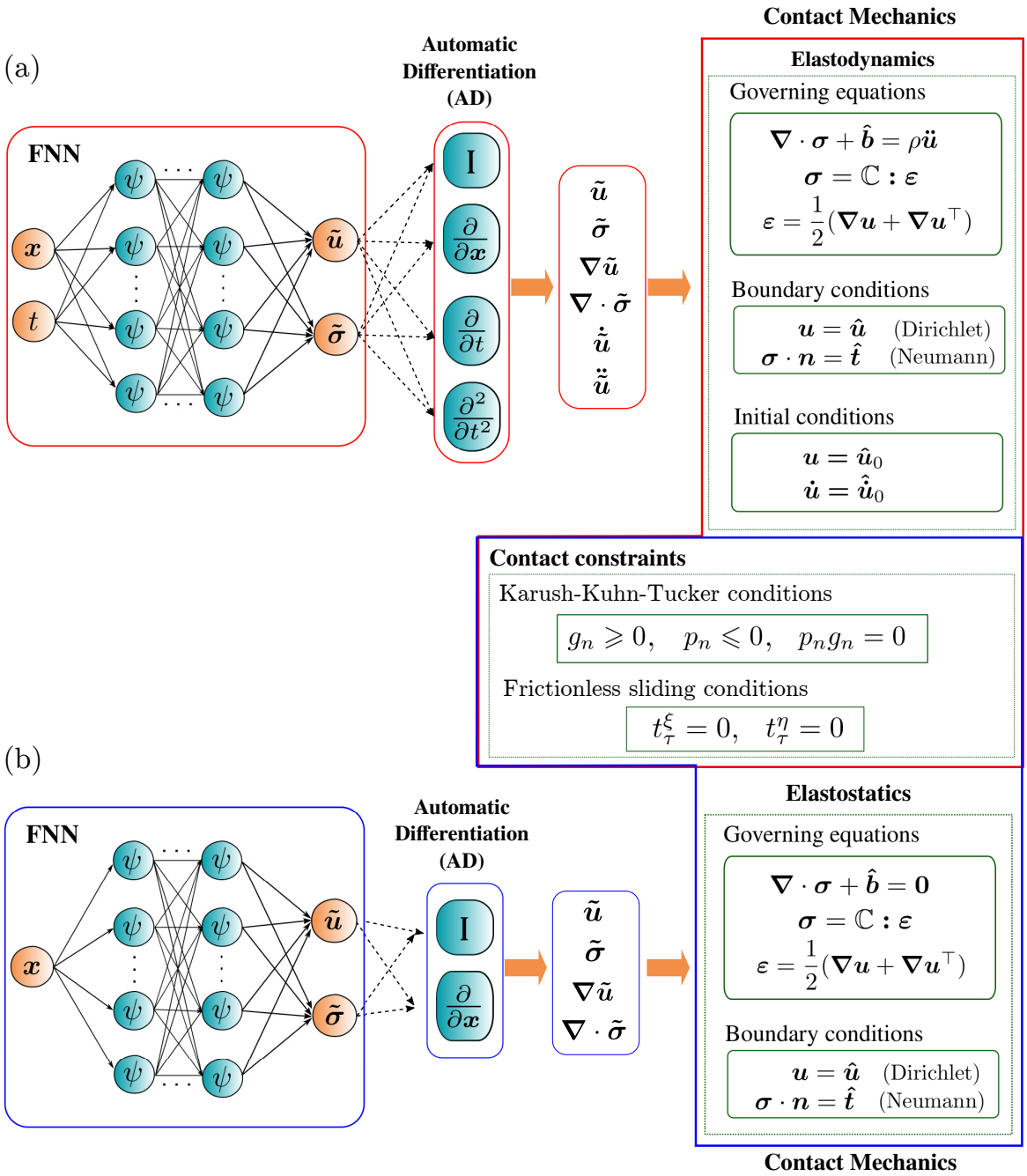


Figure 6.26.: Physics-informed neural networks in the mixed-variable form to solve normal contact problems. (a) Time-dependent, (b) quasi-static.

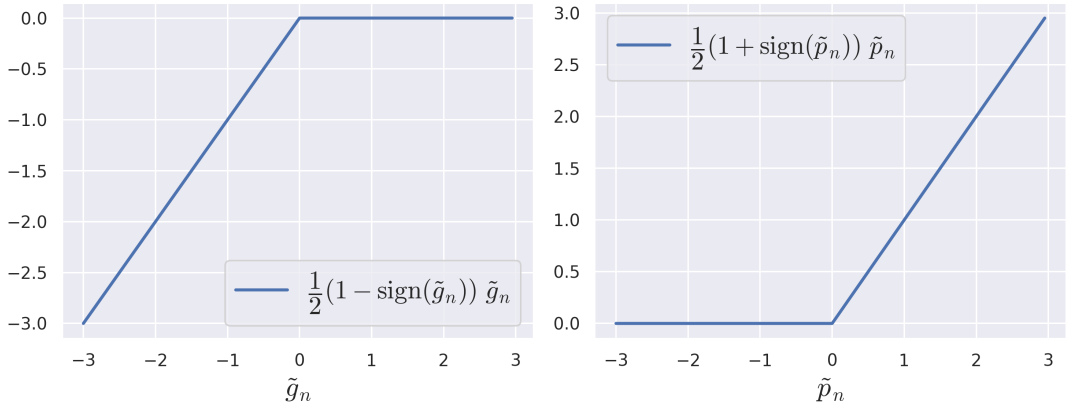


Figure 6.27.: Illustration of the sign-based functions for enforcing constraints depending on the gap $\tilde{g}_n$ and contact pressure $\tilde{p}_n$.

6.3.2.1. Sign-based method

One of the simplest methods to enforce the KKT conditions is by using the sign function, as suggested in [70]. This approach results in the following KKT loss function formulation

$$\begin{aligned}\mathcal{L}_{\text{KKT}} &= \mathcal{L}_{\tilde{g}_n \geqslant 0} + \mathcal{L}_{\tilde{p}_n \leqslant 0} + \mathcal{L}_{\tilde{p}_n \tilde{g}_n = 0} \\ &= w_1^{(\text{KKT})} \left| \frac{1}{2}\big(1 - \text{sign}(\tilde{g}_n)\big)\ \tilde{g}_n \right|_{\partial\Omega_c} + w_2^{(\text{KKT})} \left| \frac{1}{2}\big(1 + \text{sign}(\tilde{p}_n)\big)\ \tilde{p}_n \right|_{\partial\Omega_c} \\ &\quad + w_3^{(\text{KKT})} \left| \tilde{p}_n\ \tilde{g}_n \right|_{\partial\Omega_c}.\end{aligned} \tag{6.98}$$

Here, the terms $\{w_i^{(\text{KKT})}\}_{i=1}^3$ represent the loss weights associated with the corresponding KKT conditions. Each term ensures that the constraints are enforced: the first term enforces the non-penetration constraint ($\tilde{g}_n \geqslant 0$), the second term enforces the admissibility of the contact pressure ($\tilde{p}_n \leqslant 0$), and the third term ensures complementarity ($\tilde{p}_n \tilde{g}_n = 0$).

Fig. 6.27 illustrates the behavior of the sign-based function:

- The term $\frac{1}{2}\big(1 - \text{sign}(\tilde{g}_n)\big)\tilde{g}_n$ contributes to the loss component $\mathcal{L}_{\tilde{g}_n \geqslant 0}$ when the gap $\tilde{g}_n$ is less than zero, penalizing interpenetration.
- Similarly, $\frac{1}{2}\big(1 + \text{sign}(\tilde{p}_n)\big)\tilde{p}_n$ contributes to $\mathcal{L}_{\tilde{p}_n \leqslant 0}$ if the contact pressure $\tilde{p}_n$ becomes positive, which is physically inadmissible.

However, it is important to note that the sign function introduces gradient jumps, particularly around the origin, where its derivative is discontinuous. This lack of smoothness at the origin can hinder optimization by creating non-differentiable regions in the loss landscape.

6.3.2.2. Sigmoid-based method

An alternative approach to address the issue of discontinuous gradients is to utilize the Sigmoid function [69], which ensures smoother transitions and avoids gradient jumps. The corresponding

loss function for enforcing the KKT conditions is defined as

$$\begin{aligned}\mathcal{L}_{\mathrm{KKT}} &= \mathcal{L}_{\tilde{g}_n \geqslant 0} + \mathcal{L}_{\tilde{p}_n \leqslant 0} + \mathcal{L}_{\tilde{p}_n \tilde{g}_n = 0} \\ &= w_1^{(\mathrm{KKT})} \left| \frac{1}{1 + e^{\delta \tilde{g}_n}} \tilde{g}_n \right|_{\partial\Omega_c} + w_2^{(\mathrm{KKT})} \left| \frac{1}{1 + e^{-\delta \tilde{p}_n}} \tilde{p}_n \right|_{\partial\Omega_c} \\ &\quad + w_3^{(\mathrm{KKT})} \left| \tilde{p}_n \tilde{g}_n \right|_{\partial\Omega_c},\end{aligned} \tag{6.99}$$

where δ is the *steepness* parameter that controls the smoothness of the transition between zero and non-zero loss contributions. Unlike the sign-based approach, the Sigmoid function introduces an exponential regularization term that ensures smooth gradients, even near critical points, e.g., the origin.

As illustrated in Fig. 6.28, the Sigmoid function provides smooth transitions based on the choice of δ:

- When δ is set to a low value (e.g., $\delta = 1$), the transition becomes too smooth, leading to significant unphysical contributions to the loss function for cases where $\tilde{g}_n > 0$ or $\tilde{p}_n < 0$.
- Conversely, when δ is set to a very high value (e.g., $\delta = 20$), the Sigmoid function approximates the behavior of the sign-based implementation, potentially reintroducing the challenges associated with gradient discontinuities.

Thus, determining an optimal value for δ, denoted as δ_{opt}, is critical to balancing regularization smoothness and constraint enforcement accuracy. A parameter study is often necessary to identify a proper value of δ for a given problem.

6.3.2.3. Nonlinear complementarity function: Fischer-Burmeister

Nonlinear complementarity problem (NCP) functions are widely used to reformulate inequalities as equalities, enabling smooth optimization [98]. One of the most commonly used NCP functions is the *Fischer-Burmeister* function [52], which is defined as

$$\phi_{\mathrm{FB}}(a, b) := a + b - \sqrt{a^2 + b^2} = 0 \quad \Longleftrightarrow \quad a \geqslant 0, b \geqslant 0, ab = 0. \tag{6.100}$$

By substituting $a = \tilde{g}_n$ and $b = -\tilde{p}_n$ into the Fischer-Burmeister function, the loss function $\mathcal{L}_{\mathrm{KKT}}$ for enforcing the KKT conditions can be expressed as

$$\mathcal{L}_{\mathrm{KKT}} = w^{(\mathrm{KKT})} \left| \phi_{\mathrm{FB}}(\tilde{g}_n, -\tilde{p}_n) \right|_{\partial\Omega_c} = w^{(\mathrm{KKT})} \left| \tilde{g}_n - \tilde{p}_n - \sqrt{\tilde{g}_n^2 + \tilde{p}_n^2} \right|_{\partial\Omega_c}. \tag{6.101}$$

The Fischer-Burmeister function is particularly well-suited for loss calculations based on the mean squared error, as $(\phi_{\mathrm{FB}})^2$ is continuously differentiable, even at the critical point where $a = b = 0$, as reported in [161]. This property makes the Fischer-Burmeister function ideal for optimization problems requiring smooth loss landscapes.

Fig. 6.29 illustrates the behavior of the Fischer-Burmeister function. The 3D surface plot (Fig. 6.29a) demonstrates the loss contributions across the gap g_n and contact pressure p_n, while the 2D contour plot (Fig. 6.29b) highlights regions of significant loss. The largest loss contributions are observed in Region IV, where excessive penetration ($g_n \leqslant 0$) and large adhesive stresses ($p_n \geqslant 0$) occur simultaneously.

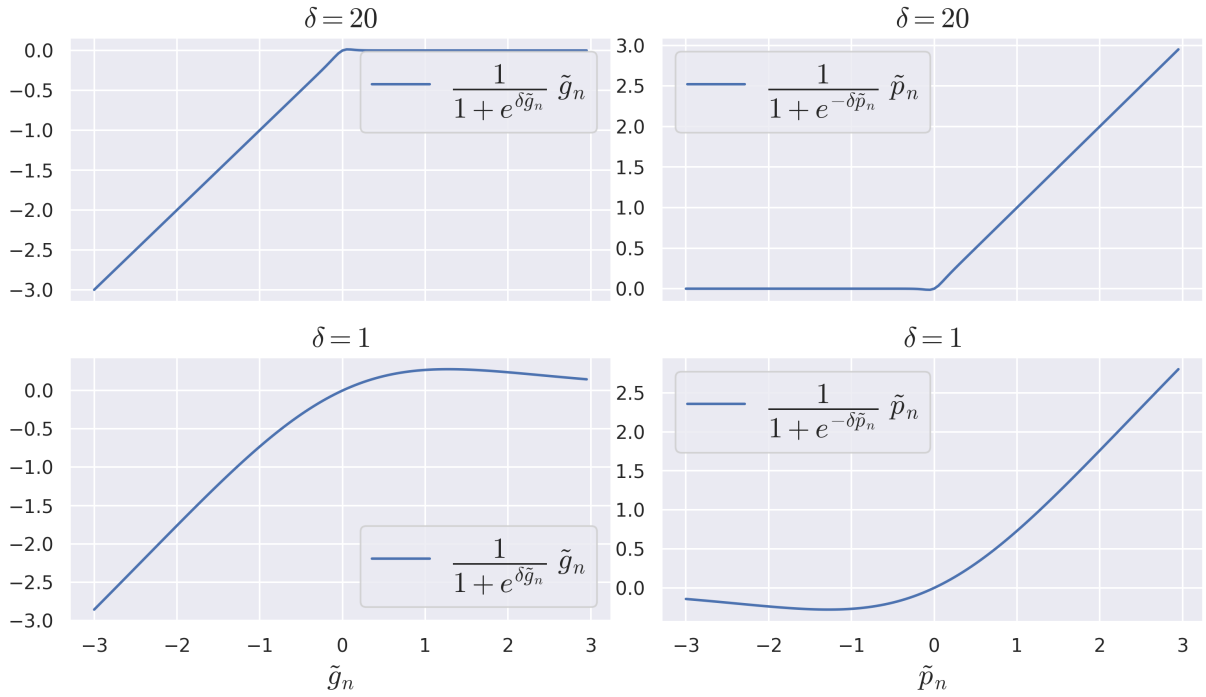
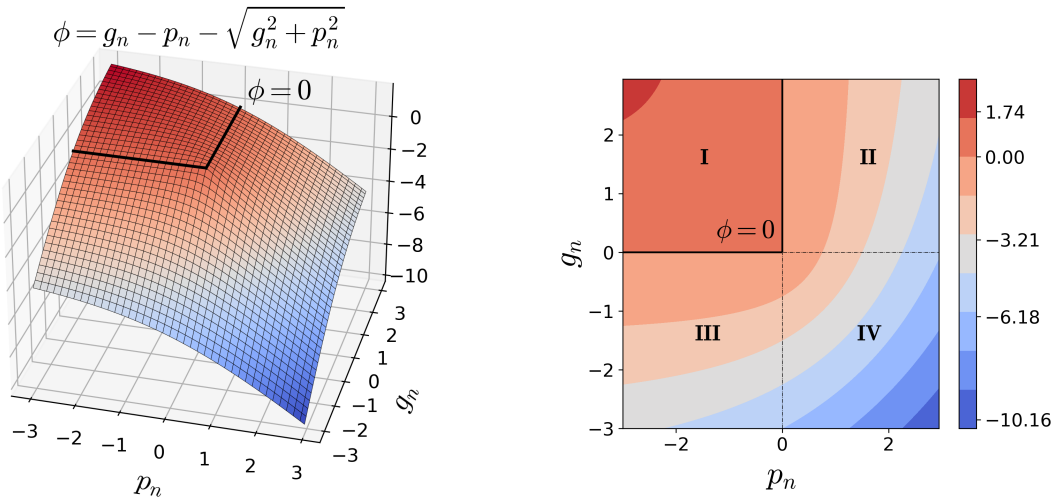


Figure 6.28.: Illustration of the Sigmoid-based function applied to the gap g_n and contact pressure p_n for different values of the steepness parameter δ, which controls the smoothness of the transition in the loss function.



(a) 3D surface plot of the Fischer-Burmeister function.

(b) 2D contour plot of the Fischer-Burmeister function.

Figure 6.29.: Visualization of the Fischer-Burmeister NCP function based on the gap g_n and contact pressure p_n, where (a) is the 3D surface plot and (b) represents the projection of the function as 2D contour plot.

A key advantage of the Fischer-Burmeister NCP function is its simplicity in implementation. Unlike the methods discussed in Sections 6.3.2.1 and 6.3.2.2, which require multiple loss terms with corresponding weights, the Fischer-Burmeister formulation uses a single loss weight $w^{(\mathrm{KKT})}$. This reduction in complexity facilitates easier parameter tuning and enhances the optimization process.

For more details on the properties and applications of the Fischer-Burmeister function, we refer to [11, 52, 98, 161].

Domination of p_n over g_n

The Fischer-Burmeister NCP function is an effective approach for reformulating inequality constraints into a single equality. However, a key challenge with this method is the potential domination of one term over another in the loss function.

As discussed in previous sections, the normal gap g_n is derived from the displacement field $\boldsymbol{u}$, while the contact pressure p_n is derived from the stress field $\boldsymbol{\sigma}$. Depending on the problem-specific parameters, such as material stiffness, the scales of the estimated quantities $\boldsymbol{u}$ and $\boldsymbol{\sigma}$ can vary significantly, leading to disparities in the scales of g_n and p_n, which have already been addressed in Section 6.2.2.3. This imbalance in scales can result in optimization issues where one term dominates the loss function.

Fig. 6.30 illustrates this phenomenon. When g_n and p_n are of similar scales (Fig. 6.30a), the optimization process is balanced, and no term dominates. However, as the scale of p_n increases relative to g_n (Fig. 6.30b and Fig. 6.30c), p_n begins to dominate. This dominance forces the neural network to prioritize minimizing the larger-scale quantity p_n, potentially leading to significant

violations of constraints associated with g_n. Such constraint violations can cause the unphysical phenomena, such as penetration across domains.

To address this issue, non-dimensionalization techniques can be employed to ensure that p_n and g_n are rescaled to comparable magnitudes [182]. By normalizing these quantities, the optimization process is balanced, preventing the domination of one term over the other and ensuring that the constraints for both g_n and p_n are enforced effectively. Additionally, output scaling can serve as an effective strategy to address this challenge (see Section 6.2.2.3).

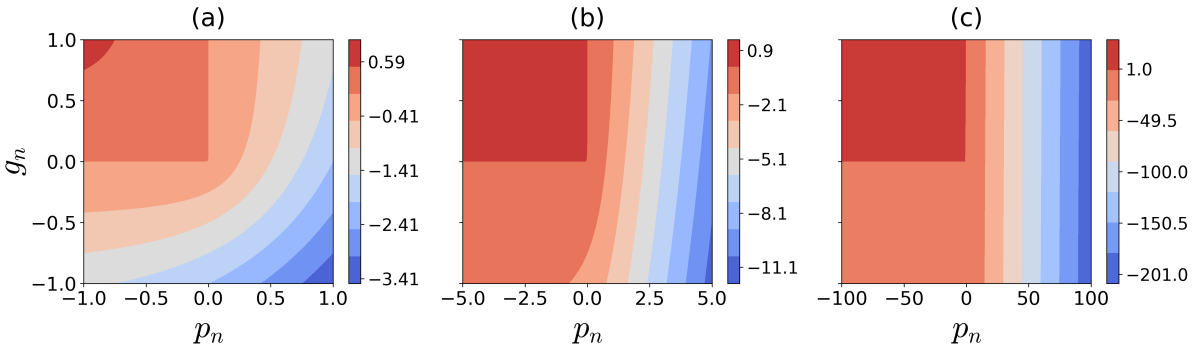


Figure 6.30.: Illustration of the domination effect of p_n over g_n in the Fischer-Burmeister NCP function. (a) No domination when g_n and p_n are of similar scales. (b) Intermediate domination as the scale of p_n increases. (c) Strong domination where p_n significantly outweighs g_n, causing optimization challenges.

6.3.3. Numerical examples

In the following sections, we explore four distinct contact examples:

- 2D contact problem between an elastic block and a rigid flat surface
- Hertzian contact problem
- Single contact patch test
- Cylindrical contact problem

The first example investigates a 2D contact problem between an elastic block and a rigid flat surface. This serves as a foundational case to validate the implementation. The second example focuses on the well-known Hertzian contact problem, comprising three different cases to emphasize the importance of data enhancement. A key distinction between these two problems lies in the contact area: for the Hertzian contact problem, the actual contact area must be determined by the PINN, whereas in the elastic block example, the potential and actual contact areas are identical.

Subsequently, we increase the complexity by transitioning to 3D contact problems, including the single contact patch test and the cylindrical contact problem. It is important to note that plane strain conditions are assumed for the 2D examples, and body forces are neglected in all cases.

Across all numerical examples, consistent settings are applied. Since the PINNs are employed using the mixed-variable approach, the networks map spatial coordinates $\boldsymbol{x} = (x, y, z)$ as inputs to mixed-form outputs $(\tilde{\boldsymbol{u}}, \tilde{\boldsymbol{\sigma}}) = (\tilde{u}_x, \tilde{u}_y, \tilde{\sigma}_{xx}, \tilde{\sigma}_{yy}, \tilde{\sigma}_{xy})$. The neural networks are initialized using the *Glorot uniform* initializer, and the *tanh* function is selected as the activation function.

For training, the models first utilize the *Adam* stochastic gradient descent optimizer with a learning rate of $lr = 0.001$ for 2000 epochs. Afterward, the training transitions to the *L-BFGS-B* optimizer until convergence is achieved. These consistent training settings ensure reliable and efficient optimization for all examples. Additionally, we utilize the vector-based relative L_2 errors for comparison, as defined in Section 6.2.4.

6.3.3.1. 2D contact between an elastic block and a rigid surface

In the first example, we analyze a contact problem involving a 2D linear-elastic block and a rigid flat surface, as shown in Fig. 6.31a. The elastic block is subjected to an external pressure on its top surface while its left surface is constrained in the horizontal direction. The analytical solution for this setup, derived from [142], is given as

$$u_x = \frac{-p}{E}\nu(1+\nu)x, \quad u_y = \frac{p}{E}(1-\nu^2)y,$$
$$\sigma_{yy} = -p, \quad \sigma_{xy} = 0.$$

For this specific problem, the material parameters are set as $E = 1.33$, $\nu = 0.33$, with the edge length of the square block $l = 1$, and the applied pressure $p = 0.1$.

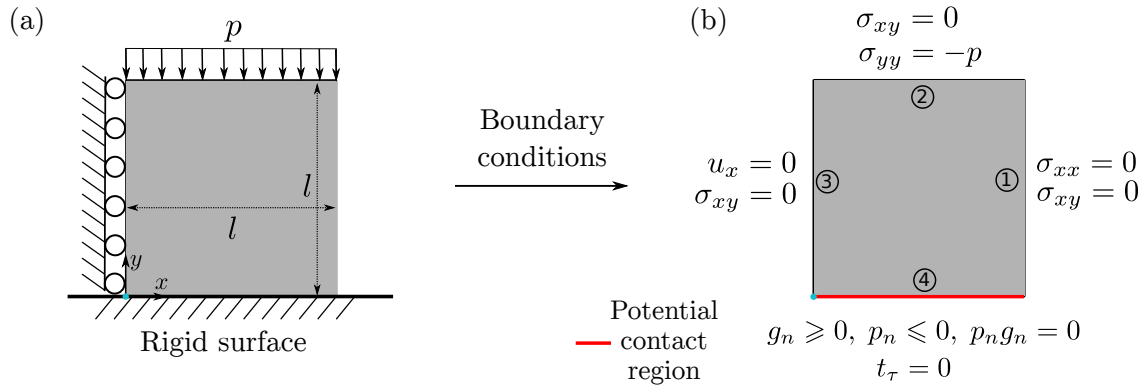


Figure 6.31.: Contact problem between a linear-elastic block and a rigid flat surface: (a) geometry, supports, and loading, and (b) equivalent system including all relevant boundary conditions.

To enforce displacement and traction boundary conditions on the edges numbered 1, 2, and 3, we employ the following output transformations

$$\begin{aligned} \tilde{u}_x &= x\mathcal{N}_{u_x}, \quad \tilde{\sigma}_{xx} = (l-x)\mathcal{N}_{\sigma_{xx}}, \\ \tilde{\sigma}_{yy} &= -p + (l-y)\mathcal{N}_{\sigma_{xy}}, \quad \tilde{\sigma}_{xy} = x(l-y)(l-x)\mathcal{N}_{\sigma_{xy}}. \end{aligned} \tag{6.102}$$

These transformations are derived based on the geometry and boundary conditions illustrated in Fig. 6.31b. For example, the normal stress σ_{yy} in the loading direction is equal to $-p$ at $y = l$.

Consequently, we define $g(\boldsymbol{x}) = -p$ and $h(\boldsymbol{x}) = (l - y)$ to satisfy the requirements outlined in Eq. 5.6.

The contact constraints at the bottom edge (potential contact region) are enforced as soft constraints using three methods that are explained in Section 6.3.2: the sign-based method, the Sigmoid-based method, and the Fischer-Burmeister NCP function. For the Sigmoid-based method, we select steepness parameters $\delta_{g_n} = 10$ and $\delta_{p_n} = 100$.

For training, a total of 514 points are used, with 434 points sampled within the domain and 80 points located on the boundary. For testing, a denser set of $11{,}827$ points is employed to evaluate the model's performance comprehensively.

As illustrated in Fig. 6.32, all the investigated methods successfully capture the linear distribution of u_y and its near-zero value at the bottom surface, which is a result of the soft enforcement of contact constraints. The maximum absolute error for the displacement component u_y, denoted as $E^{u_y}_{abs,max}$, is noticeably larger for the sign-based formulation compared to the Sigmoid-based and Fischer-Burmeister methods (see Table 6.6).

For the stress components, the normal stress σ_{yy} is approximately equal to the applied load of -0.1 across the entire domain, while the shear stress field σ_{xy} remains close to zero. Since the traction boundary condition in the y-direction on the top surface and the shear stress boundary conditions are enforced as hard constraints on all edges, the absolute errors in these regions are strictly zero across all methods. However, the sign-based formulation exhibits the highest maximum errors for $E^{\sigma_{yy}}_{abs,max}$ and $E^{\sigma_{xy}}_{abs,max}$ compared to the other two methods.

The relative L_2 errors indicate that the Fischer-Burmeister NCP function outperforms the other methods, achieving errors up to an order of magnitude smaller than those of the sign-based and Sigmoid-based formulations, as provided in Table 6.6. Furthermore, the training time remains comparable across all methods, and the prediction requires approximately $0.39\,\mathrm{s}$ for all approaches.

Overall, the Fischer-Burmeister NCP function demonstrates superior performance in terms of both accuracy and computational efficiency, along with the key advantages explained in Section 6.3.2.3. Therefore, we enforce the KKT constraints using the Fischer-Burmeister method for all the contact problems presented in the next sections.

	Hidden layers	Training time (s)	$E^{\boldsymbol{u}}_{L_2}$ (%)	$E^{\boldsymbol{\sigma}}_{L_2}$ (%)	$E^{u_y}_{abs,max}$	$E^{\sigma_{yy}}_{abs,max}$	$E^{\sigma_{xy}}_{abs,max}$
(a)	5×50	20.15	0.382	0.154	2.66×10^{-4}	4.34×10^{-4}	1.11×10^{-4}
(b)	5×50	20.14	0.090	0.094	8.23×10^{-5}	1.51×10^{-4}	6.30×10^{-5}
(c)	5×50	18.20	0.024	0.031	2.71×10^{-5}	6.10×10^{-5}	2.55×10^{-5}

Table 6.6.: Comparison of hidden layer architectures, performance metrics, and error measurements for the 2D contact problem involving an elastic block and a rigid surface. The KKT constraint enforcement methods include (a) sign-based, (b) Sigmoid-based, and (c) Fischer-Burmeister approaches.

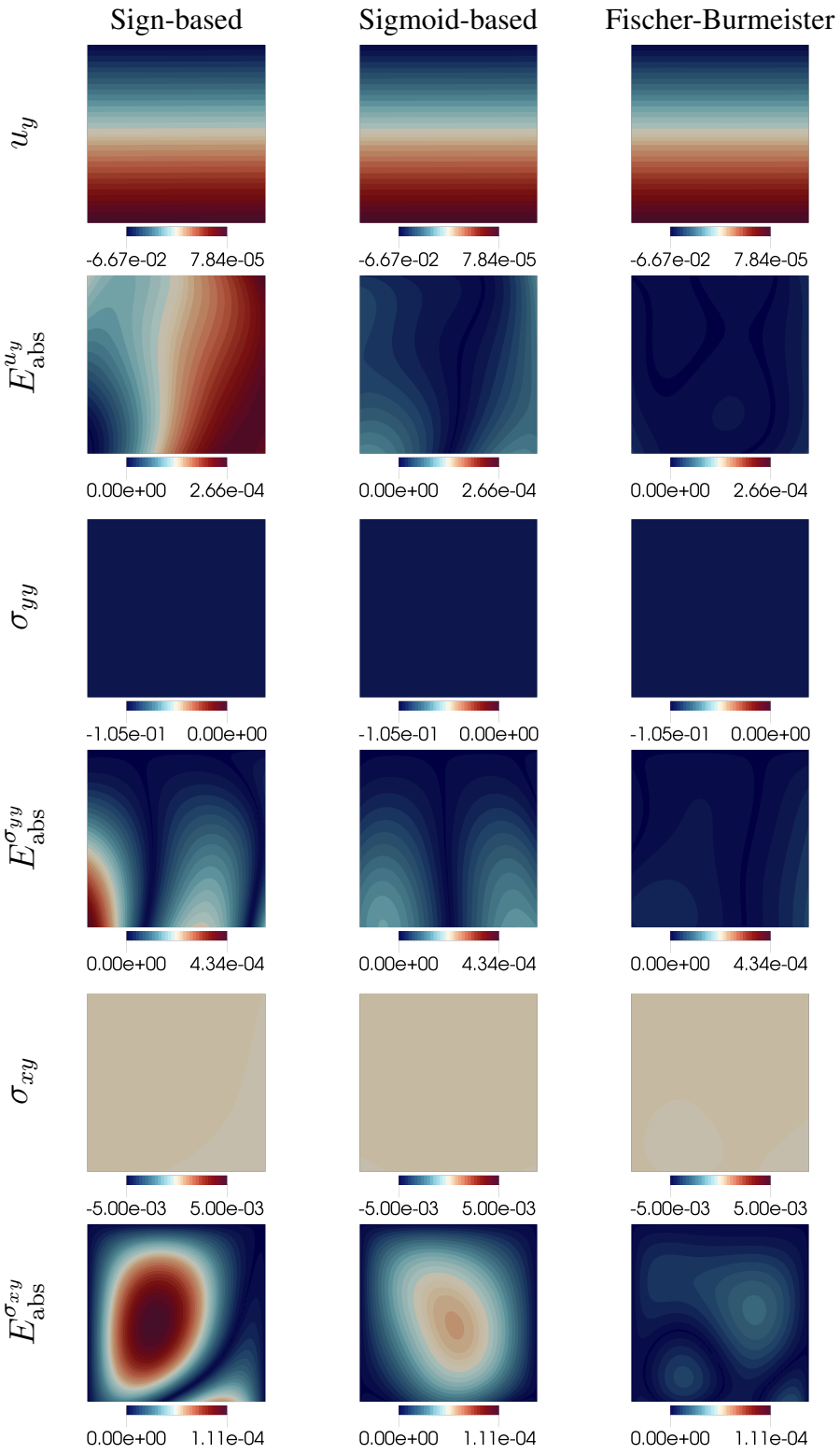


Figure 6.32.: Predictions of the displacement component u_y, stresses σ_{yy} and σ_{xy} with corresponding absolute errors $E^{u_y}_{\text{abs}}$, $E^{\sigma_{yy}}_{\text{abs}}$, $E^{\sigma_{xy}}_{\text{abs}}$. The contact constraints $\mathcal{L}_{\text{KKT}}$ is enforced via three different methods. Absolute error is defined as $E^{*}_{\text{abs}} = \text{abs}(\tilde{*} - *)$.

6.3.3.2. 2D Hertzian contact problem

In this example, we examine a long, linearly elastic half-cylinder with material parameters $E = 200$ and $\nu = 0.3$, resting on a rigid flat surface. The cylinder is subjected to a uniform pressure $p = 0.5$ applied on its top surface, as depicted in Fig. 6.33a. The analytical expression for the contact pressure p_c is given by [90, 125]

$$p_c = \frac{4Rp}{\pi b^2}\sqrt{b^2 - x^2}, \quad \text{where} \quad b = 2\sqrt{\frac{2R^2p(1-\nu^2)}{E\pi}}. \tag{6.103}$$

Here, b represents the width of the contact zone, and R denotes the radius of the cylinder. Using the specified parameters ($p = 0.5$, $E = 200$, $\nu = 0.3$, $R = 1$), the width of the contact zone is calculated as $b = 0.076$.

The analytical solution is limited to the contact pressure p_c. Reference solutions for the displacement and stress fields within the half-cylinder are generated using well-established finite element method (FEM) algorithms. These simulations are carried out using our in-house multiphysics research code, 4C: A Comprehensive Multi-Physics Simulation Framework, https://www.4c-multiphysics.org [1].

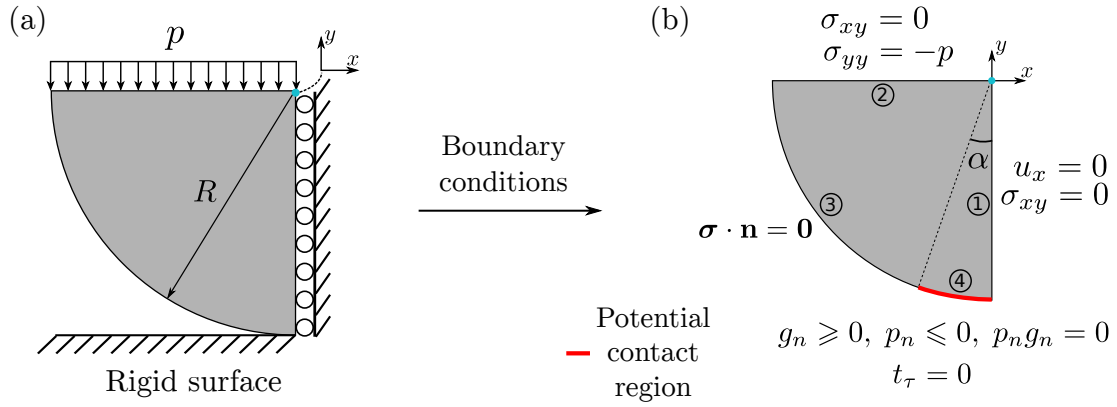


Figure 6.33.: The Hertzian contact problem between an elastic half-cylinder and a rigid flat surface (a) domain under uniform pressure on top and making use of symmetry, (b) accompanying boundary conditions.

The following output transformation is applied to enforce displacement and traction BCs as hard constraints on the edges numbered 1 and 2 (see Fig. 6.33b)

$$\tilde{u}_x = \frac{-x}{E}\mathcal{N}_{u_x}, \quad \tilde{u}_y = \frac{1}{E}\mathcal{N}_{u_y}, \quad \tilde{\sigma}_{yy} = -p + (-y)\mathcal{N}_{\sigma_{yy}}, \quad \tilde{\sigma}_{xy} = xy\mathcal{N}_{\sigma_{xy}}. \tag{6.104}$$

The displacement field is scaled by the inverse of Young's modulus, $1/E$. Additionally, only one non-zero helper function, $g(x) = -p$, is required for $\tilde{\sigma}_{yy}$. The traction boundary conditions on edge 3 and the contact constraints on edge 4 are imposed as soft constraints. To define the potential contact area, we set $\alpha = 15°$, corresponding to $b = 0.259$, which extends well beyond the actual contact area. Furthermore, the KKT constraints are enforced using the *Fischer-Burmeister* method.

In the subsequent analysis, we examine three distinct application scenarios of PINNs for the Hertzian contact problem

- *Case 1*: PINNs as a pure forward model / PDE solver.
- *Case 2*: PINNs as a data-enhanced forward model.
- *Case 3*: PINNs as an inverse solver for parameter identification.
- *Case 4*: PINNs as a fast-to-evaluate surrogate model.

Details on the network architecture, as well as the training and testing points, are summarized in Table 6.7. To facilitate reproducibility, the table also includes the weights assigned to individual loss terms.

