universität wien

DISSERTATION | DOCTORAL THESIS

Titel | Title

Physics Informed Machine Learning in the Field of Micromagnetism

verfasst von | submitted by

Sebastian Alexander Schaffer BSc MSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Doktor der Naturwissenschaften (Dr.rer.nat.)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 796 605 405
Dissertationsgebiet lt. Studienblatt | Field of study as it appears on the student record sheet:	Mathematik
Betreut von | Supervisor:	Univ.-Prof. Dr. Norbert Mauser
Mitbetreut von | Co-Supervisor:	Dipl.-Ing. Dr.techn. Lukas Exl Privatdoz.

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Teile dieser Arbeit sind wurden in angesehenen Wissenschaftlichen Journalen publiziert oder dort eingereicht. Diese sind im Quellenverzeichniss unter [78, 76, 79] zu finden.

Abstract

This thesis investigates advanced neural network methodologies, with a primary focus on Physics-Informed Neural Networks (PINNs), Extreme Learning Machines (ELMs), and hard constrained PINNs and the application to computational micromagnetism. These methods are developed to solve complex physical problems, such as the stray field problem, by embedding physical laws directly into the neural network training process, thereby achieving efficient and accurate solutions without extensive supervised data. The development of highly scalable models could pave the way for large scale micromagnetic simulations and its applications to sustainable technologies such as wind turbines and electric motors.

The initial chapters provide a comprehensive exploration of neural networks, detailing how PINNs and ELMs can be tailored for time-independent partial differential equations (PDEs) such as the Poisson equation. Emphasis is placed on hard constrained PINNs, which enhance optimization by removing constraints, thus achieving higher accuracies comparable to traditional numerical solvers like the Finite Element Method (FEM) and Finite Difference Method (FDM). This approach offers meshfree optimization without the need for complex discretization schemes, allowing for flexible, reduced-order modeling that mitigates the curse of dimensionality.

Subsequent chapters introduce Constructive Solid Geometry (CSG) using R-functions to enforce hard constraints within the PINN framework. This robust modeling technique ensures the exact satisfaction of essential boundary conditions on intricate geometries, arising in the course of the stray field computation. Further, the challenges of magnetostatics are addressed. Particularly the computation of magnetostatic self-energy. By employing the splitting ansatz proposed by Garcia-Cervera and Roma, the research derives energy bounds on finite domains and develops an alternating scheme for the efficient computation of scalar or vector potentials. This scheme facilitates the minimization of magnetostatic self-energy, essential for precise micromagnetic simulations.

The culmination of this work is the application of hard constrained PINNs to the full 3D minimization of Gibbs free energy. By incorporating the Cayley transform, the neural network outputs are constrained to the Lie group of rotation matrices $SO(3)$, ensuring the physical integrity of the magnetization configuration. This innovative approach accurately models continuous magnetization distributions while minimizing total Gibbs free energy, including exchange energy, anisotropy energy, and magnetostatic self-energy. The thesis demonstrates the efficacy of this method through the solution of the NIST μMAG Standard Problem #3 and the simulation of demagnetization processes in hard magnetic materials.

Overall, this thesis makes its contribution to computational micromagnetism by

integrating hard constraint physics informed neural network techniques into energy minimization frameworks. However, the developed methods have further applicability beyond magnetostatics to other areas of physics and engineering.

Deutsche Zusammenfassung

Diese Dissertation untersucht fortgeschrittene neuronale Netzwerkmethoden, insbesondere Physics-Informed Neural Networks (PINNs), Extreme Learning Machines (ELMs) sowie hart beschränkte PINNs und die Anwendung im computergestützten Mikromagnetismums. Diese Methoden wurden entwickelt, um komplexe physikalische Probleme wie das Streufeldproblem zu lösen, indem physikalische Gesetze direkt in den Trainingsprozess des neuronalen Netzwerks eingebettet werden, wodurch effiziente und genaue Lösungen ohne umfangreiche überwachte Daten erreicht werden. Die Entwicklung hochskalierbarer maschineller Lernmodelle könnte groß angelegte mikromagnetische Simulationen und deren Anwendung in nachhaltigen Technologien, wie Windturbinen und Elektromotoren, ermöglichen.

Die ersten Kapitel bieten eine umfassende Einführung in neuronale Netzwerke und erläutern, wie PINNs und ELMs für zeitunabhängige partielle Differentialgleichungen (PDEs) wie die Poisson-Gleichung angepasst werden können. Der Schwerpunkt liegt auf hart beschränkten PINNs, die die Optimierung durch den Wegfall von Randbedingungen verbessern und so Genauigkeiten erzielen, die mit herkömmlichen numerischen Verfahren wie der Finite-Elemente-Methode (FEM) und der Finite-Differenzen-Methode (FDM) vergleichbar sind. Dieser Ansatz bietet eine netzfreie Optimierung ohne die Notwendigkeit komplexer Diskretisierungsschemata und ermöglicht flexible, reduzierte Ordnungsmodelle, die das Fluch der Dimensionalität mildern.

Die nachfolgenden Kapitel führen die Konstruktive Festkörpergeometrie (CSG) mit Hilfe von R-Funktionen ein, um harte Beschränkungen innerhalb des PINN-Rahmens durchzusetzen. Diese robuste Modellierungstechnik gewährleistet die exakte Einhaltung essentieller Randbedingungen bei komplexen Geometrien, die im Rahmen der Streufeldberechnung auftreten.

Ein wesentlicher Teil der Dissertation befasst sich mit den Herausforderungen der Magnetostatik, insbesondere der Berechnung der magnetostatischen Selbstenergie. Durch die Anwendung des von Garcia-Cervera und Roma vorgeschlagenen Splitting-Ansatzes werden Energiebeschränkungen auf endlichen Domänen abgeleitet und ein alternierendes Schema für die effiziente Berechnung des Skalar- oder Vektorpotentials entwickelt. Dieses Schema erleichtert die Minimierung der magnetostatischen Selbstenergie, die für präzise mikromagnetische Simulationen unerlässlich ist.

Der Höhepunkt dieser Arbeit ist die Anwendung von hart beschränkten PINNs auf die vollständige 3D-Minimierung der Gibbs'schen freien Energie. Durch die Einbindung der Cayley-Transformation werden die Ausgaben des neuronalen Netzwerks auf die Lie-Gruppe der Rotationsmatrizen $SO(3)$ beschränkt, wodurch die physikalische Integrität der Magnetisierungskonfiguration sichergestellt wird. Dieser

innovative Ansatz modelliert kontinuierliche Magnetisierungsverteilungen genau und minimiert die gesamte Gibbs'sche freie Energie, einschließlich Austauschenergie, Anisotropieenergie und magnetostatischer Selbstenergie. Die Dissertation zeigt die Wirksamkeit dieser Methode durch die Lösung des NIST μMAG-Standardproblems #3 und die Simulation von Entmagnetisierungsprozessen in hartmagnetischen Materialien.

Insgesamt leistet diese Arbeit einen Beitrag zum computergestützten Mikromagnetismums, indem sie hart beschränkte physikinformierte neuronale Netzwerktechniken in Energie-Minimierungsmethoden integriert. Die entwickelten Methoden haben jedoch eine weitergehende Anwendbarkeit über die Magnetostatik hinaus in anderen Bereichen der Physik und Technik.

Acknowledgments

First and foremost, I want to thank my two PhD supervisors, Prof. Dr.techn. DDipl.Ing. Norbert J. Mauser and Doz. Dr.techn. Dipl.Ing. Lukas Exl – they were a perfect team where I always had someone taking care of all aspects and constant scientific discussions and encouragement. Special thanks to Lukas for guiding me into the world of computational micromagnetism, introducing me to the renowned specialists in the field and the financial support through his (machine learning) projects, which defined the scientific path of this thesis. His advice played a key role in covering many aspects of this thesis. Special thanks to Norbert for his invaluable insights, support and open ear, as well as for creating wonderful working conditions for me and everyone employed at MMM and WPI.

Writing this thesis was a rewarding journey of growth, discipline, and self-reflection. I appreciate the possibility to attend international conferences, workshops, and research visits–among others at the AIM Conference in Italy and my stay at DTU Den mark–I had the privilege of meeting many insightful and generous people and I would like to thank all of them. The collaboration with my project partners in the group of Prof. Schrefl at the Christian Doppler Laboratory "CD Laboratory for Magnet design through physics informed machine learning" at Danube University was also very insightful. Alexander Kovacs and Harald Oezelt deserve to be mentioned for the fun and enriching time we had at several project meetings and conferences. Here I want to thank especially my office colleague at WPI, Jan Frederik Mennemann, for the lively discussions during our time in the office. I warmly thank the WPI adm. director Stefanie Preuss for her continuous support and perfect administration. I wish to express my sincere thanks to my family and friends for their unconditional support and encouragement. I am especially grateful to Claus Trost, a long-time friend who has also become a trusted research colleague, for his support and friendship.

Finally, I acknowledge financial support by the Austrian Science Foundation (FWF) through the projects of Doz. Exl at WPI and Univ. Wien: "Reduced Order Approaches for Micromagnetics" (ROAM) (Grant-DOI: 10.55776/P31140) at WPI, "Data-driven Reduced Order Approaches for Micromagnetism" (Data-ROAM) (Grant-DOI: 10.55776/PAT7615923) at WPI, "Design of Nanocomposite Magnets by Machine Learning" (DeNaMML) (Grant-DOI: 10.55776/P35413) at Univ. Wien research platform MMM "Mathematics–Magnetism–Materials". A good portion of the computational work was done on the Vienna Scientific Cluster (VSC) under the project No. 71140, and the CPU / GPU workstations of the group of Prof. Mauser. Special thanks go to the strong support from the research platform MMM "Mathematics–Magnetism–Materials" of Univ. Wien and it's head, Prof. Mauser,

who also made it possible that I gain teaching experience (in the “Applied Machine Learning” courses of my PhD advisors) which is difficult/rare for PhD students of Univ. Wien funded by grants. I also thank the Wolfgang Pauli Institute (WPI) for the financial support, which also covered other expenses such as the binding of this thesis and provided me office space in the same floor as my supervisors.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1 Introduction

The study of magnetostatics, a branch of electromagnetism focusing on magnetic fields in systems with steady currents, is pivotal for the development and optimization of various technological applications. Understanding and accurately modeling magnetic fields is crucial for the development of magnetic materials and devices. Traditional numerical methods, such as the finite element method (FEM) and finite difference method (FDM), have long been used to solve the partial differential equations (PDEs) governing magnetostatics [58, 81, 29]. However, these methods often require extensive computational resources, fine discretizations, and intricate mesh generation, limiting their efficiency and scalability.

The magnetization can be represented by a low parametric ansatz. Kernel methods can be used to compress magnetization data and learn the magnetization dynamics in some lower dimensional latent space for fast prediction of the response curve to some change in the external field [26, 25, 77, 49]. Multilinear low-rank tensor methods [22] as well as a spectral decomposition to compute the eigenmode development of the effective field operator [17, 64] are examples of such models. Recent advances in machine learning, particularly neural networks, have introduced new paradigms for solving PDEs. Physics Informed Neural Networks (PINNs) [67] have emerged as a powerful tool that incorporates physical laws directly into the learning process, enabling efficient and accurate solutions without the need for large datasets or extensive supervision. PINNs offer a mesh-free approach, significantly reducing the computational complexity and eliminating the need for complex discretization schemes. Especially, *automatic differentiation* (AD) plays a crucial role and is used to compute partial derivatives with respect to the input of very complex models in forward mode, followed by gradient computation with respect to the model parameters in backward mode (reverse-over-forward). Such models can be applied in a fully unsupervised fashion or even in a semi-supervised setting by incorporating additional data from simulations or measurements. In [47], a physics-informed neural network (PINN) [67] was used to model the magnetostatic vector potential and minimize an upper energy bound for the magnetostatic energy. Further, it was shown that such models can be used to approximate an irreversible switching process of a $Nd_2Fe_{14}B$ particle in 2d. Neural network models offer some computational advantages, such as the capability of learning a whole class of solutions with respect to some conditional parameter [48].

In the course of machine learning, one often deals with the problem of the unconstrained minimization of some objective function. However, sometimes, one needs to deal with additional constraints as often in the case of physics-informed machine learning, where one needs to account for some initial or boundary conditions. There

are many different frameworks to deal with such kind of problems, and the traditional PINN ansatz uses inexact constraint optimization. Therefore, a quadratic penalty term is added to the objective or the *Augmented Lagrangian Method* [57] is used, both requiring subsequent optimization problems to be solved until convergence. These frameworks are simple in their implementation but come with the disadvantage of a more complicated optimization problem where errors in the constraint fulfillment propagate over the domain. An exact (nonsmooth) penalty method could also be used in theory, but the objective is usually not differentiable once the constraint is satisfied. This limits its use for gradient based optimization. A fully hard constraint formulation will impose the constraints of the optimization task in the model architecture itself, hence, there is no need for exact constraint fulfillment. Consequently, one can apply a standard optimizer for unconstrained optimization. For instance, micromagnetic energy minimization, requires the satisfaction of the unit norm constraint of the magnetization which needs to be fulfilled across the domain. In [78] a quadratic penalty method was used to account for this constraint. The penalty parameter can be increased until the constraint falls under some tolerance, but exactly satisfying the constraint is not possible since the penalty parameter is not bounded. This motivated us to further research PINN architectures, with utilize the Cayley transform to fulfill the unit norm constraint exactly.

In addition to PINNs, Extreme Learning Machines (ELMs) [41] present another promising alternative for fast and accurate prediction of the solution. ELMs can transform complex learning tasks into linear least squares problems, facilitating faster and more efficient computations. The integration of domain knowledge into ELMs through a hard constrained ansatz further enhances their applicability and accuracy.

In micromagnetism, the most expensive task is the computation of the self energy induced by the magnetization. Traditional methods use some discretization of the domain. To speed up calculations, a stray field can also be computed from only a few known points, by using a neural network to fill in the remaining point, in a physically correct way, as is done in [65]. However, in our research, we use a continuous PINN model. Recent research [87, 91] showed how implicit functions can be used to describe the domain and put constraints on the model architecture. The theory of R-functions can be used as a framework for Constructive Solid Geometry (CSG) to derive a continuous differentiable implicit function describing the domain of interest. Even if the function becomes very complex, AD can still be applied, and it can be used for a hard constraint formulation for all kinds of linear boundary conditions [72, 91].

We use the theory of R-functions to derive a hard constraint formulation for the Poisson problem with essential boundary conditions, which arises in the course of stray field computation. In combination with ELMs, a fast solver, similar to the idea in [80], is established. Further, the continuous differentiable nature of neural networks is used for a Taylor Series expansion to create a very accurate solver for the single layer potential.

This thesis aims to explore and expand the application of advanced neural network methodologies, particularly PINNs and ELMs, in the context of computational magnetostatics. The primary objectives are:

- Development of hard constrained PINNs: Enhancing the PINN framework by integrating hard constraints to remove required constraints from the optimization problem, thereby simplifying the optimization process and improving accuracy.
- Incorporation of CSG using R-Functions: Utilizing R-functions to model complex geometries and ensure exact satisfaction of boundary conditions, making the neural network outputs physically plausible and accurate while maintaining differentiability.
- Efficient computation of magnetostatic fields: Addressing the computational challenges in evaluating magnetostatic energy through advanced techniques such as the splitting ansatz proposed by Garcia-Cervera and Roma, and developing an alternating scheme for joint optimization of magnetization and magnetic fields.
- 3D Gibbs free energy minimization: Applying the hard constrained PINN ansatz to the full 3D minimization of Gibbs free energy, incorporating the Cayley transform to maintain the physical integrity of the magnetization configuration.

The thesis is structured as follows:

- Chapter 2: Brief overview of micromagnetism and current challenges of traditional numerical methods, as well as solutions offered by modern machine learning techniques.
- Chapter 3: An introduction to neural networks, PINNs, ELMs, and the development of hard constrained PINNs.
- Chapter 4: Detailed exploration of Constructive Solid Geometry using R-functions for hard constrained PINNs.
- Chapter 5: Comprehensive study of magnetostatics, energy bounds on the finite domains, the splitting ansatz of Garcia-Cervera and Roma, and an alternating scheme for magnetostatic field computation.
- Chapter 6: Application of the hard constrained PINN ansatz with the Cayley transform for 3D Gibbs free energy minimization, including case studies and practical implementations.
- Chapter 7: Summary of contributions, discussion of the broader impact, and future research directions.

In conclusion, this thesis demonstrates the power and potential of integrating advanced neural network techniques with rigorous physical modeling frameworks to achieve efficient and versatile simulations of complex physical systems.

2 Micromagnetic Background

Micromagnetism is a powerful theoretical framework that provides insights into the magnetic behavior of materials at the microscopic scale, often encompassing dimensions ranging from nanometers to micrometers. It is particularly crucial in the design and optimization of advanced magnetic materials and devices, such as magnetic storage devices [4], electric motors [33], efficient permanent magnets [31], to advanced medical imaging techniques [70, 46] and magnetic sensors [90, 42]. This chapter delves into the foundational aspects of micromagnetism, focusing on the Gibbs Free Energy, the Landau–Lifshitz–Gilbert (LLG) equation, and the computation of hysteresis using traditional numerical methods. We will also discuss the limitations of these methods and how machine learning approaches, such as Physics-Informed Neural Networks (PINNs), could potentially enhance their performance.

2.1 Gibbs free energy in micromagnetism

The Gibbs Free Energy in micromagnetism is a central concept that dictates the stable and metastable states of a magnetic system. The total Gibbs free energy E with units $[E] = \mathrm{J}$, of a ferromagnetic system is composed of several key contributions, each representing a different physical interaction within the material.

- The **exchange energy** is a fundamental concept in micromagnetism, rooted in the quantum mechanical interactions between neighboring electron spins. At the atomic level, the Pauli exclusion principle only allows two atoms at the same location if they have opposite spins. However, two neighboring electrons with the same spin can lower the magnetostatic energy and this gain can be large enough such that the parallel alignment is preferred, as in ferromagnetic materials [21].

  In the continuum model, the exchange energy is conceptualized as a measure of the spatial variation in the magnetization $\boldsymbol{M} = M_s\boldsymbol{m}$ within a material, where M_s is the *saturation magnetization*. It is given in SI-units $[\boldsymbol{M}] = \mathrm{A/m}$ and it satisfies the so-called *unit norm constraint* $\|\boldsymbol{m}\| = 1$. The magnetization is treated as a smooth, continuous field, which assumes that the variations in the direction and magnitude of magnetization occur gradually over space. This approach is effective when the characteristic length scales, such as the distance over which the magnetization changes, are much larger than the atomic lattice spacing. The exchange energy penalizes rapid changes in the direction of magnetization, favoring a more uniform alignment across the material. This concept is particularly important in understanding how magnetic

domains form and how they interact with each other. The energy associated with these variations plays a crucial role in determining the overall magnetic configuration of a material.

However, this continuum description comes with certain limitations. One of the primary assumptions is that the material is homogeneous and continuous, which simplifies the underlying atomic structure into a smooth field. This approximation holds true in many cases, especially in bulk materials or structures where the magnetization changes slowly over large distances. However, in systems where the magnetic variations occur on the scale of the atomic lattice, or in the presence of defects, this model may not capture the full complexity of the interactions.

Additionally, the continuum model is most accurate at low temperatures, where thermal fluctuations are minimal, and the alignment of spins remains relatively stable. At higher temperatures, thermal energy can disrupt this alignment, making the model less accurate. This is particularly significant near the Curie temperature, where the material transitions from a ferromagnetic to a paramagnetic state, and the continuum approximation breaks down.

- The **anisotropy energy** originates from the crystal structure of the material, which creates preferred directions, known as easy axes, along which the magnetization tends to align. This energy is minimized when the magnetization aligns with these axes. In uniaxial anisotropy, for example, there is a single easy axis, while in cubic anisotropy, there are multiple easy axes corresponding to the symmetry of the crystal lattice. The magnitude and direction of the anisotropy energy dictate the stability of the magnetization direction, influencing phenomena such as magnetic hysteresis and coercivity. In materials with strong anisotropy, the magnetization remains stable along the easy axis, contributing to a high coercivity, which is desirable for permanent magnets.

- The **magnetostatic self-energy**, also known as the demagnetizing energy, arises due to the magnetic field $\boldsymbol{H}_d = M_s \boldsymbol{h}_d$ with units $[\boldsymbol{H}_d] = \mathrm{A/m}$ created by the magnetization itself. The field $\boldsymbol{H}_d$ is scale invariant and is a linear function of the magnetization $\boldsymbol{M}$. Thus, the self-energy depends on the shape and the magnetic configuration of the material. It tends to favor configurations that minimize the magnetic field outside the material, such that it reduces the overall magnetostatic energy. The demagnetizing field is particularly important in determining the behavior of thin films, nanostructures, and soft magnetic materials, where it significantly influences the overall magnetic properties. In elongated particles or thin films, the shape anisotropy induced by the demagnetizing energy can dominate over the crystalline anisotropy, leading to an effective easy axis along the long dimension of the particle or film.

- The **Zeeman energy** describes the interaction between the magnetization and an externally applied magnetic field $\boldsymbol{H}_{ext}$. It is minimized when the mag-

netization aligns with the external field, thereby favoring the magnetization's orientation along the direction of the applied field. This term plays a critical role during magnetization reversal processes, such as those observed in hysteresis loops, where the external field drives the magnetization from one stable state to another. The Zeeman energy competes with anisotropy and magnetostatic energies, leading to the complex magnetization dynamics observed in many magnetic systems.

The minimization of this energy is crucial in determining the equilibrium configuration of the magnetization. Often, for numerical simulations, the reduced form $e = \frac{1}{\mu_0 M_s^2} E$ with units $[e] = \mathrm{m}^3$, where $\mu_0 = 4\pi 10^{-7}\,\mathrm{Tm/A}$ is the *vacuum permeability*, is used. A minimum determines an equilibrium magnetization configuration, balancing the competing influences of exchange, anisotropy, magnetostatic, and Zeeman energies. Provided with an uniaxial anisotropy, this energy functional is a quadratic functional. However, in other functional spaces, such as $\boldsymbol{M} \in L^2(\Omega)^3$, accounting for the unit norm constraint, it is posing a challenging non-linear minimization problem [30]. During numerical computations, this requires re-normalization and can put a limit on the step size of first order methods like gradient descent [23]. Minimization of the energy functional lead to stable or metastable local optimum which might not correspond to the global minimum. These states are particularly important in the study of magnetic hysteresis, where the system's history affects its current state.

Traditional numerical methods, such as finite element methods (FEM) [1, 81] and finite difference methods (FDM) [58], are employed to minimize the Gibbs Free Energy. These methods discretize the magnetic material into small elements or grids and numerically solve for the magnetization configuration that minimizes the total energy. However, these methods can be computationally expensive, particularly for complex geometries and 3D problems, and may require significant computational resources to achieve convergence.

2.2 Landau–Lifshitz–Gilbert (LLG) equation

The LLG equation is given by

$$\frac{\mathrm{d}\boldsymbol{M}}{\mathrm{d}t} = -\boldsymbol{M} \times \boldsymbol{H}_{eff} - \alpha \boldsymbol{M} \times (\boldsymbol{M} \times \boldsymbol{H}_{eff}), \tag{2.1}$$

where α is the Gilbert damping parameter and t is time. It is a fundamental equation in micromagnetism that describes the time evolution of the magnetization vector, $\boldsymbol{M}$, under the influence of various effective fields. It incorporates both precessional motion and damping, leading the magnetization toward a stable equilibrium. The LLG equation is essential for simulating dynamic magnetic processes, such as magnetic switching, domain wall motion, and spin wave propagation, but can also be applied for energy minimization by fully relaxing the system. The equation takes into account the effective magnetic field, $\boldsymbol{H}_{eff} = M_s \boldsymbol{h}_{eff}$, which is derived from the

Gibbs Free Energy by taking the Fréchet derivative:

$$\boldsymbol{H}_{eff} = -\frac{\partial E}{\partial \boldsymbol{M}}$$

The effective field encompasses contributions from exchange, anisotropy, magnetostatic, and Zeeman energies, making it a comprehensive descriptor of the forces acting on the magnetization. The Gibbs free energy can be minimized by fully relaxing the LLG equation. In a relaxed state, the damping term is vanishing which results in Brown's equilibrium equation

$$\boldsymbol{H}_{eff} \times \boldsymbol{M} = 0. \tag{2.2}$$

FEM and FDM, iteratively solve the LLG equation at each time step. This approach is highly effective for studying the time-dependent behavior of magnetic materials, but also computationally intensive, particularly in large or complex systems. Specialized extrapolation schemes can be used to reduce the computational cost [27].

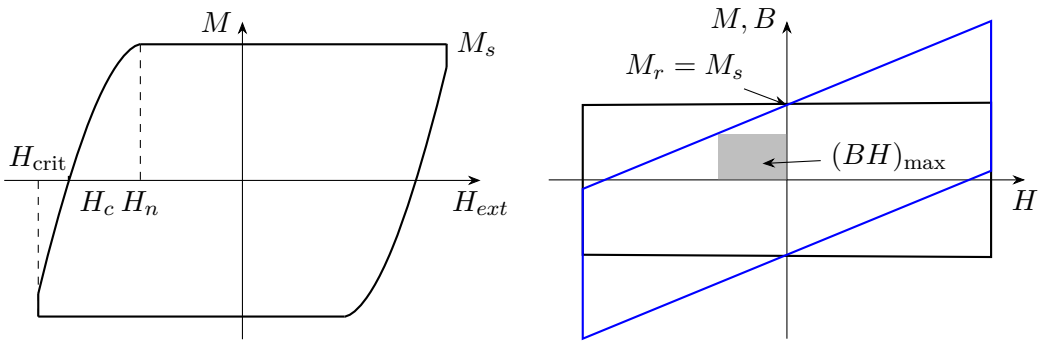
2.3 Hysteresis Curve

The hysteresis curve is a critical characteristic of ferromagnetic materials, depicting the relationship between the applied magnetic field and the magnetization. To compute the hysteresis curve using numerical methods, one minimizes the Gibbs free energy functional under varying external magnetic fields, slowly cycling the field from positive to negative saturation and back while tracking the resulting magnetization. Plotting $M(H_{ext})$ results in a hysteresis loop, where M is obtained by the mean of the projection onto the axis of the external field.

Key properties of the hysteresis curve include:

- **Coercive field (H_c)**: This is the external magnetic field strength at which the magnetization of the material is reduced to zero after being magnetized to saturation. The coercive field is a measure of the material's resistance to demagnetization. A high coercive field implies that the material is hard to demagnetize, characteristic of hard magnetic materials like permanent magnets. Conversely, a low coercive field indicates that the material is easy to demagnetize, typical of soft magnetic materials.

- **Nucleation field (H_n)**: At this field strength, the first reverse magnetic domain starts to form. This field is associated with the initial instability in the magnetization direction, leading to the nucleation of a reverse domain. It is related to the material's microstructure and is often influenced by imperfections, grain boundaries, and thermal fluctuations. It signifies the point at which a small perturbation in the material can cause a significant change in its magnetic state, initiating the reversal process. For single domain particles, with a rectangular hysteresis curve, the nucleation field is equal to the coercive field and the critical field.



(a) Hysteresis loop of a magnet with the key properties: coercive field H_c; nucleation field H_n; critical field H_{crit} and saturation magnetization M_s.

(b) Hysteresis and $B(H)$ curve (in blue) of an ideal permanent magnet. The maximum energy product $(BH)_{\max}$ is shown as the shaded area.

Figure 2.1: Schematic hysteresis loop.

- **Critical field (H_{crit})**: This value usually corresponds to the external field strength at which most of the grains reverse their magnetization.
- **Remanent magnetization (M_r)**: This is the magnetization remaining in the material when the external magnetic field is removed. Remanent magnetization indicates the material's ability to retain a magnetized state. For an ideal permanent magnet, this equals the saturation magnetization.
- **Saturation magnetization (M_s)**: The maximum magnetization that the material can achieve in the presence of a strong external magnetic field, beyond which further increases in the field do not result in additional magnetization.

Figure 2.1a shows a schematic hysteresis loop with those key properties. The curve in the second quadrant is usually referred as demagnetization curve.

Similar to the $M(H_{ext})$ loop, the $B(H)$ curve illustrates the relationship between the magnetic flux density $\boldsymbol{B}$ and the magnetic field $\boldsymbol{H} = \boldsymbol{H}_{ext} + \boldsymbol{H}_d$ of a magnet. In general, one has the relation

$$\boldsymbol{B} = \mu_0(\boldsymbol{H} + \boldsymbol{M}). \tag{2.3}$$

The curve is drawn by computing the projection onto the applied field and taking the mean of those projected fields.