	Hidden layers	# input neurons	# training points	# test points	Loss weights
Case 1	5×50	2	47,935	26,185	w^{KKT}=10^3
Case 2,3	5×50	2	47,935	26,185	w^{KKT}=10^3, w_1^{EXPs}=10^4, w_2^{EXPs}=10^4, w_3^{EXPs}=10^{-1}, w_4^{EXPs}=10^{-1}, w_5^{EXPs}=10^{-1}
Case 4	8×75	3	1,946	604	w^{KKT}=10^4

Table 6.7.: The structure of hidden layers, number of input neurons, training and test points, and loss weights are given for different cases of the Hertzian contact problem. The remaining loss weights are set as 1.

Case 1: PINNs as pure forward model / PDE solver

In the first use case, we employ PINNs as a pure forward solver to validate their application to contact problems. The training process requires a total of $4{,}049\,\mathrm{s}$, while the prediction time is only $0.091\,\mathrm{s}$. In comparison, FE simulation for the same test case takes $19.2\,\mathrm{s}$, significantly longer than the prediction time of a trained PINN, emphasizing that the computational cost of PINNs is dominated by the training phase.

Displacement and stress components computed using PINNs and FEM are compared through contour plots in Fig. 6.34. The errors in the displacement and stress fields are quantified using the relative L_2 norm. As summarized in Table 6.8, the relative error for the displacement field is $E_{L_2}^{\boldsymbol{u}} = 2.24\%$, while for the stress field it is $E_{L_2}^{\boldsymbol{\sigma}} = 3.74\%$.

Additionally, Fig. 6.35 compares the contact pressure distributions obtained through the analytical solution, PINNs, and FEM. Since the zero traction boundary condition and the KKT constraints on the curved surface (edge 3 in Fig. 6.33) are implemented as soft constraints rather than hard constraints, the PINN solution approximates the kink at $x = \pm 0.076$ based on the chosen training points. As a result, the contact pressure p_c transitions to zero smoothly and slightly violates the zero traction boundary condition in the non-contact region. This contributes to the relatively large error values $E_{L_2}^{\boldsymbol{\sigma}}$, primarily due to the violation of the zero traction boundary condition and KKT constraints near the kink.

It is worth noting that a similar phenomenon is observed in FEM modeling of contact problems, where the transition between contact and non-contact zones cannot be perfectly resolved. In some cases, this can even lead to spurious oscillations when higher-order interpolation functions are employed [128, 129, 154].

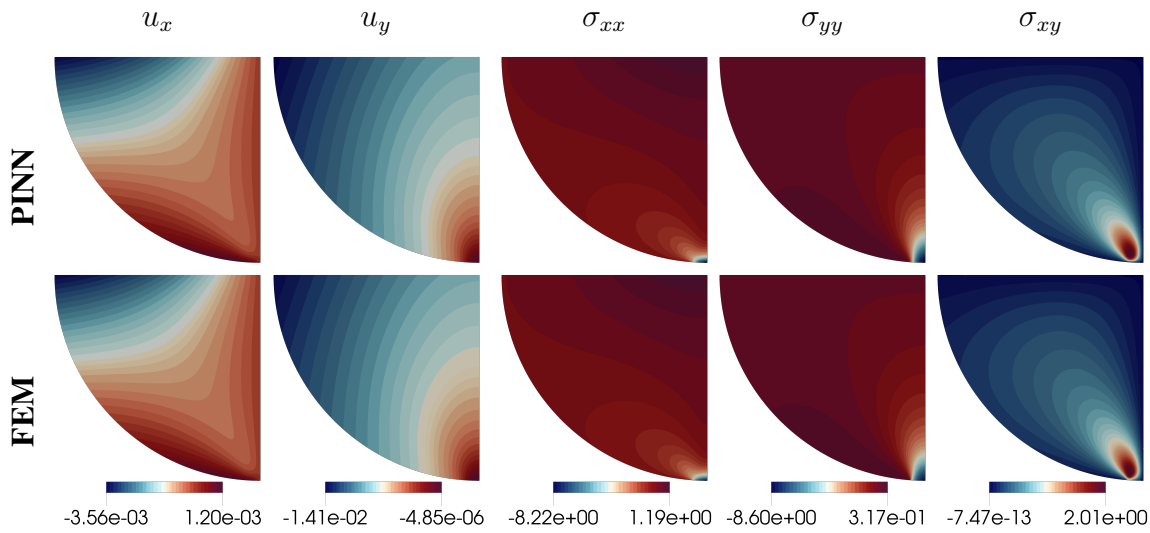


Figure 6.34.: PINNs as pure forward solver for the Hertzian contact problem. Comparison of stress and displacement components obtained by PINN and FEM.

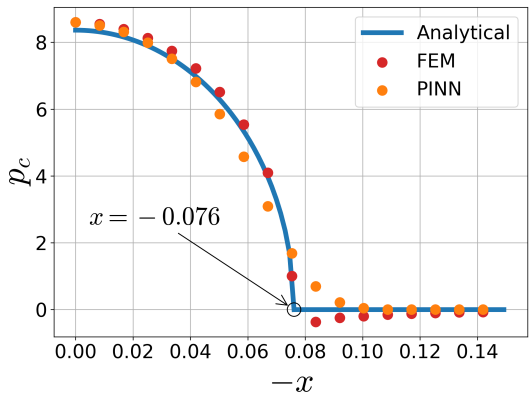


Figure 6.35.: Comparison of contact pressure distributions $p_c(x)$ obtained by analytical solution, PINN and FEM for case 1.

Case 2: PINNs as data-enhanced forward model

As explained earlier, a key advantage of PINNs is their ability to seamlessly integrate external data, such as measurements or simulation results, into the loss function. In this section, we improve the accuracy of our PINN model by incorporating ”artificial” measurement data generated from FEM simulations. Specifically, we use data points for displacement and stress fields. The FEM

	$E^{\boldsymbol{u}}_{L_2}$ (%)	$E^{\boldsymbol{\sigma}}_{L_2}$ (%)	Training time (s)	Prediction time (s)	$\int_{\Omega_c} \tilde{p}_c$
Case 1	2.24	3.74	4,049	0.091	0.4993
Case 2	0.11	2.79	7,126	0.092	0.4978

Table 6.8.: Comparison of relative L_2 errors, training and prediction time for case 1 and case 2. The term $\int_{\Omega_c} \tilde{p}_c$ denotes an integral of the predicted contact pressure $\tilde{p}_c$ over the potential contact boundary.

data points are randomly sampled, with 100 points taken from inside the domain and another 100 from the boundary, as shown in Fig. 6.36b.

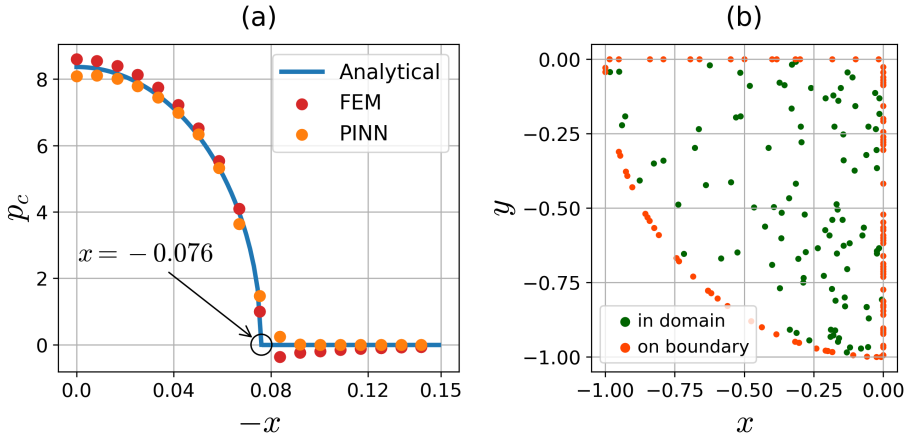


Figure 6.36.: Comparison of contact pressure distributions $p_c(x)$ obtained by analytical solution, PINN and FEM for case 2. (a) PINNs as a data-enhanced forward solver, (b) distribution of data points generated through FEM simulations in the domain and on the boundary.

Fig. 6.36a compares the contact pressure p_c in the data-enhanced case with the analytical and FEM reference solutions. While the PINN's accuracy near the kink at $x = \pm 0.076$ is significantly improved, the data-enhanced model underestimates the normal contact pressure around the origin. The relative L_2 errors support this observation. While $E^{\boldsymbol{\sigma}}_{L_2}$ shows only minor improvement, $E^{\boldsymbol{u}}_{L_2}$ greatly benefits from incorporating data. As shown in Table 6.8, the integrated contact pressure remains close to the applied load $p = 0.5$, due to the conservation of momentum. To satisfy this equation, overshooting beyond the kink is balanced by undershooting near the origin. Similarly, in case 1, overshooting after the kink is offset by undershooting from $x = 0.04$ to the kink.

We also find that results are highly sensitive to the choice of additional data loss weights w^{EXPs} (detailed in Table 6.7). For this study, the weights were manually tuned, but in general, a hyperparameter analysis is recommended to identify optimal loss weights. The data-enhanced model also requires more training time compared to the pure forward model, as additional loss terms must be evaluated. However, once trained, the more accurate predictions are computed in

essentially the same time as in case 1.

Case 3: PINNs as inverse solver for parameter identification

Next, we identify the applied pressure p on the half-cylinder using FEM-derived "artificial" measurement data. The unknown parameter p is added to the trainable parameters $\boldsymbol{\Theta}$ as described in Section 5.1.1.3, leading to

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}_{\mathrm{C}}(\boldsymbol{\Theta}) \quad \text{where,} \quad \boldsymbol{\Theta} = (p, \boldsymbol{\theta}). \tag{6.105}$$

Fig. 6.37 illustrates the convergence of the identified pressure $\tilde{p}$ compared to the true pressure p, across epochs, for different initial guesses $\tilde{p}_o$. First, we use a "good" initial guess $\tilde{p}_o = 0.1$ close to $p = 0.5$. Then, we test $\tilde{p}_o = 20$ to examine sensitivity to the initial guess. As shown in Fig. 6.37c, convergence occurs even for large, unphysical guesses. The convergence rate steeply increases after 2000 epochs when switching to the *L-BFGS-B* optimizer. Similarly, we use *Adam* for 2000 epochs to prevent the optimization from getting stuck in local minima. However, switching to *L-BFGS-B* can introduce oscillations in the transition region, requiring careful handling.

The relative error E measures the difference between identified and actual pressures, yielding 1.2% for all initial guesses $\tilde{p}_o = 0.1$, $\tilde{p}_o = 5$, and $\tilde{p}_o = 20$ (see Table 6.9). Larger initial guesses require more time and epochs, as expected.

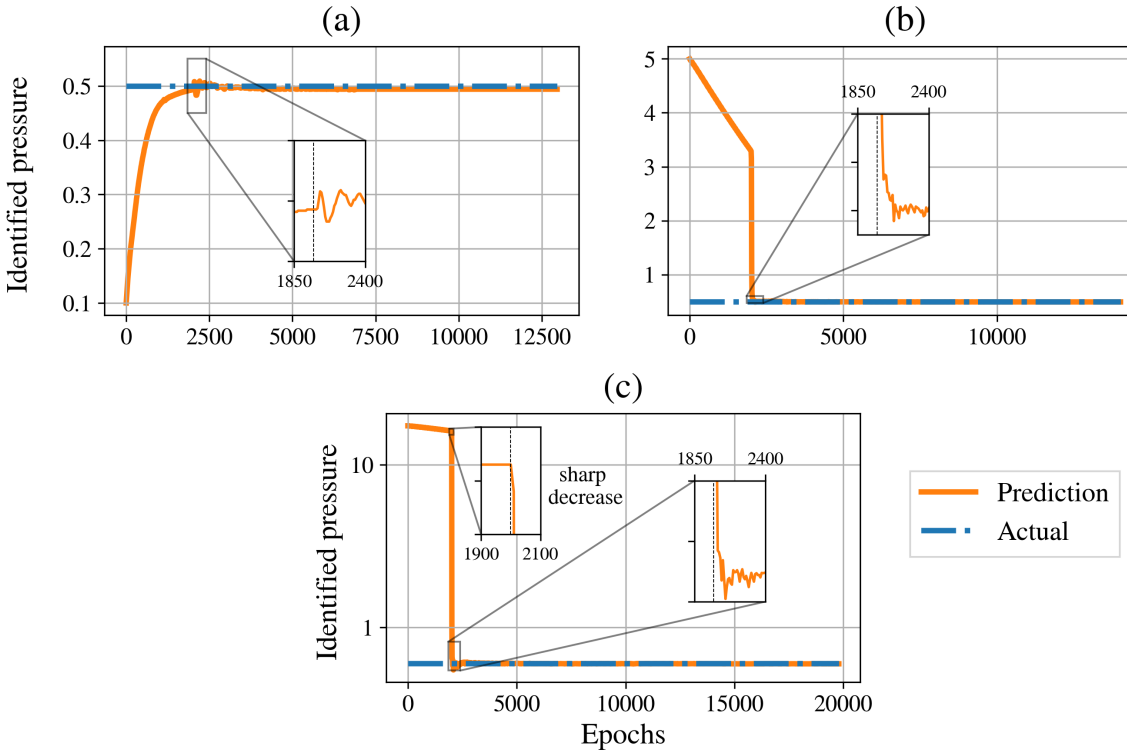


Figure 6.37.: Identification of the applied external pressure on the half-cylinder for different initial guesses (a) $\tilde{p}_o = 0.1$, (b) $\tilde{p}_o = 5$, (c) $\tilde{p}_o = 20$.

	Initial guess $\tilde{p}_o$	Identified $\tilde{p}_f$	Relative error $E(\%)$	Training time (s)	# epochs
Case 3	0.1	0.494	1.2	3,472	12,958
	5	0.494	1.2	3,762	14,019
	20	0.494	1.2	5,394	19,793

Table 6.9.: Comparison of relative errors, training time, and epochs for identifying the applied external pressure. The relative error is defined as $E = \text{abs}\left((\tilde{*} - *)/*\right)$.

Case 4: PINNs as fast-to-evaluate surrogate model

In the last use case, the load (applied external pressure) is considered as an additional network input, i.e. $\boldsymbol{z} = (\boldsymbol{x}, p)$, to construct a fast-to-evaluate surrogate model that is capable of predicting displacement and stress fields for different pressure values. Since a single network is deployed, the length of each network input must be the same. We sample the three-dimensional input space with $1{,}946$ training points. While the spatial coordinates $\boldsymbol{x}$ are selected from a two-dimensional mesh (as in Section 6.3.3.2, 6.3.3.2, and 6.3.3.2), the third (pressure) component of the input vector is drawn randomly from a uniform distribution over the considered the pressure range. To improve the accuracy of the prediction, the N sampling points are repeated k times. In the context of network training, we refer to one instance of the N distinct sampling points as one *chunk*, so that k is the number of chunks as depicted in Fig. 6.38. Indeed, this process increases the computing time and complexity of the model, since the input size increases from $(n, 2)$ to $(n \cdot k, 3)$. However, such a method can lead to better accuracy since the network is trained with a larger data set and also it enables batch training to reduce computational effort [59]. For our specific example, we sample pressure values from a range of $[0.2, 1.0]$ and we consider only two different numbers of chunks, namely $k = 1$ and $k = 5$.

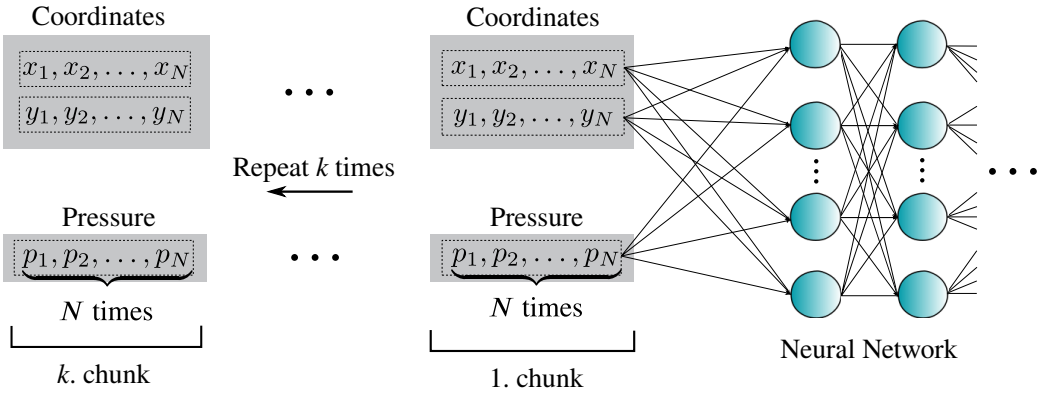


Figure 6.38.: An illustration of the procedure to include the applied external pressure as a neural network input.

As shown in Figure 6.39, employing a single chunk is insufficient to accurately capture the influence of the applied external pressure p on the contact pressure distribution $p_c(\boldsymbol{x}, p)$. However,

		Applied pressure		
		$p = 0.45$	$p = 0.98$	$p = 1.5$
Number of chunks	$k = 1$	22.81%	17.23%	16.93%
	$k = 5$	16.30%	11.47%	10.87%

Table 6.10.: Comparison of relative L_2 errors for the contact pressure distribution $p_c(\boldsymbol{x}, p)$ between the analytical solution and the predictions of our surrogate PINN models. Predictions are based on unseen pressure values.

increasing the number of chunks from $k = 1$ to $k = 5$ increases the accuracy. Table 6.10 provides the relative L_2 errors for the contact pressure distribution $p_c(\boldsymbol{x}, p)$ between the analytical solution and the predictions of surrogate PINN models, and it can be seen that increasing the number of chunks indeed results in improved accuracy of the surrogate model. Nonetheless, the overall error level of 10%-16%, even in the case $k = 5$, is still too high from an engineering perspective and will be subject to further investigations. This example is only intended as a very first proof of concept, and we have already identified several algorithmic modifications that could increase the accuracy of the surrogate model. For example, we use fixed loss weights even though pressure values in the input layer vary. Thus, adaptive loss weights should be implemented since the PINN accuracy highly depends on choosing appropriate loss weights. Additionally, the accuracy of PINNs could be further improved by increasing the number of chunks.

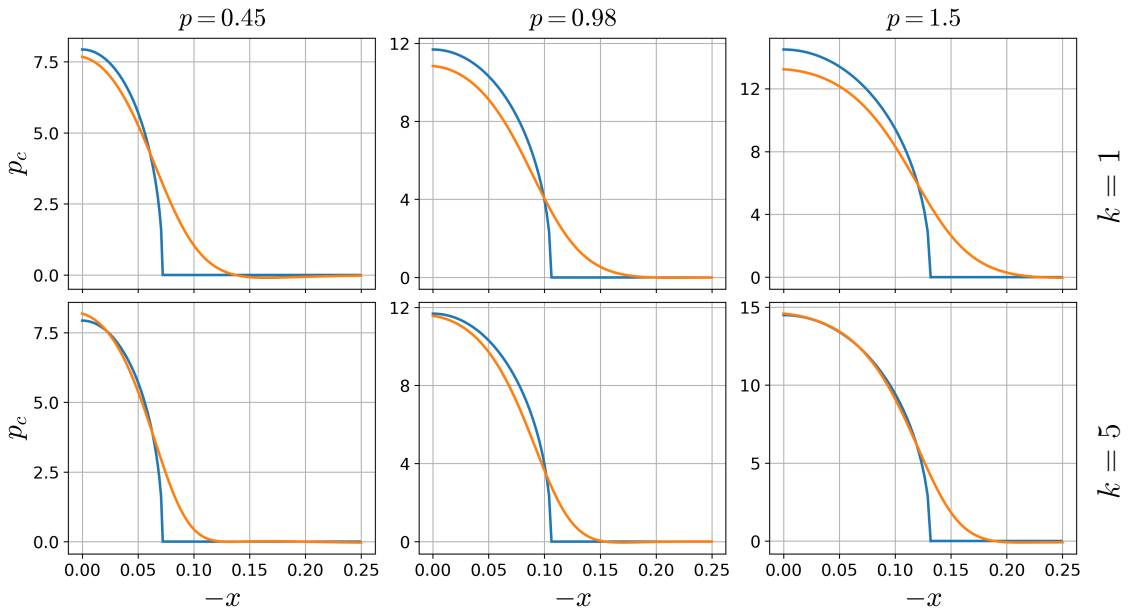


Figure 6.39.: Comparison of the contact pressure distribution obtained through a PINN-based surrogate model to the analytical solution. The PINNs are trained using different numbers of chunks.

6.3.3.3. 3D single contact patch test

To further increase the complexity of the input space, we examine a single patch contact problem, as illustrated in Fig. 6.40a. In this setup, an elastic cube is subjected to a uniform external pressure applied to its top surface. The geometry is constrained in the x-direction at $x = 0$ and in the z-direction at $z = 0$. The analytical solution for this problem, derived from [164], is given by:

$$u_x = \nu \frac{p}{E} x, \quad u_y = -\frac{p}{E} y, \quad u_z = \nu \frac{p}{E} z, \quad \sigma_{yy} = -p.$$

The parameters for this test case are specified as $E = 1.33$, $\nu = 0.33$, $p = 0.1$, and $l = h = w = 1$.

To enforce the DBCs and NBCs shown in Fig. 6.40b as hard constraints, the following output transformations are applied:

$$\begin{aligned}
\tilde{u}_x &= x\mathcal{N}_{u_x}, \\
\tilde{u}_z &= z\mathcal{N}_{u_z}, \\
\tilde{\sigma}_{xx} &= (l - x)\mathcal{N}_{\sigma_{xx}}, \\
\tilde{\sigma}_{yy} &= -p + (h - y)\mathcal{N}_{\sigma_{yy}}, \\
\tilde{\sigma}_{zz} &= (w - z)\mathcal{N}_{\sigma_{zz}}, \\
\tilde{\sigma}_{xy} &= x(h - y)(l - x)\mathcal{N}_{\sigma_{xy}}, \\
\tilde{\sigma}_{yz} &= z(h - y)(w - z)\mathcal{N}_{\sigma_{yz}}, \\
\tilde{\sigma}_{xz} &= xz(l - x)(w - z)\mathcal{N}_{\sigma_{xz}}.
\end{aligned}$$

These transformations enforce DBCs and NBCs as hard constraints, while the contact conditions are incorporated as soft constraints, as described earlier. The set of KKT constraints is enforced using the Fischer-Burmeister function. The employed PINN architecture consists of a fully connected neural network with 5 hidden layers, each containing 50 neurons.

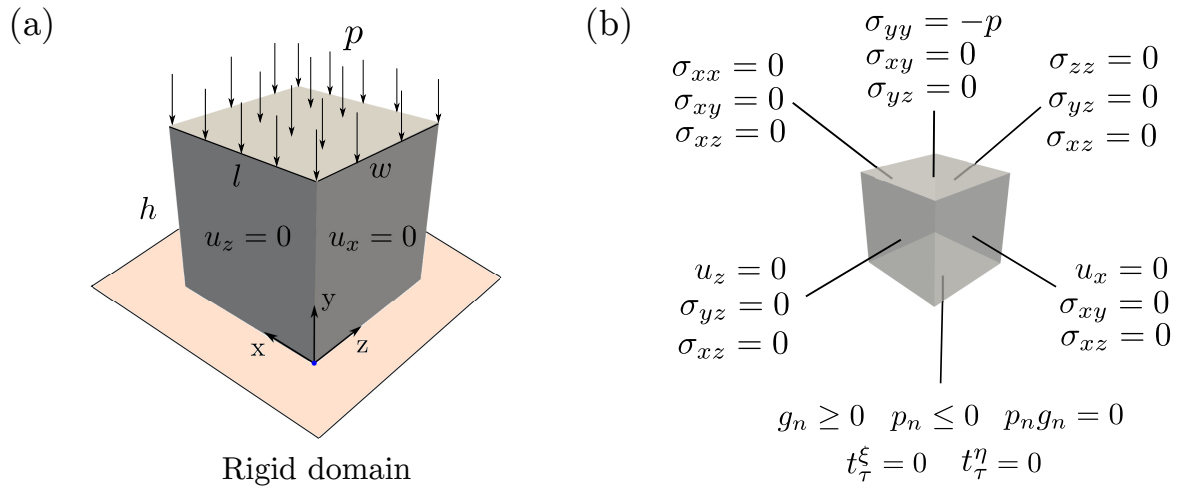


Figure 6.40.: Single patch contact problem: (a) geometry, constraints, and loading, (b) relevant boundary conditions including NBCs, DBCs, KKT, and FSCs.

The comparison between the predicted and analytical solutions for the single contact patch test is shown in Fig. 6.31. The results are only provided for the displacement components u_x

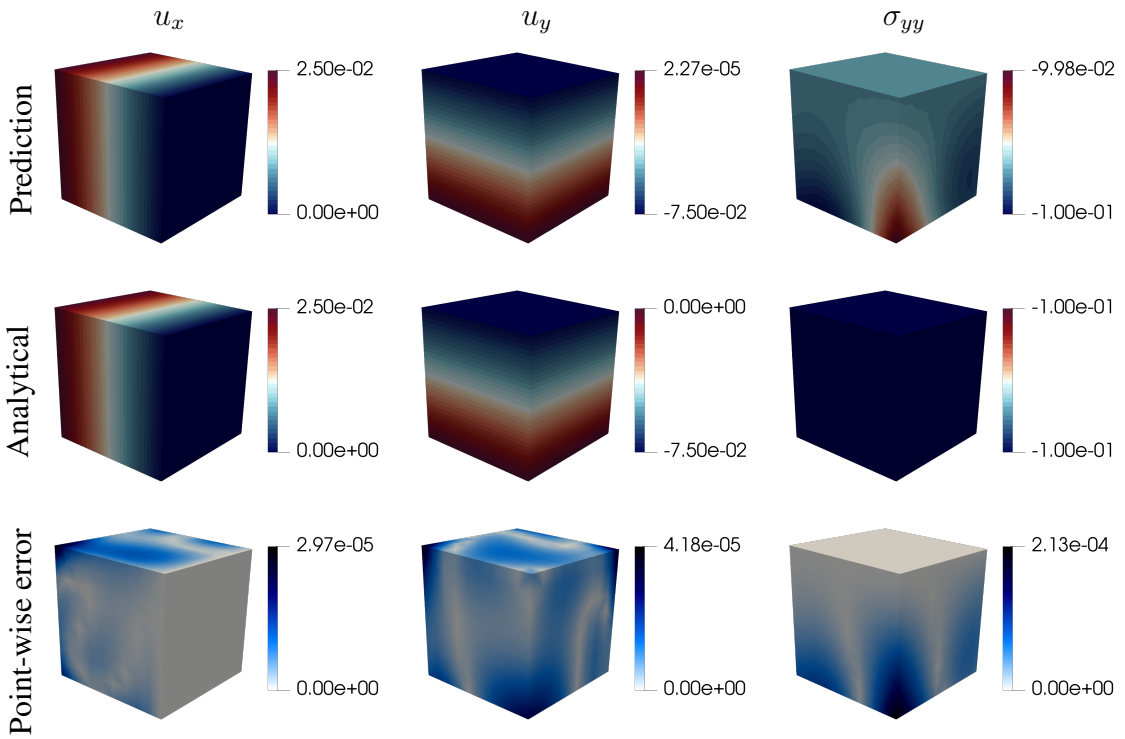


Figure 6.41.: Comparison of the PINN predictions and the analytical solution for the single contact patch test. Point-wise absolute error is defined as $E^*_{\text{abs}} = \text{abs}(\tilde{*} - *)$.

and u_y, as the behavior of u_z is identical to that of u_x. Both predicted displacement fields exhibit a linear distribution that closely matches the analytical solution. While the contact constraints are imposed softly, the displacement component u_y approaches zero near the contact surface, reflecting the expected behavior. Thanks to the output transformation, the boundary conditions at $x = 0$ and $z = 0$ are enforced as hard constraints, ensuring that the corresponding displacement components are strictly zero. This accuracy is further supported by point-wise error analysis, which confirms the close agreement between predictions and the analytical solution. Similarly, the normal stress component in the y-direction, σ_{yy}, remains close to the expected value of -0.1 across the entire domain, consistently aligning with the analytical solution.

6.3.3.4. 3D cylindrical contact problem

Now, we consider a finite linear elastic cylinder of height w and radius R lying on a rigid flat domain under a contact force F as depicted in Fig. 6.42a. As depicted in Fig. 6.42b, the contact area is a rectangle of a width of $2b$ and length of w with an elliptical pressure distribution. Due to the axisymmetric nature of both geometry and loading, the equivalent problem under a uniform

pressure p is given in Fig. 6.42c The analytical solution is expressed as [35]

$$\sigma_{xx} = -p_{\max}\left(\frac{1+2\frac{y^2}{b^2}}{\sqrt{1+\frac{y^2}{b^2}}} - 2\left|\frac{y}{b}\right|\right), \tag{6.106}$$

$$\sigma_{yy} = \frac{-p_{\max}}{\sqrt{1+\frac{y^2}{b^2}}}, \tag{6.107}$$

$$\sigma_{zz} = -2\nu p_{\max}\left(\sqrt{1+\frac{y^2}{b^2}} - \left|\frac{y}{b}\right|\right), \tag{6.108}$$

where

$$p_{\max} = \frac{2F}{\pi b w}, \tag{6.109}$$

$$b = \sqrt{\frac{2F}{\pi w}\frac{(1-\nu^2)/E}{1/(2R)}}, \tag{6.110}$$

$$F = 2Rwp. \tag{6.111}$$

By inserting $p = 0.5$, $R = 1$, $w = 1$, $\nu = 0.3$ and $E = 200$, the half-width of the contact area can be computed as $b = 0.076$, and the maximum pressure is calculated as $p_{\max} = 8.36$. Additionally, the maximum shear stress can be expressed as

$$\tau_{\max} = \frac{\sigma_{zz} - \sigma_{yy}}{2} \quad \text{for} \quad 0 \leq y \leq 0.436b, \tag{6.112}$$

$$\tau_{\max} = \frac{\sigma_{xx} - \sigma_{yy}}{2} \quad \text{for} \quad y \geq 0.436b. \tag{6.113}$$

Similarly, the following output transformation is applied to impose the Dirichlet and Neumann boundary conditions (BCs) shown in Fig. 6.42d as hard constraints:

$$\tilde{u}_x = \frac{x}{E}\mathcal{N}_{u_x},\ \tilde{u}_y = \frac{1}{E}\mathcal{N}_{u_y},\ \tilde{u}_z = \frac{z(w+z)}{E}\mathcal{N}_{u_z},\ \tilde{\sigma}_{yy} = -p + (-y)\mathcal{N}_{\sigma_{yy}},$$
$$\tilde{\sigma}_{xy} = xy\mathcal{N}_{\sigma_{xy}},\ \tilde{\sigma}_{yz} = (w+z)zy\mathcal{N}_{\sigma_{yz}},\ \tilde{\sigma}_{xz} = (w+z)zx\mathcal{N}_{\sigma_{xz}}.$$

This output transformation ensures that the DBCs and NBCs on the flat surfaces are strictly enforced. For instance, $sigma_{yy}$ is equal to $-p$ at the top of the quarter-cylinder. To simplify the optimization process, the displacements are scaled using the inverse of Young's modulus. Moreover, NBCs on the curved surface are enforced using soft constraints due to the coupling between the normal and shear stress components on those surfaces. To define the potential contact area, we set $\alpha = 15^\circ$ and enforce the contact constraints exclusively on the points lying on that surface (highlighted in red in Fig 6.42d).

In the following, we explore two distinct applications of PINNs for the cylindrical contact problem. In the first approach, a PINN is used solely as a PDE solver, referred to as the *plain vanilla PINN*. In the second approach, a data-enhanced PINN is employed, incorporating

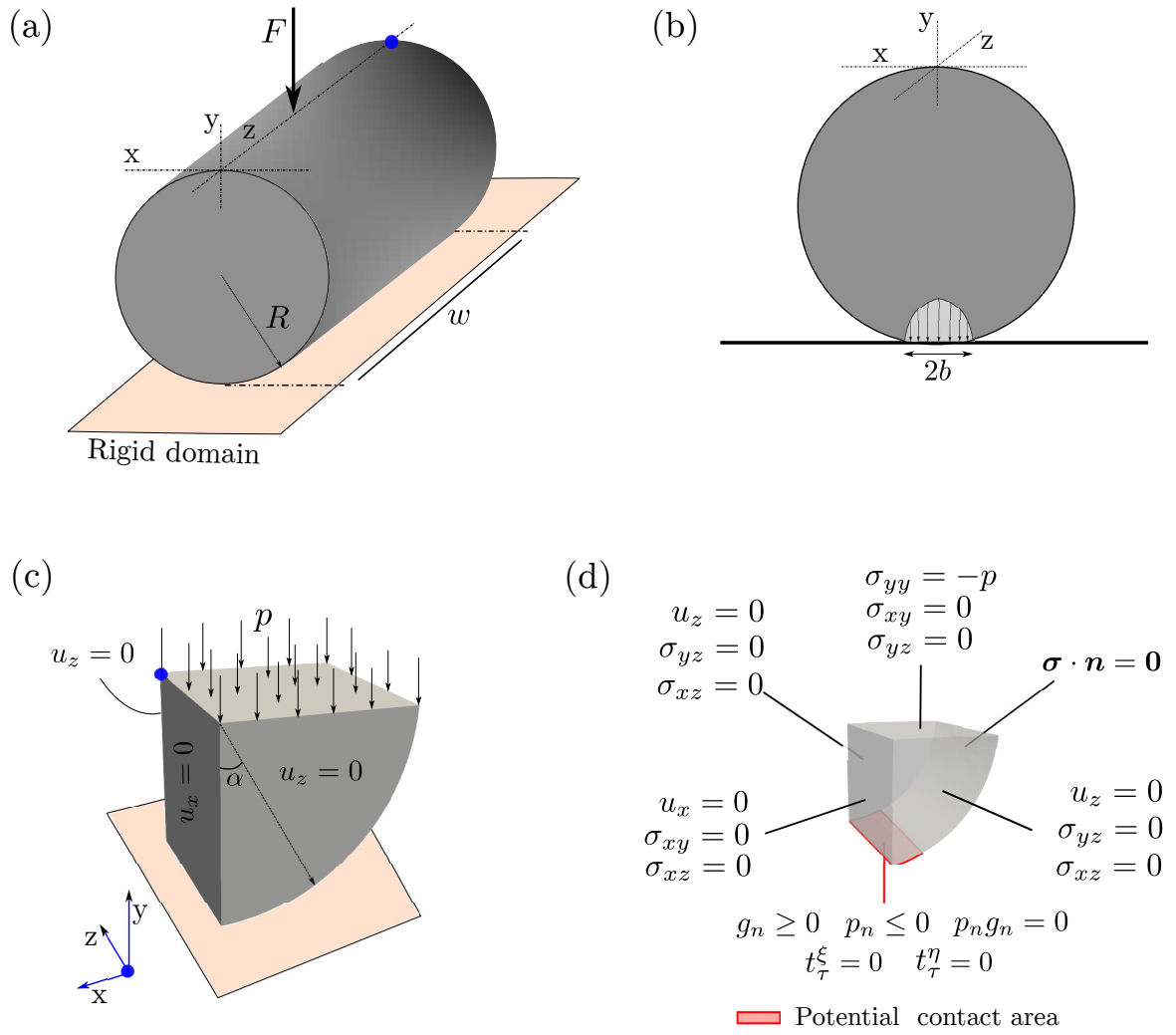


Figure 6.42.: The Hertzian contact problem between an elastic quarter-cylinder and a rigid flat domain: (a) The full contact problem under load F, including geometry and dimensions, (b) the width of the contact zone, (c) the equivalent problem due to the axisymmetrical nature of both geometry and loading (d) all relevant boundary conditions.

additional data points derived from the analytical solution (see Eqs. 6.106-6.108). A total of 150 supplementary data points are distributed along three different lines, each containing 50 points, positioned at $z = -1$, $z = -0.5$, and $z = 0$. Each line extends from $y = -1$ to $y = -0.7642$.

Additionally, the loss weight for the KKT term in Eq. 6.101 is set to $w_{\mathrm{KKT}} = 500$, ensuring that the PINN model emphasizes learning the contact constraints. The neural network architecture remains consistent with that used in the previous example.

The predictions of the plain vanilla PINN for the cylindrical contact problem are shown in Fig. 6.43. These predictions include the displacement fields and the stress fields in both the normal and shear directions. Furthermore, the normal components of the stress tensor are represented in cylindrical coordinates to illustrate the normal contact pressure. It can be observed that the output transformation explicitly enforces the displacement and stress field components on the target surfaces, such as $u_x(0, y, z) = 0$. Additionally, the maximum pressure is found to be $\sigma_{rr} = 7.44$, which differs from the maximum analytical pressure of $p_{\max} = 8.36$ (Eq. 6.109).