The *maximum energy product* $(BH)_{\max}$ is a key performance metric for permanent magnets. It represents the maximum value of the product of B and H within the second quadrant of the hysteresis loop, where the material can provide the highest magnetic energy per unit volume. Mathematically, the maximum energy product is given by the point on the $B(H)$ curve where the area BH is largest. This value is a crucial indicator of the efficiency and strength of a permanent magnet. In practice, achieving a high maximum energy product requires optimizing both

the material's intrinsic properties, such as remanent magnetization and coercivity, and its microstructural characteristics, which can influence the shape of the hysteresis loop. Modern permanent magnets, such as *NdFeB* magnets, are designed to maximize $(BH)_{\max}$. Figure 2.1b schematically shows both hysteresis loops for an ideal permanent magnet, where the $M(H_{ext})$ loop is characterized by its rectangular shape. The squareness of the $M(H_{ext})$ curve is an important measure of the quality of a magnetic material, indicating that the material retains most of its magnetization until the critical field is reached, at which point it rapidly demagnetizes. In real magnetic materials, the presence of microstructures such as grain boundaries, defects, and domain walls leads to deviations from the ideal square hysteresis loop. These imperfections create local variations in the Gibbs Free Energy landscape, resulting in an early switching of single grains and a more rounded hysteresis curve. The shape of the hysteresis curve varies significantly between soft and hard magnetic materials:

- **Soft Magnetic Materials**: These materials are characterized by low coercivity and a narrow hysteresis loop, indicating that they are easy to magnetize and demagnetize. Soft magnetic materials are typically used in applications requiring rapid switching of magnetization, such as in transformer cores and electromagnetic actuators.
- **Hard Magnetic Materials**: Hard magnetic materials, in contrast, have high coercivity and a broad hysteresis loop, making them resistant to demagnetization. These properties make them suitable for use in permanent magnets, where maintaining a stable magnetic field is crucial. Examples include neodymium-iron-boron (*NdFeB*) and samarium-cobalt (SmCo) magnets, which are widely used in motors, generators, and magnetic storage devices.

Modern permanent magnets, such as *NdFeB* magnets, are engineered to exhibit high coercivity and a large maximum energy product. Achieving these properties requires precise control over the material's composition, microstructure, and processing techniques. Additionally, techniques like grain refinement and sintering are used to optimize the microstructure, improving the material's remanence and overall magnetic performance.

2.4 Challenges and Potential Improvements Using Machine Learning

Machine learning (ML) may be revolutionizing the field of micromagnetism by offering innovative solutions to some of the longstanding challenges associated with traditional numerical methods. These traditional methods, such as finite element methods (FEM) and finite difference methods (FDM), are commonly used to minimize Gibbs free energy and integrate the Landau–Lifshitz–Gilbert (LLG) equation. While effective, these approaches face several significant issues.

One of the primary challenges is the high computational cost. The accurate simulation of magnetization dynamics requires a fine spatial discretization and small time steps. This is particularly problematic in complex systems with intricate geometries or in simulations that involve many grains or extended microstructures.

To address these challenges, data-driven models have gained traction as an alternative or complement to conventional methods. Kernel methods or autoencoders, for instance, offer a powerful way to predict magnetization dynamics. Novel approaches use machine learning techniques to learn the magnetic evolution in a reduced space, allowing fast, low-parametric predictor models [26, 49, 77]. These models can approximate the magnetization behavior by capturing underlying patterns and relationships in the data without the need to solve the governing partial differential equations explicitly. This approach can significantly reduce computational costs and enable faster simulations, especially in scenarios where quick predictions are essential.

Graph Neural Networks (GNNs) are another promising tool in the micromagnetic community. GNNs could be used to effectively model the complex interactions within grain structures, which are crucial in determining the macroscopic magnetic properties of materials. By representing the microstructure as a graph, where nodes correspond to grains and edges to their interactions, GNNs could learn to predict important quantities such as the demagnetization field or the effective field within a material. This capability would be particularly useful in the study of polycrystalline materials or systems with intricate microstructures, where traditional methods may struggle due to the complexity of the interactions.

Physics-Informed Neural Networks (PINNs) offer yet another innovative approach. PINNs integrate physical laws directly into the neural network's training process, allowing for the solution of differential equations governing micromagnetic systems, such as those arising in the minimization of Gibbs free energy [78, 76]. They can be used to learn the solution in an unsupervised or semi-supervised setting, while opening the possibilities to massive parallel computing on clusters. Further, they can incorporate additional conditional parameters to learn a whole family of solutions in a reduced order model [48].

In conclusion, while traditional numerical methods remain essential tools in micromagnetism at this time, the integration of machine learning approaches offers significant improvements in terms of computational efficiency and the ability to handle complex, high-dimensional problems. Data-driven models such as kernel methods and GNNs, along with PINNs, represent powerful opportunities that can address the limitations of conventional techniques, opening up new possibilities for the simulation, understanding, and inverse design of magnetic materials.

3 Neural Network Primer

Parts of this chapter were previously published or submitted for publication as [76, 79]. Permission for use has been granted by the co-authors.

Machine learning and neural networks have recently transformed the field of physics by offering efficient methods to solve complex and inverse problems that are difficult to solve with traditional numerical techniques. Neural networks excel at approximating nonlinear functions, which is crucial for simulating physical systems governed by differential equations. One significant advancement are Physics Informed Neural Networks (PINNs), which integrate physical laws directly into the training process, enabling the solution of partial differential equations with fewer data requirements. This chapter gives a short overview of neural networks and introduces PINNs as an alternative approach for solving such problems.

3.1 Neural Networks

A feedforward neural network is a type of artificial neural network that consists of layers of interconnected nodes, where data flows through the network from the input layer to the output layer without loops. Each node represents a unit that performs a simple computation on the data, and the connections between nodes represent the weights that adjust the influence of the data on the computation. In a feedforward neural network, the output of each node is determined by the weighted sum of its inputs, passed through an activation function that determines the output of the node. Figure 3.1 shows the architecture of a feedforward neural network. It can be described by a series of affine transformations and nonlinear activation functions, such as a sigmoid or exponential linear unit (elu), in order to generate a final output. This can be expressed as a mapping from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$, e.g., $\mathcal{N}(\,.\,;\boldsymbol{\omega}) : \mathcal{X} \to \mathcal{Y}$. In more detail, a network with T layers, which maps the input $\boldsymbol{x}$ to the output $\boldsymbol{y}$, can be written in a recursive way as

$$\begin{aligned} \boldsymbol{z}^{(0)} &= \boldsymbol{x} \\ \boldsymbol{z}^{(t)} &= \sigma^{(t)}(\boldsymbol{W}^{(t)}\boldsymbol{z}^{(t-1)} + \boldsymbol{b}^{(t)}) \quad \text{for } t = 1, \dots, T, \\ \boldsymbol{z}^{(T)} &= \boldsymbol{y}, \end{aligned} \tag{3.1}$$

where $\boldsymbol{W}^{(t)}$ and $\boldsymbol{b}^{(t)}$ are the weight matrix and bias vector of the t^{th} layer and $\sigma^{(t)}$ is the respective activation function. The output of the t^{th} hidden layer is denoted with $\boldsymbol{z}^{(t)}$. The trainable parameters are summarized with $\boldsymbol{\omega} = \{(\boldsymbol{W}^{(t)}, \boldsymbol{b}^{(t)}) \,|\, t =$

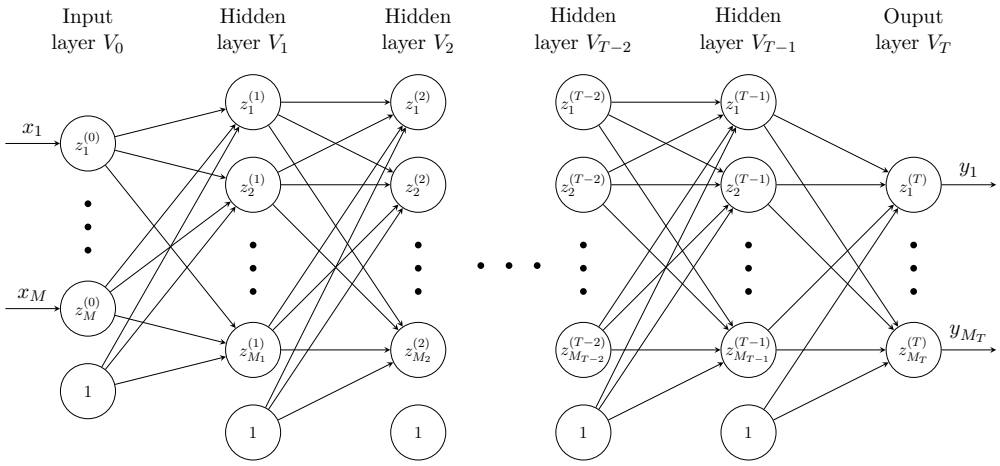


Figure 3.1: Neural network architecture with input dimension M and output dimension M_T.

$1, \ldots, T\}$. Typical activation functions are the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$, the Hyperbolic Tangent function $\tanh(x) = 2/(1 + \exp(-2x)) - 1$ or the Exponential Linear Unit (ELU)

$$\text{elu}(x, \alpha) = \begin{cases} x & x \geq 0 \\ \alpha(\exp(x) - 1) & x < 0, \end{cases}$$

where α is an adjustable parameter. The final activation function is often problem specific. For instance, for regression tasks it might just be the identity mapping.

With traditional supervised learning, the parameters of the network are adjusted during training in order to minimize an objective function. In a regression framework, this is often the mean squared error (MSE). Given a training set with n samples $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \,|\, i = 1, \ldots, N\} \subseteq \mathcal{X} \times \mathcal{Y}$, the neural network parameters $\boldsymbol{\omega}$ should minimize

$$\text{MSE}(\boldsymbol{\omega}) = \frac{1}{|\mathcal{S}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{S}} \|\mathcal{N}(\boldsymbol{x}; \boldsymbol{\omega}) - \boldsymbol{y}\|^2. \tag{3.2}$$

Typically, this objective function is minimized with first order methods. Since the training set can be very large, it is common to use variants of Stochastic Gradient Descent (SGD). This requires the computation of the gradient of the objective, which is usually computed with Automatic Differentiation (AD).

3.2 Physics-Informed Neural Networks

Physics-informed machine learning and in particular a PINN is a novel approach to numerically solve a number of physical problems. Contrary to the supervised learning task (3.2), in a physics-informed setting there is only partial or no label

information available and the objective is designed to minimize the residual of some governing algebraic or differential equation.

Due to the flexibility of neural networks, it can handle forward as well as inverse problems and integrate additional experimental or simulation data. PINNs have diverse applications in scientific and engineering domains such as fluid dynamics [13, 68], material science [102], and geophysics [45].

For instance, consider a Poisson equation with Dirichlet boundary conditions on the domain Ω with boundary $\partial\Omega$

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega. \end{aligned} \tag{3.3}$$

We can minimize the physics-informed objective

$$\mathcal{L}_{\text{PINN}}(\boldsymbol{\omega}) = \underbrace{\int_\Omega |\Delta u_{\boldsymbol{\omega}}(\boldsymbol{x}) + f(\boldsymbol{x})|^2 \mathrm{d}\boldsymbol{x}}_{\mathcal{L}_{\text{PDE}}(\boldsymbol{\omega})} + \lambda \underbrace{\int_{\partial\Omega} |u_{\boldsymbol{\omega}}(\boldsymbol{x}) - g(\boldsymbol{x})|^2 \mathrm{d}s(\boldsymbol{x})}_{\mathcal{L}_b(\boldsymbol{\omega})}. \tag{3.4}$$

Here $u_{\boldsymbol{\omega}} = u(\,\cdot\,;\boldsymbol{\omega})$ is a PINN model which is parameterized by $\boldsymbol{\omega}$ and λ is a penalty parameter to account for boundary conditions.

Partial derivatives can be computed by AD either in forward or reverse mode. If the input is low-dimensional, forward-mode AD can be used efficiently. Hence, the gradient $\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\text{PINN}}$ can be computed with forward AD followed by reverse mode, i.e., *reverse-over-forward* (see Appendix A).

It is not obvious how to choose the penalty parameter λ in (3.4). One could for instance apply the squared penalty framework and solve successive optimization problems with increasing penalty parameter. However, the gain in accuracy would come with increased computational cost and progressively enlarging the penalty parameter amplifies the condition number of the Hessian, making the problem ill-conditioned. Instead of the penalty formulation (3.4) one could as well use an augmented Lagrangian formulation to minimize the PDE residual with respect to the boundary conditions [57].

A priori and a posteriori error estimates of residual based minimization of linear PDEs, using PINNs with soft constraints, have been derived by Shin et al. [88]. This shows that a minimizer of (3.4) is actually a physically meaningful approximation to the true solution of the PDE. Extensive research was done on the balancing of the different loss terms in soft constraint PINNs in order to find a good trade-off between computational cost and performance. In [93] it is shown that gradients of different loss terms can be very unbalanced, leading to poor convergence. Therefore, a balancing algorithm is proposed in [93] which is based on the neural tangent kernel (NTK) perspective. Another approach is a self adapting loss, which was proposed in [100] and is able to outperform classical PINNs for various problems. An important insight of NTK theory is the existence of spectral bias of PINNs. The eigenvalues of the NTK rapidly decay, which leads to extremely slow convergence of high frequency components of the target function [95]. *Fourier Feature Networks* are more capable

of learning such high frequency solutions [92, 94]. *Quadratic Residual Networks* [12] might be another important network architecture to capture higher non-linearities and achieve better performance.

Another example of PINNs would be the minimization of some energy functional with possible additional constraints in a penalty term, i.e.,

$$E(\boldsymbol{\omega}) = \int_{\Omega} \mathcal{E}(u_{\boldsymbol{\omega}}(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} + \text{Penalty}, \tag{3.5}$$

where $\mathcal{E}$ is the energy density.

To compute the respective integrals of the objective function accurately, one could for instance choose some quadrature rule. For arbitrary geometries it is usually easier to use Monte Carlo integration and select a set of collocation points $\mathcal{S} = \{\boldsymbol{x}_i \,|\, i = 1, \dots, N\} \subseteq \Omega$, drawn from some distribution, to approximate the integral. A comprehensive study of different sampling strategies, as well as residual based sampling algorithms, is given in [98] for various problem settings. It can be seen that quasi-random low discrepancy sequence leads to a lower relative error and importance sampling [60] further improves the performance and convergence of PINNs. However, this study does not take into account hard constraint PINNs (see section 3.2.2), but it is expected that these results also hold. For an energy functional as (3.5), where the energy density is non-zero, and the sampling distribution is non-uniform, importance sampling can be applied.