The PINN predictions for the cylindrical contact problem with data enhancement are presented in Fig. 6.45. Similarly, these predictions include the displacement fields, stress fields in both normal and shear directions, and stress fields in cylindrical coordinates.

The results of the data-enhanced model exhibit similar qualitative distributions to those of the plain vanilla PINN model. However, in contrast to the plain vanilla PINN, the maximum pressure achieves a value of $\sigma_{rr} = 8.38$, which is remarkably close to the maximum analytical pressure of $p_{\max} = 8.36$.

Additionally, a comparison between the PINN predictions and the analytical solution for both the plain vanilla and data-enhanced PINNs is shown in Fig. 6.44. The plain vanilla PINN tends to underpredict the analytical solution, particularly in regions with higher nonlinearity. In contrast, the data-enhanced PINN exhibits significantly improved performance, closely matching the analytical solution.

This improvement is further corroborated by the relative L_2 errors provided in Table 6.11. For example, the relative error $E_{L_2}^{\sigma_{yy}}$ is calculated as 5.29% for the plain vanilla PINN, whereas it is drastically reduced to 0.17% for the data-enhanced PINN, highlighting the impact of data enhancement. It is important to note that the predictions are evaluated at unseen data points along the line $z = -0.75$, demonstrating the model's generalization capabilities.

	$E_{L_2}^{\sigma_{xx}}$ (%)	$E_{L_2}^{\sigma_{yy}}$ (%)	$E_{L_2}^{\sigma_{zz}}$ (%)	$E_{L_2}^{\tau_{\max}}$ (%)
Plain vanilla PINN	3.81	5.29	3.31	7.41
Data-enhanced PINN	0.15	0.17	0.16	0.18

Table 6.11.: Comparison of relative L_2 errors (%) between the predictions and the analytical solution for the plain vanilla PINN and the data-enhanced PINN.

6.4. Use Case 4: Large deformations

This section introduces a systematic approach for developing PINN frameworks tailored to the modeling of large deformation problems in solid and contact mechanics, specifically under finite strain hyperelasticity.

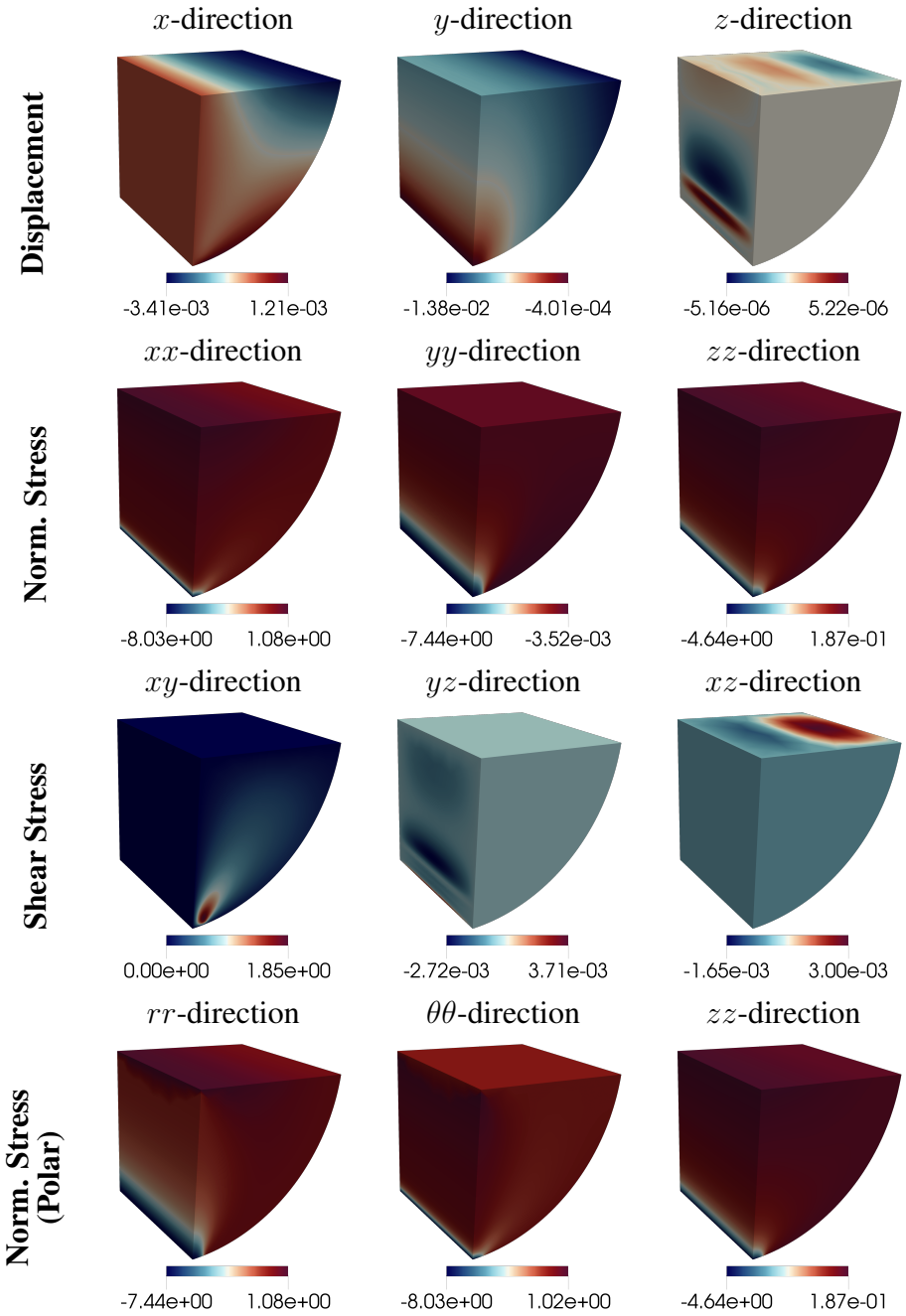


Figure 6.43.: The predictions of the plain vanilla PINN for the cylindrical contact example. Polar components are computed based on the cylindrical coordinates.

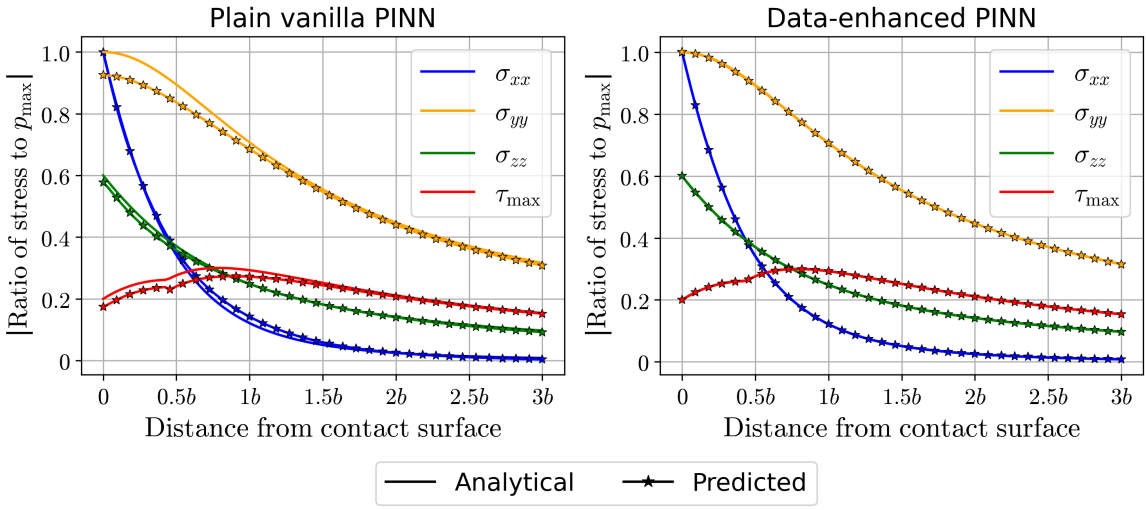


Figure 6.44.: Comparison of the predictions with the analytical solution for the cylindrical contact problem between the plain vanilla PINN and data-enhanced PINN.

In contrast to the previous examples, where traditional PINN formulations were employed, the methodologies presented here are grounded in the principle of minimum potential energy (refer to Section 2.2.3) and are implemented using the element-based energy PINNs described in Section 5.2.2. This variational formulation enables a more physically consistent and numerically stable treatment of nonlinear mechanics problems.

We begin by formulating the EBE-PINN framework for finite strain solid mechanics, incorporating geometric nonlinearities. The approach is then extended to handle contact interactions, facilitating the simulation of large deformation contact problems within the same energy-based framework.

In the numerical examples, we present a diverse range of examples, including the 2D beam bending example, torsion of a square prism, spherical contact problem, and torus contact instability problem. Note that the investigated examples include only static problems.

6.4.1. EBE-PINNs for solid mechanics

For quasi-static finite strain solid mechanics, an FNN maps the given spatial coordinates $\boldsymbol{x}$ to the approximated displacement $\tilde{\boldsymbol{u}}$ (see Fig. 6.46) such that

$$\tilde{\boldsymbol{u}} := \left(\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\right)'. \tag{6.114}$$

An illustration of the EBE-PINN formulation for finite strain solid mechanics in static problems is presented in Fig. 6.46. In this approach, a FNN is employed to approximate the displacement field by taking spatial coordinates $\boldsymbol{x}$ as input. The network outputs a continuous representation of the displacement field, denoted by $\tilde{\boldsymbol{u}}$.

To compute the internal energy, automatic differentiation (AD) is employed to obtain the required spatial derivatives of $\tilde{\boldsymbol{u}}$, such as the deformation gradient $\boldsymbol{F}$. The resulting fields are then used to assemble the total potential energy functional, posed as an integral over the computational domain and discretized into elementwise contributions, as detailed in Section 5.2.2.

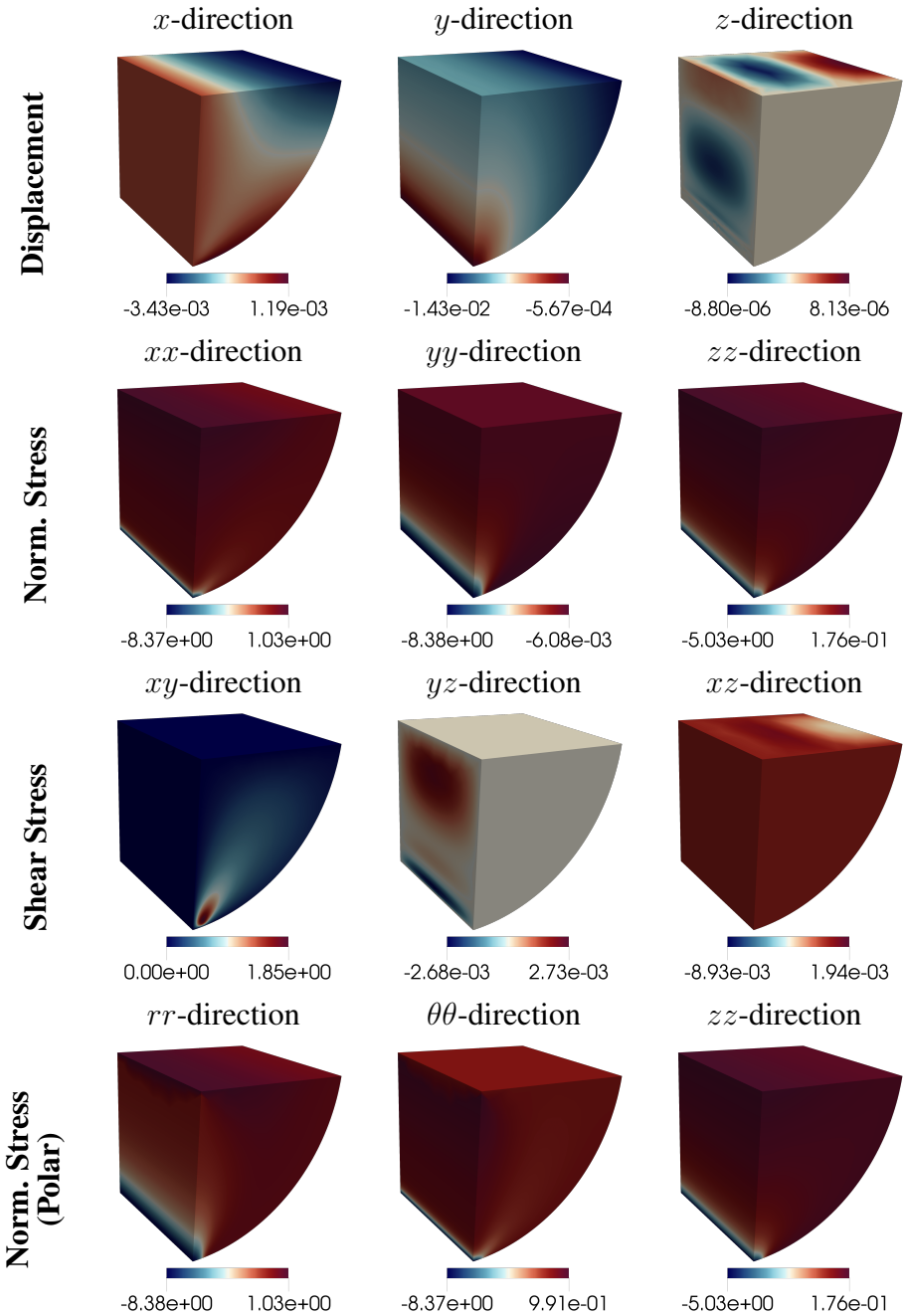


Figure 6.45.: The data-enhanced PINN predictions for the Hertzian contact example. Polar components are computed based on the cylindrical coordinates.

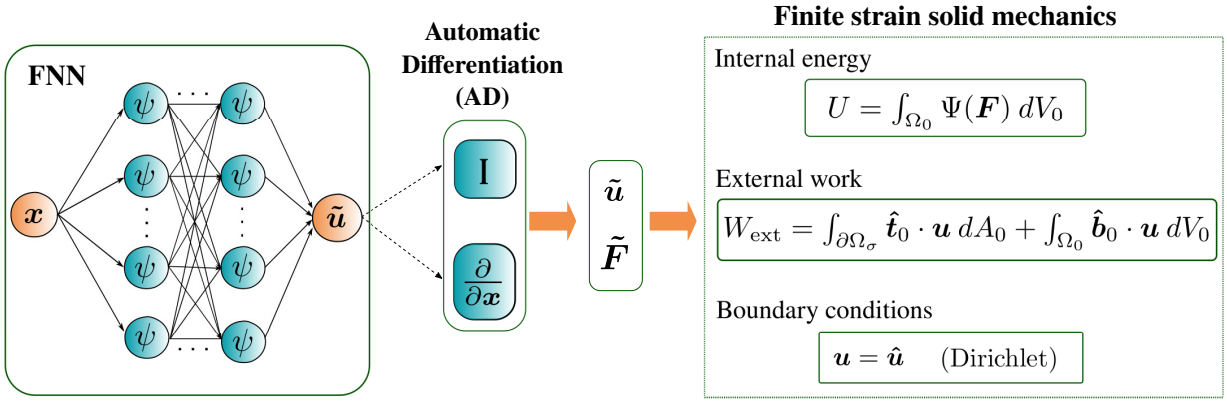


Figure 6.46.: Element-based energy physics-informed neural networks (EBE-PINNs) solve quasi-static finite strain solid mechanics problems.

By combining the information provided in Fig. 6.46 for losses of general EBE-PINNs with the governing equations of finite strain solid mechanics as explained in Section 2.2.3.3, we obtain the total loss $\mathcal{L}_{\text{S}}$, including additional experimental data as

$$\mathcal{L}_{\text{S}} = \mathcal{L}_{\mathbb{E}} + \mathcal{L}_{\text{DBCs}} + \mathcal{L}_{\text{EXPs}}, \tag{6.115}$$

where

$$\mathcal{L}_{\mathbb{E}} = w_{\mathbb{E}} \sum_{i_n=1}^{N^e} \mathbb{E}_{\text{S}}^e, \tag{6.116}$$

$$\mathcal{L}_{\text{DBCs}} = \sum_{i_{bc,D}=1}^{N_{bc,D}} w_{i_{bc,D}} \frac{1}{N_{bp,D}} \sum_{i_{bp,D}=1}^{N_{bp,D}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) - \hat{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{bp,D}} \right) \right]_{i_{bc,D}} \right]^2, \tag{6.117}$$

$$\mathcal{L}_{\text{EXPs}} = \sum_{i_{e,\boldsymbol{u}}=1}^{N_{e,\boldsymbol{u}}} w_{i_{e,\boldsymbol{u}}} \frac{1}{N_{ep,\boldsymbol{u}}} \sum_{i_{ep,\boldsymbol{u}}=1}^{N_{ep,\boldsymbol{u}}} \left[\left[\tilde{\boldsymbol{u}} \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) - \boldsymbol{u}^* \left(\boldsymbol{x}^{i_{ep,\boldsymbol{u}}} \right) \right]_{i_{e,\boldsymbol{u}}} \right]^2, \tag{6.118}$$

with

$$\mathbb{E}_{\text{S}}^e = \int_{\Omega^e} \Pi_{\text{S}}^e \left[\tilde{\boldsymbol{u}}(\boldsymbol{x}) \right] d\boldsymbol{x}. \tag{6.119}$$

Equations 6.117 and 6.118 have been introduced in the previous section. Here, N^e denotes the number of elements, Π_{S}^e is the element-wise strain energy density functional, $w_{\mathbb{E}}$ is the weighting coefficient applied to the potential energy loss term, and $\mathbb{E}_{\text{S}}^e$ represents the element-wise strain energy potential.

By explicitly introducing the internal energy and external work components as defined in Eqs. 2.109 and 2.110, we can expand the element-wise strain energy potential as

$$\mathbb{E}_{\text{S}}^e = \int_{\Omega_0^e} \Psi \left[\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\boldsymbol{x})) \right] dV_0 - \int_{\partial\Omega_{\sigma,0}^e} \hat{\boldsymbol{t}}_0 \cdot \tilde{\boldsymbol{u}}(\boldsymbol{x})\, dA_0 - \int_{\Omega_0^e} \hat{\boldsymbol{b}}_0 \cdot \tilde{\boldsymbol{u}}(\boldsymbol{x})\, dV_0. \tag{6.120}$$

Here, Ψ is the strain energy density function, $\tilde{\boldsymbol{F}}$ denotes the deformation gradient derived via automatic differentiation of the neural network displacement field, $\hat{\boldsymbol{t}}_0$ is the prescribed traction on the Neumann boundary $\partial\Omega_\sigma^e$, and $\hat{\boldsymbol{b}}_0$ denotes the body force per unit reference volume.

The strain energy density function for Neo-Hookean materials is expressed as

$$\Psi(\tilde{\boldsymbol{F}}) = \frac{\mu}{2}(\text{tr}(\tilde{\mathbf{C}}) - 3) - \mu \log \tilde{J} + \frac{\lambda}{2}(\log \tilde{J})^2, \tag{6.121}$$

with

$$\tilde{\mathbf{C}} = \tilde{\boldsymbol{F}}^\top \tilde{\boldsymbol{F}}, \tag{6.122}$$

$$\tilde{J} = \det \tilde{\boldsymbol{F}}, \tag{6.123}$$

$$\tilde{\boldsymbol{F}} = \boldsymbol{I} + \boldsymbol{\nabla}\tilde{\boldsymbol{u}}(\boldsymbol{x}). \tag{6.124}$$

This energy-based formulation naturally enforces mechanical equilibrium by minimizing the total potential energy of the system through all elements.

As the next step, the integrals in Eq. 6.120 must be evaluated using numerical integration, as discussed in Section 5.2.3. For problems defined in 3D space, volume integrals can be computed using numerical integration over hexahedral or tetrahedral elements. Similarly, surface integrals are evaluated using quadrature rules on quadrilateral or triangular surface elements. In the case of 2D problems, boundary integrals reduce to line integrals, which can be evaluated using numerical integration over 1D elements.

If the target problem is 3D and the domain is discretized using hexahedral elements, then the total potential energy of an element can be expressed as

$$\begin{aligned}\mathbb{E}^e_{\text{S}} = &\sum_{i_x=1}^{N_x^{gp}} \sum_{i_y=1}^{N_y^{gp}} \sum_{i_z=1}^{N_z^{gp}} c_{i_x} c_{i_y} c_{i_z}\, \Psi\big[\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta,\zeta)))\big] \det(\mathbf{J}) - \\ &\sum_{i_x=1}^{N_x^{gp}} \sum_{i_y=1}^{N_y^{gp}} c_{i_x} c_{i_y} \hat{\boldsymbol{t}}_0 \cdot \tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta)) \det(\mathbf{J}) - \sum_{i_x=1}^{N_x^{gp}} \sum_{i_y=1}^{N_y^{gp}} \sum_{i_z=1}^{N_z^{gp}} c_{i_x} c_{i_y} c_{i_z} \hat{\boldsymbol{b}}_0 \cdot \tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta,\zeta)) \det(\mathbf{J}),\end{aligned} \tag{6.125}$$

where N^{gp} denotes the number of Gauss points in each respective direction, c are the Gauss quadrature weights, and $\bar{\boldsymbol{x}}$ are the mapped physical coordinates obtained via linear isoparametric mapping (see Section 5.2.3). The term $\det(\mathbf{J})$ represents the determinant of the Jacobian matrix associated with the transformation from reference to physical coordinates. For specific expressions, refer to Eq. 5.42 and Eq. 5.55, which detail the Jacobian transformations for two- and three-dimensional settings, respectively. Note that, since the domain is discretized using hexahedral elements, surface integrals are naturally evaluated over quadrilateral elements, as these represent the projections of the hexahedral faces onto the boundary (with the surface projection taken along the z-axis in this representative case). In the representative case considered here, the surface projection is taken along the z-axis so that the projected plane is the z-plane. Depending on the chosen projection, the formulation of the surface integral can be adapted accordingly.

When the domain is discretized using tetrahedral elements, the total potential energy of an element can be expressed as

$$\begin{aligned}\mathbb{E}^e_{\text{S}} = &\frac{1}{6}\sum_{i=1}^{N^{gp}} c_i\, \Psi\big[\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta,\zeta)))\big] \det(\mathbf{J}) - \\ &\frac{1}{2}\sum_{i=1}^{N^{gp}} c_i\, \hat{\boldsymbol{t}}_0 \cdot \tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta)) \det(\mathbf{J}) - \frac{1}{6}\sum_{i=1}^{N_x^{gp}} c_i\, \hat{\boldsymbol{b}}_0 \cdot \tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta,\zeta)) \det(\mathbf{J}).\end{aligned} \tag{6.126}$$

In contrast to the hexahedral formulation, this expression includes explicit scaling factors: $1/6$ for volume integrals and $1/2$ for surface integrals, which account for the area and volume of the reference tetrahedral elements. Moreover, the Gauss quadrature weights are defined in a single dimension due to the simplex geometry. Similarly, since the domain is meshed with tetrahedral elements, surface integrals are naturally evaluated over triangular elements, which correspond to the projections of the tetrahedral faces onto the boundary.

In the next scenario, if the target problem is 2D and the domain is discretized using quadrilateral elements, then

$$\begin{aligned}\mathbb{E}_{\mathrm{S}}^{e} = &\sum_{i_x=1}^{N_x^{gp}}\sum_{i_y=1}^{N_y^{gp}} c_{i_x}c_{i_y}\ \Psi\big[\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta)))\big]\det(\mathbf{J}) - \\ &\sum_{i_x=1}^{N_x^{gp}} c_{i_x}\ \hat{\boldsymbol{t}}_0\cdot\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi))\det(\mathbf{J}) - \sum_{i_x=1}^{N_x^{gp}}\sum_{i_y=1}^{N_y^{gp}} c_{i_x}c_{i_y}\ \hat{\boldsymbol{b}}_0\cdot\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta))\det(\mathbf{J}).\end{aligned} \tag{6.127}$$

Here, the contribution of external work due to traction is evaluated over 1D elements, as the projection of a triangular face onto the boundary is a line segment. When the domain is discretized using triangular elements, the total potential energy of an element can be expressed as

$$\begin{aligned}\mathbb{E}_{\mathrm{S}}^{e} = &\frac{1}{2}\sum_{i=1}^{N^{gp}} c_i\ \Psi\big[\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta)))\big]\det(\mathbf{J}) - \\ &\sum_{i_x=1}^{N_x^{gp}} c_{i_x}\ \hat{\boldsymbol{t}}_0\cdot\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi))\det(\mathbf{J}) - \frac{1}{2}\sum_{i=1}^{N^{gp}} c_i\ \hat{\boldsymbol{b}}_0\cdot\tilde{\boldsymbol{u}}(\bar{\boldsymbol{x}}(\xi,\eta))\det(\mathbf{J}).\end{aligned} \tag{6.128}$$

As with the tetrahedral case, surface integration in triangular elements includes a scaling factor of $1/2$, corresponding to the area of the reference triangle. Additionally, note that triangular and tetrahedral elements do not have directions for the Gauss points.

6.4.2. EBE-PINNs for contact mechanics

Analogous to solid mechanics, in the case of quasi-static finite strain contact mechanics, an FNN maps the spatial coordinates $\boldsymbol{x}$ to the predicted displacement field $\tilde{\boldsymbol{u}}$, as illustrated in Fig. 6.47

$$\tilde{\boldsymbol{u}} := \left(\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\right)'. \tag{6.129}$$

In the quasi-static case, the neural network input involves only the spatial coordinates $\boldsymbol{x}$ as input. Fig. 6.47 illustrates the EBE-PINN framework tailored for large deformation contact problems. Compared to the finite strain solid mechanics formulation, this framework additionally incorporates a contact potential, modeled here using the penalty method as introduced in Section 2.3.3.

For frictionless contact under large deformation, the total loss function $\mathcal{L}_{\mathrm{C}}$ is defined as

$$\mathcal{L}_{\mathrm{C}} = \mathcal{L}_{\mathbb{E}} + \mathcal{L}_{\mathrm{DBCs}} + \mathcal{L}_{\mathrm{EXPs}}, \tag{6.130}$$

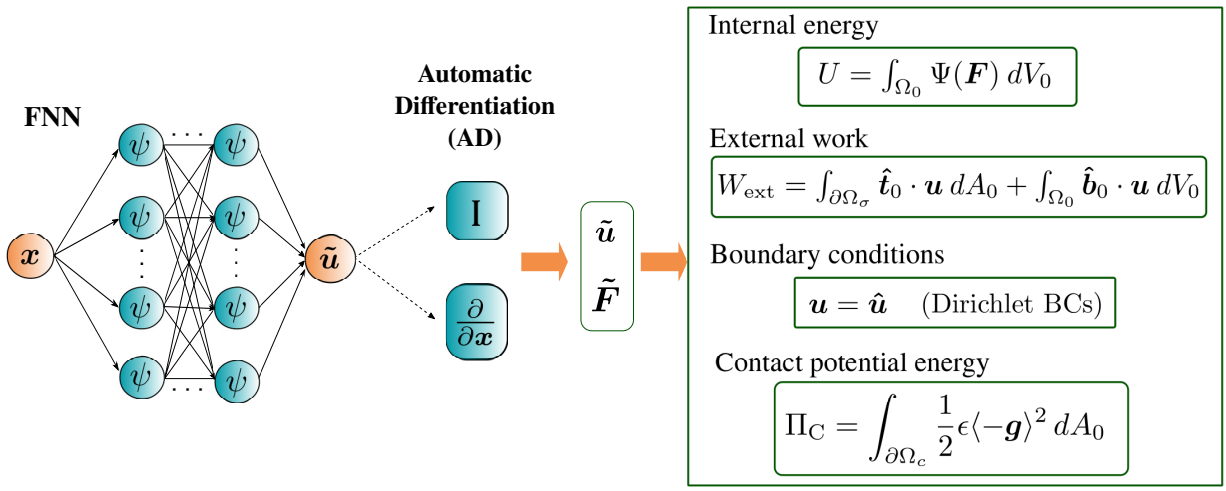


Figure 6.47.: Element-based energy physics-informed neural networks (EBE-PINNs) solve quasi-static finite strain contact mechanics problems.

where the energy loss term is given by

$$\mathcal{L}_{\mathbb{E}} = w_{\mathbb{E}} \sum_{i_n=1}^{N^e} \mathbb{E}_{\text{C}}^e, \tag{6.131}$$

with the element-wise integrated energy defined as

$$\mathbb{E}_{\text{C}}^e = \int_{\Omega^e} \Pi_{\text{S}}^e\big[\tilde{\boldsymbol{u}}(\boldsymbol{x})\big]\, d\boldsymbol{x} + \int_{\partial\Omega_c^e} \Pi_{\text{C}}^e\big[\tilde{\boldsymbol{u}}(\boldsymbol{x})\big]\, d\boldsymbol{x}. \tag{6.132}$$

The first term in Eq. 6.132 accounts for the strain or internal energy due to finite strain deformation, as discussed in Section 6.4.1. In this section, we focus on the second term, which represents the contribution of the contact potential. As shown in Fig. 6.47, the element-wise contact potential energy is expressed using the penalty formulation (see Eq. 2.127) as

$$\frac{1}{2} \int_{\partial\Omega_c^e} \epsilon\langle -\tilde{\boldsymbol{g}}\rangle^2\, dA_0, \tag{6.133}$$

where the predicted gap vector is defined as

$$\tilde{\boldsymbol{g}} = \boldsymbol{X} + \tilde{\boldsymbol{u}} - \hat{\boldsymbol{x}}. \tag{6.134}$$

Following the same approach used for evaluating the strain energy, the contact potential integral in Eq. 6.133 must also be computed numerically. In 3D problems, the contact surface integral is evaluated using quadrature rules on quadrilateral or triangular surface elements. For 2D problems, the surface integral reduces to a line integral, which is numerically computed over 1D boundary elements.

For 3D problems, depending on the surface element type, the contact potential energy per element is computed as

$$\begin{aligned} &\frac{1}{2}\sum_{i_x=1}^{N_x^{gp}}\sum_{i_y=1}^{N_y^{gp}} c_{i_x}c_{i_y}\epsilon\langle-\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}(\xi,\eta))\rangle^2\det(\mathbf{J}), \quad \rightarrow \text{for quad elements} \\ &\frac{1}{2}\frac{1}{2}\sum_{i=1}^{N^{gp}} c_i\epsilon\langle-\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}(\xi,\eta))\rangle^2\det(\mathbf{J}). \quad \rightarrow \text{for triangular elements} \end{aligned} \tag{6.135}$$

For 2D problems, the contact potential energy is computed over a 1D boundary, regardless of the element type as

$$\frac{1}{2}\sum_{i_x=1}^{N_x^{gp}} c_{i_x}\ \epsilon\langle-\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}(\xi,\eta))\rangle^2\det(\mathbf{J}). \tag{6.136}$$

6.4.3. Numerical examples

In this section, we present four numerical examples based on quasi-static large deformation theory to demonstrate the capabilities and performance of the proposed framework.

The first example considers a 2D bending beam, serving as a baseline model to validate the formulation under plane strain assumptions.

The second example investigates the torsion of a square prism, highlighting the framework's ability to handle 3D geometries. In this case, all boundary conditions are enforced via hard constraints. Furthermore, this example illustrates the potential sources of discrepancy between FEM results and those obtained using the EBE-PINN approach.

The third example focuses on a 3D large deformation contact problem involving spherical contact. This scenario showcases the method's effectiveness in resolving contact interactions under significant geometrical nonlinearity.

Finally, the fourth example examines the contact instability of a torus geometry, which serves as a challenging benchmark to evaluate the robustness of EBE-PINNs in handling unstable contact behavior.

Unless stated otherwise, all numerical examples follow a consistent configuration. The neural networks are initialized using the *Glorot uniform* initializer, and the tanh function is employed as the activation function. The models are first trained using the Adam optimizer [91] with a learning rate of $lr = 0.001$ for a predefined number of epochs. Subsequently, the training is refined using the L-BFGS-B optimizer [195] with box constraints, a quasi-Newton method known for its fast convergence [36, 105].

Furthermore, all simulations are conducted assuming hyperelastic material behavior modeled by a Neo-Hookean formulation. Body forces are neglected unless explicitly stated otherwise.

6.4.3.1. 2D bending beam

In the first example, we investigate a 2D bending beam subjected to shear loading, as illustrated in Fig. 6.48a. The beam has a length of $L = 20$ and a height of $h = 2$. The left boundary at $x = 0$ is fully constrained in both the x- and y-directions. A uniform shear loading of magnitude

$t_{0y} = -0.01$ is applied on the right edge in the vertical direction. The material is characterized by Young's modulus $E = 10$ and Poisson's ratio $\nu = 0.2$.

The employed PINN architecture maps spatial coordinates (x, y) to the displacement components $(\mathcal{N}_{u_x}, \mathcal{N}_{u_y})$. The network consists of a fully connected feedforward neural network with 5 hidden layers, each containing 64 neurons. Training is performed using $1{,}248$ quadrilateral elements, as shown in Fig. 6.48b. For numerical integration, we employ $2 \times 2 = 4$ Gauss points for surface integrals and 4 Gauss points for line integrals. Additionally, the model is trained for $5{,}000$ epochs using the Adam optimizer, and we switch to L-BFGS-B to smooth the solution.

To impose the Dirichlet boundary conditions as hard constraints, we apply the following output transformation:

$$\tilde{u}_x = \frac{x}{L}\mathcal{N}_{u_x}, \quad \tilde{u}_y = \frac{x}{L}\mathcal{N}_{u_y}. \tag{6.137}$$

This formulation ensures that the displacements vanish at the fixed boundary ($x = 0$), satisfying the prescribed boundary conditions exactly. Additionally, the normalization by the beam length L introduces a non-dimensional scaling, which helps improve the conditioning of the optimization problem and mitigates numerical instabilities during training.

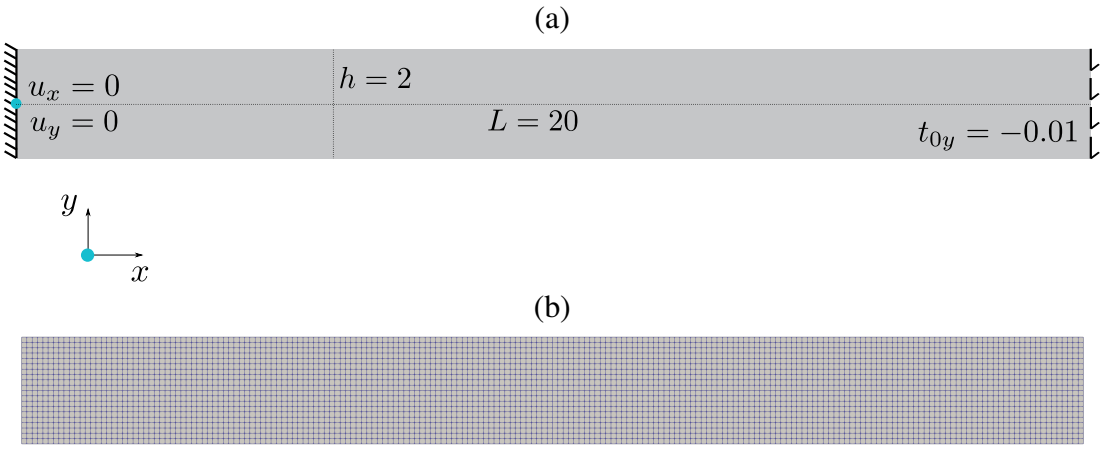


Figure 6.48.: 2D bending beam under shear loading. (a) Geometry, boundary conditions, and applied traction, (b) corresponding mesh with quadrilateral elements.

Fig. 6.49 presents a comparison between the displacement fields predicted by the EBE-PINN and the FEM reference solution for the 2D beam under shear loading. The displacement components u_x and u_y obtained from both methods show good qualitative and quantitative agreement, as indicated by the nearly identical color distributions in the top two rows. The bottom row illustrates the corresponding point-wise absolute errors. The maximum absolute errors are on the order of 10^{-3} for both u_x and u_y, which confirms the accuracy of the EBE-PINN framework. As expected, larger errors are concentrated in regions where larger displacements occur, as the deformation gradients are more pronounced and harder to approximate. Furthermore, the computational efficiency of the EBE-PINN model is noteworthy: while the FEM solution requires approximately $232\,\mathrm{s}$, the EBE-PINN training takes only $25\,\mathrm{s}$, offering a substantial reduction in computation time. This highlights the potential of EBE-PINNs as a fast and reliable alternative for solving nonlinear solid mechanics problems.

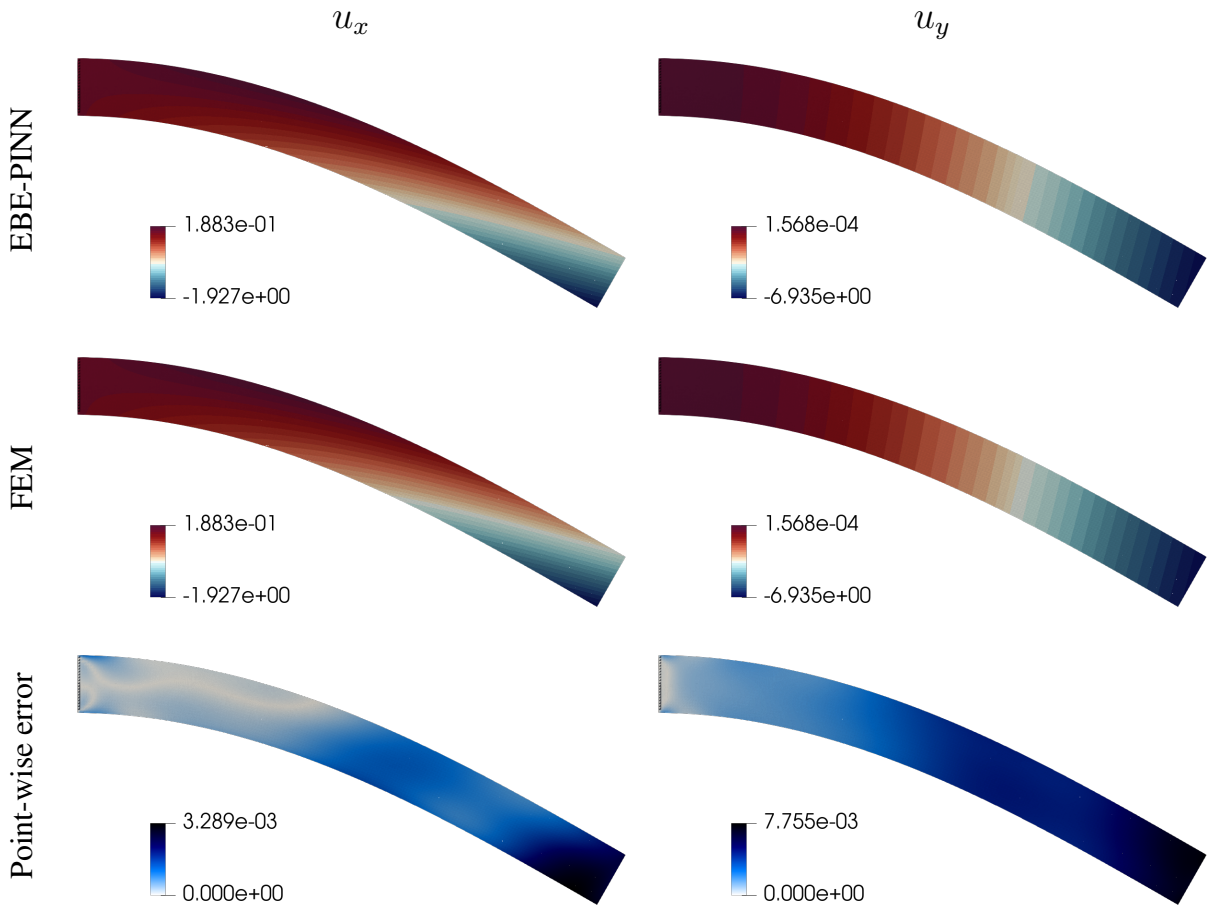


Figure 6.49.: Comparison of predicted and reference displacement components (u_x, u_y) for the 2D beam under shear loading between EBE-PINN and FEM. The bottom row shows the corresponding point-wise absolute errors, defined as $E^*_{abs} = \mathrm{abs}(\tilde{*} - *)$, where $\tilde{*}$ denotes the predicted quantity.

Fig. 6.50 compares the stress components σ_{xx}, σ_{yy}, and σ_{xy} predicted by the EBE-PINN and FEM methods for the 2D beam under shear loading. While the EBE-PINN results exhibit good overall agreement with the FEM reference across all stress components, the accuracy is slightly lower compared to the displacement fields. This is primarily due to the presence of stress concentrations near boundary edges or corners, where the deformation gradients are sharp and more challenging to approximate. The point-wise absolute error plots confirm that the largest discrepancies are confined to these localized regions, with maximum errors reaching the order of 10^{-1}. In contrast, the error remains below 10^{-2} across most of the domain. Despite these localized differences, the EBE-PINN model still captures the global stress distribution with reasonable fidelity, validating its effectiveness for large deformation problems. Quantitatively, the average absolute errors are computed to be approximately 1.8×10^{-3} for σ_{xx}, 2.2×10^{-3} for σ_{yy}, and 1.0×10^{-3} for σ_{xy}, indicating that while EBE-PINNs achieve good global agreement. However, special attention may be needed in high-gradient regions, where adaptive refinement strategies could further improve accuracy.

As depicted in Fig. 6.49, the displacement predictions from EBE-PINNs exhibit a closer match with FEM results compared to the stress components shown in Fig. 6.50. This discrepancy is primarily attributed to stress concentrations that arise near boundary edges and corners–regions where stress gradients are steep and numerical sensitivity is high. It is well known that even in conventional FEM, stress concentrations lead to localized inaccuracies unless highly refined meshes are used. In the case of EBE-PINNs, Cauchy stresses are recovered in a post-processing step from the predicted displacements. Therefore, when displacement fields are accurate, the associated stress fields are generally expected to align well. However, due to the nature of energy-based formulations and the absence of direct stress supervision, EBE-PINNs, like FEM, may exhibit stress discontinuities or jumps, particularly at geometric singularities or sharp edges. This highlights the need for improved stress recovery techniques or localized refinement strategies to enhance stress accuracy in such regions.

6.4.3.2. 3D torsion of a square prism

In the second example, we analyze the torsional deformation of a square prism, as illustrated in Fig. 6.51a. The prism has a length of $L = 4$, a height of $h = 1$, and a width of $w = 1$. The surface at $x = 0$ is fully constrained in all three spatial directions (x-, y-, and z-directions). The material is modeled using Young's modulus $E = 1.33$ and Poisson's ratio $\nu = 0.33$.

To induce torsion, a prescribed displacement is applied at the front face ($x = L$), corresponding to a rigid rotation about the x-axis. The displacement field is defined as:

$$\hat{\boldsymbol{u}}_L = \begin{bmatrix} 0 \\ y(\cos\theta - 1) - z\sin\theta \\ y\sin\theta + z(\cos\theta - 1) \end{bmatrix}, \tag{6.138}$$

where $\theta = 150^\circ$ represents the applied torsional angle.

The PINN architecture maps the spatial coordinates (x, y, z) to the displacement components $(\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_z})$. The model employs a fully connected feedforward neural network consisting of 5 hidden layers with 50 neurons each. Training is conducted using $4{,}961$ hexahedral elements, as shown in Fig. 6.51b. Numerical integration is performed using $2 \times 2 \times 2 = 8$ Gauss points for

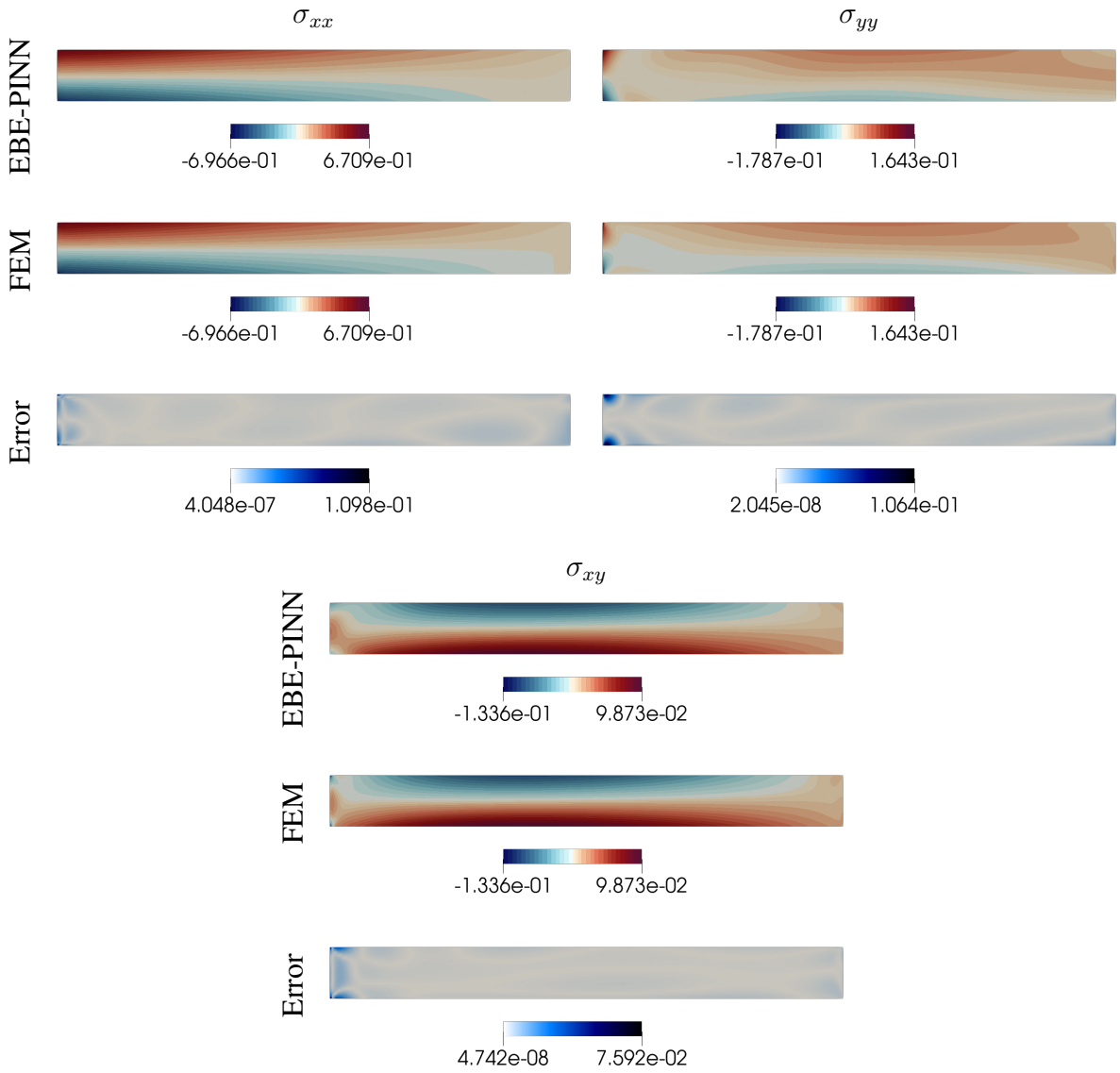


Figure 6.50.: Comparison of predicted and reference Cauchy stress components (σ_{xx}, σ_{yy}, σ_{xy}) for the 2D beam under shear loading using the EBE-PINN and FEM. Similar to displacement results, corresponding point-wise absolute errors are defined as $E^*_{abs} = \mathrm{abs}(\tilde{*} - *)$, where $\tilde{*}$ denotes the predicted quantity.

volume integrals and $2 \times 2 = 4$ Gauss points for surface integrals. The model is trained for $3{,}000$ epochs using the Adam optimizer and we switch to L-BFGS-B to smooth the solution.

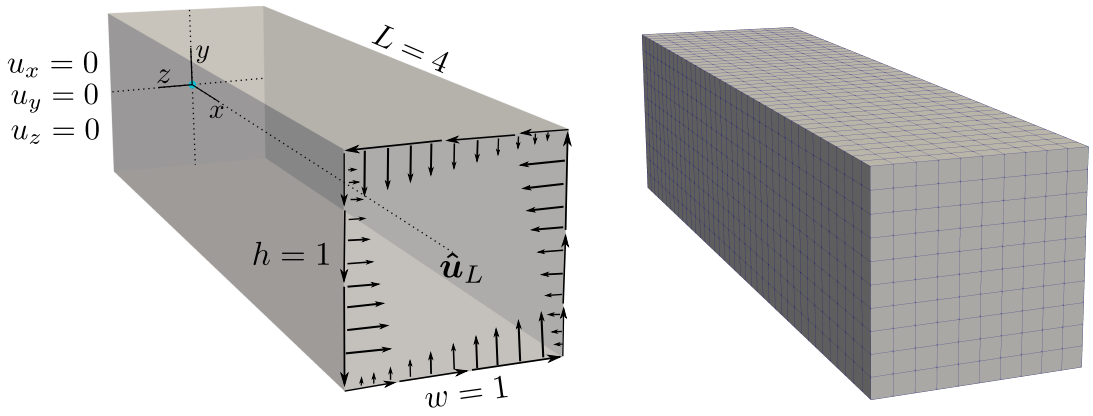
To impose the Dirichlet boundary conditions as hard constraints, we use the following output transformation:

$$\tilde{\boldsymbol{u}} = s\,\hat{\boldsymbol{u}}_L + s(1-s)\,\mathcal{N}_{\boldsymbol{u}}, \tag{6.139}$$

where the scaling parameter is defined as $s = x/L$. Substituting the prescribed displacement field from Eq. 6.138, the transformed displacement vector becomes as

$$\begin{bmatrix} \tilde{u}_x \\ \tilde{u}_y \\ \tilde{u}_z \end{bmatrix} = \begin{bmatrix} \dfrac{x}{L}(1-\dfrac{x}{L})\mathcal{N}_{u_x} \\ \dfrac{x}{L}\left[y(\cos\theta - 1) - z\sin\theta\right] + \dfrac{x}{L}(1-\dfrac{x}{L})\mathcal{N}_{u_y} \\ \dfrac{x}{L}\left[y\sin\theta + z(\cos\theta - 1)\right] + \dfrac{x}{L}(1-\dfrac{x}{L})\mathcal{N}_{u_z} \end{bmatrix}.$$

This transformation ensures that the displacements vanish at $x = 0$, satisfying the essential boundary condition exactly. Simultaneously, it enforces the prescribed rotation at $x = L$, while allowing the interior displacement field to be identified by the neural network.



(a) (b)

Figure 6.51.: Torsion of a prism. (a) Geometry, boundary conditions, and loading; (b) corresponding mesh with hexahedral elements.

Fig. 6.52 compares the displacement components (u_x, u_y, u_z) predicted by EBE-PINN and FEM for the square prism under torsion, visualized in the deformed configuration. The EBE-PINN predictions (top row) closely match the FEM reference results (middle row), demonstrating that the network accurately captures the twisting deformation characteristic of this benchmark problem. The displacement distributions are nearly identical in both magnitude and pattern across all three components. The bottom row presents the point-wise absolute errors, which are generally small. Maximum absolute errors are on the order of 10^{-3} for all displacement components, with slightly higher values observed near the free surface and sharp edges – regions where gradients are sharper and stress concentrations typically occur.

Fig. 6.53 presents a comparison of the Cauchy stress components $(\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz})$ computed using EBE-PINN and FEM for the torsion of the square prism. The first and second

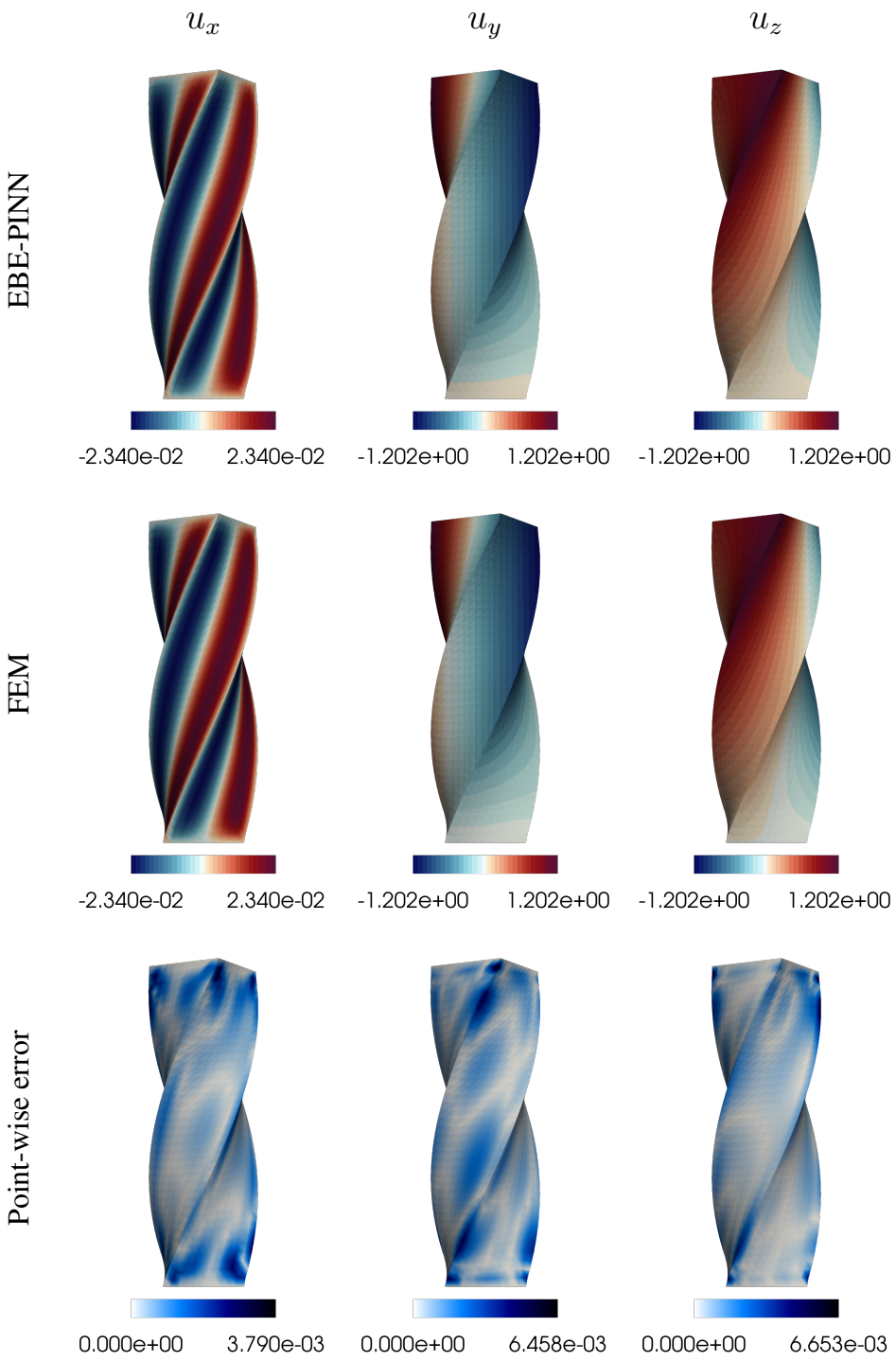


Figure 6.52.: Predicted and reference displacement components (u_x, u_y, u_z) for the torsion of the prism example shown in the deformed configuration. Results from EBE-PINN and FEM are shown in the first and second rows, respectively. The third row presents the corresponding point-wise absolute errors.

rows show the results from EBE-PINN and FEM, respectively, while the third row displays the corresponding point-wise absolute errors. Overall, the EBE-PINN stress fields exhibit good visual agreement with the FEM reference across all components. However, the errors are noticeably higher than those observed in the displacement predictions (cf. Fig. 6.52).

Similar to the bending beam example, the error distributions reveal that the largest discrepancies in stress predictions occur along the outer surfaces and near the regions experiencing significant twisting, where stress gradients and rotational deformations are most pronounced. In particular, shear stress components such as σ_{xy} and σ_{xz} exhibit localized absolute errors on the order of 10^{-2}, while the normal stress component σ_{xx} shows even higher localized errors, reaching up to 10^{-1}. Despite these localized inaccuracies, the EBE-PINN framework successfully captures the overall multiaxial stress distribution under large deformation.

Quantitatively, the average absolute errors are computed as approximately 4.7×10^{-3} for σ_{xx}, 4.8×10^{-3} for σ_{yy}, 4.6×10^{-3} for σ_{zz}, 3.7×10^{-3} for σ_{xy}, 1.7×10^{-3} for σ_{yz}, and 3.8×10^{-3} for σ_{xz}. These values confirm that EBE-PINNs deliver good global agreement with FEM.

To investigate the stress concentration phenomena in greater detail, we extract and evaluate field quantities along a line segment extending from one sharp corner to another, specifically from $(2, -0.5, -0.5)$ to $(2, 0.5, 0.5)$. Along this line, we plot the displacement, normal and shear stress components (see Fig. 6.54), as well as the Green-Lagrange strain components (see Fig. 6.55).

The displacement profiles exhibit excellent agreement between EBE-PINN and FEM, with the curves overlapping almost perfectly across the entire domain. However, deviations become apparent when examining the stress components. Notably, both normal and shear stresses show localized discrepancies near the endpoints of the line–regions that coincide with geometric corners and are prone to stress concentrations. While the overall stress trends are well-captured, the FEM solution exhibits slight oscillations near the corners, which are not present in the EBE-PINN results.

This pattern is also observed in the Green-Lagrange strain components. While EBE-PINN and FEM results align well through most of the domain, sharp kinks appear near the corners in the FEM curves. These kinks are not physically expected and are likely numerical artifacts caused by local stress singularities and mesh distortion in FEM due to large deformations. In contrast, the smooth nature of EBE-PINN predictions helps to suppress such behavior, delivering a cleaner representation of the underlying deformation field.

6.4.3.3. 3D spherical contact problem

In the third example, we increase the level of nonlinearity by incorporating contact constraints in addition to geometric nonlinearity. Our focus now shifts to contact problems, beginning with the classical spherical contact scenario illustrated in Fig. 6.56a. The problem considers a hyperelastic sphere subjected to a compressive force F applied vertically at its top. Thanks to the symmetry of both geometry and loading, we model only one-eighth of the sphere, as shown in the reduced domain in Fig. 6.56b.

The reduced domain has a radius of $R = 1$. To impose essential boundary conditions, the face at $z = 0$ is fixed in the z-direction, while the face at $x = 0$ is fixed in the x-direction. A uniform traction load $\hat{t}_{0y} = -5$ is applied along the top surface at $y = 0$. At the bottom, a rigid flat surface defines the contact interface. The material is modeled as hyperelastic with Young's modulus $E = 50$ and Poisson's ratio $\nu = 0.3$.

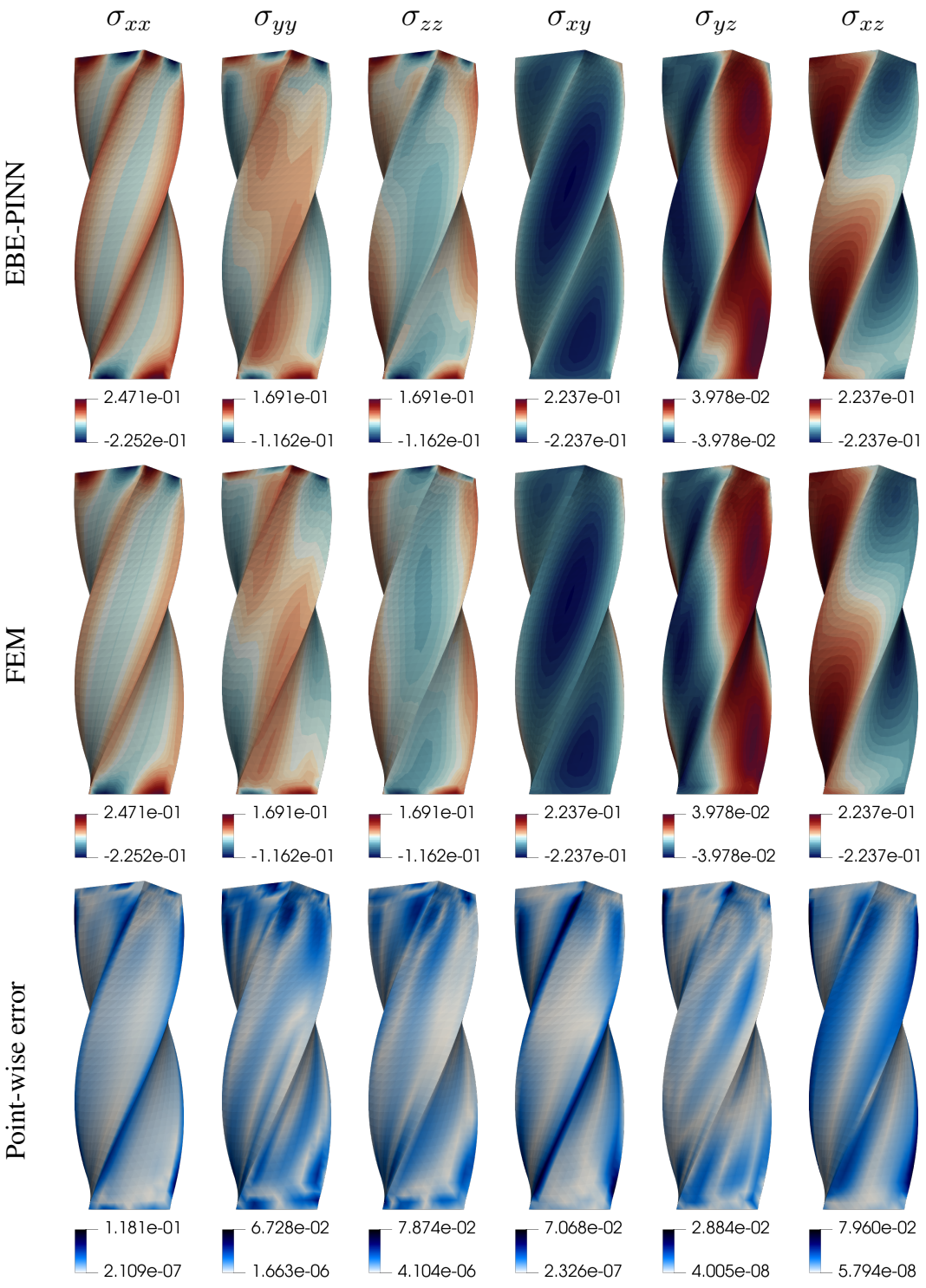


Figure 6.53.: Predicted and reference Cauchy stress components (σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{xz}) for the torsion of the prism example. Results from EBE-PINN and FEM are shown in the first and second rows, respectively. The third row presents the corresponding point-wise absolute errors.

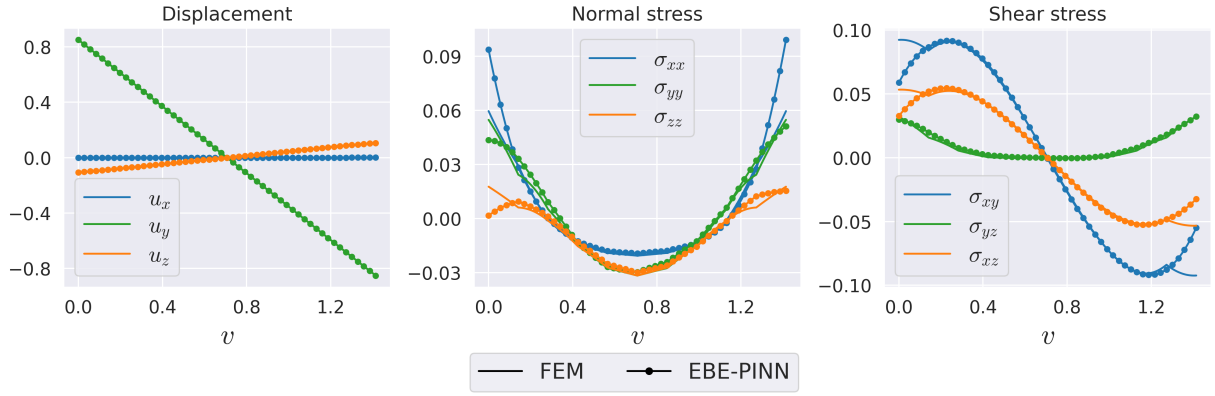


Figure 6.54.: Comparison of displacement and Cauchy stress components along a line in the 3D torsion problem between FEM and EBE-PINNs.

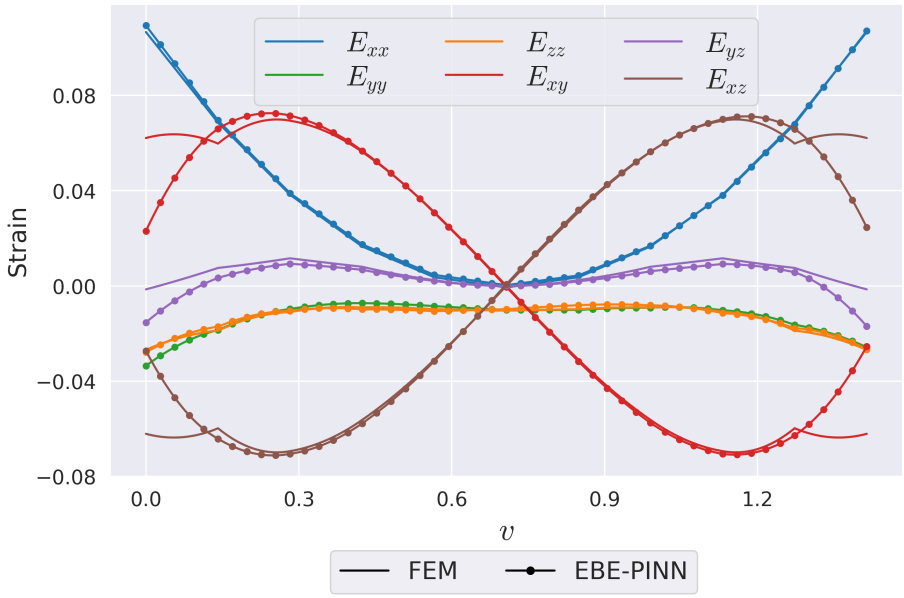


Figure 6.55.: Comparison of Green-Lagrange strain tensor components along a line in the 3D torsion problem between FEM and EBE-PINN.

The PINN architecture maps the spatial coordinates (x, y, z) to the displacement components $(\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_z})$. A fully connected feedforward neural network is employed, consisting of 5 hidden layers with 50 neurons each. The model is trained using $14{,}964$ tetrahedral elements, as shown in Fig. 6.56c. For numerical integration, 4 Gauss points are used for both volume and surface integrals. Training is first carried out for $5{,}000$ epochs using the *Adam* optimizer, followed by a refinement stage using the L-BFGS-B algorithm to enhance convergence and smoothness of the predicted solution. Note that the penalty parameter is set to $\epsilon = 3 \times 10^4$.

To exactly enforce Dirichlet boundary conditions and improve training stability, we apply the following output transformation:

$$\begin{aligned} \tilde{u}_x &= \frac{x}{E}\mathcal{N}_{u_x}, \\ \tilde{u}_y &= \frac{1}{E}\mathcal{N}_{u_y}, \\ \tilde{u}_z &= \frac{z}{E}\mathcal{N}_{u_z}. \end{aligned} \tag{6.140}$$

This formulation ensures the displacement field satisfies the Dirichlet boundary conditions exactly. Moreover, the scaling by E introduces non-dimensionalization, which improves the numerical conditioning of the optimization problem and helps mitigate instability during training.

Fig. 6.57 presents a comparison of the displacement and Green-Lagrange strain components computed using EBE-PINN and FEM for the 3D spherical contact problem. The EBE-PINN results are shown in the first and third rows for displacement and strain, respectively, while the corresponding FEM results are displayed in the second and fourth rows. The EBE-PINN predictions exhibit excellent visual agreement with the FEM reference, particularly in the displacement field. This is further supported by the relative L_2 errors, which are 2.42×10^{-3} for displacement and 2.92×10^{-2} for strain, indicating high accuracy across both fields.

Similarly, Fig. 6.58 presents a comparison of the Cauchy stress components computed using EBE-PINN and FEM for the 3D spherical contact problem. The EBE-PINN results are shown in the first and third rows for the normal and shear stress components, respectively, while the second and fourth rows display the corresponding FEM results. The predicted stress fields from EBE-PINN show good visual agreement with the FEM reference across all components. The relative L_2 error for the full Cauchy stress tensor is 3.13×10^{-2}, which is higher than the displacement error, but still indicates reasonably accurate stress recovery under large deformation.

In terms of computational performance, the EBE-PINN model required approximately 13 minutes to complete the simulation, whereas the FEM simulation took over 78 hours (2.82×10^5 seconds), despite being parallelized. It is worth noting that the FEM setup solves the full 3D problem, while EBE-PINN is trained on a clipped subdomain. Nevertheless, this significant gap in computational time highlights the potential of EBE-PINNs to efficiently approximate high-fidelity solutions typically obtained using traditional numerical methods. Once trained, EBE-PINNs can rapidly predict displacement, strain, and stress fields, making them highly suitable for scenarios involving repeated simulations or real-time inference.

It should be emphasized that these results constitute a preliminary demonstration of the EBE-PINN framework for large-deformation contact problems. The current model accuracy could be further enhanced through targeted hyperparameter tuning (e.g., optimal penalty parameter), architectural refinements (e.g., increasing the complexity of the neural network), and algorithmic advancements such as a Lagrange-based gap formulation. Considering that the present simulation

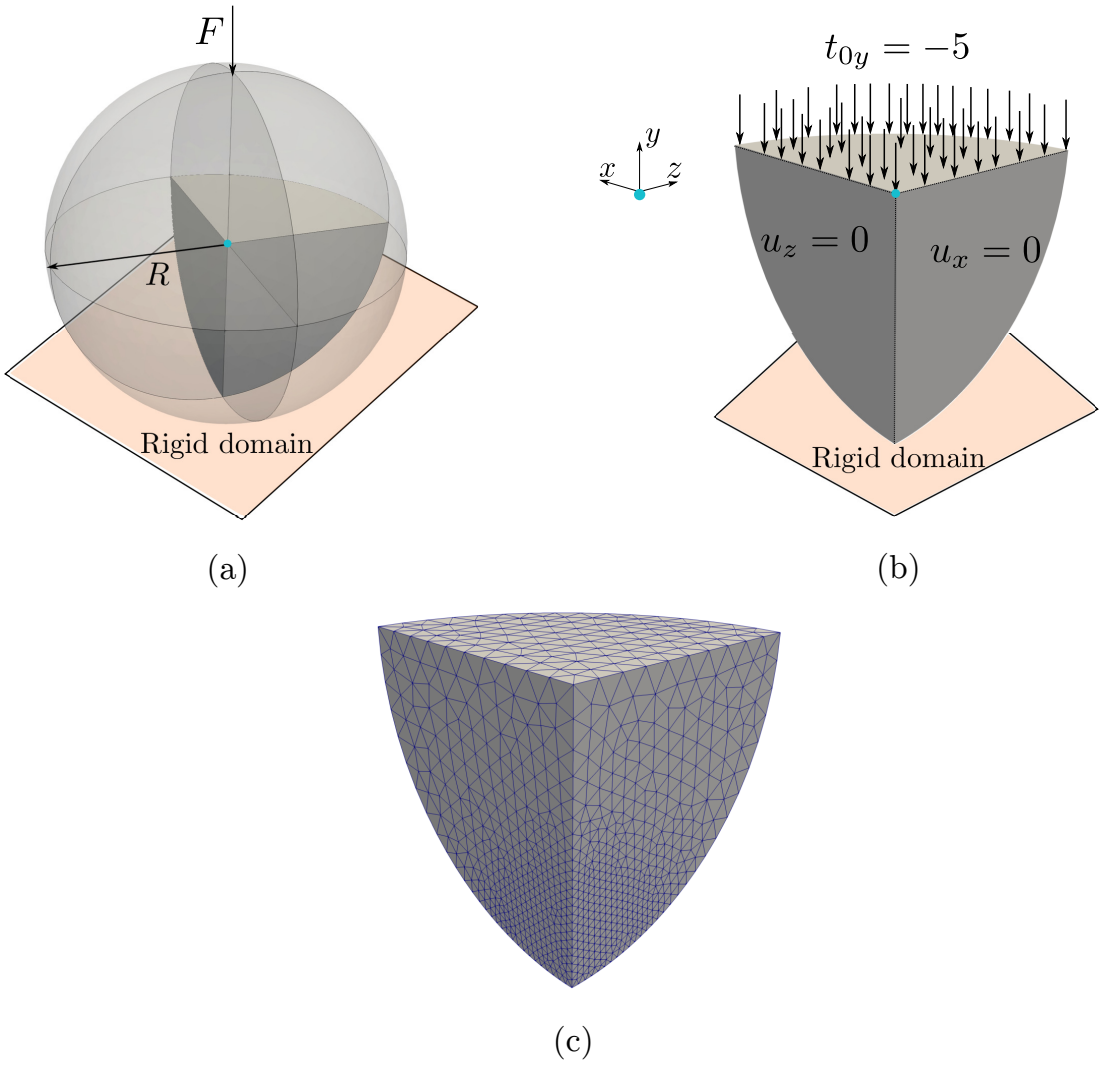


Figure 6.56.: Spherical contact problem between an elastic sphere and a rigid flat domain. (a) The full contact problem under load F, including geometry and dimensions, (b) the equivalent problem due to the axisymmetrical nature of both geometry and loading, including boundary conditions, (c) corresponding mesh with tetrahedral elements.