The main challenge is the minimization of the objective function. This is usually done with some variant of stochastic gradient descent, as well as second-order optimization methods like *L-BFGS* or *Trust-Region*. Further, hyperparameter optimization is a very challenging task for PINNs. It is not clear how to choose the perfect architecture to match the task at hand, and there is very little research regarding this issue [86]. Overparametrization of the model is usually required to gain accuracy, resulting in a more complicated optimization problem which becomes more ill-conditioned with a growing number of parameters. On the other hand, overfitting does not seem to be a particular problem for PINNs, since one can always increase the number of samples. But empirical tests suggest, that even for a small sample size, the model is still not very likely to overfit.

Example 3.1. *Given the Poisson equation on* $\Omega = (0,1)^2$

$$\begin{aligned} -\Delta u &= \sin(2\pi x_1)\sin(2\pi x_2) && \text{in } \Omega \\ u &= 0 && \text{on } \partial\Omega, \end{aligned} \tag{3.6}$$

and use a PINN $u_{\boldsymbol{\omega}}$ *with the architecture given in Table 3.1. The analytic solution of this problem is*

$$u(x_1, x_2) = \frac{\sin(2\pi x_1) sin(2\pi x_2)}{8\pi^2}. \tag{3.7}$$

Layer	Activation	Input shape	Output shape
Dense	`tanh`	2	16
Dense	`tanh`	16	16
Dense	`identity`	16	1

Table 3.1: PINN architecture for the ansatz of equation (3.6)

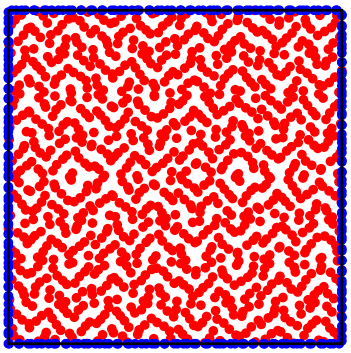


Figure 3.2: Collocation points for problem (3.6).

The objective function is

$$\mathcal{L}_{PINN}(\boldsymbol{\omega}) = \underbrace{\frac{1}{N_\Omega}\sum_{1=1}^{N_\Omega} |\Delta u_{\boldsymbol{\omega}}(\boldsymbol{x}) + \sin(2\pi x_1)\sin(2\pi x_2)|^2}_{\mathcal{L}_{PDE}(\boldsymbol{\omega})} + \underbrace{\frac{4}{N_{\partial\Omega}}\sum_{1=1}^{N_{\partial\Omega}} |u_{\boldsymbol{\omega}}(\boldsymbol{x})|^2}_{\mathcal{L}_b(\boldsymbol{\omega})}, \quad (3.8)$$

where $N_\Omega = 1024$ *and* $N_{\partial\Omega} = 256$ *are the number of collocation points within the domain and on the boundary, respectively. Figure 3.2 shows those collocation points. Those are drawn from Sobol sequence [89]. The objective was minimized with a Trust Region optimizer, where the stopping criteria,* $\|\nabla_{\boldsymbol{\omega}}\mathcal{L}_{PINN}\|_2 < \epsilon = 1e-4$*, was used. The solution as well as the error is shown in Figure 3.3 and the training curve is shown in Figure 3.4. It can be seen that the main error comes from the fulfillment of the boundary conditions.*

3.2.1 Conditional PINNs

The curse of dimensionality presents a significant challenge in various computational tasks, including the modeling of complex physical systems. As the dimensionality of the problem space increases, traditional numerical methods encounter escalating challenges due to the exponential growth in computational and data storage requirements. This phenomenon significantly impacts the accuracy scalability of these methods, often rendering them impractical for high-dimensional problems.

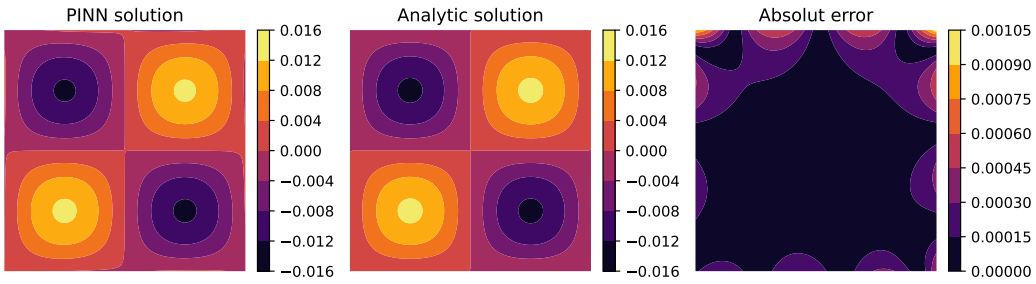


Figure 3.3: Solution of a PINN, with architecture given in Table 3.1, for the 2d Poisson problem (3.6).

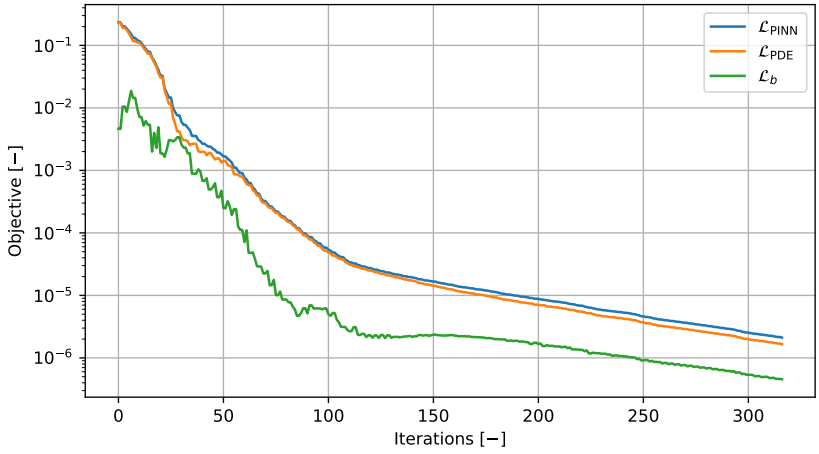


Figure 3.4: Training curve of a PINN, with architecture given in Table 3.1, for the 2d Poisson problem (3.6) with a *Trust Region* optimizer. The stopping criteria, $\|\nabla_{\omega}\mathcal{L}_{\text{PINN}}\|_2 < \epsilon = 1e-4$, is reached after 317 steps.

Layer	Activation	Input shape	Output shape
Dense	`gelu`	7	20
Dense	`gelu`	20	20
Dense	`gelu`	20	20
Dense	`identity`	20	1

Table 3.2: Conditional PINN architecture for the ansatz of equation (3.9) with 6 conditional parameters. The model has only 1021 trainable parameters.

In contrast, physics-informed neural networks (PINNs) offer a promising alternative for addressing the curse of dimensionality in the context of physical modeling. Since the linear ansatz function of traditional methods is replaced by a more powerful non-linear model, there is no exponential growth with respect to the model parameters. The model simply learns a set of parameters which represents the full manifold of solutions and hence provides a flexible and data-driven approach to model complex physical systems across high-dimensional spaces. This is termed as *Conditional PINN* [48].

Example 3.2. *Consider the following differential equation with mixed boundary conditions:*

$$\begin{aligned} &\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = 1 - 2x^2 && \text{for } x \in (0,1) \\ &\alpha_l u(0) + \beta_l \frac{\mathrm{d}u}{\mathrm{d}x}(0) = \gamma_l, && \alpha_l \in [0.8, 1], \quad \beta_l \in [-1, 0], \quad \gamma_l \in [0, 1] \\ &\alpha_h u(1) + \beta_h \frac{\mathrm{d}u}{\mathrm{d}x}(1) = \gamma_h, && \alpha_h \in [0.8, 1], \quad \beta_h \in [0, 1], \quad \gamma_h \in [0, 1]. \end{aligned} \tag{3.9}$$

The explicit solution is given by

$$u(x) = a + bx + \frac{x^2}{2} - \frac{x^4}{6}, \tag{3.10}$$

with

$$\begin{aligned} a &= \frac{\gamma_l(\alpha_h + \beta_h) - \beta_l(\gamma_h - (\alpha_h + \beta_h)/3)}{\alpha_l\alpha_h + \alpha_l\beta_h - \beta_l\alpha_h} \\ b &= \frac{\alpha_l(\gamma_h - (\alpha_h + \beta_h)/3) - \gamma_l\alpha_h}{\alpha_l\alpha_h + \alpha_l\beta_h - \beta_l\alpha_h}. \end{aligned}$$

To solve this problem with a traditional method, e.g., finite differences, one would choose a discretization for the domain and each parameter dimension. Therefore, the number of parameters would be in $\mathcal{O}(h^7)$, for a discretization size h. Instead, by using a PINN ansatz u_ω as in Table 3.2, there are only 1021 *trainable parameters. To train this conditional PINN model, the parameter space is sampled, and the strong residual is computed for all collocation points. The final objective function is the*

mean over the collocation points, as well as over the set of sampled parameters $\mathcal{C}$, i.e.,

$$\begin{aligned}\mathcal{L}_{PINN}(\boldsymbol{\omega}) = \frac{1}{|\mathcal{C}|}\sum_{c_i\in\mathcal{C}}\frac{1}{N}\sum_{i=1}^{N}&\left(\frac{\mathrm{d}^2 u_{\boldsymbol{\omega}}}{\mathrm{d}x^2}(x_i;\, c_i) - 1 + 2x_i^2\right)^2 \\ &+ \left(\alpha_l u_{\boldsymbol{\omega}}(0;\, c_i) + \beta_l \frac{\mathrm{d}u_{\boldsymbol{\omega}}}{\mathrm{d}x}(0;\, c_i) - \gamma_l\right)^2 \\ &+ \left(\alpha_h u_{\boldsymbol{\omega}}(1;\, c_i) + \beta_h \frac{\mathrm{d}u_{\boldsymbol{\omega}}}{\mathrm{d}x}(1;\, c_i) - \gamma_h\right)^2,\end{aligned} \tag{3.11}$$

where $c_i = \{\alpha_l, \beta_l, \gamma_l, \alpha_h, \beta_h, \gamma_h\}$ Figure 3.5 shows the solution for multiple different parameters for a single PINN model, as in Table 3.2. In particular, the PINN was trained on 200 000 patches with a batch size of 100 and each parameter was drawn from a uniform distribution. With $N = 100$ equidistantly spaced collocation points, the training was performed with a AdamW optimizer [55] and exponential learning rate decay of 0.1. The learning rate was 10^{-3} at the beginning and reached 10^{-5} at the end of training.

Example 3.2 should demonstrate the clear advantage of PINNs over traditional numerical methods for high dimensional problems, as well as parameter dependent PDEs. Even a very complex solution manifold can be compressed by the low parametric representation of a neural network.

3.2.2 Hard constraint formulation

Another line of research, contrary to soft constraints, is the hard constraint formulation. Boundary conditions are exactly imposed on the architecture of the PINN. Hence, the optimization problem becomes unconstrained and is easier to solve in many applications. For problem (3.3) we can define the model as [78]

$$u_{\boldsymbol{\omega}}(\boldsymbol{x}) = \ell(\boldsymbol{x})\hat{u}_{\boldsymbol{\omega}}(\boldsymbol{x}) + \tilde{g}(\boldsymbol{x}), \tag{3.12}$$

where

$$\ell(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \partial\Omega \\ > 0 & \text{if } \boldsymbol{x} \in \Omega \end{cases} \tag{3.13}$$

serves as an indicator function and $\hat{u}_{\boldsymbol{\omega}}$ being a neural network. The function $\tilde{g}$ should interpolate g on the boundary and be well-defined within the domain. Instead of (3.4) we can minimize just $\mathcal{L}_{\mathrm{PDE}}$ since $\mathcal{L}_b$ is zero. One could use a separate neural network to learn $\tilde{g}$ as in [87] or use other means of interpolation, such as transfinite interpolation [74, 19]. The former is particularly useful if g is not analytically known and only pointwise data is available.

There are many choices to construct ℓ, but as noted in [91] it is useful if ℓ is a good approximation to the exact distance function in the vicinity of the boundary (see section 4.1). In section 5.3 we see how this property leads to a practical simplification for the computation of the stray field.

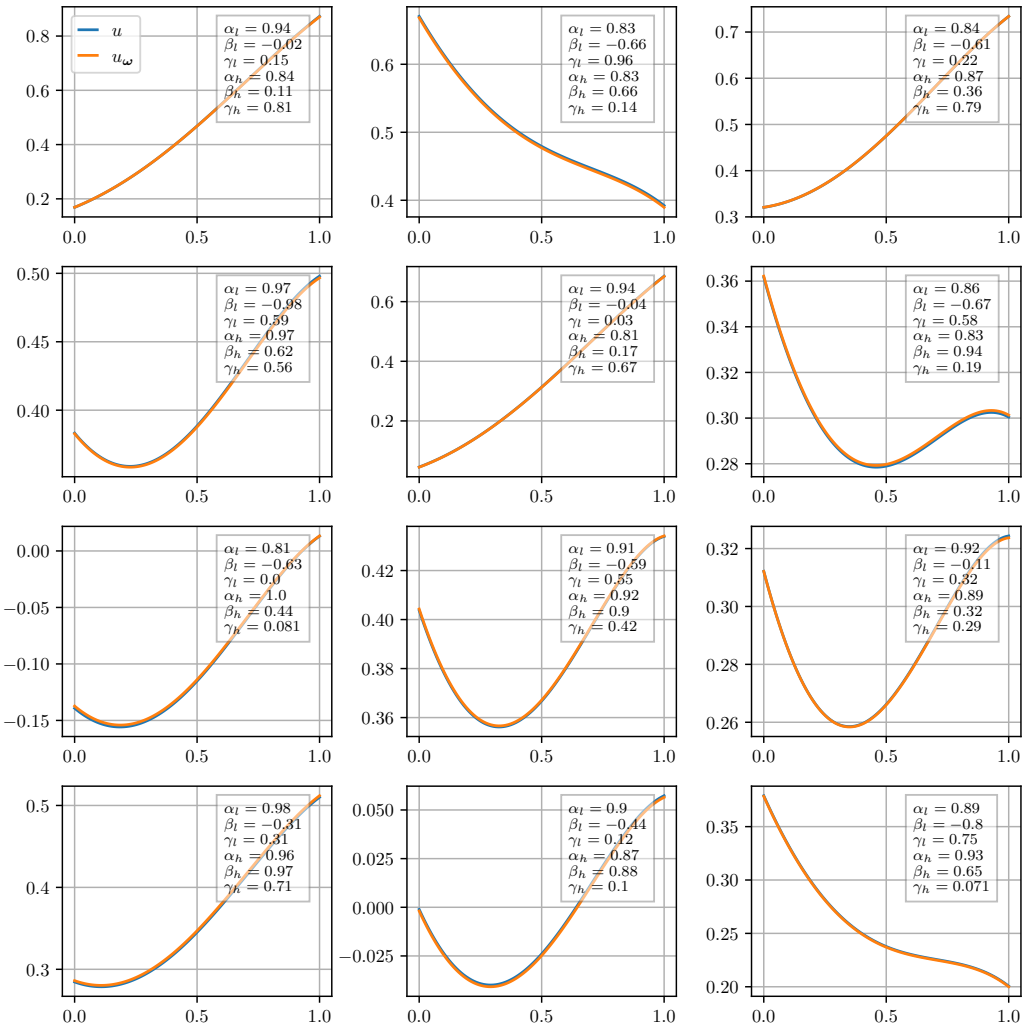


Figure 3.5: True solution and predicted solution of a conditional PINN, as in Table 3.2, of equation (3.9) for different parameters.

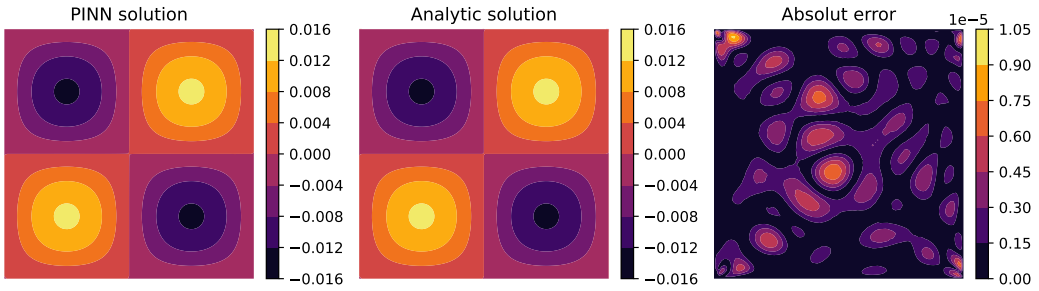


Figure 3.6: Training curve of the hard constraint PINN model (3.14) with the architecture of $\tilde{u}_{\boldsymbol{\omega}}$ given in Table 3.1 for the 2d Poisson problem (3.6).

Example 3.3. *Consider again the Poisson equation from Example 3.1. The same set of collocation points within the domain is used, and the boundary condition is satisfied with an ADF ℓ which is normalized to first order (see Chapter 4.1). The model ansatz*

$$u_{\boldsymbol{\omega}}(x_1, x_2) = \ell(x_1, x_2)\hat{u}_{\boldsymbol{\omega}}(x_1, x_2) \tag{3.14}$$

was used to train a hard constraint PINN with the architecture of $\hat{u}_{\boldsymbol{\omega}}$ given in Table 3.1. A Trust Region optimizer is used to minimize the objective $\mathcal{L}_{PDE}$ from (3.8) with the same stopping criteria ($\|\nabla_{\boldsymbol{\omega}}\mathcal{L}_{PDE}\|_2 < \epsilon = 1e-4$).

Figure 3.6 shows the solution of the hard constraint formulation and Figure 3.7 shows the training curve. The quality of the solution is considerably higher compared to the soft constraint PINN in Example 3.1 (see Figure 3.3). On the other hand, it can be seen that the optimizer needs much longer to reach the required tolerance.

3.3 Physics-Informed Extreme Learning Machines

An Extreme Learning Machine (ELM) is a type of neural network that contains only a single hidden layer. The input data is randomly mapped onto a latent space, meaning that only the output layer requires training. Formally, for an input $\boldsymbol{x} \in \mathbb{R}^d$, an ELM with M hidden nodes and a scalar output can be described as

$$u_{\beta}(\boldsymbol{x}) = \sum_{i=1}^{M} \beta_i \sigma_i(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{z}(\boldsymbol{x}), \tag{3.15}$$

where $\boldsymbol{z}$ represents the latent embedding of the hidden layer. For instance, σ_i can be given by

- an affine map followed by a nonlinear activation σ, i.e. $\sigma_i(\boldsymbol{x}) = \sigma(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)$, where the input weights $(\boldsymbol{w}_i, b_i)$ are drawn from some distribution,
- a radial basis function, centered around a point $\boldsymbol{w}_i \in \Omega$, i.e. $\sigma_i(\boldsymbol{x}) = \exp(-\gamma\|\boldsymbol{x} - \boldsymbol{w}_i\|^2)$ with kernel parameter γ,

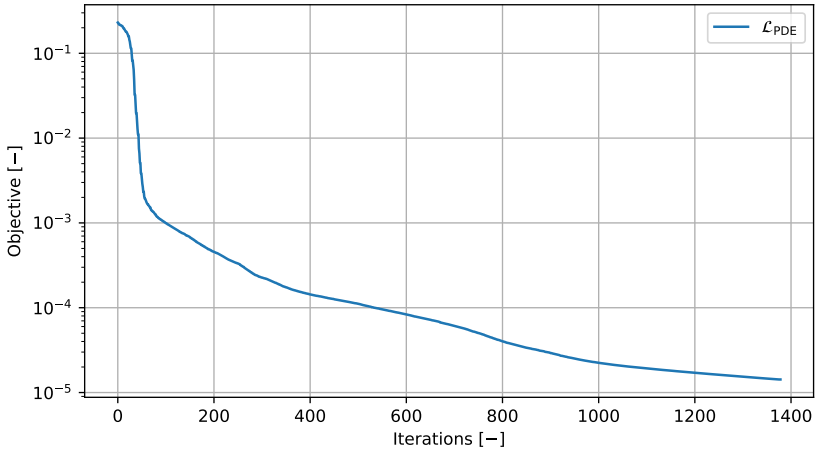


Figure 3.7: Training curve of the hard constraint PINN model (3.14) with the architecture of $\tilde{u}_{\boldsymbol{\omega}}$ given in Table 3.1 for the 2d Poisson problem (3.6) with a *Trust Region* optimizer. The stopping criteria, $\|\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\text{PDE}}\|_2 < \epsilon = 1e{-}4$, is reached after 1379 steps.

- or a basis of a B-spline function, leading to a sparse linear regression problem.

Figure 3.8 shows the architecture of an ELM. Note that also an intercept term could be added to the output layer.

If the MSE (3.2) is to be minimized for some training set $\mathcal{S} = \{(\boldsymbol{x}_i, y_i) \,|\, i = 1, \dots, N\} \subseteq \mathbb{R}^d \times \mathbb{R}$,

$$\text{MSE}(\boldsymbol{\beta}) = \frac{1}{2N} \sum_{i=1}^{N} (\boldsymbol{\beta}^T \boldsymbol{z}(\boldsymbol{x}_i) - y_i)^2, \tag{3.16}$$

the solution is given by the least squares problem

$$H\boldsymbol{\beta} = \boldsymbol{y}, \tag{3.17}$$

with

$$\begin{aligned} H_{i,j} &= \sigma_j(\boldsymbol{x}_i) \\ \boldsymbol{y}_i &= y_i \end{aligned} \qquad \text{for } \; i = 1, \dots, N \text{ and } j = 1, \dots, M.$$

with the solution operator being the pseudoinverse $H^\dagger = (H^T H)^{-1} H^T$. If the size of the system becomes restrictive, one can use an incremental method to train the ELM [40]. Nodes are incrementally added to the network, in essence, using gradient boosting to improve the model. On the one hand this reduces memory requirements but increases the model size since $\boldsymbol{\beta}$ is far from optimal. In [39] it is proposed to recompute the output weights of existing nodes using convex optimization. This would drastically improve convergence. Application of a random Fourier mapping

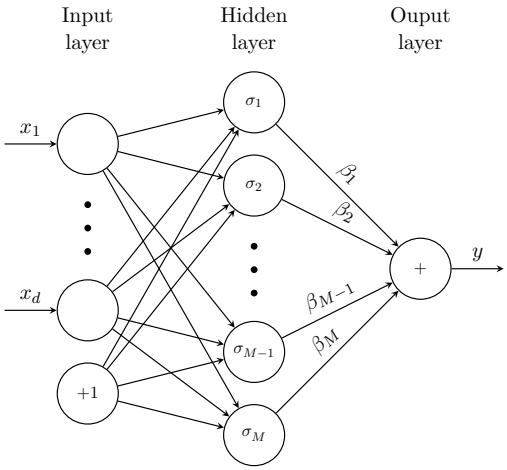


Figure 3.8: An Extreme Learning Machine with d-dimensional input $\boldsymbol{x}$, M hidden nodes and scalar output y where only the output parameter $\boldsymbol{\beta}$ is trainable.

[103] might be beneficial in the presence of high frequency solution components, hence reducing the required hidden layer size.

For an overparameterized model, computing the pseudoinverse can quickly become an ill-conditioned problem. Adding a Ridge regression term $\mu\|\boldsymbol{\beta}\|^2$ to (3.16) can be very helpful in these cases. The simplest way is by modifying the pseudoinverse, i.e., when computing the Singular Value Decomposition (SVD) $H = USV^T$, with singular values $\sigma_1 \geq \cdots \geq \sigma_M$, the regularized solution operator is given by

$$H_\mu^\dagger = VS_\mu^{-1}U^T, \tag{3.18}$$

where S_μ^{-1} is diagonal with the i^{th} diagonal entry given by $S_{\mu,ii}^{-1} = \sigma_i/(\sigma_i^2 + \mu)$.

3.3.1 Quadratic programming

Any linear operator acting on an ELM only acts on the activation functions, transforming the ELM into another ELM with different activations. For instance, the Laplace operator applied to an ELM is

$$\Delta u_{\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{i=1}^{M} \beta_i \Delta\sigma_i(\boldsymbol{x}). \tag{3.19}$$

To solve a Poisson equation (3.3), the energy functional

$$\min_{u\in H_0^1} \int_\Omega (\frac{1}{2}\|\nabla u(\boldsymbol{x})\|^2 - f(\boldsymbol{x})u(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} \tag{3.20}$$

can be minimized. Applying the same penalty term as in (3.4), this is known as the *Deep-Ritz Method* [101]. Using a linear ansatz $u_{\boldsymbol{\beta}} = \sum_{i=1}^M \beta_i\sigma_i(\boldsymbol{x})$ with $\boldsymbol{\beta} \in \mathbb{R}^M$ and

some collocation points $\{\boldsymbol{y}_i \in \partial\Omega;\ \text{for } i = 1, \dots N_{\partial\Omega}\}$, the objective (3.19) can be written in terms of a linear constraint quadratic program:

$$\begin{aligned} \min_{\boldsymbol{\beta}\in\mathbb{R}^M} \mathcal{L}_Q(\boldsymbol{\beta}) &= \frac{1}{2}\boldsymbol{\beta}^T Q\boldsymbol{\beta} - \boldsymbol{b}^T\boldsymbol{\beta} \\ \text{subject to } K\boldsymbol{\beta} &= \boldsymbol{z}, \end{aligned} \tag{3.21}$$

with $K \in \mathbb{R}^{N_{\partial\Omega}\times M}$, $Q \in \mathbb{R}^{M\times M}$, $\boldsymbol{b} \in \mathbb{R}^M$ and

$$\begin{aligned} Q_{ij} &= \int_\Omega \nabla\sigma_i(\boldsymbol{x})^T \nabla\sigma_j(\boldsymbol{x})\mathrm{d}\boldsymbol{x}, \\ \boldsymbol{b}_i &= \int_\Omega f(\boldsymbol{x})\sigma_i(\boldsymbol{x})\mathrm{d}\boldsymbol{x}, \\ K_{ij} &= \sigma_j(\boldsymbol{y}_i), \\ \boldsymbol{z}_i &= g(\boldsymbol{y}_i). \end{aligned}$$

This quadratic program can be computed efficiently and is particularly useful if the domain, as well as the Dirichlet boundary conditions, are static since all quantities but $\boldsymbol{b}$ can be precomputed. Especially if $\boldsymbol{b}$ only changes slightly, the previous solution can be used as initial guess such that the optimizer converges very rapidly.

3.3.2 Hard constraint ELMs

Another possibility is the usage of a hard-constraint ELM ansatz, as in (3.12). The indicator function ℓ only acts on the activation functions and the ansatz becomes

$$u_{\boldsymbol{\beta}}(\boldsymbol{x}) = \ell(\boldsymbol{x})\hat{u}_{\boldsymbol{\beta}}(\boldsymbol{x}) + \tilde{g}(\boldsymbol{x}) = \boldsymbol{\beta}^T\ell(\boldsymbol{x})\boldsymbol{z}(\boldsymbol{x}) + \tilde{g}(\boldsymbol{x}). \tag{3.22}$$

To solve (3.3) we can use this ansatz which always satisfies the homogeneous boundary conditions

$$-\Delta(\ell(\boldsymbol{x})\hat{u}_{\boldsymbol{\beta}}(\boldsymbol{x}) + \tilde{g}(\boldsymbol{x})) = -\boldsymbol{\beta}^T\Delta\left(\ell(\boldsymbol{x})\boldsymbol{z}(\boldsymbol{x})\right) - \Delta\tilde{g}(\boldsymbol{x}) = f(\boldsymbol{x}) \tag{3.23}$$

and find the best parameter $\boldsymbol{\beta}$ by minimizing

$$\mathcal{L}_{\mathrm{ELM}}(\boldsymbol{\beta}) = \int_\Omega |\boldsymbol{\beta}^T\Delta\left(\ell(\boldsymbol{x})\boldsymbol{z}(\boldsymbol{x})\right) + \Delta\tilde{g}(\boldsymbol{x}) + f(\boldsymbol{x})|^2\mathrm{d}\boldsymbol{x}, \tag{3.24}$$

or equivalently, by solving the system

$$H\boldsymbol{\beta} = \boldsymbol{b}, \tag{3.25}$$

with

$$\begin{aligned} H_{ij} &= -\int_\Omega \Delta(\ell(\boldsymbol{x})\sigma_i(\boldsymbol{x}))\Delta(\ell(\boldsymbol{x})\sigma_j(\boldsymbol{x}))\mathrm{d}\boldsymbol{x}, \\ \boldsymbol{b}_i &= \int_\Omega f(\boldsymbol{x})\Delta(\ell(\boldsymbol{x})\sigma_i(\boldsymbol{x})) + \Delta\tilde{g}(\boldsymbol{x})\Delta(\ell(\boldsymbol{x})\sigma_i(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} \end{aligned} \qquad \begin{aligned} &\text{for } i = 1, \dots, M \\ &\text{and } j = 1, \dots, M. \end{aligned}$$

For a dataset of collocation points $\mathcal{S} = \{\boldsymbol{x}_i \,|\, i = 1, \dots, N\} \subseteq \Omega$ we can use empirical risk minimization and use the objective

$$\mathcal{L}_{\text{ELM}}(\boldsymbol{\beta}) = \frac{1}{N} \sum_{i=1}^{N} |\boldsymbol{\beta}^T \Delta \left(\ell(\boldsymbol{x}_i)\boldsymbol{z}(\boldsymbol{x}_i)\right) + \Delta\tilde{g}(\boldsymbol{x}_i) + f(\boldsymbol{x}_i)|^2. \tag{3.26}$$

The minimizer of (3.26) is given by the solution to the least squares problem

$$A\boldsymbol{\beta} = \boldsymbol{b}, \tag{3.27}$$

with

$$\begin{aligned} A_{ij} &= -\Delta\left(\ell(\boldsymbol{x}_i)\sigma_j(\boldsymbol{x}_i)\right), \\ \boldsymbol{b}_i &= f(\boldsymbol{x}_i) + \Delta\tilde{g}(\boldsymbol{x}_i) \end{aligned} \qquad \text{for } \; i = 1, \dots, N \text{ and } j = 1, \dots, M.$$

If this system is not too large, the pseudoinverse $A^\dagger$ defines the solution operator and can be precomputed using SVD. The best parameter $\boldsymbol{\beta}$ for some right-hand side f is then given by a simple matrix multiplication. If the problem is ill conditioned, a Ridge regression term can be added. Further, for good performance and depending on the hidden layer activation, the input might be rescaled to $[-1, 1]^d$.