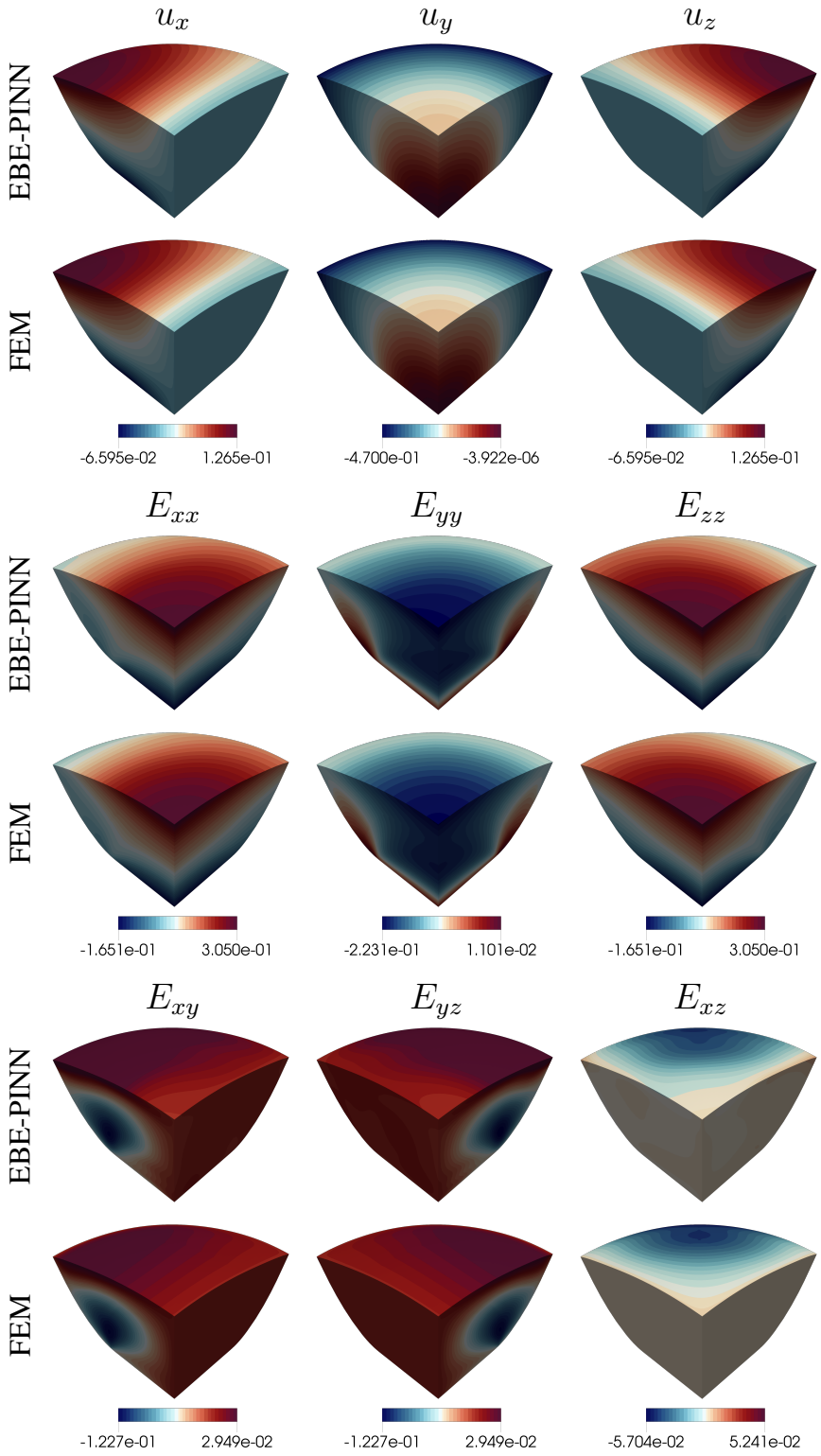


Figure 6.57.: Comparison of displacement and Green-Lagrange strain components predicted by EBE-PINN and FEM for the 3D spherical contact problem.

already yields reasonably accurate results within a fraction of the computational time required by FEM, further optimization of the EBE-PINN approach holds considerable potential for accelerating the solution of large-deformation contact problems.

6.4.3.4. 3D torus contact instability problem

This benchmark problem investigates the contact-induced instability behavior in a toroidal geometry. The setup provides a challenging test case to evaluate the effectiveness and robustness of EBE-PINNs under highly nonlinear and unstable contact conditions.

Fig 6.59a illustrates the complete geometry, where a torus lies in contact with a rigid flat domain. The top of the torus (located at $y = 0$) is subjected to a prescribed vertical displacement in the negative y-direction. The torus is characterized by two radii: the major radius R, which denotes the distance from the center of the torus to the center of the tube, and the minor radius r, which represents the radius of the tube itself.

Due to the axisymmetric nature of both the geometry and the loading, the domain can be reduced to a quarter of the half torus without loss of generality. The reduced configuration is depicted in Fig. 6.59b, leading to a significant reduction in computational cost.

The reduced domain is defined with the following geometric and material properties: the major radius is set to $R = 0.95$ and the minor radius to $r = 0.05$. Imposed displacements are applied as $u_y = -0.8$ on the top surface of the torus ($y = 0$). The material parameters are given by a Young's modulus of $E = 10$ and a Poisson's ratio of $\nu = 0.3$. This setup can be likened to a foam pool toy, exhibiting highly hyperelastic behavior that permits large deformations.

Homogeneous Dirichlet boundary conditions are specified as follows:

- $u_x = 0$ is applied on the front surface at $x = 0$ and on the top surface at $y = 0$,
- $u_z = 0$ is applied on the symmetry plane at $z = 0$.

These constraints, along with the prescribed displacements, are illustrated in Fig. 6.59(a,b).

To construct the EBE-PINN architecture, a fully connected feedforward neural network is employed. The designed architecture maps the spatial coordinates (x, y, z) to the displacement components $(\mathcal{N}_{u_x}, \mathcal{N}_{u_y}, \mathcal{N}_{u_z})$. The architecture consists of 5 hidden layers with 50 neurons each. The training mesh comprises $14{,}119$ tetrahedral elements, as illustrated in Fig. 6.59c. Numerical integration is carried out using 4 Gauss points for both the volume and contact surface integrals.

Similar to previous examples, the training is conducted in two stages. Initial training using the Adam optimizer for $5{,}000$ epochs. Fine-tuning with the L-BFGS-B optimizer to improve convergence and ensure smooth displacement fields.

To ensure exact satisfaction of Dirichlet boundary conditions and improve numerical conditioning, the output of the neural network is transformed as follows:

$$\begin{aligned}
\tilde{u}_x &= \frac{xy}{E}\mathcal{N}_{u_x}, \\
\tilde{u}_y &= \frac{1}{E}\mathcal{N}_{u_y}, \\
\tilde{u}_z &= \frac{z}{E}\mathcal{N}_{u_z}.
\end{aligned}$$

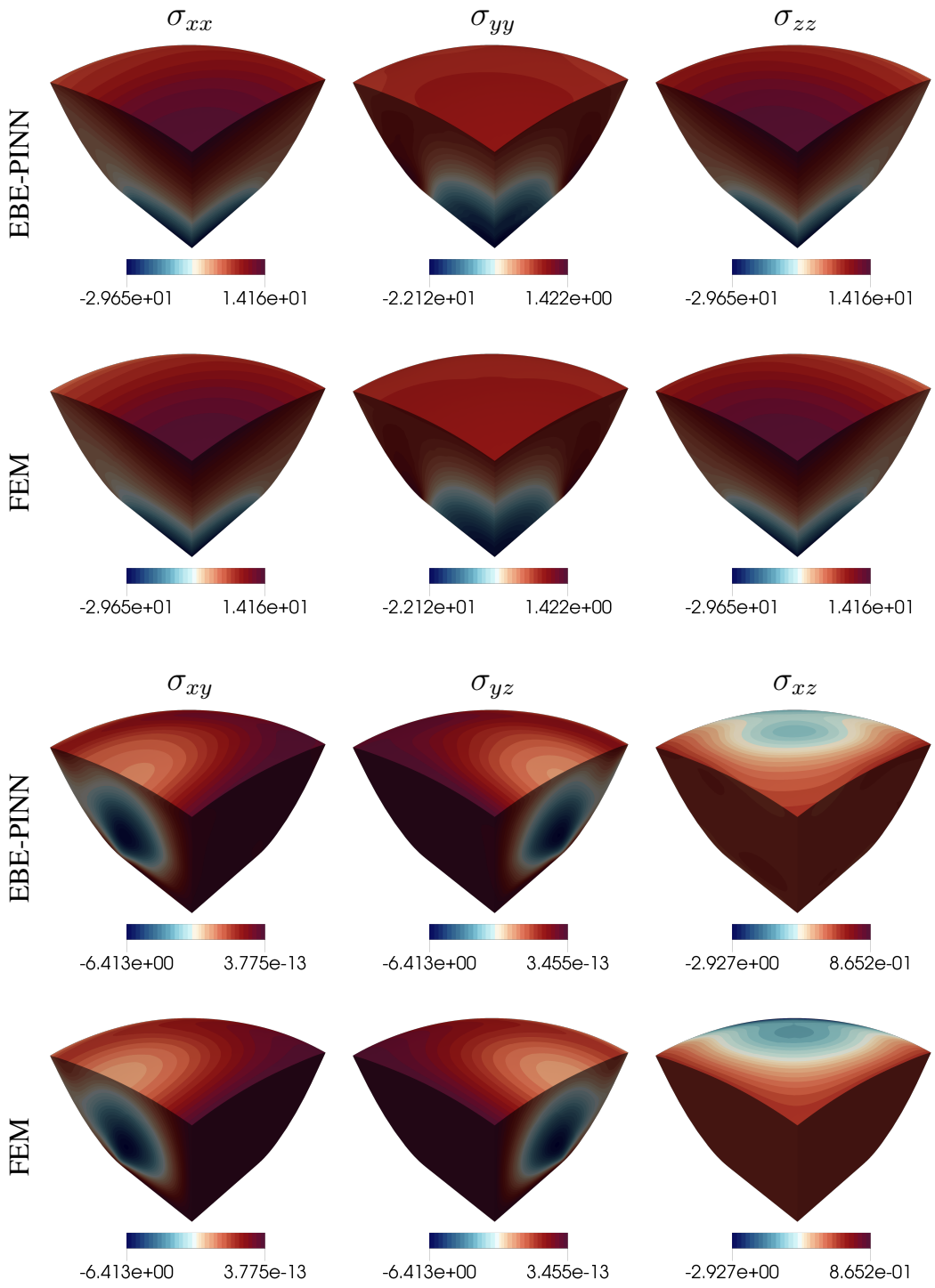


Figure 6.58.: Comparison of Cauchy stress components predicted by the EBE-PINN model and FEM for the 3D spherical contact problem. The top two rows show the normal stress components σ_{xx}, σ_{yy}, and σ_{zz}, while the bottom two rows present the shear stress components σ_{xy}, σ_{yz}, and σ_{xz}.

This transformation exactly enforces the prescribed displacements while introducing non-dimensional scaling via E. This scaling enhances numerical stability during optimization and helps mitigate instabilities in training.

The penalty parameter used in the loss formulation is set to $\epsilon = 3 \times 10^4$.

The displacement results in Fig. 6.60 show smooth, continuous deformations across the quarter-torus geometry with only a negligible penetration of approximately 4.8×10^{-4} between contacting surfaces. The predicted displacement fields (u_x, u_y, u_z) follow the expected deformation pattern under the applied loading, with large amplitudes near the free ends and reduced motion toward the fixed or symmetry-constrained boundaries. The overall deformation profile is physically meaningful and consistent with the imposed contact conditions.

The corresponding Cauchy stress components in Fig. 6.61 exhibit meaningful distributions between the compression and tension sides of the curved geometry. Regions of high positive stress align with tensile zones, while high negative stress corresponds to compressive loading, matching the bending-like deformation observed in the displacement plots. Although no FEM reference solution is available for direct comparison, the patterns are qualitatively reasonable, symmetric where expected, and free from spurious oscillations, indicating that the EBE-PINN method successfully captured both the displacement and stress states for this challenging 3D contact instability problem.

Furthermore, the shear stress components $(\sigma_{xy}, \sigma_{yz}, \sigma_{xz})$ display localized concentration patterns near regions of curvature change and contact interfaces, which is consistent with the mechanics of curved tubular structures under bending. These shear distributions are well-bounded and exhibit smooth spatial variation, indicating that the EBE-PINN formulation handled the stress coupling between directions without producing unphysical discontinuities.

The normal stress components $(\sigma_{xx}, \sigma_{yy}, \sigma_{zz})$ also exhibit balanced tensile–compressive behavior across the thickness, with symmetry about the mid-surface where applicable. The magnitudes and gradients appear physically realistic, indicating that the identified stress field fulfills equilibrium and compatibility conditions. Taken together, these results indicate that even in the absence of FEM ground truth, the EBE-PINN predictions are mechanically consistent, contact constraints are satisfied, and the stress fields align with expected physical trends for the quarter-torus contact instability scenario.

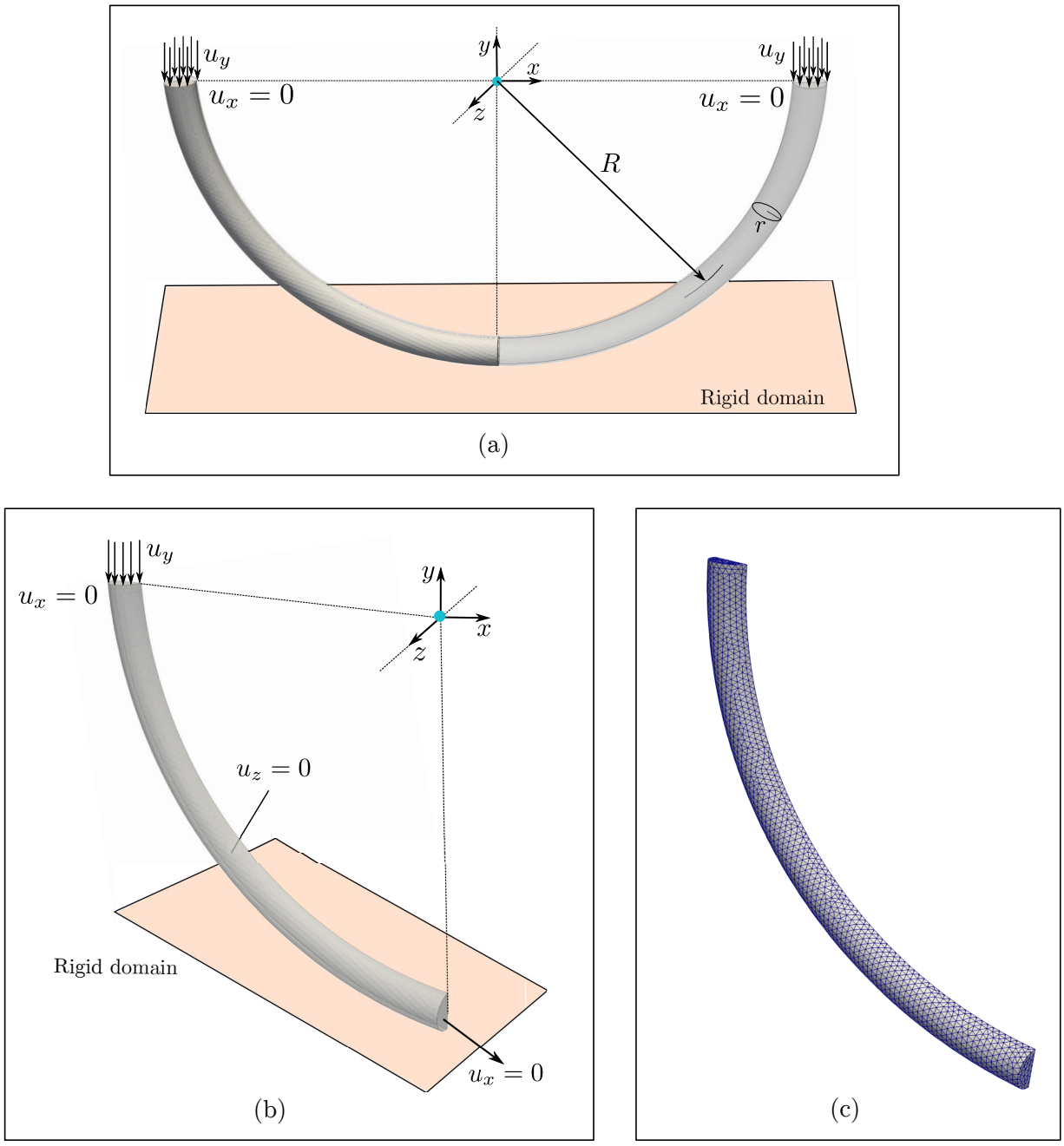


Figure 6.59.: Torus contact instability example. (a) The full contact problem under displacement loading u_y including geometry and dimensions, (b) the equivalent problem due to the axisymmetrical nature of both geometry and loading, including boundary conditions, (c) the tetrahedral mesh used for the quarter torus contact instability example.

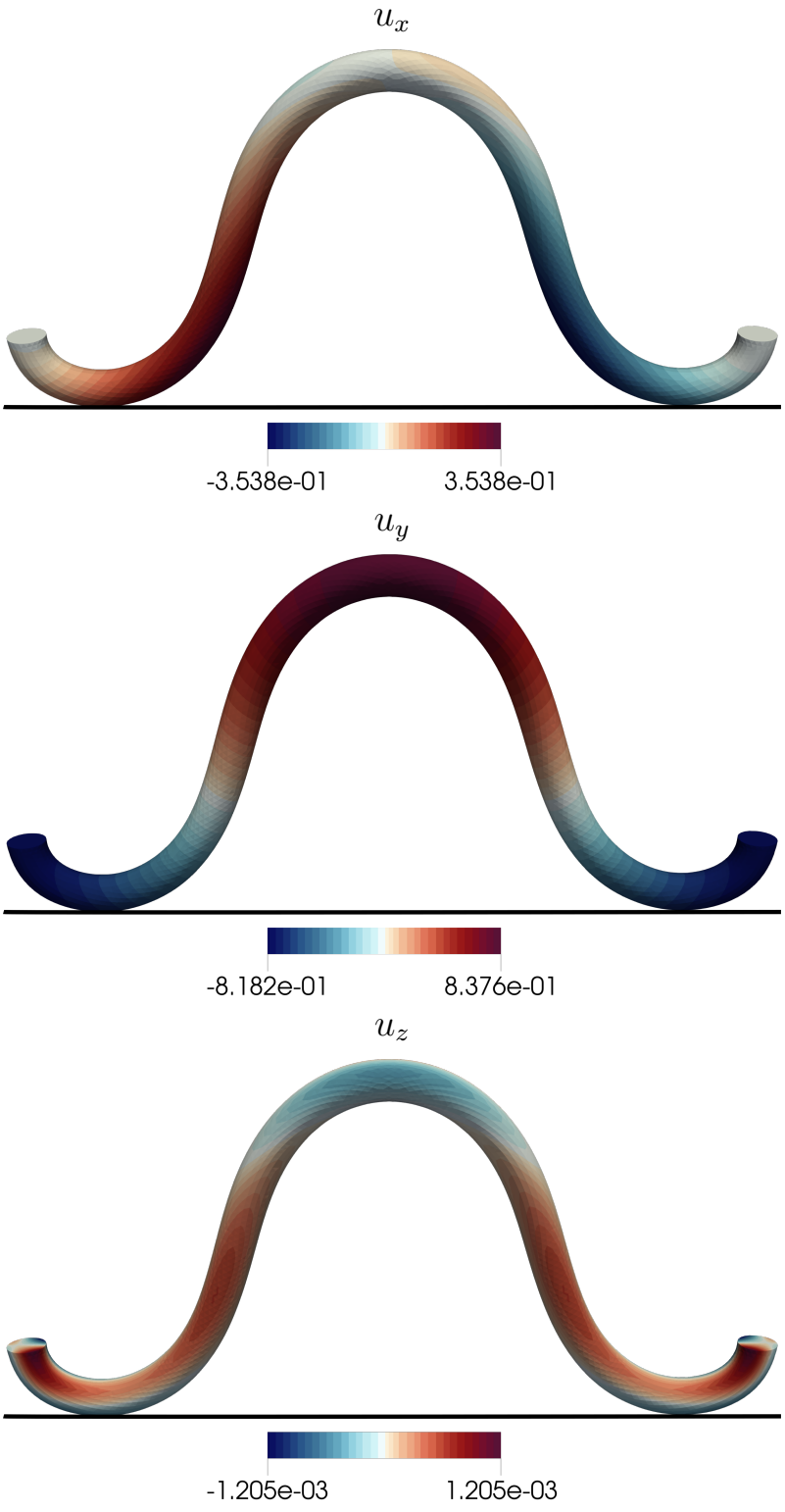


Figure 6.60.: Demonstration of predicted displacement components (u_x, u_y, u_z) obtained using EBE-PINN for the 3D quarter torus contact instability example.

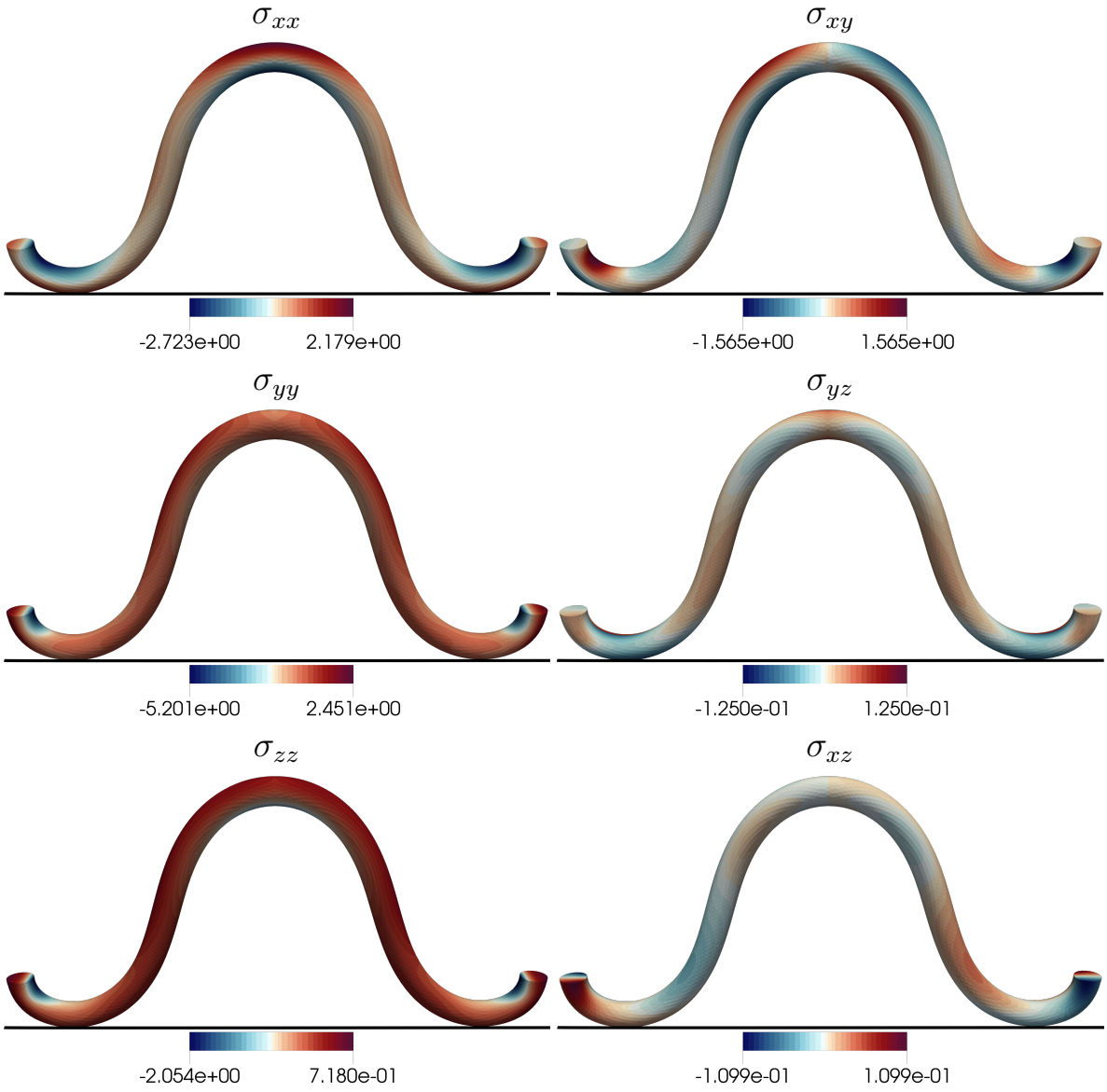


Figure 6.61.: Demonstration of predicted Cauchy stress components (σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{xz}) obtained using EBE-PINN the 3D quarter torus contact instability example.

7. Towards a Hybrid Digital Twin

A *hybrid digital twin* represents a digital replica of a physical asset or system, serving as a metamodel that integrates sensor data, simulation results, and the underlying physical principles of the system. This combination enhances the model's accuracy, reliability, and predictive capabilities. Hybrid digital twins have been widely adopted across various industries, such as construction, manufacturing, and automotive, demonstrating their versatility and value. Notable applications and case studies can be found in the works of Jiang [82], Singh [157], and San [150].

Surrogate modeling has become a cornerstone in developing hybrid digital twins, offering rapid and efficient models capable of delivering accurate, real-time predictions. Once trained, these models can be continuously updated with live sensor data, enabling applications like monitoring the structural health of critical infrastructure, such as bridges. This capability is crucial for ensuring the safety and integrity of these essential assets.

Broadly, surrogate modeling methods fall into two primary categories: purely data-driven approaches and physics-informed approaches. Purely data-driven models, often referred to as *black-box models*, rely solely on observed data to learn the underlying system dynamics. These models are typically constructed using machine learning and deep learning techniques, such as neural networks, and aim to approximate system behavior based on input-output pairs. However, they may struggle with accuracy in scenarios where data is sparse, as they lack knowledge of the system's inherent constraints.

In contrast, physics-informed methods–also known as *gray-box models*–enhance traditional data-driven approaches by embedding physical laws and governing equations directly into the learning process. This integration aims to improve the model's robustness and generalization, particularly when data is limited or the system behavior extends beyond the available observations. A promising example of this approach is physics-informed neural networks (PINNs) [134], which have recently gained significant traction due to their potential for achieving high-fidelity predictions by combining data with domain-specific physics.

7.1. The concept of hybrid digital twins

A *digital twin* is an up-to-date, virtual representation of a physical asset, system, or process designed to accurately replicate the characteristics and behaviors of its real-world counterpart [67]. In essence, it serves as a dynamic bridge between the physical and digital realms, providing a comprehensive mapping of physical entities into the virtual environment.

In contrast, a *hybrid digital twin* [42] advances this concept by merging physics-based modeling (often referred to as a virtual twin) with data-driven approaches (digital twin). This combination creates powerful simulation tools capable of monitoring and analyzing system behavior and detecting underlying trends, incorporating real-time operational data, making reliable predictions, and performing sophisticated what-if simulations.

In the context of bridge infrastructure, hybrid digital twins offer immense potential for enhancing condition and lifetime assessment. By enabling effective structural health monitoring (SHM) and facilitating proactive maintenance, these digital replicas improve the overall resilience and safety of critical infrastructure assets [166].

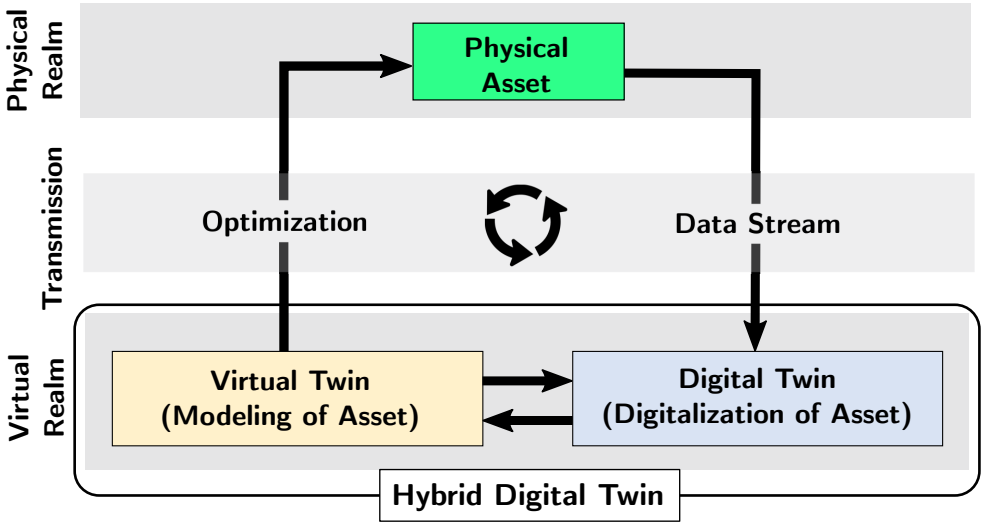


Figure 7.1.: The conceptual workflow between hybrid digital twin, virtual twin, and digital twin.

The conceptual workflow between the hybrid digital twin, virtual twin, and digital twin is depicted in Fig. 7.1. At the core of the workflow lies the physical asset, which exists within the physical realm. Data generated by the physical asset, including sensor information and operational metrics, is continuously streamed to the digital realm. Fast, efficient, and secure data transmission methods, such as advanced file transfer and streaming protocols, play a critical role in maintaining the integrity of the digital representation, as the quality of the data directly impacts the fidelity of the replication.

A hybrid digital twin has two main components: the digital twin and the virtual twin. A digital twin is a data-centric model built using data-driven methods like machine learning and deep learning techniques. This approach captures complex behaviors and patterns from real-world data, enabling predictive capabilities and insights based on historical and real-time inputs.

Conversely, the virtual twin is constructed using a physics-based modeling approach. It leverages fundamental physical principles and domain-specific knowledge to replicate the behavior of the physical asset through analytical models and numerical simulations, providing a detailed understanding of the underlying system dynamics.

The digital twin and virtual twin coexist within the virtual realm, forming the hybrid digital twin. This integrated model combines the predictive power and flexibility of data-driven techniques with the robustness and interpretability of physics-based simulations. The hybrid digital twin offers a high-fidelity representation of the physical asset, facilitating enhanced monitoring, diagnostics, and optimization. Bridging the gap between empirical data insights and theoretical simulations provides a comprehensive framework for improved decision-making and system analysis.

7.1.1. Proof of concept for a steel-reinforced concrete beam

As a part of this thesis, we aim to lay the foundation for a hybrid digital twin concept for steel-reinforced concrete beams as a representative component of bridges or, more generally, civil engineering structures.

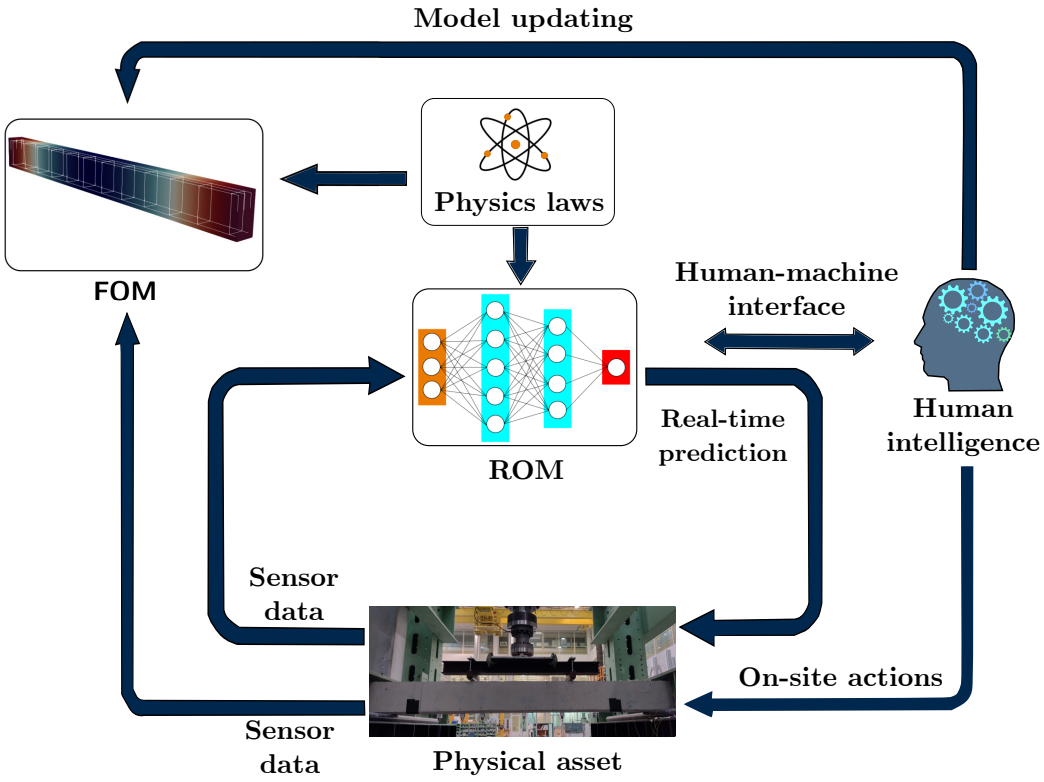


Figure 7.2.: Conceptual framework of a hybrid digital twin for a steel-reinforced concrete beam under a four-point bending test.

As illustrated in Fig. 7.2, the hybrid digital twin framework is applied to a steel-reinforced concrete beam, which is the physical asset analyzed. The experiment setup involves a four-point bending test, where forces are applied at two points to induce bending moments, simulating realistic loading conditions. Sensors embedded in the beam continuously capture structural response data, such as strain and displacement, throughout the test.

The collected sensor data is transmitted in real-time to two modeling components: a pure physics-based full-order model (FOM) and a physics-enhanced data-driven reduced-order model (ROM). The FOM utilizes detailed physical equations to provide a high-fidelity representation of the beam's behavior. In contrast, the ROM, enhanced by integrating physical principles (gray-box modeling), offers a simplified but fast-to-evaluate approximation, allowing for efficient real-time predictions of the structural response.

While the ROM's rapid evaluation capability is advantageous for real-time analysis, gray-box modeling approaches still require human expertise to manage the overall framework, as the reliability of purely data-driven models can be inconsistent. Therefore, human intelligence plays a crucial role in interpreting the outputs and guiding on-site decision-making.

The ROM is constructed using physics-informed neural networks (PINNs) to balance computational speed and predictive accuracy. The detailed formulation of the FOM is beyond the scope of this section; we refer the reader to [45] for further information.

7.1.2. Physical asset and sensor information

As shown in Fig. 7.2, the physical asset in this thesis is a steel-reinforced concrete beam, representative of structural components commonly used in civil engineering applications. The beam is subjected to a four-point bending test, a standard method for evaluating structural elements' flexural strength and behavior. The beam's dimensions and experimental setup are illustrated in Fig. 7.3. The concrete used in the beam has a Young's modulus of $E_c = 29\,\mathrm{GPa}$, while the reinforcing steel has a Young's modulus of $E_s = 200\,\mathrm{GPa}$. Detailed specifications of the beam can be found in [29, 45].

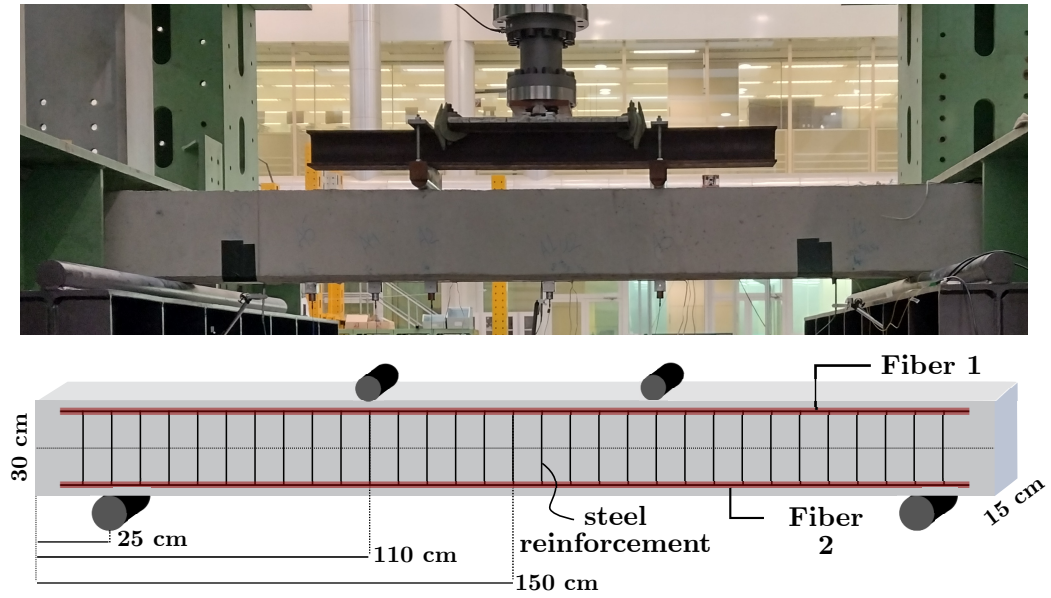


Figure 7.3.: Schematic of the four-point bending test setup for the steel-reinforced concrete beam. The experiment aims to evaluate the flexural response of the beam under cyclic loading. Image adapted from [45].

The four-point bending test is conducted to generate a comprehensive dataset for validating the hybrid digital twin framework. During the experiment, fiber-optical sensors are embedded along the longitudinal axis of the beam to measure the microstrains, providing high-resolution strain data. A total of six fiber-optic sensors are deployed: two sensors are positioned on the compression side of the beam, while four sensors are placed on the tension side. For this analysis, data from two key channels are highlighted: one sensor from the compression side (Fiber 1) and one from the tension side (Fiber 2).

A sinusoidal cyclic load with a peak amplitude of $10\,\mathrm{kN}$ is applied to the beam during the test, simulating realistic loading conditions. Each fiber-optical sensor records approximately 1000 data points, capturing the strain response of the beam under cyclic loading. The strain measurements from the fiber-optical sensors are presented in Fig. 7.4.

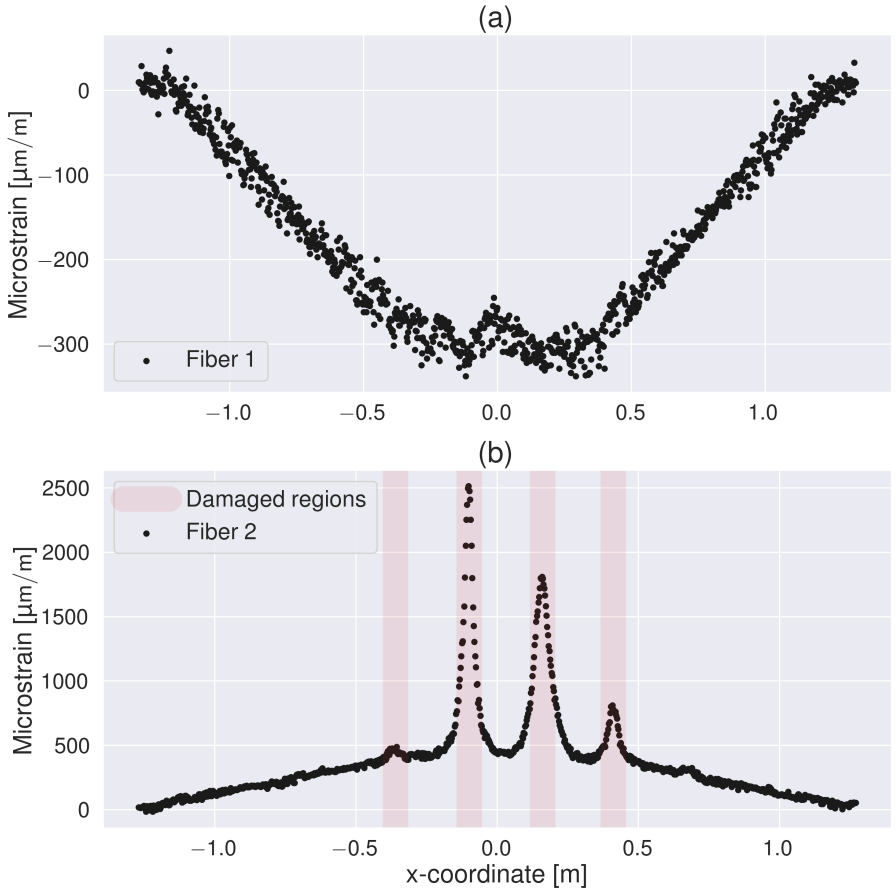


Figure 7.4.: Strain measurements obtained from fiber-optical sensors during the four-point bending test under a peak load of $10\,\mathrm{kN}$. (a) The microstrain response of Fiber 1 on the compression side shows a typical parabolic bending profile. (b) The microstrain response of Fiber 2 on the tension side highlights regions of significant strain increase due to the presence of cracks and localized damage.

In Fig. 7.4(a), the strain data from Fiber 1 (located on the compression side) displays a typical parabolic shape consistent with the expected bending behavior of a beam under flexural loading. The microstrains gradually increase toward the mid-span of the beam and decrease towards the supports.

In contrast, Fig. 7.4(b) illustrates the strain measurements from Fiber 2 (on the tension side), where the response is significantly different. The strain profile exhibits sharp peaks, indicative of localized damage regions where cracks have formed in the concrete. These cracks significantly weaken the structural integrity of the beam, resulting in a dramatic increase in microstrain values. The highlighted shaded areas in the figure represent the damaged regions where the material experiences substantial tensile stresses beyond its cracking threshold.

The collected strain data from both sensors are then transmitted to the hybrid digital twin framework, which integrates a purely physics-based full-order model (FOM) and a physics-enhanced data-driven reduced-order model (ROM). The FOM provides a detailed, high-fidelity simulation of the beam's response, incorporating the physical laws governing structural mechanics. In parallel, the ROM, constructed using physics-informed neural networks (PINNs), offers a computationally efficient alternative, enabling real-time predictions of the beam's behavior.

Despite the inclusion of physics in the ROM through gray-box modeling, human expertise is essential for managing the framework and interpreting the model outputs, particularly given the potential inconsistencies in purely data-driven predictions (see Fig. 7.2).

For further details on the full-order model formulation and additional experimental specifications, the reader is referred to [45].

In the following section, we develop fast-to-evaluate reduced-order models (ROMs) to predict strain distributions within a reinforced concrete beam as a representative example of civil engineering structures.

7.2. Surrogate modeling of a reinforced concrete beam

In this section, we develop efficient, fast-to-evaluate reduced-order models (ROMs) to predict strain distributions within a steel-reinforced concrete beam, serving as a representative example of typical civil engineering structures as explained in Section 7.1. Two distinct ROMs are constructed using PINNs, each addressing a specific aspect of the strain behavior observed during the four-point bending test.

The first ROM is designed to analyze the temporal evolution of strain at a fixed spatial location inside the beam. Starting with time series data collected from fiber-optical sensors embedded within the beam, we employ a purely data-driven neural network model trained exclusively on the available sensor data. However, due to the periodic nature of the applied cyclic load, we extend this approach by incorporating domain-specific knowledge, which is the harmonic oscillator motion. By integrating the physical insights of periodic loading with the sensor data, we demonstrate the enhanced predictive capability of the physics-augmented model over the purely data-driven approach.

The second ROM focuses on capturing the spatial variation of strain within the beam at a specific time instance. Here, we integrate the collected sensor data with the governing equations of linear elasticity to predict the strain distribution across the entire beam. The inclusion of sensor data from both the compression and tension sides of the beam enables a more comprehensive

representation of the strain field, leading to improved prediction accuracy. However, incorporating measurements from multiple sensors introduces potential challenges due to the presence of noise in the sensor data, which may result in non-smooth or unrealistic predictions.

To address these issues, we employ a loss-weight tuning strategy, carefully balancing the contributions of the physics-based loss and the data-driven sensor loss. This approach mitigates the effects of measurement noise and ensures a smoother, physically consistent strain prediction.

Overall, the comparison between the purely data-driven model and the physics-augmented model highlights the importance of incorporating physical knowledge into the ROM framework, demonstrating significant improvements in predictive accuracy and robustness.

7.2.1. Capturing the temporal response

The primary objective of this section is to develop a model capable of accurately capturing the temporal behavior of the system under cyclic loading conditions. To achieve this, we utilize measurement data collected at a fixed location along the beam, specifically at the center of fiber 1 (see Fig. 7.3 in Section 7.1.2) across multiple time instances.

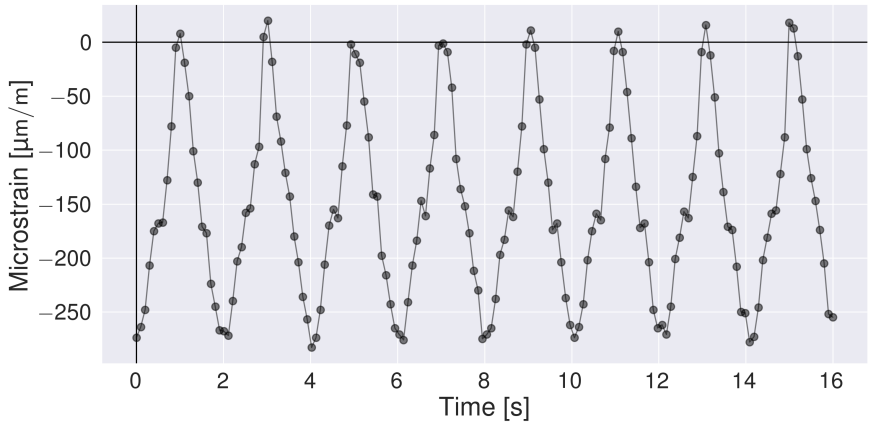


Figure 7.5.: Temporal strain measurements recorded at the center of fiber 1 over a duration of 16 seconds. Each second contains ten individual measurement points, capturing the cyclic nature of the loading.

Fig. 7.5 presents the recorded strain measurements for the first 16 seconds of the experiment. The strain data follows a characteristic harmonic pattern, consistent with the sinusoidal loading applied to the beam. Each measurement interval consists of ten data points per second, providing high-resolution temporal data that captures the cyclic response of the structure.

Although the measured strain data in this example exhibits harmonic behavior due to the known sinusoidal loading, it is important to note that, in general, the underlying data generation process may be unknown or more complex. To address such realistic scenarios, we introduce the ordinary differential equation (ODE) for a harmonic oscillator as a model for describing the data distribution shown in Fig. 7.5. For this simple case, the ODE can be solved analytically, providing a straightforward interpretation of the temporal strain response. However, as we demonstrate in the following sections, more complicated cases may require numerical or data-driven approaches due to increased complexity.

7.2.1.1. Motion of a simple harmonic oscillator

The balance of forces for a simple harmonic oscillator, governed by Newton's second law, is given by:

$$m\frac{d^2u}{dt^2} = -ku \quad \Leftrightarrow \quad \frac{d^2u}{dt^2} + \omega_0^2 u = 0, \tag{7.1}$$

where $u(t)$ denotes the displacement from the equilibrium position x_0, k is the spring constant, and m is the inertial mass of the system. The term $\omega_0 = \sqrt{\frac{k}{m}} = 2\pi f$ represents the natural angular frequency of the oscillator, with f being the natural oscillating frequency.

For a reinforced concrete beam subjected to periodic loading, the longitudinal strain ε can be related to the displacement u as follows:

$$\varepsilon = \frac{u}{x_0}. \tag{7.2}$$

This strain-displacement relationship provides a direct link between the harmonic motion of the beam and the measured microstrain data, allowing us to use the harmonic oscillator model as a simplified but an effective representation of the observed strain behavior under cyclic loading.

7.2.1.2. A PINN-based temporal ROM

Inspired by the harmonic oscillator model, we utilize an FNN to approximate the longitudinal strain within a selected fiber. The neural network predicts the strain as:

$$\tilde{\varepsilon} := \mathcal{N}_\varepsilon(t; \boldsymbol{\theta}), \tag{7.3}$$

where the time t serves as the input to the neural network, and the predicted longitudinal strain $\tilde{\varepsilon}$ is the output. Using automatic differentiation, we compute the necessary second-order derivatives of the neural network output with respect to time.

The total loss function consists of two components: $\mathcal{L}_{\mathrm{ODE}}$ and $\mathcal{L}_{\mathrm{EXPs}}$, expressed as

$$\mathcal{L}_{\mathrm{ODE}} = \frac{1}{N_{rp}} \sum_{i_{rp}=1}^{N_{rp}} \left[\frac{d^2\tilde{\varepsilon}(t^{i_{rp}})}{dt^2} + \omega_0^2 \tilde{\varepsilon}(t^{i_{rp}}) \right]^2, \tag{7.4}$$

$$\mathcal{L}_{\mathrm{EXPs}} = \sum_{i_{e,\varepsilon}=1}^{N_{e,\varepsilon}} w_{i_{e,\varepsilon}} \frac{1}{N_{ep,\varepsilon}} \sum_{i_{ep,\varepsilon}=1}^{N_{ep,\varepsilon}} \left[\left[\tilde{\varepsilon}\left(t^{i_{ep,\varepsilon}}\right) - \varepsilon^*\left(t^{i_{ep,\varepsilon}}\right) \right]_{i_{e,\varepsilon}} \right]^2. \tag{7.5}$$

The term $\mathcal{L}_{\mathrm{ODE}}$ represents the loss contribution from the ODE given in Eq. (7.1), and ε^* represents the measurement data. Since the measurement data loss $\mathcal{L}_{\mathrm{EXPs}}$ already includes the initial condition, $\varepsilon(t=0) = -293\,\mu\mathrm{m\,m^{-1}}$, the total loss function does not include $\mathcal{L}_{\mathrm{ICs}}$.

While the square of the oscillating frequency can be estimated as $\omega_0^2 \approx 9.87\,\mathrm{rad^2\,Hz^2}$ using the measurement data, we treat this parameter as unknown during training and allow the PINN to infer it from the data. The detailed workflow of the temporal ROM is depicted in Fig. 7.6. The neural network architecture consists of three fully-connected layers, each with 30 neurons, and uses the tanh activation function.

For the training process, we initially employ the *Adam* optimizer with a learning rate of 0.001 for $5{,}000$ epochs to obtain a preliminary model. Subsequently, we refine the model using the *L-BFGS-B* algorithm.

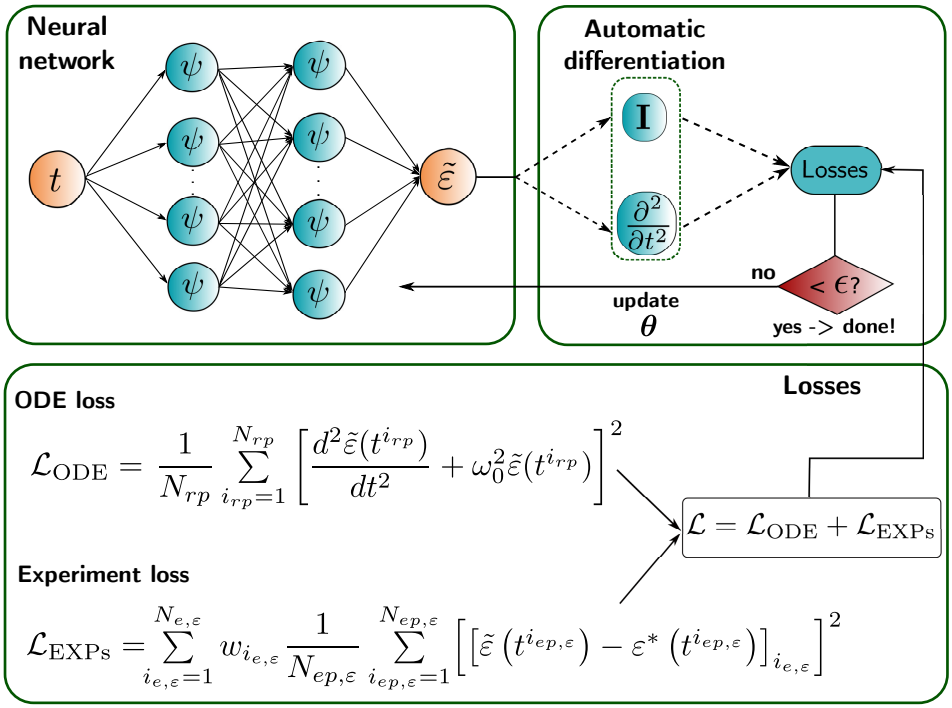


Figure 7.6.: Schematic representation of the temporal reduced-order model (ROM) using a physics-informed neural network (PINN).

7.2.1.3. Comparison of temporal models

Here, we assess the performance of two modeling approaches for predicting the temporal strain response of a steel-reinforced concrete beam. The first approach employs a purely data-driven neural network trained solely on time-dependent sensor data collected from the compression side of the beam (see Fig. 7.5). This network follows the same architecture as the temporal ROM but uses a sin activation function instead of tanh. The second approach leverages the physics-informed temporal ROM, which integrates sensor data with physical insights obtained from the harmonic oscillator model in Eq. (7.1).

Fig. 7.7 compares the predictions of the purely data-driven model and the physics-informed temporal ROM. Both models are trained using the first $6\,\mathrm{s}$ of sensor data. The data-driven model fits the training data almost perfectly, achieving close alignment. However, while the temporal ROM does not perfectly match the training data, it successfully captures the overall trend. This discrepancy arises from inherent measurement noise in the sensor data, which is not explicitly accounted for in the physical model.

The advantage of the temporal ROM becomes apparent when evaluating predictions beyond the training interval ($t > 6\,\mathrm{s}$). While the data-driven model significantly deviates from the actual sensor data, the temporal ROM continues to capture the general strain response trend. This highlights the limitation of the purely data-driven model in generalizing beyond the training data range, whereas the temporal ROM, benefiting from embedded physical laws, demonstrates robust predictive capability even in regions without measurement data.

Regarding training performance, the purely data-driven model required $10.2\,\mathrm{s}$ using the *Adam* optimizer, followed by an additional $12.7\,\mathrm{s}$ using the *L-BFGS-B* optimizer. The temporal ROM

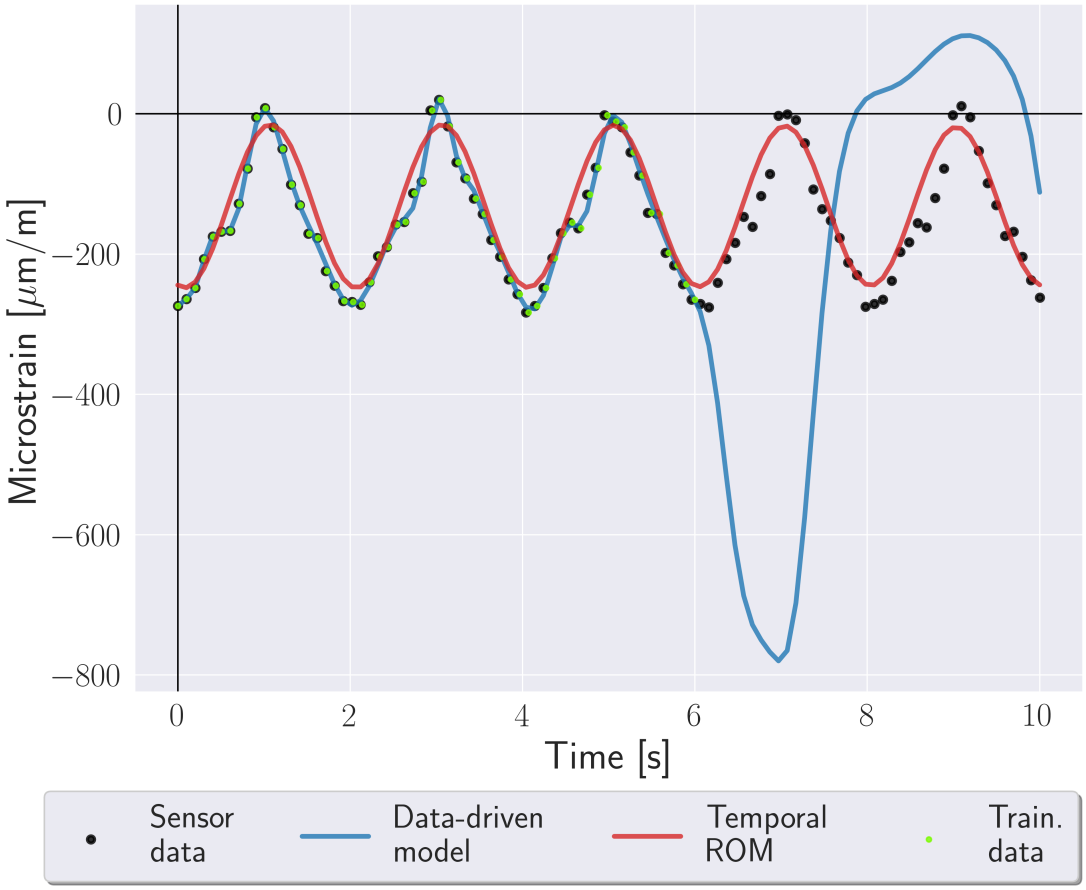


Figure 7.7.: Comparison of the purely data-driven model and the physics-informed temporal ROM, including training data and sensor measurements. The data-driven model aligns well with the training data, but diverges outside the training interval, whereas the temporal ROM maintains accurate predictions beyond the training period.

took approximately $12.6\,\mathrm{s}$ for the initial training with *Adam*, and $23\,\mathrm{s}$ for the subsequent refinement with *L-BFGS-B*. Despite the longer training time for the temporal ROM, including physical laws significantly improves its generalization ability. The prediction times for both models are comparable, with the purely data-driven model taking $0.003\,\mathrm{s}$ and the temporal ROM requiring $0.004\,\mathrm{s}$ per prediction.

The robustness of the temporal ROM is further enhanced by its capability to identify key physical parameters, leveraging the PINN framework as an inverse solver. Specifically, we included the natural oscillating frequency ω_0 as a trainable parameter within the neural network's parameter set $\boldsymbol{\theta}$. Starting from an initial guess of 1, the model successfully converged to an estimated value of $\tilde{\omega}_0^2 \approx 9.74\,\mathrm{rad}^2\,\mathrm{Hz}^2$, as shown in Fig. 7.8. This estimate is very close to the value of $\omega_0^2 \approx 9.87\,\mathrm{rad}^2\,\mathrm{Hz}^2$ obtained from direct measurement analysis. The convergence plot highlights the significant improvement achieved by switching from the *Adam* optimizer to *L-BFGS-B*, demonstrating the effectiveness of this two-stage optimization approach.

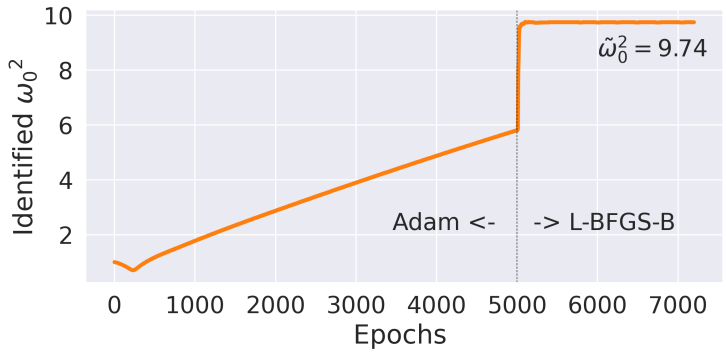


Figure 7.8.: Convergence of the identified natural frequency ω_0. The optimization begins with the *Adam* optimizer and switches to *L-BFGS-B*, resulting in rapid convergence to $\tilde{\omega}_0^2 \approx 9.74\,\mathrm{rad}^2\,\mathrm{Hz}^2$.

Overall, the results clearly demonstrate the benefits of incorporating physical knowledge into the model. While the purely data-driven approach shows excellent fit within the training range, it fails to generalize beyond this interval. In contrast, the temporal ROM leverages the governing physical laws, allowing it to make reliable predictions even in regions with limited or no data. This highlights the potential of hybrid modeling approaches, combining data-driven techniques with physics-based insights, for robust and accurate predictions in engineering applications.

7.2.2. Capturing the spatial response

After developing a ROM for the temporal response in the previous section, we now introduce PINN-based ROMs aimed at predicting the spatial strain distribution within the steel-reinforced concrete beam. To simplify the problem, we model the beam as a two-dimensional rectangular domain and disregard its depth. It is important to note that accurately modeling the physical behavior of a steel-reinforced concrete beam, especially considering the damage depicted in Fig. 7.4 (Section 7.1.2), is quite challenging. Thus, our spatial ROM employs a simplified physical model, which is enhanced by incorporating the available sensor data. This data-driven enhancement corrects the model predictions, and we demonstrate that, with appropriate weighting

of the competing loss components during training, the combined approach yields sufficiently accurate results.

This integration of sensor data with simplified physics in the context of PINNs allows for constructing a surrogate model without explicitly modeling the complex underlying physics. By leveraging this hybrid modeling strategy, we bypass the need for detailed physical simulations while still achieving high-fidelity predictions of the spatial strain distribution.

7.2.2.1. A PINN-based spatial ROM

The spatial reduced-order model (ROM) is formulated using a mixed-variable approach under quasi-static conditions (refer to Section 6.2.2.2). As shown in Fig. 7.9, an FNN maps the spatial coordinates $\boldsymbol{x} = (x, y)$ onto both the estimated displacement $\tilde{\boldsymbol{u}}$ and stress $\tilde{\boldsymbol{\sigma}}$, such that

$$\tilde{\boldsymbol{u}} := \left(\mathcal{N}_{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\right)' \quad \text{and} \quad \tilde{\boldsymbol{\sigma}} := \left(\mathcal{N}_{\boldsymbol{\sigma}}(\boldsymbol{x};\boldsymbol{\theta})\right)'. \tag{7.6}$$

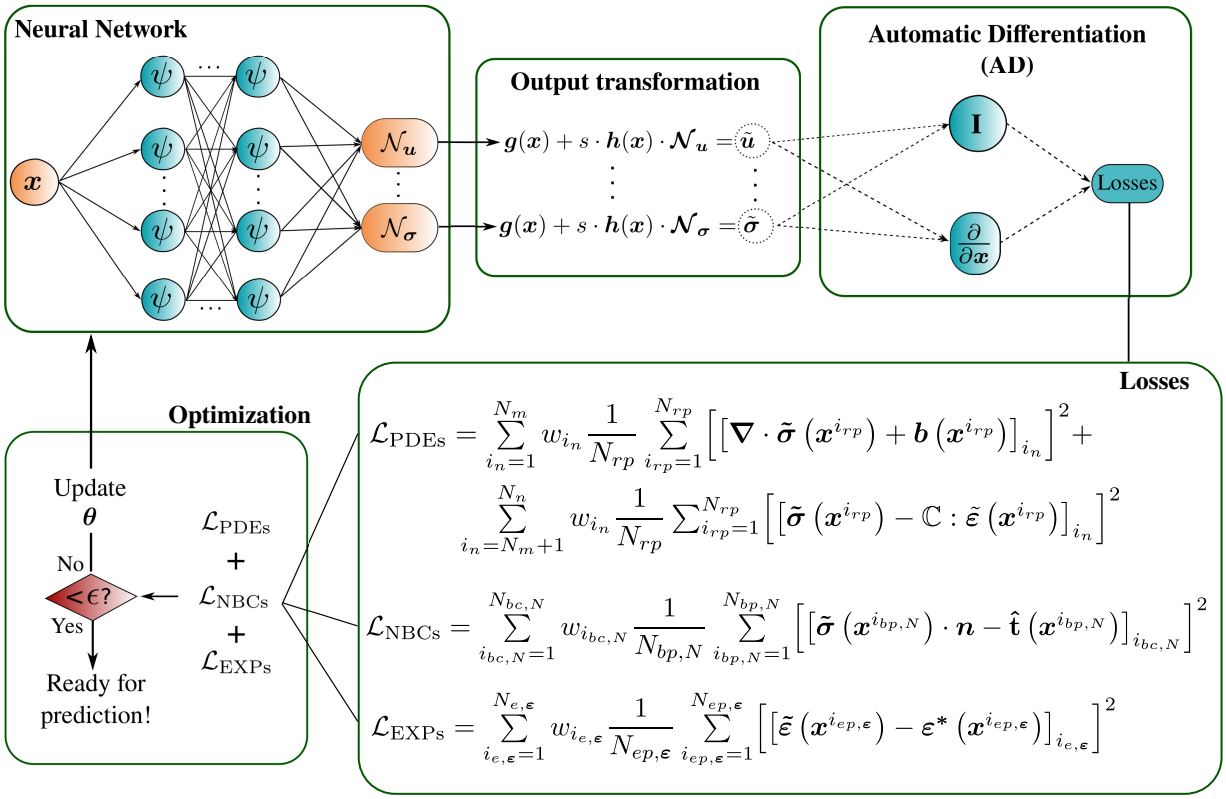


Figure 7.9.: Schematic of the spatial ROM based on the PINNs in the mixed-variable formulation.

To model the beam's physical behavior, we adopt a relaxed representation of the experiment (see Fig. 7.10a). The supports are replaced by traction boundary conditions, and the beam is fixed in the x-direction at its center to prevent rigid body motion. The boundary conditions (BCs) of this relaxed model are illustrated in Fig. 7.10b. Specifically, the displacement is constrained at $x = 0$ such that $u_x = 0$.

Since both displacement and stress components are directly defined as neural network outputs, certain boundary conditions are enforced as hard constraints through output transformations.

Based on the relaxed model, the following output transformations are applied:

$$\begin{aligned} \tilde{u}_x &= x\mathcal{N}_{u_x}, \quad \tilde{\sigma}_{xx} = \left(\frac{l}{2} - x\right)\left(\frac{l}{2} + x\right)\mathcal{N}_{\sigma_{xx}}, \\ \tilde{\sigma}_{xy} &= \left(\frac{l}{2} - x\right)\left(\frac{l}{2} + x\right)\mathcal{N}_{\sigma_{xy}}, \end{aligned} \tag{7.7}$$

where the displacement constraint at the beam's center and traction boundary conditions on the left and right edges are strongly enforced. For additional details on the formulation of the output transformations, we refer to Sahin et al. [144]. The traction boundary conditions on the upper and lower edges are incorporated as soft constraints via additional loss terms.

The sensor dataset includes longitudinal strain measurements from both the compression and tension sides of the beam. The sensor data loss $\mathcal{L}_{\text{EXPs}}$ integrates data from 200 points on the compression side (fiber 1) and 100 points on the tension side (fiber 2). This combination of data helps enhance the model's accuracy by providing comprehensive information about the strain distribution.

The spatial ROM utilizes a fully connected neural network with four hidden layers, each containing 50 neurons. The $\tanh$ activation function is chosen to ensure smooth predictions. Given the 2D nature of the problem, the network inputs are the spatial coordinates (x, y), while the network outputs are $\tilde{u}_x, \tilde{u}_y, \tilde{\sigma}_{xx}, \tilde{\sigma}_{yy}$, and $\tilde{\sigma}_{xy}$.

As with the temporal ROM, the initial training phase employs the *Adam* optimizer for 2000 epochs, followed by the *L-BFGS-B* optimizer to refine the pre-trained model.

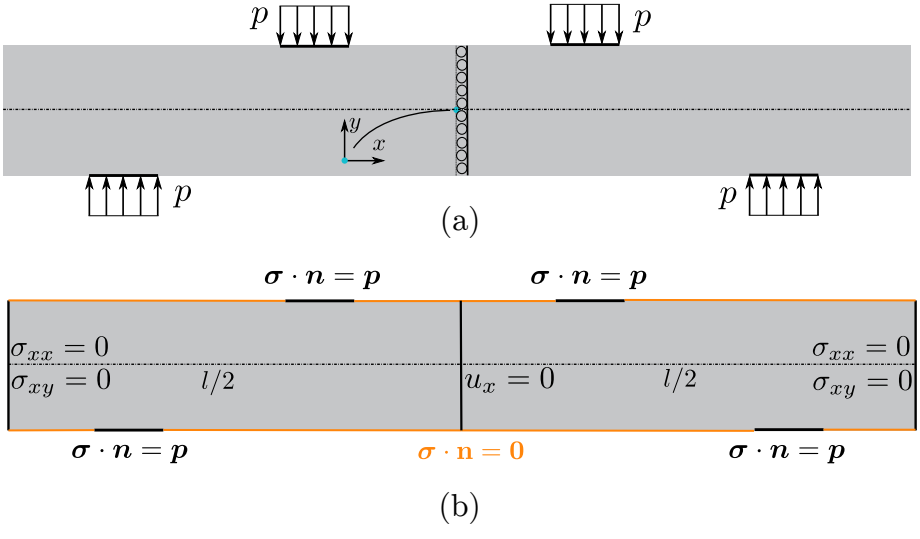


Figure 7.10.: (a) Schematic representation of the relaxed system with simplified boundary conditions. (b) Detailed boundary conditions of the relaxed model, showing the fixed displacement at $x = 0$ and traction BCs on the left and right edges.

7.2.2.2. Comparison of spatial ROM scenarios

In this section, we evaluate the performance of the spatial ROM across three distinct scenarios to assess the impact of incorporating different types of sensor data. The scenarios are defined as follows:

- Scenario 1: The sensor loss component of the spatial ROM includes only the compression data from fiber 1.
- Scenario 2: The sensor loss incorporates both compression and tension data. However, tension data from the damaged regions (refer to Fig. 7.4 in Section 7.1.2) is excluded to avoid inaccuracies caused by structural weakening.
- Scenario 3: Both compression and tension data are included, but the weight for the sensor loss from the tension data is reduced ($w_{\mathrm{EXPs,T}} = 0.01$) to mitigate the influence of noisy measurements.

In Fig. 7.11, we compare the strain predictions across the three scenarios. In Scenario 1 (Fig. 7.11a and Fig. 7.11b), the spatial ROM accurately aligns with the training and sensor data on the compression side. However, it significantly underestimates the strains on the tension side, primarily due to the absence of tension data in the loss component. Additionally, the simplified physical model does not capture the increased deformations caused by cracks.

To address this issue, Scenario 2 incorporates tension data, resulting in better alignment on both the compression and tension sides (Fig. 7.11c and Fig. 7.11d). However, the predictions exhibit oscillatory behavior, especially near the beam's center, due to the noisy tension data contributing to the sensor loss.

In Scenario 3, we reduce the sensor loss weight for the tension data to $w_{\mathrm{EXPs,T}} = 0.01$. This adjustment improves the model's robustness by mitigating the influence of noisy measurements. The resulting predictions (Fig. 7.11e and Fig. 7.11f) show a smoother strain distribution without significant underestimation on the tension side, striking a balance between fitting the data and maintaining physically realistic behavior.

The spatial ROM's extrapolation capabilities are demonstrated in Fig. 7.12, where the predicted strain distribution is shown across the entire beam domain for each scenario. In Scenario 1 (Fig. 7.12a), the strain distribution is smooth, with nearly zero horizontal strains at the beam's center, as expected. In Scenario 2 (Fig. 7.12b), fluctuating strains are observed near the center line, indicating the impact of the noisy tension data. Scenario 3 (Fig. 7.12c) achieves a compromise, restoring a smooth strain distribution while capturing the larger strains on the tension side.

The training with the *Adam* optimizer averages $25\,\mathrm{s}$ across all scenarios. The *L-BFGS-B* optimizer requires longer times, with $56.96\,\mathrm{s}$, $171.17\,\mathrm{s}$, and $70.49\,\mathrm{s}$ for Scenarios 1, 2, and 3, respectively. The increased time for Scenario 2 is due to the inclusion of more data points, resulting in a higher computational cost. However, the prediction time remains consistent across scenarios at approximately $0.35\,\mathrm{s}$.