Example 3.4. *Consider once again the Poisson problem from Example 3.1. The problem was solved using an ELM with $M = 64$ hidden nodes and input weights sampled from a Sobol sequence [89]. Figure 3.9 shows the distribution of the input weights over the input space. The ELM is given by*

$$u_{\boldsymbol{\beta}}(x_1, x_2) = \sum_{i=1}^{M} \beta_i \tanh(w_{1,i}(x_1 - 0.5) + w_{2,i}(x_2 - 0.5) + b_i). \tag{3.28}$$

Figure 3.10 shows the solution of the quadratic program (3.21) with $N_{\partial\Omega} = 64$. Figure 3.11 shows the solution of the hard constraint ELM ansatz (3.22) which was obtained from solving (3.25). All integrals were evaluated with a Newton-Cotes formula and 5^{th} order Gauss–Legendre quadrature.

However, in terms of accuracy, it is usually better to solve (3.27) instead of (3.22). Empirical risk minimization often performs better than using some quadrature rule for PINN training in general. Figure 3.12 shows the same hard constraint model, but trained with a sample set drawn from a Sobol sequence with 256 samples and Ridge regularization with $\mu = 10^{-3}$. It can be seen that the error is significantly lower. Figure 3.13 shows the error of the ELM (trained with (3.27)) in the L_2 norm with respect to the number of hidden nodes M. It can be seen that even for an ELM with more hidden nodes than samples $M > N$, using Ridge regularization, the solution is still becoming more and more accurate. Hence, training is more stable with this approach. Also, it can be seen from Figure 3.13, that the convergence rate of the ELM is about $\mathcal{O}(M^{-2})$. The loss of this convergence rate for $M > 2^7$ is most likely due to ill conditioning, Ridge regularization as well as single precision arithmetic.

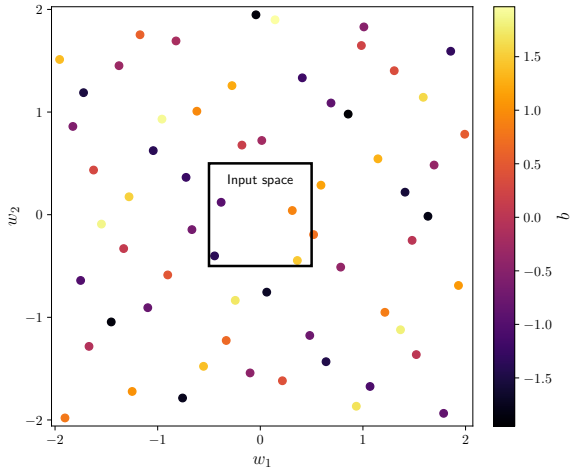


Figure 3.9: Distribution of the input weights of the ELM (3.28) for problem (3.6) over the input space. Note that the input space of the ELM has been centered. The color represents the bias term.

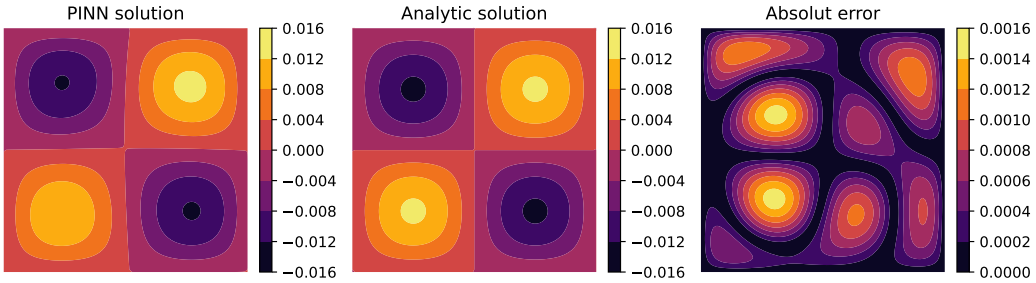


Figure 3.10: ELM solution of (3.6) with the quadratic programming ansatz (3.21).

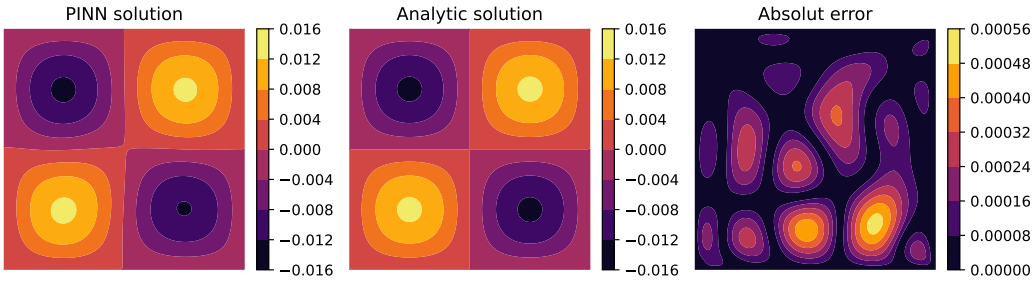


Figure 3.11: ELM solution of (3.6) with the hard constraint ansatz (3.22).

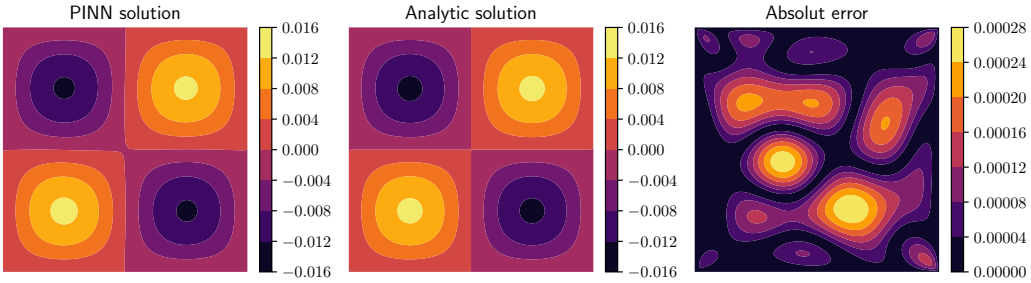


Figure 3.12: ELM solution of (3.6) with the hard constraint ansatz (3.22) with empirical risk minimization.

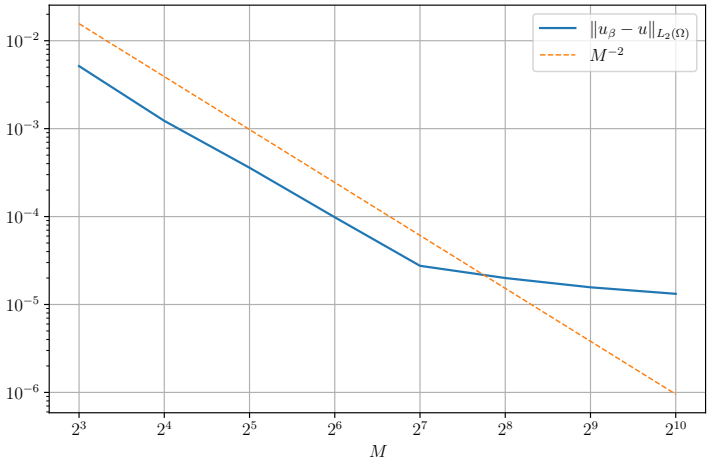


Figure 3.13: ELM error with respect to the number of hidden nodes M.

3.3.2.1 Kernel perspective

The regularized solution operator can also be computed with the dual formulation, i.e.,

$$A^\dagger = (A^T A + \mu I)^{-1} A^T = A^T (A A^T + \mu I)^{-1}, \tag{3.29}$$

by application of the push-through identity. This is equivalent to Kernel Ridge regression (KRR) [59] with the kernel function given by

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \Delta\left(\ell(\boldsymbol{x}_1)\boldsymbol{z}(\boldsymbol{x}_1)\right)^T \Delta\left(\ell(\boldsymbol{x}_2)\boldsymbol{z}(\boldsymbol{x}_2)\right). \tag{3.30}$$

Thus, the kernel matrix is given by $K = AA^T$.

There is a clear connection of ELMs and the low-rank KRR [77, 96]. Consider a KRR with kernel function $a(\boldsymbol{x}_i, \boldsymbol{x}_j) = \ell(\boldsymbol{x}_i)\psi(\boldsymbol{x}_i) \cdot \ell(\boldsymbol{x}_j)\psi(\boldsymbol{x}_j)$, where $\psi : \Omega \to \mathcal{F}_\Omega, \boldsymbol{x} \mapsto \psi(\boldsymbol{x})$ is a feature map which transforms the input to a reproducing kernel Hilbert space (RKHS) and $\cdot$ denotes the inner product in this space. The regularized physics-informed loss is given by

$$\mathcal{L}_{\text{KRR}}(\boldsymbol{\xi}) = \frac{1}{N}\sum_{i=1}^{N} \left|\Delta\left(\ell(\boldsymbol{x}_i)\psi(\boldsymbol{x}_i)\right) \cdot \boldsymbol{\xi} + \Delta\tilde{g}(\boldsymbol{x}_i) + f(\boldsymbol{x}_i)\right|^2 + \mu(\boldsymbol{\xi} \cdot \boldsymbol{\xi}). \tag{3.31}$$

Let $\psi_\ell(\boldsymbol{x}) = \Delta\left(\ell(\boldsymbol{x})\psi(\boldsymbol{x})\right)$ and denote $\Psi[\mathcal{S}]$ as the matrix of size $N \times \dim(\mathcal{F}_\Omega)$ which holds the samples mapped to the RKHS

$$\Psi[\mathcal{S}] := [\psi_\ell(\boldsymbol{x}_1)| \ldots |\psi_\ell(\boldsymbol{x}_N)]^T. \tag{3.32}$$

The dual solution would then be given by

$$\boldsymbol{\xi} = \Psi[\mathcal{S}]^T (\Psi[\mathcal{S}] \cdot \Psi[\mathcal{S}] + \mu I)^{-1} \boldsymbol{b} = \Psi[\mathcal{S}]^T (G[\mathcal{S}, \mathcal{S}] + \mu I)^{-1} \boldsymbol{b}, \tag{3.33}$$

where $G[\mathcal{S}, \mathcal{S}]$ is the corresponding kernel matrix. If the kernel matrix $G[\mathcal{S}, \mathcal{S}]$ is now approximated with a low rank version of rank M, i.e, $G[\mathcal{S}, \mathcal{S}] \approx \widetilde{\Psi}[\mathcal{S}]\widetilde{\Psi}[\mathcal{S}]^T$ with $\widetilde{\Psi}[\mathcal{S}]$ being a low rank matrix of size $N \times M$ of Ψ, the solution operator has the same form as for the ELM ansatz. The problem with this ansatz is the computation of the Laplace operator, since it is not clear what the corresponding kernel to $\Delta\left(\ell(\boldsymbol{x}_i)\psi(\boldsymbol{x}_i)\right)\Delta\left(\ell(\boldsymbol{x}_j)\psi(\boldsymbol{x}_j)\right)$ should be. The ELM solves this by mapping the input directly onto a randomized low-rank RKHS. The feature space mapping $\widetilde{\Psi}[\mathcal{S}]$ is directly given by the ELM's hidden layer output.

In our experiments, selecting for instance a radial basis activation function with $\ell(\boldsymbol{x})\sigma_i(\boldsymbol{x}) = \ell(\boldsymbol{x})\exp(-\gamma\|\boldsymbol{x} - \boldsymbol{w}_i\|^2)$ with $\boldsymbol{w}_i \in \Omega$ being the weight of the i^{th} node of the ELM, achieves similar performance to other activation functions, such as `tanh` or `gelu` with the advantage that the weights can be drawn from the domain and there is no intercept term required. For difficult functions, such as the approximation of a vortex state with vector potential (see Section 5.5.3), this kind of ELM even outperforms ELMs with other activations. The disadvantage lies in the selection

of the kernel parameter γ. Hyperparameter tuning could be performed to select an appropriate value, but it is still unclear if the selected value would perform equally well for different input functions f. It has been empirically shown that kernel learning can achieve similar performance on various regression and classification tasks [66]. Considering the activation $\sigma_i(\boldsymbol{x}) = \exp(-(\boldsymbol{x} - \boldsymbol{w}_i)^T \Sigma (\boldsymbol{x} - \boldsymbol{w}_i))$, one could learn the SPD kernel parameter Σ for even better performance with a smaller model. This if left for future research.