7.2.3. Conclusion

Here, we demonstrated how sensor data and non-sensor contextual knowledge, specifically the underlying physics of the system, can be effectively integrated using physics-informed neural

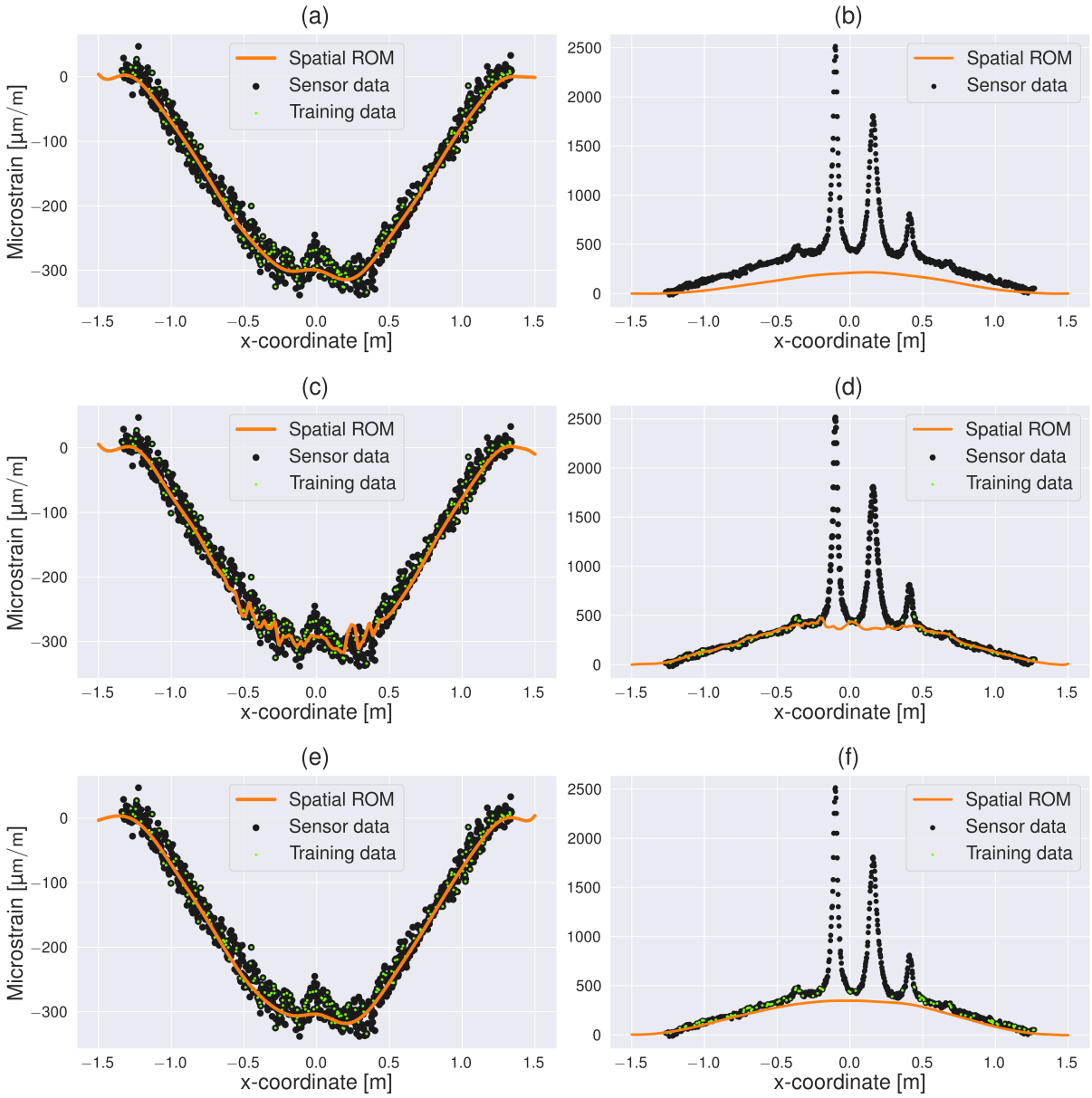
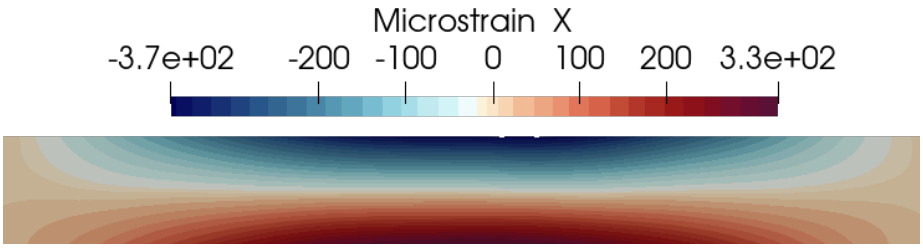
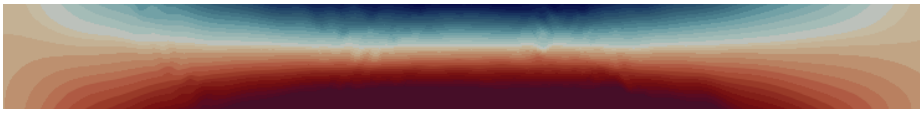


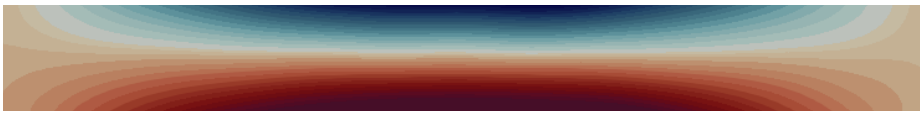
Figure 7.11.: Strain predictions using the spatial ROM along fiber 1 (compression side, top row) and fiber 2 (tension side, bottom row). (a, b) Scenario 1: Only compression data is included. (c, d) Scenario 2: Both compression and tension data are included. (e, f) Scenario 3: Both compression and tension data are included, but the sensor loss weight for tension data is set to $w_{\mathrm{EXPs,T}} = 0.01$.



(a) Scenario 1: Only compression data.



(b) Scenario 2: Both compression and tension data included.



(c) Scenario 3: Both compression and tension data included with adjusted loss weight.

Figure 7.12.: Strain predictions using the spatial ROM within the beam domain for the three scenarios. The strain distribution is visualized, showing how the model predictions adapt with varying sensor data contributions.

networks (PINNs) to develop surrogate models for a steel-reinforced concrete beam. These models serve as core components in hybrid digital twins, combining data-driven insights with physical laws for enhanced predictive capabilities.

Unlike traditional machine learning models, which rely solely on sensor data, PINNs leverage additional contextual knowledge by embedding physics directly into the loss function. This fusion of experimental data from a four-point bending test with governing physical laws forms the basis of our modeling approach.

For the first surrogate model, we integrated time-dependent strain measurements collected at fixed locations along the beam's center line with an ODE representing the motion of a harmonic oscillator. Additionally, we trained a purely data-driven model for comparison. Our results indicate that the data-driven model yields accurate predictions in regions where sensor data is available (strong interpolation) but performs poorly in regions lacking data (poor extrapolation). In contrast, the PINN-based surrogate model demonstrated strong performance in both interpolation and extrapolation regimes. Furthermore, by utilizing the inverse solving capability of PINNs, we accurately identified the natural frequency of the system.

In the second surrogate modeling approach, we combined spatial sensor data with a simplified PDE representing the equations of linear elasticity to capture the spatial strain distribution across the beam at a fixed time instance. We explored several scenarios by incorporating sensor data from different beam locations and adjusting the loss weight for the sensor data. The results showed that including both compression and tension data enhanced the accuracy of the model, particularly on the tension side, where cracks formed during the experiment. However, the model tended to overfit the compression side, resulting in fluctuating strain predictions. To address

this, we reduced the loss weight for tension-side sensor data, which led to smoother and more physically realistic strain predictions. Overall, incorporating physical laws significantly improved the extrapolation capability of the surrogate models, enabling reliable predictions across the entire beam domain, including regions with no available sensor data.

In conclusion, these two predictive models can integrate sensor data and physical knowledge, making them suitable components for hybrid digital twins. Depending on whether temporal or spatial phenomena are of interest, these models can be adapted to provide accurate and robust predictions, highlighting the potential of PINNs for enhanced modeling in civil engineering applications.

8. Summary and Outlook

In this thesis, traditional machine learning and deep learning methods, along with physics-informed machine learning techniques, have been investigated in the context of computational mechanics—including rough surface contact problems, beam mechanics, solid and contact mechanics under small and large deformation regimes–and extended toward a hybrid digital twin framework by integrating real sensor data with physics-informed neural networks, demonstrated through a proof-of-concept application to a steel-reinforced concrete beam for an accurate temporal and spatial surrogate modeling under realistic measurement conditions.

In the first part of this thesis, we developed and evaluated a data-driven surrogate modeling framework to predict the effective contact area in rough surface contact problems involving a linear elastic half-space and a rigid rough surface. The surrogate approach is purely statistical in nature and does not incorporate governing physical laws, positioning it within the class of *black-box* models. While such models offer high flexibility and predictive power, they lack inherent interpretability and do not guarantee physically consistent behavior outside the training distribution. To enable model training, a comprehensive and computationally expensive database was generated using boundary element method simulations. The dataset included far-field displacements as well as 22 statistical descriptors derived from synthetic surface height fields. Several machine learning regression models were assessed, and optimal configurations were identified through cross-validated grid search. Among all models considered, the Kernel Ridge approach emerged as the most effective, providing a favorable balance between accuracy, generalization, training cost, and prediction speed. Despite the high offline cost of data generation, the trained surrogate model proved highly efficient in online evaluation and demonstrated strong generalization to unseen inputs. These features make it a promising tool for multi-query tasks such as uncertainty quantification, sensitivity analysis, and parameter exploration, where repeated model evaluations are required, but high-fidelity simulations would be computationally prohibitive.

In order to mitigate the black-box nature of purely data-driven approaches, we explored the development and application of physics-informed neural networks (PINNs) in beam mechanics, solid mechanics, and contact mechanics, covering both small and large deformation regimes. Each of these problem domains introduces distinct challenges that help illustrate the capabilities, strengths, and limitations of PINNs.

In the context of beam mechanics, we addressed challenges of solving governing equations that contain fourth-order derivatives, such as those in the Euler-Bernoulli beam theory. To demonstrate the capability of PINNs in handling high-order derivatives, we developed models for both static and dynamic beam problems. The complexity was further increased by introducing nonlinear loading conditions and beams with spatially varying cross-sections. Despite these challenges, PINNs successfully captured the physical behavior with high accuracy. However, when applied to inverse problems, a well-documented issue with PINNs is the non-uniqueness of the identified parameters. That is, multiple sets of parameters may satisfy the governing equations and observed data equally well. To highlight this limitation, we considered a data-driven discovery problem

where the goal was to identify the unknown applied loading from observed structural response data. As expected, the PINN successfully predicted a valid loading distribution that satisfied the equations, but it was not the unique solution we were targeting. To address this issue, we introduced prior knowledge in the form of additional constraints on the loading. Incorporating this information helped guide the solution toward the physically correct configuration. This finding emphasizes the importance of embedding domain knowledge into the PINN formulation to enhance the reliability and interpretability of inverse modeling tasks.

In the context of solid mechanics, we first focused on problems governed by infinitesimal strain theory, i.e., linear elasticity. We introduced and compared displacement-based and mixed-variable PINN formulations for solving elasticity problems. The role of output transformation, particularly output scaling, was emphasized as a crucial step for improving training efficiency and solution accuracy. While the displacement-based approach requires fewer loss terms, the mixed-variable formulation allows traction boundary conditions to be enforced as hard constraints and reduces the order of required derivatives. Both formulations were developed using the strong form of the linear elasticity equations. In the first example–the classical Lamé problem–we demonstrated that output transformation is essential. Without output transformation, the optimizer failed to handle large output magnitudes and could not recover the true solution. In contrast to the beam mechanics case, material parameters were treated as scalar values rather than distributed functions, and PINNs successfully identified these parameters with high accuracy. To further evaluate the performance of PINNs, we applied the framework to high-dimensional problems, such as a 3D hollow sphere subjected to time-dependent loading, resulting in a 4D input space. The results showed that PINNs could accurately capture the underlying physics, although computational cost increases with higher input dimensionality.

Next, we shifted our focus to contact problems under small deformations. In this part, PINNs were applied using the mixed-variable formulation due to the advantages discussed earlier, such as the ability to impose traction boundary conditions and reduce derivative orders. Contact problems inherently involve inequality constraints in the form of Karush-Kuhn-Tucker (KKT) conditions. To address these, we implemented several strategies, including sign-based and sigmoid-based formulations inspired by similar approaches in elastoplasticity studies via PINNs. As an alternative to standard soft constraint enforcement, we also investigated the use of nonlinear complementarity problem functions. Specifically, the Fischer-Burmeister function was employed to impose contact constraints in a smooth and differentiable manner. The methodology was tested on benchmark problems, including a simple contact patch test and the classical 2D Hertzian contact problem. For the Hertzian case, we explored multiple setups, including a data-enhanced PINN model guided by finite element simulations and a PINN-based surrogate model using the applied load as input. The data-enhanced model showed superior accuracy by leveraging simulation results to better inform the learning process. The surrogate model, however, struggled to generalize well due to fixed loss weights and a lack of adaptive loss weights balancing, which is critical when applied pressure significantly influences the contact response. PINNs, in this case, were sensitive to weight scaling and captured the target solution only partially. To further challenge the framework, we extended our study to 3D contact problems. These results demonstrated that PINNs retain their ability to handle geometrically complex scenarios, but performance strongly depends on proper constraint formulation and loss weighting strategies.

In the final application of PINNs to solid and contact mechanics, we addressed problems involving large deformations. This part of the study employed the principle of minimum potential

energy rather than the strong form residuals used in vanilla PINNs. Accordingly, we developed element-based energy PINNs (EBE-PINNs), which minimize a scalar energy functional over the domain to approximate the solution. Since the energy formulation involves integral evaluations, we partitioned the domain into smaller subdomains, inspired by the finite element method (FEM). To define the strain energy functional, we employed a hyperelastic Neo-Hookean material model capable of capturing large nonlinear deformations. The approach was first validated on a 2D beam bending problem, demonstrating the ability of EBE-PINNs to handle geometric nonlinearity effectively. We then extended the formulation to 3D settings, exemplified by a torsion problem applied to a square prism. The displacement fields computed by EBE-PINNs showed excellent agreement with FEM results. However, localized discrepancies in strain and stress fields were observed near sharp edges due to stress concentration effects, a well-known challenge in FEM as well. Next, we further increased the level of complexity by incorporating contact constraints in addition to geometric nonlinearity. The contact interactions were modeled using a penalty-based contact potential. This formulation was applied to highly nonlinear 3D problems, including a spherical contact scenario and a torus contact instability problem. Moreover, it was observed that EBE-PINNs are competitive with FEM in terms of overall computational cost. While the training phase may require substantial time, it remains comparable to or faster than the total run time of FEM. More importantly, once trained, the prediction time of EBE-PINNs is exceptionally fast and cannot be matched by traditional numerical solvers, making them highly suitable for real-time or multi-query scenarios. These examples demonstrated the potential of EBE-PINNs to handle complex contact phenomena in the large deformation regime, highlighting the flexibility of quasi-variational PINN formulations in advanced solid mechanics applications.

Until now, PINNs and their variations have primarily been formulated without incorporating experimental or sensor data. While some studies have enhanced PINN models using synthetic data generated from analytical solutions or finite element simulations, such data types are inherently deterministic and noise-free. In contrast, real-world structures generate large amounts of data through sensors, often accompanied by significant measurement noise and uncertainty. To address this gap, the final part of this thesis introduced the concept of hybrid digital twins, demonstrating their potential by integrating physics-informed models with real sensor data. A proof of concept was established using a steel-reinforced concrete beam, showcasing the end-to-end workflow for linking physical assets with virtual models through sensor-based monitoring systems. We explored both temporal and spatial surrogate modeling strategies for reduced-order modeling (ROM) of structural response. In the temporal domain, a PINN-based ROM was developed by taking inspiration from the motion of a simple harmonic oscillator and was evaluated against a purely data-driven model. For the spatial domain, PINN-based ROMs were constructed using the momentum equation, with different weightings applied to balance the contributions of physics and data. Our findings show that PINNs can accurately capture both time-dependent and spatial responses when trained on well-posed data. These observations suggest that hybrid digital twins hold considerable promise for structural health monitoring applications. By combining sensor-driven data with physics-informed models, they offer a flexible and interpretable framework for real-time digital twin development in engineering practice. However, model performance is highly sensitive to ROM design factors, including challenges in integrating noisy sensor data into the governing physics of the target problem, as well as the choice and tuning of loss weighting strategies.

A question that has gained increasing attention is whether data-driven methods such as PINNs

will eventually replace classical numerical techniques like the finite element method (FEM). This thesis has two perspectives on the matter. The first perspective considers scenarios where large amounts of data are available. In such cases, data-driven models can serve as powerful tools–not to replace, but to enhance classical solvers, or vice versa. While some argue that they may eventually supersede traditional methods, one of the key limitations of purely data-driven models is their lack of extrapolation capability. These models cannot inherently detect whether a given input scenario lies outside their training distribution. Thus, to ensure robustness, one would need to collect an extensive dataset covering nearly all relevant scenarios–an effort that demands a massive amount of data. The second perspective assumes limited data availability but leverages physical knowledge of the underlying system, as exemplified by PINNs. In this thesis, we applied PINNs and their variants to a range of benchmark problems, progressively increasing the complexity up to large deformation and contact mechanics applications. The results demonstrate that PINNs exhibit strong potential to act as alternatives to traditional numerical methods. However, they also introduce challenges, particularly in terms of training stability, loss balancing, and convergence–issues that have been discussed throughout this work. It is important to note that these challenges are not unique to PINNs. Even classical methods like FEM can be difficult to implement and solve efficiently, especially for complex geometries or nonlinear problems. Both data-driven and traditional approaches have their respective strengths and limitations. Therefore, I strongly believe that hybrid strategies, which combine the physical grounding of classical methods with the flexibility and efficiency of data-driven models, hold great promise. Such mixed frameworks can capitalize on the advantages of both paradigms to offer accurate, efficient, and scalable solutions for computational mechanics.

This thesis opens several possibilities for future research and exploration. In the context of PINNs, we have addressed nonlinearities arising from geometry and contact interactions. However, material nonlinearity remains an open area for investigation. For instance, it would be of interest to apply the Fischer-Burmeister NCP function to elastoplasticity problems under small deformations–a direction that has not yet been explored in existing literature. Moreover, the EBE-PINN framework could be further extended to handle time-dependent problems, enabling its application to dynamic large deformation scenarios involving complex material behavior and evolving contact conditions. Another potential direction involves the use of augmented Lagrangian formulations to improve the enforcement of contact constraints within energy PINN frameworks. Furthermore, all contact examples in this thesis have involved interactions between an elastic body and a rigid domain. A natural next step would be to investigate contact between two elastic bodies, which would require the development of more advanced gap formulations. On the other hand, the development of variational PINNs may offer further improvements in solution accuracy and physical consistency, though this remains a challenging and ongoing area of study. Beyond formulation extensions, PINNs are known to be sensitive to hyperparameter choices. Future work could focus on systematic hyperparameter optimization, including tuning of loss weights, network architectures, and activation functions. Going a step further, establishing a generic guideline for PINN hyperparameter selection–independent of the target problem–could significantly enhance model reliability and usability in practical applications.

In parallel with PINN-based methods, surrogate modeling techniques present another promising research direction—particularly for rough surface contact problems. In this thesis, surface geometries were generated based on the Random Midpoint Displacement (RMD) method. However, more advanced and diverse surface generation techniques, such as spectral synthesis or

generative models, could be employed to improve the representativeness and diversity of the training data. Moreover, the current database could be expanded to include a broader set of statistical descriptors—such as higher-order moments or frequency domain features—to better capture the complexity of rough surfaces. Leveraging such enriched datasets, future work could explore the development of more accurate and generalizable surrogate models, potentially based on neural networks, to predict contact mechanics quantities (e.g., pressure distribution, real contact area) with high efficiency and accuracy. Moreover, convolutional neural networks (CNNs) could be employed to automatically extract spatial features directly from surface height maps. Instead of limiting predictions to scalar outputs like effective contact area, these models could be trained to predict full-field outputs—such as spatially varying contact pressure or contact area distributions—offering a more detailed and robust surrogate modeling approach.

A. Comparison of GP Regressor and Kernel Ridge

In the kernel-based regression, the predicted output $\hat{y}$ is calculated as

$$\hat{y} = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}, \boldsymbol{x}_i), \tag{A.1}$$

where α is the dual parameter, κ is the kernel function and N denotes number of sample points. The dual parameters for both models are calculated as

$$\begin{aligned} \alpha &= (\mathbf{K} + {\sigma_n}^2\mathbf{I})^{-1}\boldsymbol{y} \rightarrow \text{GP Regressor} \\ \alpha &= (\mathbf{K} + \lambda\mathbf{I})^{-1}\boldsymbol{y} \quad \rightarrow \text{Kernel Ridge} \end{aligned}$$

where $\mathbf{K}$ is the kernel matrix, λ is the regularization term, $\mathbf{I}$ is the identity matrix, and ${\sigma_n}^2$ is the noise variance. As long as the same kernel function with identical kernel parameters is used and the condition $\lambda = {\sigma_n}^2$ is met, the mean predictions of the GP Regressor closely follow the predictions of Kernel Ridge. To verify this, two cases are considered. In the first case, a GP Regressor using the Radial Basis Function (RBF) kernel with the length scale of 1 and ${\sigma_n}^2 = 1 \times 10^{-10}$, along with a Kernel Ridge using the same kernel with $\lambda = 1 \times 10^{-10}$, are trained. In this case, both models demonstrate strong alignment between observed data and their respective predictions. Additionally, both predictions closely match each other, as depicted in Fig. A.1. In the second case, illustrated in Fig. A.2, a GP Regressor using the DotProduct kernel with ${\sigma_n}^2 = 0$ (an equivalent representation of a linear kernel) is compared to a Kernel Ridge employing the linear kernel. Here, although there is a noticeable misalignment between observed data and predictions, the predictions of both models closely match each other, resembling the behavior observed in the first case. Moreover, in both scenarios, point-wise errors between predictions of GP Regressor and Kernel Ridge confirm these observations. Note that the kernel used in the first case is the kernel of the baseline GP Regressor, while the kernel used in the second case is the kernel of the baseline Kernel Ridge (see Fig. 4.5). Therefore, we can see similar behavior on effects on the predictions.

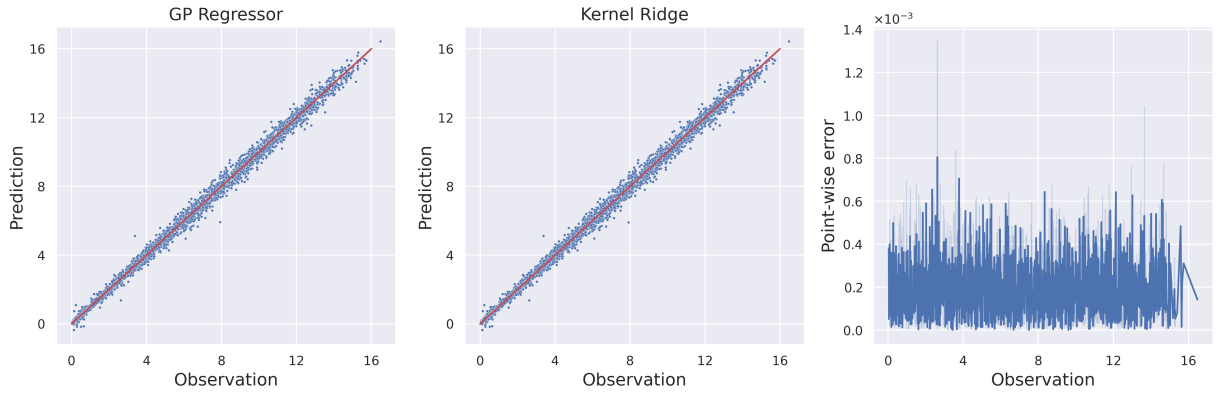


Figure A.1.: Case 1: Good generalization.

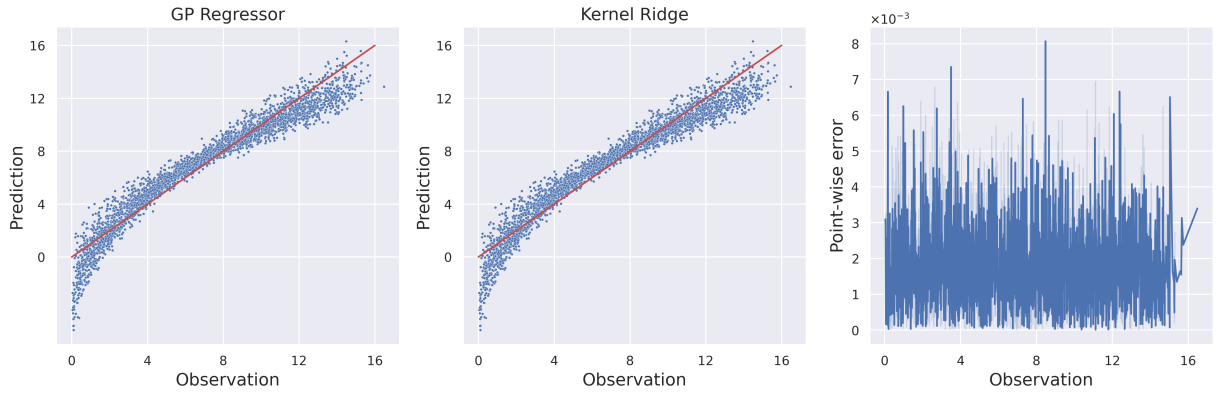


Figure A.2.: Case 2: Bad generalization.

Figure A.3.: Comparison of GP Regressor and Kernel Ridge under the assumption of both models having the same kernel including identical kernel parameters and the regularization parameter.

Bibliography

[1] 4C: 4C: A Comprehensive Multi-Physics Simulation Framework. https://www.4c-multiphysics.org (2025). Accessed: 09.02.2025

[2] Adams, M.F.: Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics. Numerical Linear Algebra with Applications **11**(2–3), 141–153 (2004). DOI 10.1002/nla.374. URL https://onlinelibrary.wiley.com/doi/10.1002/nla.374

[3] An, G.: The effects of adding noise during backpropagation training on a generalization performance. Neural Computation **8**(3), 643–674 (1996). DOI 10.1162/neco.1996.8.3.643. URL https://doi.org/10.1162/neco.1996.8.3.643

[4] Arend Torres, F., Massimo Negri, M., Nagy-Huber, M., Samarin, M., Roth, V.: Mesh-free eulerian physics-informed neural networks. arXiv preprint (2022)

[5] Arf, C.: Makine düşünebilir mi ve nasıl düşünebilir? Atatürk Üniversitesi, Halk Konferansları pp. 91–103 (1959)

[6] Aydin, I., Karakose, M., Akin, E.: Artificial immune based support vector machine algorithm for fault diagnosis of induction motors. In: 2007 International Aegean Conference on Electrical Machines and Power Electronics, pp. 217–221 (2007). DOI 10.1109/ACEMP.2007.4510505

[7] Bai, J., Lin, Z., Wang, Y., Wen, J., Liu, Y., Rabczuk, T., Gu, Y., Feng, X.Q.: Energy-based physics-informed neural network for frictionless contact problems under large deformation. Computer Methods in Applied Mechanics and Engineering **437**, 117787 (2025). DOI https://doi.org/10.1016/j.cma.2025.117787. URL https://www.sciencedirect.com/science/article/pii/S0045782525000593

[8] Barber, J.R.: Contact Mechanics, *Solid Mechanics and Its Applications*, vol. 250. Springer, Cham (2018). DOI 10.1007/978-3-319-70939-0

[9] Barber, J.R., Ciavarella, M.: Contact mechanics. International Journal of Solids and Structures **37**(1), 29–43 (2000). DOI https://doi.org/10.1016/S0020-7683(99)00075-X. URL https://www.sciencedirect.com/science/article/pii/S002076839900075X

[10] Barnsley, M.F., Devaney, R.L., Mandelbrot, B.B., Peitgen, H.O., Saupe, D., Voss, R.F.: The Science of Fractal Images, 1 edn. Springer New York, NY (1988). DOI 10.1007/978-1-4612-3784-6

[11] Bartel, T., Schulte, R., Menzel, A., Kiefer, B., Svendsen, B.: Investigations on enhanced fischer–burmeister ncp functions: application to a rate-dependent model for ferroelectrics. Archive of Applied Mechanics **89**, 995–1010 (2019). DOI 10.1007/s00419-018-1466-7

[12] Batu, T., Lemu, H.G., Shimels, H.: Application of artificial intelligence for surface roughness prediction of additively manufactured components. Materials **16**(18) (2023). DOI 10.3390/ma16186266. URL https://www.mdpi.com/1996-1944/16/18/6266

[13] Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res. **18**(1), 5595–5637 (2017)

[14] Beljkaš, v., Baša, N.: Neural networks–deflection prediction of continuous beams with gfrp reinforcement. Applied Sciences **11**(8) (2021). DOI 10.3390/app11083429. URL https://www.mdpi.com/2076-3417/11/8/3429

[15] Beltrán-Pulido, A., Bilionis, I., Aliprantis, D.: Physics-informed neural networks for solving parametric magnetostatic problems. IEEE Transactions on Energy Conversion **37**(4), 2678–2689 (2022). DOI 10.1109/TEC.2022.3180295

[16] Bemporad, A., Paggi, M.: Optimization algorithms for the solution of the frictionless normal contact between rough surfaces. International Journal of Solids and Structures **69-70**, 94–105 (2015). DOI 10.1016/j.ijsolstr.2015.06.005. URL https://linkinghub.elsevier.com/retrieve/pii/S0020768315002693

[17] Bemporad, A., Paggi, M.: Optimization algorithms for the solution of the frictionless normal contact between rough surfaces. International Journal of Solids and Structures **69-70**, 94–105 (2015). DOI https://doi.org/10.1016/j.ijsolstr.2015.06.005

[18] Bharadwaja, B., Nabian, M.A., Sharma, B., Choudhry, S., Alankar, A.: Physics-informed machine learning and uncertainty quantification for mechanics of heterogeneous materials. Integrating Materials and Manufacturing Innovation pp. 1–21 (2022). DOI 10.1007/s40192-022-00283-2

[19] Bhatti, M.A.: Advanced Topics in Finite Element Analysis of Structures: With Mathematica and MATLAB Computations. John Wiley & Sons, Inc., USA (2006)

[20] Boidi, G., da Silva, M.R., Profito, F.J., Machado, I.F.: Using machine learning radial basis function (rbf) method for predicting lubricated friction on textured and porous surfaces. Surface Topography: Metrology and Properties **8**(4), 044002 (2020). DOI 10.1088/2051-672X/abae13. URL https://dx.doi.org/10.1088/2051-672X/abae13

[21] Bonari, J., Marulli, M.R., Hagmeyer, N., Mayr, M., Popp, A., Paggi, M.: A multi-scale FEM-BEM formulation for contact mechanics between rough surfaces. Computational Mechanics **65**(3), 731–749 (2020). DOI 10.1007/s00466-019-01791-3. URL http://link.springer.com/10.1007/s00466-019-01791-3

[22] Bonari, J., Paggi, M., Dini, D.: A new finite element paradigm to solve contact problems with roughness. International Journal of Solids and Structures **253**, 111643 (2022). DOI 10.1016/j.ijsolstr.2022.111643. URL https://linkinghub.elsevier.com/retrieve/pii/S0020768322001640

[23] Bonari, J., Paggi, M., Reinoso, J.: A framework for the analysis of fully coupled normal and tangential contact problems with complex interfaces. Finite Elements in Analysis and Design **196**, 103605 (2021). DOI 10.1016/j.finel.2021.103605. URL https://linkinghub.elsevier.com/retrieve/pii/S0168874X21000895

[24] Bordón, J.D., Álamo, G.M., Padrón, L.A., Aznárez, J.J., Maeso, O.: Multifebe: A multi-domain finite element–boundary element solver for linear mixed-dimensional mechanical problems. SoftwareX **20**, 101265 (2022). DOI https://doi.org/10.1016/j.softx.2022.101265

[25] Borri, C., Paggi, M.: Topological characterization of antireflective and hydrophobic rough surfaces: are random process theory and fractal modeling applicable? Journal of Physics D: Applied Physics **48**(4), 045301 (2015). DOI 10.1088/0022-3727/48/4/045301

[26] Botchkarev, A.: A new typology design of performance metrics to measure errors in machine learning regression algorithms. Interdisciplinary Journal of Information, Knowledge, and Management **14**, 45 – 79 (2019). DOI https://doi.org/10.28945/4184

[27] Bourdeau, M., qiang Zhai, X., Nefzaoui, E., Guo, X., Chatellier, P.: Modeling and forecasting building energy consumption: A review of data-driven techniques. Sustainable Cities and Society **48**, 101533 (2019). DOI https://doi.org/10.1016/j.scs.2019.101533. URL https://www.sciencedirect.com/science/article/pii/S2210670718323862

[28] Bowden, F.P., Tabor, D.: The Friction and Lubrication of Solids, 1 edn. International Series of Monographs on Physics. Clarendon Press, Oxford (1950)

[29] Braml, T., Wimmer, J., Varabei, Y.: Erfordernisse an die datenaufnahme und -verarbeitung zur erzeugung von intelligenten digitalen zwillingen im ingenieurbau. In: J. Berger (ed.) Innsbrucker Bautage 2022 : Festschrift zum 60. Geburtstag von Univ.-Prof. Dr.-Ing. Jürgen Feix, *Innsbruck – Massivbau und Brückenbau*, vol. 7, pp. 31–49. Studia Universitätsverlag Innsbruck, Innsbruck (2022). Erschien am 17. Mai 2022, 479 Seiten Gesamtumfang des Festschriftenbandes

[30] Brandstaeter, S., Fuchs, S.L., Biehler, J., Aydin, R.C., Wall, W.A., Cyron, C.J.: Global Sensitivity Analysis of a Homogenized Constrained Mixture Model of Arterial Growth and Remodeling. Journal of Elasticity **145**(1-2), 191–221 (2021). DOI 10.1007/s10659-021-09833-9

[31] Brebbia, C., Dominguez, J.: Boundary element methods for potential problems. Applied Mathematical Modelling **1**(7), 372–378 (1977). DOI https://doi.org/10.1016/0307-904X(77)90046-4. URL https://www.sciencedirect.com/science/article/pii/0307904X77900464

[32] Brebbia, C.A. (ed.): Progress in Boundary Element Methods: Volume 2. Springer Science+Business Media / Pentech Press, New York (1983). DOI 10.1007/978-1-4757-6300-3

[33] Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001). DOI 10.1023/A:1010933404324

[34] Brucherseifer, E., Winter, H., Mentges, A., Mühlhäuser, M., Hellmann, M.: Digital twin conceptual framework for improving critical infrastructure resilience. at - Automatisierungstechnik **69**(12), 1062–1080 (2021). DOI doi:10.1515/auto-2021-0104

[35] Budynas, R.G., Nisbett, J.K.: Shigley's Mechanical Engineering Design, 10th edn. McGraw-Hill Education, New York (2014)

[36] Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing **16**(5), 1190–1208 (1995). DOI 10.1137/0916069

[37] Campañá, C., Müser, M.H.: Practical Green's function approach to the simulation of elastic semi-infinite solids. Physical Review B **74**(7), 075420 (2006). DOI 10.1103/PhysRevB.74.075420. URL `https://link.aps.org/doi/10.1103/PhysRevB.74.075420`

[38] Carbone, G., Bottiglione, F.: Asperity contact theories: Do they predict linearity between contact area and load? Journal of the Mechanics and Physics of Solids **56**(8), 2555–2572 (2008). DOI 10.1016/j.jmps.2008.03.011. URL `https://linkinghub.elsevier.com/retrieve/pii/S0022509608000586`

[39] Carvalho, R.P., Carneiro, A.M.C., Pires, F.M.A., Popp, A.: An efficient algorithm for rigid/deformable contact interaction based on the dual mortar method. Computational Mechanics **71**(1), 143–167 (2023). DOI 10.1007/s00466-022-02226-2. URL `https://link.springer.com/10.1007/s00466-022-02226-2`

[40] Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pp. 785–794. Association for Computing Machinery, New York, NY, USA (2016). DOI 10.1145/2939672.2939785. URL `https://doi.org/10.1145/2939672.2939785`

[41] Chen, Y., Lu, L., Karniadakis, G.E., Dal Negro, L.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express **28**(8), 11618–11633 (2020). DOI 10.1364/OE.384875

[42] Chinesta, F., Cueto, E., Abisset-Chavanne, E., Duval, J.L., Khaldi, F.E.: Virtual, digital and hybrid twins: A new paradigm in data-based engineering and engineered data. Archives of Computational Methods in Engineering **27**(1), 105–134 (2020). DOI 10.1007/s11831-018-9301-4

[43] Choi, D.K.: Data-driven materials modeling with xgboost algorithm and statistical inference analysis for prediction of fatigue strength of steels. International Journal of Precision Engineering and Manufacturing **20**(1), 129–138 (2019). DOI 10.1007/s12541-019-00048-6. URL https://doi.org/10.1007/s12541-019-00048-6

[44] Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory **13**(1), 21–27 (1967). DOI 10.1109/TIT.1967.1053964

[45] von Danwitz, M., Kochmann, T.T., Sahin, T., Wimmer, J., Braml, T., Popp, A.: Hybrid digital twins: A proof of concept for reinforced concrete beams. Proceedings in Applied Mathematics and Mechanics **22**(1), e202200146 (2023). DOI https://doi.org/10.1002/pamm.202200146. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.202200146

[46] Depina, I., Jain, S., Mar Valsson, S., Gotovac, H.: Application of physics-informed neural networks to inverse problems in unsaturated groundwater flow. Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards **16**(1), 21–36 (2022). DOI 10.1080/17499518.2021.1971251

[47] Dinkel, M., Geitner, C.M., Robalo Rei, G., Nitzler, J., Wall, W.A.: Solving Bayesian inverse problems with expensive likelihoods using constrained Gaussian processes and active learning. Inverse Problems **40**(9), 095008 (2024). DOI 10.1088/1361-6420/ad5eb4

[48] Draper, N., Smith, H.: Applied Regression Analysis. Wiley Series in Probability and Statistics. Wiley (1998). URL https://books.google.de/books?id=d6NsDwAAQBAJ

[49] Eivazi, H., Vinuesa, R.: Physics-informed deep-learning applications to experimental fluid mechanics. arXiv preprint (2022)

[50] Elangovan, M., Sakthivel, N., Saravanamurugan, S., Nair, B., Sugumaran, V.: Machine learning approach to the prediction of surface roughness using statistical features of vibration signal acquired in turning. Procedia Computer Science **50**, 282–288 (2015). DOI https://doi.org/10.1016/j.procs.2015.04.047. URL https://www.sciencedirect.com/science/article/pii/S1877050915005487. Big Data, Cloud and Computing Challenges

[51] Feurer, M., Hutter, F.: Hyperparameter optimization. In: F. Hutter, L. Kotthoff, J. Vanschoren (eds.) Automated Machine Learning: Methods, Systems, Challenges, pp. 3–33. Springer International Publishing, Cham (2019). DOI 10.1007/978-3-030-05318-5_1

[52] Fischer, A.: A special newton-type optimization method. Optimization **24**(3-4), 269–284 (1992). DOI 10.1080/02331939208843795

[53] Fix, E., Hodges, J.L.: Discriminatory analysis. nonparametric discrimination: Consistency properties. Technical Report 4, USA Air Force School of Aviation Medicine, Randolph Field, Texas (1951)

[54] Fournier, A., Fussell, D., Carpenter, L.: Computer rendering of stochastic models. Commun. ACM **25**(6), 371–384 (1982). DOI 10.1145/358523.358553

[55] Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55**(1), 119–139 (1997). DOI https://doi.org/10.1006/jcss.1997.1504

[56] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics **29**(5), 1189 – 1232 (2001). DOI 10.1214/aos/1013203451

[57] Frérot, L., Anciaux, G., Rey, V., Pham-Ba, S., Molinari, J.F.: Tamaas: a library for elastic-plastic contact of periodic rough surfaces. Journal of Open Source Software **5**(51), 2121 (2020). DOI 10.21105/joss.02121. URL https://doi.org/10.21105/joss.02121

[58] Ganti, H., Khare, P.: Data-driven surrogate modeling of multiphase flows using machine learning techniques. Computers & Fluids **211**, 104626 (2020). DOI https://doi.org/10.1016/j.compfluid.2020.104626

[59] Géron, A.: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc. (2022)

[60] Geuzaine, C., Remacle, J.F.: Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering **79**(11), 1309–1331 (2009). DOI 10.1002/nme.2579

[61] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Y.W. Teh, M. Titterington (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, *Proceedings of Machine Learning Research*, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, Italy (2010). URL https://proceedings.mlr.press/v9/glorot10a.html

[62] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org

[63] Goodman, R., Kish, L.: Controlled selection—a technique in probability sampling. Journal of the American Statistical Association **45**(251), 350–372 (1950). DOI 10.1080/01621459.1950.10501130. URL https://doi.org/10.1080/01621459.1950.10501130

[64] Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A.: Revisiting deep learning models for tabular data. In: Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS '21. Curran Associates Inc., Red Hook, NY, USA (2021)

[65] Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization. In: Proceedings of the 18th International Conference on Neural Information Processing Systems, NIPS'04, pp. 529–536. MIT Press, Cambridge, MA, USA (2004)

[66] Greenwood, J.A., Williamson, J.B.P., Bowden, F.P.: Contact of nominally flat surfaces. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **295**(1442), 300–319 (1966). DOI 10.1098/rspa.1966.0242

[67] Grieves, M.: Digital twin: manufacturing excellence through virtual factory replication. White paper **1**(2014), 1–7 (2014)

[68] Grohs, P., Hornung, F., Jentzen, A., von Wurstemberger, P.: A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, vol. 284. American Mathematical Society (2023)

[69] Haghighat, E., Abouali, S., Vaziri, R.: Constitutive model characterization and discovery using physics-informed deep learning. Engineering Applications of Artificial Intelligence **120**, 105828 (2023). DOI 10.1016/j.engappai.2023.105828

[70] Haghighat, E., Raissi, M., Moure, A., Gomez, H., Juanes, R.: A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Computer Methods in Applied Mechanics and Engineering **379**, 113741 (2021). DOI 10.1016/j.cma.2021.113741

[71] Hasan, M.S., Nosonovsky, M.: Triboinformatics: machine learning algorithms and data topology methods for tribology. Surface Innovations **10**(4-5), 229–242 (2022). DOI 10.1680/jsuin.22.00027. URL https://doi.org/10.1680/jsuin.22.00027

[72] Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2 edn. Springer Series in Statistics. Springer, New York (2009). DOI 10.1007/978-0-387-84858-7. URL https://link.springer.com/book/10.1007/978-0-387-84858-7

[73] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV '15, p. 1026–1034. IEEE Computer Society, USA (2015). DOI 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123

[74] Heidenreich, J.N., Bonatti, C., Mohr, D.: Transfer learning of recurrent neural network-based plasticity models. International Journal for Numerical Methods in Engineering **125**(1), e7357 (2024). DOI https://doi.org/10.1002/nme.7357. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.7357

[75] Hoerl, A.E., Kennard, R.W.: Ridge regression: Biased estimation for nonorthogonal problems. Technometrics **12**(1), 55–67 (1970). URL http://www.jstor.org/stable/1267351

[76] Hu, Z., Zhang, J., Ge, Y.: Handling vanishing gradient problem using artificial derivative. IEEE Access **9**, 22371–22377 (2021). DOI 10.1109/ACCESS.2021.3054915

[77] Hürkamp, A., Gellrich, S., Ossowski, T., Beuscher, J., Thiede, S., Herrmann, C., Dröder, K.: Combining simulation and machine learning as digital twin for the manufacturing of overmolded thermoplastic composites. Journal of Manufacturing and Materials Processing **4**(3) (2020). DOI 10.3390/jmmp4030092. URL https://www.mdpi.com/2504-4494/4/3/92

[78] Hyndman, R.J., Koehler, A.B.: Another look at measures of forecast accuracy. International Journal of Forecasting **22**(4), 679–688 (2006). DOI https://doi.org/10.1016/j.ijforecast.2006.03.001

[79] Hyun, S., Pei, L., Molinari, J.F., Robbins, M.O.: Finite-element analysis of contact between elastic self-affine surfaces. Phys. Rev. E **70**, 026117 (2004). DOI 10.1103/PhysRevE.70.026117. URL https://link.aps.org/doi/10.1103/PhysRevE.70.026117

[80] Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, pp. 448–456. JMLR.org (2015)

[81] Jacq, C., Nélias, D., Lormand, G., Girodin, D.: Development of a three-dimensional semi-analytical elastic-plastic contact code. Journal of Tribology **124**(4), 653 – 667 (2002). DOI 10.1115/1.1467920. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0036799432&doi=10.1115%2f1.1467920&partnerID=40&md5=956f3cae91a72df18c01f43f63a2f3e4. Type: Article

[82] Jiang, Y., Yin, S., Li, K., Luo, H., Kaynak, O.: Industrial applications of digital twins. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **379**(2207), 20200360 (2021). DOI 10.1098/rsta.2020.0360

[83] Jin, H., Zhang, E., Espinosa, H.D.: Recent advances and applications of machine learning in experimental solid mechanics: A review. Applied Mechanics Reviews **75**(6), 061001 (2023). DOI 10.1115/1.4062966. URL https://doi.org/10.1115/1.4062966

[84] Johnson, K.L.: Contact Mechanics. Cambridge University Press (1985). DOI 10.1017/CBO9781139171731

[85] Kaczmarek, M., Szymańska, A.: Application of artificial neural networks to predict the deflections of reinforced concrete beams. Studia Geotechnica et Mechanica **38**(2), 37–46 (2016). DOI 10.1515/sgem-2016-0017. URL https://doi.org/10.1515/sgem-2016-0017

[86] Kalliorinne, K., Larsson, R., Pérez-Ràfols, F., Liwicki, M., Almqvist, A.: Artificial neural network architecture for prediction of contact mechanical response. Frontiers in Mechanical Engineering **6**, 579825 (2021). DOI 10.3389/fmech.2020.579825

[87] Karapiperis, K., Stainier, L., Ortiz, M., Andrade, J.: Data-driven multiscale modeling in mechanics. Journal of the Mechanics and Physics of Solids **147**, 104239 (2021). DOI https://doi.org/10.1016/j.jmps.2020.104239. URL https://www.sciencedirect.com/science/article/pii/S0022509620304531

[88] Khan, A., Lowther, D.A.: Physics informed neural networks for electromagnetic analysis. IEEE Transactions on Magnetics **58**(9), 1–4 (2022). DOI 10.1109/TMAG.2022.3161814

[89] Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-vpinns: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering **374**, 113547 (2021). DOI https://doi.org/10.1016/j.cma.2020.113547

[90] Kikuchi, N., Oden, J.T.: Contact Problems in Elasticity. Society for Industrial and Applied Mathematics (1988). DOI 10.1137/1.9781611970845

[91] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Y. Bengio, Y. LeCun (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015). URL http://arxiv.org/abs/1412.6980

[92] Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, pp. 1137–1143. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)

[93] Kollmannsberger, S., D'Angella, D., Jokeit, M., Herrmann, L.: Deep Learning in Computational Mechanics: An Introductory Course, vol. 977, pp. 1–3. Springer International Publishing, Cham (2021). DOI 10.1007/978-3-030-76587-3_1

[94] Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks **9**(5), 987–1000 (1998). DOI 10.1109/72.712178

[95] Laursen, T.A.: Computational Contact and Impact Mechanics: Fundamentals of Modeling Interfacial Phenomena in Nonlinear Finite Element Analysis. Springer, Berlin, Heidelberg (2003). DOI 10.1007/978-3-662-04864-1. URL https://link.springer.com/book/10.1007/978-3-662-04864-1

[96] Lawrence, J.: Introduction to Neural Networks. California Scientific Software, USA (1993)

[97] LeCun, Y., Bottou, L., Orr, G., Müller, K.: Efficient backprop, pp. 9–48. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag (2012). DOI 10.1007/978-3-642-35289-8_3. Copyright: Copyright 2021 Elsevier B.V., All rights reserved.

[98] Li, M., Dankowicz, H.: Optimization with equality and inequality constraints using parameter continuation. Applied Mathematics and Computation **375**, 125058 (2020). DOI 10.1016/j.amc.2020.125058

[99] Li, S., Shen, Z., Xiong, G.: A k-nearest neighbor locally weighted regression method for short-term traffic flow forecasting. In: 2012 15th International IEEE Conference on Intelligent Transportation Systems, pp. 1596–1601 (2012). DOI 10.1109/ITSC.2012.6338648

[100] Liu, Q., Wu, Y.: Supervised learning. In: N.M. Seel (ed.) Encyclopedia of the Sciences of Learning, pp. 3243–3245. Springer US, Boston, MA (2012). DOI 10.1007/978-1-4419-1428-6_451. URL https://doi.org/10.1007/978-1-4419-1428-6_451

[101] Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: A deep learning library for solving differential equations. SIAM Review **63**(1), 208–228 (2021). DOI 10.1137/19M1274067

[102] Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G.: Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing **43**(6), B1105–B1132 (2021). DOI 10.1137/21M1397908

[103] Mandelbrot, B.B.: The Fractal Geometry of Nature, 3 edn. W. H. Freeman and Company, New York (1983)

[104] Maria Antony, A.N., Narisetti, N., Gladilin, E.: Linel2d-net: A deep learning approach to solving 2d linear elastic boundary value problems on image domains. iScience **27**(4), 109519 (2024). DOI https://doi.org/10.1016/j.isci.2024.109519. URL https://www.sciencedirect.com/science/article/pii/S2589004224007405

[105] Markidis, S.: The old and the new: Can physics-informed deep-learning replace traditional linear solvers? Frontiers in Big Data **4** (2021). DOI 10.3389/fdata.2021.669097

[106] Maurizi, M., Gao, C., Berto, F.: Predicting stress, strain and deformation fields in materials and structures with graph neural networks. Scientific Reports **12**(1), 21834 (2022). DOI 10.1038/s41598-022-26424-3. URL https://doi.org/10.1038/s41598-022-26424-3

[107] Mayr, M., Popp, A.: Scalable computational kernels for mortar finite element methods. Engineering with Computers **39**(5), 3691–3720 (2023). DOI 10.1007/s00366-022-01779-3. URL https://doi.org/10.1007/s00366-022-01779-3

[108] Mitchell, T.M.: Machine Learning, 1 edn. McGraw-Hill, Inc., USA (1997)

[109] Montáns, F.J., Chinesta, F., Gómez-Bombarelli, R., Kutz, J.N.: Data-driven modeling and learning in science and engineering. Comptes Rendus Mécanique **347**(11), 845–855 (2019). DOI https://doi.org/10.1016/j.crme.2019.11.009. URL https://www.sciencedirect.com/science/article/pii/S1631072119301809. Data-Based Engineering Science and Technology

[110] Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)

[111] Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge, MA (2012)

[112] Müser, M.H., Dapp, W.B., Bugnicourt, R., Sainsot, P., Lesaffre, N., Lubrecht, T.A., Persson, B.N.J., Harris, K., Bennett, A., Schulze, K., Rohde, S., Ifju, P., Sawyer, W.G., Angelini, T., Ashtari Esfahani, H., Kadkhodaei, M., Akbarzadeh, S., Wu, J.J., Vorlaufer, G., Vernes, A., Solhjoo, S., Vakis, A.I., Jackson, R.L., Xu, Y., Streator, J., Rostami, A., Dini, D., Medina, S., Carbone, G., Bottiglione, F., Afferrante, L., Monti, J., Pastewka, L., Robbins, M.O., Greenwood, J.A.: Meeting the Contact-Mechanics Challenge. Tribology Letters **65**(4), 118 (2017). DOI 10.1007/s11249-017-0900-2. URL http://link.springer.com/10.1007/s11249-017-0900-2

[113] Nie, Z., Jiang, H., Kara, L.B.: Stress field prediction in cantilevered structures using convolutional neural networks. Journal of Computing and Information Science in Engineering **20**(1), 011002 (2019). DOI 10.1115/1.4044097

[114] Oishi, A., Yagawa, G.: A surface-to-surface contact search method enhanced by deep learning. Computational Mechanics **65**(4), 1125–1147 (2020). DOI 10.1007/s00466-019-01811-2. URL https://doi.org/10.1007/s00466-019-01811-2

[115] Paggi, M., Ciavarella, M.: The coefficient of proportionality k between real contact area and load, with new asperity models. Wear **268**(7-8), 1020–1029 (2010). DOI 10.1016/j.wear.2009.12.038. URL https://linkinghub.elsevier.com/retrieve/pii/S0043164809006607

[116] Paggi, M., Ciavarella, M.: The coefficient of proportionality κ between real contact area and load, with new asperity models. Wear **268**(7), 1020–1029 (2010). DOI https://doi.org/10.1016/j.wear.2009.12.038

[117] Paggi, M., Reinoso, J.: A variational approach with embedded roughness for adhesive contact problems. Mechanics of Advanced Materials and Structures **27**(20), 1731–1747 (2020). DOI 10.1080/15376494.2018.1525454. URL https://www.tandfonline.com/doi/full/10.1080/15376494.2018.1525454

[118] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[119] Pei, L., Hyun, S., Molinari, J., Robbins, M.: Finite element modeling of elasto-plastic contact between rough surfaces. Journal of the Mechanics and Physics of Solids **53**(11), 2385–2409 (2005). DOI 10.1016/j.jmps.2005.06.008. URL https://linkinghub.elsevier.com/retrieve/pii/S0022509605001225

[120] Persson, B.N.J.: On the fractal dimension of rough surfaces. Tribology Letters **54**(1), 99–106 (2014). DOI 10.1007/s11249-014-0313-4

[121] Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q.: Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. International Journal of Automation and Computing **14**(5), 503–519 (2017). DOI 10.1007/s11633-017-1054-2

[122] Pohrt, R., Li, Q.: Complete boundary element formulation for normal and tangential contact problems. Physical Mesomechanics **17**(4), 334–340 (2014). DOI 10.1134/S1029959914040109. URL https://doi.org/10.1134/S1029959914040109

[123] Popov, V.L., Heß, M., Willert, E.: Handbook of Contact Mechanics: Exact Solutions of Axisymmetric Contact Problems. Springer, Berlin, Heidelberg (2019). DOI 10.1007/978-3-662-58709-6. Open access; 347 pages

[124] Popp, A.: Mortar methods for computational contact mechanics and general interface problems. phdthesis. Technische Universität München, Fakultät für Maschinenwesen, München (2012). URL https://mediatum.ub.tum.de/node?id=1109994

[125] Popp, A., Gee, M.W., Wall, W.A.: A finite deformation mortar contact formulation using a primal–dual active set strategy. International Journal for Numerical Methods in Engineering **79**(11), 1354–1391 (2009). DOI 10.1002/nme.2614

[126] Popp, A., Gitterle, M., Gee, M.W., Wall, W.A.: A dual mortar approach for 3d finite deformation contact with consistent linearization. International Journal for Numerical Methods in Engineering **83**(11), 1428–1465 (2010). DOI 10.1002/nme.2866

[127] Popp, A., Wall, W.A.: Dual mortar methods for computational contact mechanics – overview and recent developments. GAMM Mitteilungen: Gesellschaft für Angewandte Mathematik und Mechanik **37**(1), 66–84 (2014). DOI 10.1002/gamm.201410004

[128] Popp, A., Wohlmuth, B.I., Gee, M.W., Wall, W.A.: Dual quadratic mortar finite element methods for 3d finite deformation contact. SIAM Journal on Scientific Computing **34**(4), B421–B446 (2012). DOI 10.1137/110848190

[129] Popp, A., Wriggers, P. (eds.): Contact Modeling for Solids and Particles, *CISM International Centre for Mechanical Sciences*, vol. 585. Springer International Publishing (2018). DOI 10.1007/978-3-319-90155-8

[130] Prajapati, D.K., Katiyar, J.K., Prakash, C.: Machine learning approach for the prediction of mixed lubrication parameters for different surface topographies of non-conformal rough contacts. Industrial Lubrication and Tribology **75**(9), 1022–1030 (2023). DOI https://doi.org/10.1108/ILT-04-2023-0121

[131] Prechelt, L.: Early stopping — but when? In: G. Montavon, G.B. Orr, K.R. Müller (eds.) Neural Networks: Tricks of the Trade: Second Edition, pp. 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-35289-8_5. URL https://doi.org/10.1007/978-3-642-35289-8_5

[132] Preetham, S., Ravi, K.H., Prema, K.W.P., Shivaraj, M.: Prediction of defection of reinforced concrete beams using machine learning tools. International Journal of Engineering

Research & Technology (IJERT) **4**(5), 883–886 (2015). DOI 10.17577/IJERTV4IS050936. URL https://doi.org/10.17577/IJERTV4IS050936

[133] Putignano, C., Carbone, G.: A review of boundary elements methodologies for elastic and viscoelastic rough contact mechanics. Physical Mesomechanics **17**(4), 321–333 (2014). DOI 10.1134/S1029959914040092. URL https://doi.org/10.1134/S1029959914040092

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

[135] Rao, C., Sun, H., Liu, Y.: Physics-informed deep learning for incompressible laminar flows. Theoretical and Applied Mechanics Letters **10**(3), 207–212 (2020). DOI 10.1016/j.taml.2020.01.039

[136] Rao, C., Sun, H., Liu, Y.: Physics-informed deep learning for computational elastodynamics without labeled data. Journal of Engineering Mechanics **147**(8), 04021043 (2021). DOI 10.1061/(ASCE)EM.1943-7889.0001947

[137] Rapetto, M., Almqvist, A., Larsson, R., Lugt, P.: On the influence of surface roughness on real area of contact in normal, dry, friction free, rough contact by using a neural network. Wear **266**(5), 592–595 (2009). DOI https://doi.org/10.1016/j.wear.2008.04.059. URL https://www.sciencedirect.com/science/article/pii/S0043164808003177. Metrology and Properties of Engineering Surfaces

[138] Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. The MIT Press (2005). DOI 10.7551/mitpress/3206.001.0001

[139] Reissner, E.: On a variational theorem in elasticity. Journal of Mathematics and Physics **29**(1-4), 90–95 (1950). DOI 10.1002/sapm195029190

[140] Ren, P., Rao, C., Chen, S., Wang, J.X., Sun, H., Liu, Y.: Seismicnet: Physics-informed neural networks for seismic wave modeling in semi-infinite domain. Computer Physics Communications **295**, 109010 (2024). DOI https://doi.org/10.1016/j.cpc.2023.109010. URL https://www.sciencedirect.com/science/article/pii/S0010465523003557

[141] Rocha, F., Deparis, S., Antolin, P., Buffa, A.: Deepbnd: A machine learning approach to enhance multiscale solid mechanics. Journal of Computational Physics **479**, 111996 (2023). DOI https://doi.org/10.1016/j.jcp.2023.111996. URL https://www.sciencedirect.com/science/article/pii/S0021999123000918

[142] S. N. Atluri H. T. Liu, Z.D.H.: Meshless local petrov-galerkin (mlpg) mixed collocation method for elasticity problems. Computer Modeling in Engineering & Sciences **14**(3), 141–152 (2006). DOI 10.3970/cmes.2006.014.141

[143] Sahin, T., Bonari, J., Brandstaeter, S., Popp, A.: Data-driven surrogate modeling techniques to predict the effective contact area of rough surface contact problems. arXiv preprint arXiv:2504.17354 (2025)

[144] Sahin, T., Von Danwitz, M., Popp, A.: Solving forward and inverse problems of contact mechanics using physics-informed neural networks. Advanced Modeling and Simulation in Engineering Sciences **11**(1), 11 (2024). DOI 10.1186/s40323-024-00265-3

[145] Sahin, T., Wolff, D., von Danwitz, M., Popp, A.: Towards a hybrid digital twin: Fusing sensor information and physics in surrogate modeling of a reinforced concrete beam. In: 2024 Sensor Data Fusion: Trends, Solutions, Applications (SDF), pp. 1–8 (2024). DOI 10.1109/SDF63218.2024.10773885

[146] Sahin, T., Wolff, D., Popp, A.: Physics-informed neural networks for solving contact problems in three dimensions. arXiv preprint arXiv:2412.09022 (2024)

[147] Saidaoui, H., Espath, L., Tempone, R.: Deep nurbs–admissible neural networks. arXiv preprint (2022)

[148] Samaniego, E., Anitescu, C., Goswami, S., Nguyen-Thanh, V.M., Guo, H., Hamdia, K., Zhuang, X., Rabczuk, T.: An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. Computer Methods in Applied Mechanics and Engineering **362**, 112790 (2020). DOI 10.1016/j.cma.2019.112790

[149] Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3**(3), 210–229 (1959). DOI 10.1147/rd.33.0210

[150] San, O., Rasheed, A., Kvamsdal, T.: Hybrid analysis and modeling, eclecticism, and multifidelity computing toward digital twin revolution. GAMM-Mitteilungen **44**(2), e202100007 (2021). DOI https://doi.org/10.1002/gamm.202100007

[151] Sanhueza, R.D., Akkerman, I., Peeters, J.W.: Machine learning for the prediction of the local skin friction factors and nusselt numbers in turbulent flows past rough surfaces. International Journal of Heat and Fluid Flow **103**, 109204 (2023). DOI https://doi.org/10.1016/j.ijheatfluidflow.2023.109204. URL `https://www.sciencedirect.com/science/article/pii/S0142727X23001030`

[152] Santapuri, S., Lowe, R.L., Bechtel, S.E.: Chapter 9 - modeling of thermo-electro-magneto-mechanical behavior, with application to smart materials. In: S.E. Bechtel, R.L. Lowe (eds.) Fundamentals of Continuum Mechanics, pp. 249–303. Academic Press (2015). DOI 10.1016/B978-0-12-394600-3.00009-5

[153] Schillinger, D., Evans, J.A., Frischmann, F., Hiemstra, R.R., Hsu, M.C., Hughes, T.J.: A collocated c0 finite element method: Reduced quadrature perspective, cost comparison with standard finite elements, and explicit structural dynamics. International Journal for Numerical Methods in Engineering **102**(3-4), 576–631 (2015). DOI https://doi.org/10.1002/nme.4783. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4783`

[154] Seitz, A., Farah, P., Kremheller, J., Wohlmuth, B.I., Wall, W.A., Popp, A.: Isogeometric dual mortar methods for computational contact mechanics. Computer Methods in Applied Mechanics and Engineering **301**, 259–280 (2016). DOI 10.1016/j.cma.2015.12.018

[155] Shaheen, B., Ádám Kocsis, Németh, I.: Data-driven failure prediction and rul estimation of mechanical components using accumulative artificial neural networks. Engineering Applications of Artificial Intelligence **119**, 105749 (2023). DOI https://doi.org/10.1016/j.engappai.2022.105749. URL https://www.sciencedirect.com/science/article/pii/S0952197622007394

[156] Singh, D., Singh, B.: Investigating the impact of data normalization on classification performance. Applied Soft Computing **97**, 105524 (2020). DOI https://doi.org/10.1016/j.asoc.2019.105524

[157] Singh, M., Srivastava, R., Fuenmayor, E., Kuts, V., Qiao, Y., Murray, N., Devine, D.: Applications of digital twin across industries: A review. Applied Sciences **12**(11) (2022). DOI 10.3390/app12115727

[158] Smith, J.D., Ross, Z.E., Azizzadenesheli, K., Muir, J.B.: Hyposvi: Hypocentre inversion with stein variational inference and physics informed neural networks. Geophysical Journal International **228**(1), 698–710 (2022). DOI 10.1093/gji/ggab309

[159] Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'12, p. 2951–2959. Curran Associates Inc., Red Hook, NY, USA (2012)

[160] Solomatine, D.P., Ostfeld, A.: Data-driven modelling: some past experiences and new approaches. Journal of Hydroinformatics **10**(1), 3–22 (2008). DOI 10.2166/hydro.2008.015. URL https://doi.org/10.2166/hydro.2008.015

[161] Sun, D., Qi, L.: On ncp-functions. Computational Optimization and Applications **13**, 201–220 (1999). DOI 10.1023/A:1008669226453

[162] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. MIT Press (2018). URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/

[163] Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) **58**(1), 267–288 (1996). URL http://www.jstor.org/stable/2346178

[164] Timoshenko, S., Goodier, J.N.: Theory of Elasticity. McGraw-Hill (1951)

[165] Tonti, E.: The reason for analogies between physical theories. Applied Mathematical Modelling **1**(1), 37–50 (1976). DOI 10.1016/0307-904X(76)90023-8

[166] Torzoni, M., Tezzele, M., Mariani, S., Manzoni, A., Willcox, K.E.: A digital twin framework for civil engineering structures. Computer Methods in Applied Mechanics and Engineering **418**, 116584 (2024). DOI https://doi.org/10.1016/j.cma.2023.116584

[167] Vakis, A., Yastrebov, V., Scheibert, J., Nicola, L., Dini, D., Minfray, C., Almqvist, A., Paggi, M., Lee, S., Limbert, G., Molinari, J., Anciaux, G., Aghababaei, R., Echeverri Restrepo, S., Papangelo, A., Cammarata, A., Nicolini, P., Putignano, C., Carbone, G., Stupkiewicz, S., Lengiewicz, J., Costagliola, G., Bosia, F., Guarino, R., Pugno, N., Müser, M., Ciavarella, M.: Modeling and simulation in tribology across scales: An overview. Tribology International **125**, 169–199 (2018). DOI 10.1016/j.triboint.2018.02.005. URL `https://linkinghub.elsevier.com/retrieve/pii/S0301679X18300756`

[168] Vakis, A., Yastrebov, V., Scheibert, J., Nicola, L., Dini, D., Minfray, C., Almqvist, A., Paggi, M., Lee, S., Limbert, G., Molinari, J., Anciaux, G., Aghababaei, R., Echeverri Restrepo, S., Papangelo, A., Cammarata, A., Nicolini, P., Putignano, C., Carbone, G., Stupkiewicz, S., Lengiewicz, J., Costagliola, G., Bosia, F., Guarino, R., Pugno, N., Müser, M., Ciavarella, M.: Modeling and simulation in tribology across scales: An overview. Tribology International **125**, 169–199 (2018). DOI https://doi.org/10.1016/j.triboint.2018.02.005

[169] Vapnik, V.N.: The nature of statistical learning theory. Springer-Verlag New York, Inc. (1995)

[170] Vollebregt, E.A.H.: A new solver for the elastic normal contact problem using conjugate gradients, deflation, and an FFT-based preconditioner. Journal of Computational Physics **257**, 333–351 (2014). DOI 10.1016/j.jcp.2013.10.005. URL `https://www.sciencedirect.com/science/article/pii/S0021999113006724`

[171] Vovk, V.: Kernel ridge regression. In: B. Schölkopf, Z. Luo, V. Vovk (eds.) Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik, pp. 105–116. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-41136-6_11

[172] Wang, J., Qiao, P.: Vibration of beams with arbitrary discontinuities and boundary conditions. Journal of Sound and Vibration **308**(1), 12–27 (2007). DOI https://doi.org/10.1016/j.jsv.2007.06.071. URL `https://www.sciencedirect.com/science/article/pii/S0022460X07004981`

[173] Whitehouse, D.J.: Handbook of Surface Metrology. Routledge & CRC Press (1994). URL `https://www.routledge.com/Handbook-of-Surface-Metrology/Whitehouse/p/book/9780750300391`

[174] Wiesner, T.A., Mayr, M., Popp, A., Gee, M.W., Wall, W.A.: Algebraic multigrid methods for saddle point systems arising from mortar contact formulations. International Journal for Numerical Methods in Engineering **122**(15), 3749–3779 (2021). DOI 10.1002/nme.6680. URL `https://doi.org/10.1002/nme.6680`

[175] Wiesner, T.A., Popp, A., Gee, M.W., Wall, W.A.: Algebraic multigrid methods for dual mortar finite element formulations in contact mechanics. International Journal for Numerical Methods in Engineering **114**(4), 399–430 (2018). DOI 10.1002/nme.5748. URL https://doi.org/10.1002/nme.5748

[176] Wirthl, B., Brandstaeter, S., Nitzler, J., Schrefler, B.A., Wall, W.A.: Global sensitivity analysis based on Gaussian-process metamodelling for complex biomechanical problems. International Journal for Numerical Methods in Biomedical Engineering **39**(3), e3675 (2023). DOI 10.1002/cnm.3675

[177] Wriggers, P.: Advanced discretization methods for contact mechanics. In: A. Popp, P. Wriggers (eds.) Contact Modeling for Solids and Particles, pp. 87–123. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-319-90155-8_2. URL https://doi.org/10.1007/978-3-319-90155-8_2

[178] Wriggers, P., Laursen, T.A.: Computational contact mechanics, vol. 2. Springer (2006). DOI 10.1007/978-3-540-32609-0

[179] Wriggers, P., Zavarise, G.: Computational contact mechanics. Computational Mechanics **32**, 141 (2002). URL https://api.semanticscholar.org/CorpusID:86572264

[180] Wu, M., Wang, J.: Prediction of 3d contact force chains using artificial neural networks. Engineering Geology **296**, 106444 (2022). DOI https://doi.org/10.1016/j.enggeo.2021.106444. URL https://www.sciencedirect.com/science/article/pii/S0013795221004555

[181] Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowledge and Information Systems **14**(1), 1–37 (2008). DOI 10.1007/s10115-007-0114-2

[182] Xu, C., Cao, B.T., Yuan, Y., Meschke, G.: Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. Computer Methods in Applied Mechanics and Engineering **405**, 115852 (2023). DOI 10.1016/j.cma.2022.115852

[183] Xu, G., Yue, Q., Liu, X.: Deep learning algorithm for real-time automatic crack detection, segmentation, qualification. Engineering Applications of Artificial Intelligence **126**, 107085 (2023). DOI https://doi.org/10.1016/j.engappai.2023.107085. URL https://www.sciencedirect.com/science/article/pii/S0952197623012691

[184] Xu, Y., Jackson, R.L.: Boundary element method (BEM) applied to the rough surface contact vs. BEM in computational mechanics. Friction **7**(4), 359–371 (2019). DOI 10.1007/s40544-018-0229-3. URL http://link.springer.com/10.1007/s40544-018-0229-3

[185] Yang, C., Zhang, Z., Gu, G.X.: Chapter 2 - machine learning for solid mechanics. In: Y. Zheng, Z. Wu (eds.) Intelligent Nanotechnology, Materials Today, pp. 33–45. Elsevier (2023). DOI https://doi.org/10.1016/B978-0-323-85796-3.00002-0. URL https://www.sciencedirect.com/science/article/pii/B9780323857963000020

[186] Yang, L., Shami, A.: On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing **415**, 295–316 (2020). DOI https://doi.org/10.1016/j.neucom.2020.07.061

[187] Yastrebov, V.A.: Numerical methods in contact mechanics. John Wiley & Sons (2013). DOI 10.1002/9781118647974

[188] Yastrebov, V.A., Anciaux, G., Molinari, J.F.: From infinitesimal to full contact between rough surfaces: Evolution of the contact area. International Journal of Solids and Structures **52**, 83–102 (2015). DOI https://doi.org/10.1016/j.ijsolstr.2014.09.019. URL https://www.sciencedirect.com/science/article/pii/S0020768314003667

[189] Yastrebov, V.A., Anciaux, G., Molinari, J.F.: The role of the roughness spectral breadth in elastic contact of rough surfaces. Journal of the Mechanics and Physics of Solids **107**, 469–493 (2017). DOI 10.1016/j.jmps.2017.07.016. URL https://linkinghub.elsevier.com/retrieve/pii/S0022509617303010

[190] Yastrebov, V.A., Durand, J., Proudhon, H., Cailletaud, G.: Rough surface contact analysis by means of the Finite Element Method and of a new reduced model. Comptes Rendus. Mécanique **339**(7-8), 473–490 (2011). DOI 10.1016/j.crme.2011.05.006. URL https://comptes-rendus.academie-sciences.fr/mecanique/articles/10.1016/j.crme.2011.05.006/

[191] Yucesan, Y.A., Viana, F.A.: A hybrid physics-informed neural network for main bearing fatigue prognosis under grease quality variation. Mechanical Systems and Signal Processing **171**, 108875 (2022). DOI 10.1016/j.ymssp.2022.108875

[192] Zavarise, G., Borri-Brunetto, M., Paggi, M.: On the reliability of microscopical contact models. Wear **257**(3), 229–245 (2004). DOI https://doi.org/10.1016/j.wear.2003.12.010

[193] Zhang, X., Liu, C.A.: Model averaging prediction by k-fold cross-validation. Journal of Econometrics **235**(1), 280–301 (2023). DOI https://doi.org/10.1016/j.jeconom.2022.04.007

[194] Zhao, J., Vollebregt, E.A., Oosterlee, C.W.: Extending the BEM for Elastic Contact Problems Beyond the Half-Space Approach. Mathematical Modelling and Analysis **21**(1), 119–141 (2016). DOI 10.3846/13926292.2016.1138418. URL https://doi.org/10.3846/13926292.2016.1138418

[195] Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Trans. Math. Software **23**(4), 550–560 (1997). DOI 10.1145/279232.279236

[196] Zhu, X., Goldberg, A.B.: Introduction to Semi-Supervised Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer (2009). DOI 10.1007/978-3-031-01548-9. URL https://link.springer.com/book/10.1007/978-3-031-01548-9

[197] Zienkiewicz, O.: Displacement and equilibrium models in the finite element method by b. fraeijs de veubeke, chapter 9, pages 145–197 of stress analysis, edited by oc zienkiewicz and gs holister, published by john wiley & sons, 1965. International Journal for Numerical Methods in Engineering **52**(3), 287–342 (2001). DOI 10.1002/nme.339

[198] Zienkiewicz, O., Taylor, R., Zhu, J.: Chapter 6 - shape functions, derivatives, and integration. In: O. Zienkiewicz, R. Taylor, J. Zhu (eds.) The Finite Element Method: its Basis and Fundamentals (Seventh Edition), seventh edition edn., pp. 151–209. Butterworth-Heinemann, Oxford (2013). DOI https://doi.org/10.1016/B978-1-85617-633-0.00006-X. URL https://www.sciencedirect.com/science/article/pii/B978185617633000006X

[199] Zienkiewicz, O.C., Taylor, R.L.: The finite element method for solid and structural mechanics. Elsevier (2005)