3.3.2.2 Neural operator perspective

The physics-informed ELM can also be seen as a kind of operator network, like Deep Operator Networks (DeepONets) [56]. A DeepONet G has the basic structure

$$G(\boldsymbol{x};\, u) = B(u) \cdot T(\boldsymbol{x}), \tag{3.34}$$

where B is the so-called branch network and T the trunk network. The branch network takes as input a function u (or discrete measurements of the function) and the trunk network takes a point from the input domain. The result is given by the inner product of the branch and the trunk. The DeepONet can the then be trained in a physics-informed setting for various input functions u, sampled from some function space for the goal to approximate the respective inverse operator. For an ELM and the objective (3.26), the branch is given by $B(f, g) = A^{\dagger}\boldsymbol{b}(f, g)$ and the trunk by $T(\boldsymbol{x}) = \ell(\boldsymbol{x})\boldsymbol{z}(\boldsymbol{x})$. The main difference is, that only a shallow network is applied and there is no complicated training procedure required for the determination of the solution operator. Pre-training a DeepONet might be a versatile alternative to ELMs. Especially, Graph Neural Networks [99] are of interest for the task of learning the solution operator [52, 85, 15]. The main challenge lies in the training of the model, which requires sampling from some function space. ELMs could be utilized for efficient sampling of such a space. Especially, training of the branch network can be challenging. But considering also the kernel perspective of ELMs, a further question emerges. How does the optimal low-rank embedding look like for some input function f. It is naive to think that a good embedding can be independent of f and achieve good performance. One could condition the trunk network on the function f with the neural network $\mathcal{N}(\boldsymbol{x}, f(\boldsymbol{x}); \boldsymbol{\omega}) = \boldsymbol{z}(\boldsymbol{x}, f(\boldsymbol{x}))$ and let the output β of the branch network be given in its implicit form $A\beta = \boldsymbol{b}$. The trunk, or ELM embedding, could then be trained with SGD with the input functions f being sampled from some function space. This kind of ELM could be able to learn highly specialized low-rank kernel functions and remove the intricacy of training a branch network by removing it completely. The drawback is that the solution operator could not be pre-computed. However, if the low-rank approximation is good, i.e., M is low, computing the SVD should not be too expensive and the computational cost for evaluating the implicit branch network would be $\mathcal{O}(NM^2)$. It is still unclear if this approach works well in practice and is left for future research. If this research shows successful, one could further condition the embedding on the Approximate

Distance Function (ADF) and thus generalize to arbitrary geometries.

All in all, those seemingly simple ELM models are very powerful function approximators and are capable to solve quite complex problems with ease. For small scale problems, the solution can be computed without the need of the weak formulation, allowing a stable computation without preconditioning. However, for larger irregular domains, it is still questionable how to select the input weights properly. A Gradient Boosting technique could be applied to solve for consecutive small ELMs and the error information might be applied as information for the input weight distribution. Since ELMs are connected to KRR, a low-rank version of KRR could also be applied with a *Nyström* approximation of the kernel matrix.

Since the hard constrained ansatz formulation allows easy and flexible training of PINNs, the next chapter covers the construction of the required indicator functions for arbitrary domains.

3.4 Second Order Optimization of PINNs

Most research focuses on gradient-based optimization methods to train PINN models. This approach is particularly powerful in high-dimensional sample spaces, such as for conditional PINNs. However, PINNs are often trained in relatively simple 2D or 3D domains. In such cases, second-order methods can provide faster and more accurate training. Despite their potential to improve convergence, achieving a true quadratic convergence rate with second-order seems intractable. This difficulty stems from the loss landscape of the objective function [69]. When model parameters are far from the optimum, the landscape is typically characterized by a high number of saddle points, which exponentially increase with the parameter dimension M [18]. When optimizing an objective function f, the Newton method, which uses the update $-\nabla^2 f(\boldsymbol{x})\nabla f(\boldsymbol{x})$, often converges quickly to such a saddle point, making minimal progress. This phenomenon is directly linked to directions of negative curvature. While gradient descent naturally avoids saddle points, Newton's method dynamics are attracted to them because directions of negative curvature correspond to ascent directions of the objective. Therefore, for training a PINN, using the Newton search direction is a bad choice. Gradient descent, on the other hand, easily gets stuck on large plateaus. The saddle-free Newton method [18] addresses this by replacing the Hessian $\nabla^2 f$ with $|\nabla^2 f|$, where $|\nabla^2 f|$ is obtained by taking the absolute value of the eigenvalues, thus converting directions of negative curvature into descent directions again. The challenge in large-scale optimization is that $|\nabla^2 f|$ is not easily obtainable if only relying on Hessian-vector products (HVPs) (see Section A.1).

Subspace methods, such as those in [5] and [35], though practical within a trust region framework, showed to be inefficient because the random subspace is unlikely to capture directions associated with small eigenvalues.

L-BFGS circumvents this issue with a positive definite approximation to the in-

verse Hessian. However, this approximation can be poor, causing the optimizer to stagnate on large plateaus in the loss landscape. In the trust region method (see Appendix B), the Steihaug Conjugate Gradient (Steihaug-CG) (see Algorithm 7) method also avoids negative curvature by halting iterations once negative curvature is detected. However, in practice, the trust region is often exceeded before negative curvature is encountered because the Hessian of a PINN loss typically has many small positive eigenvalues. Especially for hard constrained PINNs as for instance in Example 3.3, using a trust region framework is particularly effective for optimization. Minimizing the quadratic subproblem automatically avoids saddle points and allows effective minimization. Figure 3.14 shows a comparison of solving the Poisson problem in Example 3.3 of the trust region method and L-BFGS. One can see how L-BFGS stalls, while the trust region method achieves a much better optimum. Especially in the beginning, where large negative curvatures are present, both methods converge very fast. Problems start to arise once the parameters are closer to the optimum and the energy landscape becomes very flat. This results in a very ill-conditioned Hessian matrix, which impedes further progress. A quadratic convergence rate is not achieved, neither for the trust region method nor for L-BFGS, although this can actually not be seen from Figure 3.14, since the minimum is unknown.

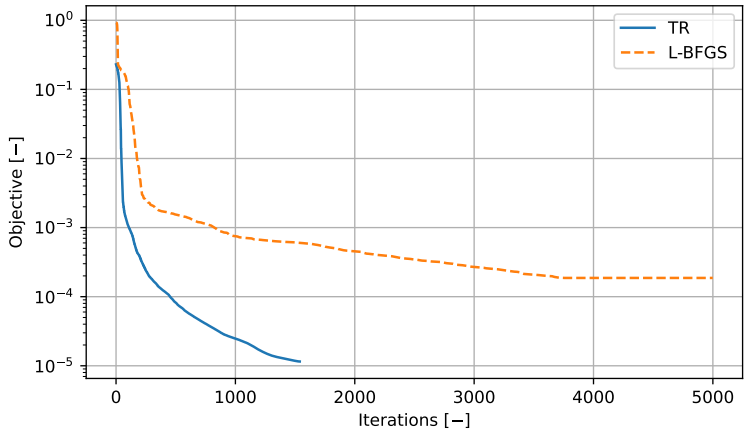


Figure 3.14: Comparison of the trust region method and L-BFGS.

Though, trust region performs better in this example, L-BFGS can also be very efficient in other cases, such as computing a demagnetization process with a hard constrained ansatz as in (6.8). It seems to be easier for L-BFGS to move along the path of minimal energy.

One might argue that a quadratic rate is only given if the parameters are already close to a stationary point. Further, the true solution usually does not lie in the function space of the PINN model and numerical integration introduces further errors. Therefore, a simple test was designed, where the true solution lies in the

same space as the model. This is done by using a random hard constrained PINN (2 hidden layers and 5 nodes per hidden layer) and use its Laplacian as the right-hand side of a Poisson problem with homogeneous boundary conditions. The domain is a square $\Omega = [-1, 1]^2$. The parameters $\boldsymbol{\omega} = \hat{\boldsymbol{\omega}} + \epsilon$ with $\epsilon \sim N(0, 0.01^2)$, with $\hat{\boldsymbol{\omega}}$ being the solution parameters, are used for initialization. Further, to mitigate errors from Monte Carlo integration, a 5^{th} order Gauss-Legendre quadrature scheme was used for each tile in a regular 9×9 partitioning of the domain. The objective was then minimized with a trust region framework with Steihaug-CG and double precision until the gradient norm satisfied $\|\nabla_{\boldsymbol{\omega}} \mathcal{L}\| < 1 \times 10^{-13}$. Figure 3.15 shows the condition

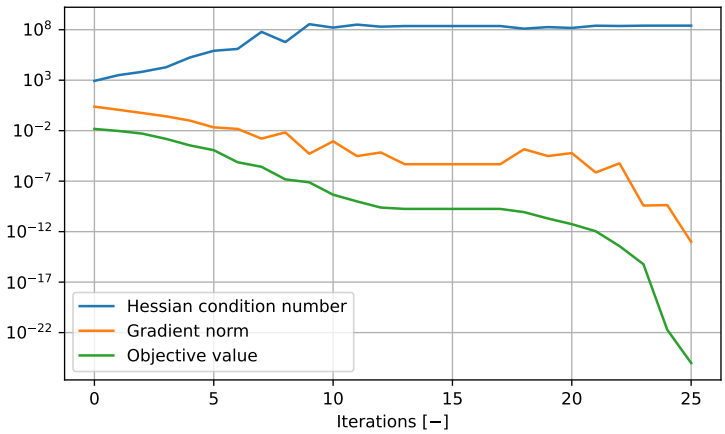


Figure 3.15: Condition number of Hessian, gradient norm and objective value of a hard constrained trial problem to test convergence.

number of the Hessian, the gradient norm of the objective and the objective value. It can be seen, that the model parameters converge quickly to a critical point with the same parameters as given by the solution model. Optimization quickly moves into a positive definite region of the objective with a constant condition number of 2.5×10^{8}. It can also be seen that even for this small toy problem, the Hessian is highly ill-conditioned. So in principle, it could be possible to achieve better convergence rates for PINNs, but we were only able to achieve this with this simple toy example. Also, if the solution does not lie in the parameter space of the PINN, verification of the convergence rate is difficult, since the minimum of the objective is a number bigger than zero. However, for this toy example, the Hessian is positive definite, which makes optimization easy. In more realistic settings, this will certainly not be the case and the Hessian will be indefinite and almost singular. The Steihaug-CG method does not effectively avoid saddle points in such a case, and a more sophisticated solver might be the better choice. In this work, we still achieved highly accurate training with this method. Interestingly, limiting the iteration number for Steihaug-CG can perform better. It can also be seen from this toy example, that a good initialization of the network can have a drastic impact on the convergence. Altogether, it is a very

challenging task to find a global optimal solution for a hard constrained PINN and that raises the question if there are better architectures with better conditioning or if the objective function can be posed such that optimization is easier.

Efficient optimization schemes are critical for PINN training and remain one of the main bottlenecks preventing PINNs from replacing traditional numerical methods. Although there is extensive research on second-order optimization, efficient Hessian-free second-order schemes are still rare and lack implementations. In this work, a trust region method with the Steihaug-CG method as a subproblem solver is applied. However, as discussed, there is significant room for improvement over the results achieved in this work. Future research should focus on developing more effective second-order optimization techniques that can better handle the complex loss landscapes encountered in PINN training. Also, another line of research could be the improvement of the network architecture such that the Hessian of the objective is better conditioned.

4 Constructive Solid Geometry

Parts of this chapter were previously published as [78, 76]. Permission for use has been granted by the co-authors.

CSG offers a powerful framework for representing complex. Accurately representing the domains of PDEs, especially for complex geometries, can be challenging. CSG provides a systematic approach to define intricate geometries through Boolean operations on simple geometric primitives such as spheres, cubes, cylinders, and their combinations. By leveraging CSG techniques, it is possible to efficiently generate detailed geometries that precisely conform to the requirements of PDE-based simulations.

4.1 *R*-Functions

R-functions are a fascinating concept of CSG with a rich literature [83, 72, 82]. Implicit functions of the form $\ell(\boldsymbol{x}) = 0$ are composed to describe geometric objects. In principle, it is very easy to describe most geometric objects via implicit functions, but often differential properties are essential for modeling, as is the case for physics-informed machine learning. A *R*-function is a function whose sign is fully determined by its arguments. Such functions can be interpreted as a form of Boolean algebra, where a positive sign indicates `true` and a negative sign `false`. Using respective logical predicates, those functions can be further composed and since Boolean logic is closely related to set theory it is no surprise that geometric objects can be described that way.

Usually the partition of the real axis is chosen to be $\{(-\infty, 0], [0, \infty)\}$. Note that 0 is part of both intervals. This definition ensures continuity. If 0 would only be part of one interval, logical negation would require a discontinuity at 0. However, there are other definitions. For more details, the reader is referred to [83].

The simplest example of a sufficiently complete system of R_1-functions, with their Boolean companion function in parentheses, is

$$\begin{aligned} \overline{\ell}_1 &\equiv -\ell_1 && \text{(logical negation } \neg) \\ \ell_1 \wedge_1 \ell_2 &\equiv \min(\ell_1, \ell_2) && \text{(logical conjunction } \wedge) \\ \ell_1 \vee_1 \ell_2 &\equiv \max(\ell_1, \ell_2) && \text{(logical disjunction } \vee), \end{aligned} \tag{4.1}$$

which corresponds to *R*-negation, *R*-conjunction and *R*-disjunction. For ℓ_i being some exact distance function, the composition with R_1 results also in the exact distance function of the composed object. Even though this system is not minimal,

it is intuitive to define R-conjunction and R-disjunction for geometric modeling [82]. Logical negation is usually always given by real negation, hence, one only needs to define conjunction and disjunction to form a complete system. All further operations such as the difference, equivalence or xor operation can be constructed from those primitives, together with their direct translation to set theory. We can write

$$\begin{aligned} \min(\ell_1, \ell_2) &= \frac{1}{2}(\ell_1 + \ell_2 - \sqrt{(\ell_1 - \ell_2)^2}) \\ \max(\ell_1, \ell_2) &= \frac{1}{2}(\ell_1 + \ell_2 + \sqrt{(\ell_1 - \ell_2)^2}) \end{aligned} \tag{4.2}$$

to see that those operations are not differentiable along $\ell_1 = \ell_2$. Especially in the course of machine learning, where we are very concerned about differentiability, we can generalize the system R_1 as

$$R_\alpha : \quad \frac{1}{1+\alpha}(\ell_1 + \ell_2 \pm \sqrt{\ell_1^2 + \ell_2^2 - 2\alpha\ell_1\ell_2}), \tag{4.3}$$

for $-1 < \alpha \leq 1$ and $(+)$ defining R-disjunction and $(-)$ R-conjunction. By now, the purpose of the subscript 1 in (4.1) should be clear. For us, one important system, due to its simplicity and its differentiable properties, is R_0

$$R_0 : \quad \ell_1 + \ell_2 \pm \sqrt{\ell_1^2 + \ell_2^2}. \tag{4.4}$$

The derivative is well-defined everywhere but the origin $(0, 0)$. For more information on different systems of R-functions and logic properties, the reader is referred to [83, 82, 84].

Since we are concerned about differentiability, especially in the course of AD, we want to select a system of R-functions which satisfy some important properties. Especially, when describing our respective domain Ω, we want this function to be differentiable everywhere within the domain. The systems R_0 satisfy this condition but R_1 does not. Differentiation of the system R_0 with $r(\ell_1, \ell_2)$ being either R-conjunction or R-disjunction results in

$$\partial_{\ell_i} r = 1 \pm \frac{\ell_i}{\sqrt{\ell_1^2 + \ell_2^2}}, \; i = 1, 2 \tag{4.5}$$

and is well-defined, except at $\ell_1 = \ell_2 = 0$. On the boundary of the composed function, where either $\ell_1 = 0$ or $\ell_2 = 0$ and the sign switches, we find that $\partial_{\ell_i} r|_{\ell_i = 0} = 1$ and the derivative with respect to the other variable is 0. In this work, we denote the inward pointing normal vector by $\boldsymbol{\nu}$ and the outward pointing normal vector by $\boldsymbol{n} = -\boldsymbol{\nu}$. We call ℓ normalized to the k^{th} order if it satisfies

$$\ell|_{\partial\Omega} = 0, \quad D_{\boldsymbol{\nu}}\ell|_{\partial\Omega} = 1, \quad D_{\boldsymbol{\nu}}^k \ell\Big|_{\partial\Omega} = 0 \;\text{ for } k = 2, \ldots, k, \tag{4.6}$$

where $D_{\boldsymbol{\nu}}\ell$ the directional derivative in direction $\boldsymbol{\nu}$. This implies that the $\nabla\ell = \boldsymbol{\nu}$, which is a useful property for computation of the normal vector. If a function ℓ is

at least normalized to first order, it approximates the exact distance function in the vicinity of the boundary. We call such a function an approximate distance function (ADF).

4.1.1 Exact imposition of homogeneous Dirichlet boundary conditions

For instance, to solve a homogenous Dirichlet problem, an ansatz function

$$u(\boldsymbol{x}) = \ell(\boldsymbol{x})\tilde{u}(\boldsymbol{x}), \quad \ell(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \partial\Omega \\ > 0 & \text{if } \boldsymbol{x} \in \Omega \end{cases}, \tag{4.7}$$

where ℓ is a fixed indicator function, can be used. The approximation function $\tilde{u}$ can be freely chosen such that u approximates γ. In particular, for this solution structure, it is important that the gradient $\nabla\ell$ does not vanish on the boundary. In fact, there is the following theorem about the completeness for homogeneous Dirichlet problems.

Theorem 1 (Completeness for homogeneous Dirichlet problems (Theorem 1 in [73]))**.** *Suppose that Ω is a bounded region of d-dimensional space with boundary $\partial\Omega$. Let k and r be positive integers ($k > r$) and $\ell(\boldsymbol{x})$ be a function defined in an open region that contains Ω and satisfying the following conditions:*

1. $\ell(\boldsymbol{x})|_{\boldsymbol{x}\in\partial\Omega} = 0, \quad \ell(\boldsymbol{x})|_{\boldsymbol{x}\notin\partial\Omega} \neq 0$;
2. *$\ell(\boldsymbol{x})$ is k times continuously differentiable and its derivatives of order k satisfy the Lipschitz condition;*
3. $\nabla\ell(\boldsymbol{x})|_{\boldsymbol{x}\in\partial\Omega} \neq 0$.

If the function $\gamma(\boldsymbol{x})$ is k times continuously differentiable in the region Ω and vanishes on the boundary $\partial\Omega$ together with its derivatives up to order $(r-1)$, then for any positive ϵ it is possible to construct a sequence of polynomials $P_n(\boldsymbol{x})$ such that:

$$\|\gamma - \ell^r P_n\|_{C^p(\Omega)} < \epsilon, \tag{4.8}$$

where $p = 1, 2, \ldots, k$.

Completeness for a hard constraint PINN solution follows from the universal approximation theorem of neural networks [38]. In [73] the completeness for inhomogeneous Dirichlet boundary conditions is shown as well. In our case, where $r = 1$, using an ADF ℓ which is normalized to first order is therefore sufficient for the solution structure of the Poisson problem (3.3).

We are only concerned about imposition of Dirichlet boundary conditions. But it should be noted that R-functions can also be used to exactly impose higher order or mixed boundary conditions by means of the generalized Taylor polynomial [71, 72, 91].

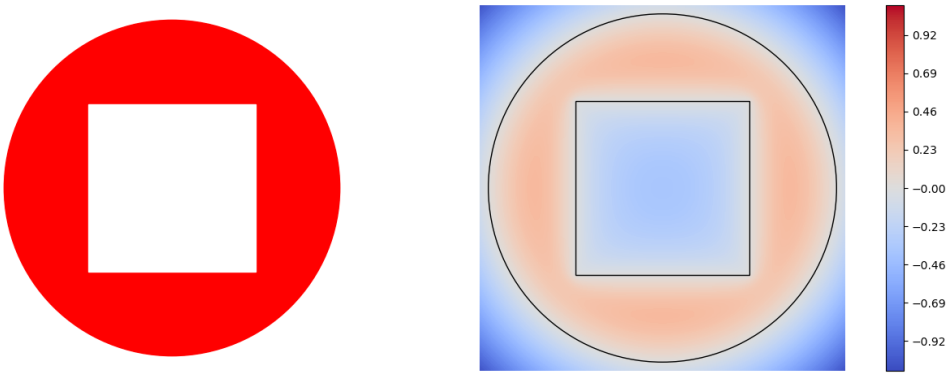


Figure 4.1: Simple shape of circle with a square remove from its interior and the corresponding ADF constructed with R_0.

4.1.2 *R*-Function modeling

To ensure that an implicit function ℓ is normalized to first order, one can use

$$\frac{\ell}{\sqrt{\ell^2 + \|\nabla \ell\|^2}} \tag{4.9}$$

in order to derive a normalized version of ℓ. This can be useful if we don't know some normalized version beforehand but can lead to a complicated ADF if ℓ is already very complex. Therefore, the better approach is to construct the ADF from simpler already normalized distance functions using the system R_0, since normalization is preserved up to first order. To be more precise, the following theorem describes the conditions for the directional derivative to be preserved.

Theorem 2 (Theorem 3 from [83]). *Suppose that argument x_i appears in the R_α-function $f(x_1, \ldots, x_n)$ only once and has an inversion degree of r. Let the functions $\ell_1, \ldots, \ell_n$ be in C^s and the boundary $\partial\Omega = (f(\ell_1, \ldots, \ell_n) = 0)$. At every regular point $\boldsymbol{p} \in \partial\Omega$ where*

$$\ell_i(\boldsymbol{p}) = 0, \quad \ell_j(\boldsymbol{p}) \neq 0, \quad j = 1, \ldots, n,\ i \neq j,$$

and some direction $\boldsymbol{v}$, the following equality holds

$$[D_{\boldsymbol{v}} f(\ell_1, \ldots, \ell_n)]|_{\boldsymbol{p}} = (-1)^r\ (D_{\boldsymbol{v}} \ell_i)|_{\boldsymbol{p}}\,.$$

When joining two ADF with R-conjunction or R-disjunction, it is important to note that those ADF should not partially describe the same boundary, since the resulting function will not be differentiable on this segment.

Example 4.1. *Figure 4.1 shows the construction of the ADF ℓ of a circle with a*

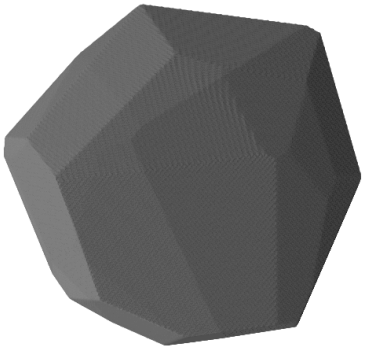


Figure 4.2: Zero level-set of a randomly sampled convex grain. Each face is described by its exact distance function. The ADF of the grain is given by the conjunction of all exact distance functions. We sampled 24 normal vectors and offsets from a uniform distribution to assemble the ADF. The offsets are drawn from $[0.9, 1.0]$.

square area removed from its interior using R_0, i.e.

$$\begin{aligned} \ell_{circ}(\boldsymbol{x}) &= 2 - \sqrt{\boldsymbol{x}_1^2 + \boldsymbol{x}_2^2} \\ \ell_{sq}(\boldsymbol{x}) &= ((\boldsymbol{x}_1 + 1) \wedge_0 (1 - \boldsymbol{x}_1)) \wedge_0 ((\boldsymbol{x}_2 + 1) \wedge_0 (1 - \boldsymbol{x}_2)) \\ \ell &= \ell_{circ} \setminus_0 \ell_{sq} = \ell_{circ} \wedge_0 \overline{\ell_{sq}}. \end{aligned} \tag{4.10}$$

Example 4.2. *For micromagnetic modeling, we could describe a single convex polyhedron (grain) by the normal vectors of its surface elements. Each face could be described by its exact distance function and the ADF would be given by R-conjunction, i.e.,*

$$\begin{aligned} \ell_i(\boldsymbol{x}) &= -\boldsymbol{x} \cdot \boldsymbol{n}_i + b_i, \quad \text{for } i = 1, \dots, S, \\ \ell &= \ell_1 \wedge_0 \ell_2 \wedge_0 \cdots \wedge_0 \ell_S, \end{aligned} \tag{4.11}$$

where S is the number of faces. The non-associativity is neglected since the ADF will describe the same domain nonetheless. For the i^{th} face, $\boldsymbol{n}_i$ is the outward looking normal vector and b_i is the offset from the origin. Figure 4.2 shows a randomly sampled grain.

A useful feature is the possibility to translate, rotate and scale the ADF without losing any of its differential properties. For example, to translate any ADF in $\mathbb{R}$ by a vector $\boldsymbol{x}_0$, we can apply the transformation $\boldsymbol{x} \mapsto \boldsymbol{x} - \boldsymbol{x}_0$. To rotate the shape in Figure 4.1 counterclockwise by some angle, we can first rotate the input $\boldsymbol{x} = (x, y)$ clockwise by the same angle. To scale some ADF $\ell(\boldsymbol{x})$ we can use $\ell(\boldsymbol{x}/s)s$, where s is the scaling factor. This preserves the gradient of ℓ. Further, one could also project the ADF onto some hyperplane or create the body of revolution from a 2d ADF.

With those primitives, we can create a modeling language which works similarly as OpenSCAD's [62] modeling language and, hence, create a powerful framework for CSG using implicit functions.

To establish a user-friendly framework for CSG with R-functions, one should consider higher-order functions (HOF) from the functional programming paradigm. That is, functions which accept other functions as arguments, apply some kind of transformation and return an altered function. With this in mind, we can define HOFs for R-conjunction, R-disjunction, and R-negation for some system of R-functions. All further operations can be composed of these fundamentals. All transformations, e.g., translation, rotation, scaling, or extrusion can be abstractly defined as HOF as well and work for any system of R-functions. With basic building blocks, such as the ADF for a cube, a sphere, or a cylinder, the user can compose the desired geometry. Further, R-functions are a great way to build toy problems on complex domains, where the solution is given by the ADF.

Such a functional framework can result in poorly performant code when using an interpreted programming language like Python. Since `JAX` offers *just-in-time* (jit) compilation, we can use the XLA compiler to generate highly performant XLA-optimized kernels of the ADF for the respective device. AD and jit compilation can be arbitrarily composed. Listing 4.1 shows our modeling framework for a "Pac-Man" Rfunction model. The final ADF can be jit compiled. Figure 4.3 shows the zero level-set of the same ADF. This example should demonstrate the rich capabilities of such a modeling framework to describe complex geometries.

Listing 4.1: Modeling of Pac-Man using R-functions.

```
unit_vec = lambda x: x / norm(x, axis=-1, keepdims=True)
mouth = r0.intersection(
    lambda x: x @ unit_vec(jnp.array([1., 0., -1.8])),
    lambda x: x @ unit_vec(jnp.array([1., 0., 1.8]))
)
left_eye = translate(sphere(0.25), [0.4, -0.5, 0.6])
right_eye = translate(sphere(0.25), [0.4, 0.5, 0.6])
pacman = r0.difference(sphere(1.), mouth)
pacman = r0.union(pacman, left_eye)
pacman = r0.union(pacman, right_eye)
pacman = r0.union(pacman, translate(sphere(0.15),
                                    [1., 0., 0.]))
pacman = r0.union(pacman, translate(sphere(0.15),
                                    [2., 0., 0.]))
pacman = jax.jit(pacman)
```

There are many other interesting concepts including blending or trimming [82] which are not mentioned here but can be useful for some modeling applications.

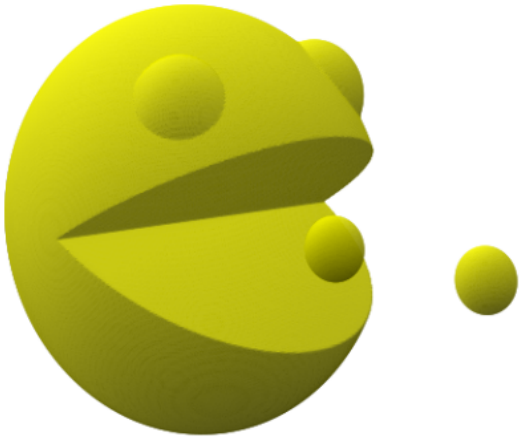


Figure 4.3: Zero level-set of Pac-Man from Listing 4.1. Visualization of the surface was done with Marching Cubes [54].

4.1.3 Other systems of *R*-functions

For solving PDEs with a hard constraint ansatz as in (4.7), the system R_0 will often suffice. However, if the domain has sharp corners or edges (in 3d), it can become an impediment for training since second order derivatives are unbounded. This can impair model accuracy and training speed. Using another system of R-functions can serve as a form of regularization by smoothing the ADF at those areas, i.e., reducing extreme curvatures. For instance, the system R_0^m [82] is given by

$$R_0^m : \quad \ell_1 + \ell_2 \pm \sqrt{\ell_1^2 + \ell_2^2}(\ell_1^2 + \ell_2^2)^{\frac{m}{2}}. \tag{4.12}$$

With some small value $m < 1$, this system will smooth the shape of the ADF. Figure 4.4 shows the ADF of a square constructed with the associative R_0^m-equivalence operation $\sim_m$, i.e.,

$$\ell_{sq}(\boldsymbol{x}) = \boldsymbol{x}_1 \sim_m (1 - \boldsymbol{x}_1) \sim_m \boldsymbol{x}_2 \sim_m (1 - \boldsymbol{x}_2), \tag{4.13}$$

together with the norm of the Jacobian $\|J_{\ell_{sq}}\|_2$ and the Laplace $\Delta\ell_{sq}$. The R_0^m-equivalence operation is the material biconditional: $\ell_1 \sim_m \ell_2 = -(\ell_1 \oplus_m \ell_2)$, where $\oplus_m$ is the exclusive or operation.

For the system R_0^0, the curvature rapidly goes to infinity at the corners of the square. This effect is reduced for values $m > 0$, waiving first order normalization. But it can also be seen, that for very small values of $m \ll 1$, normalization is almost preserved.

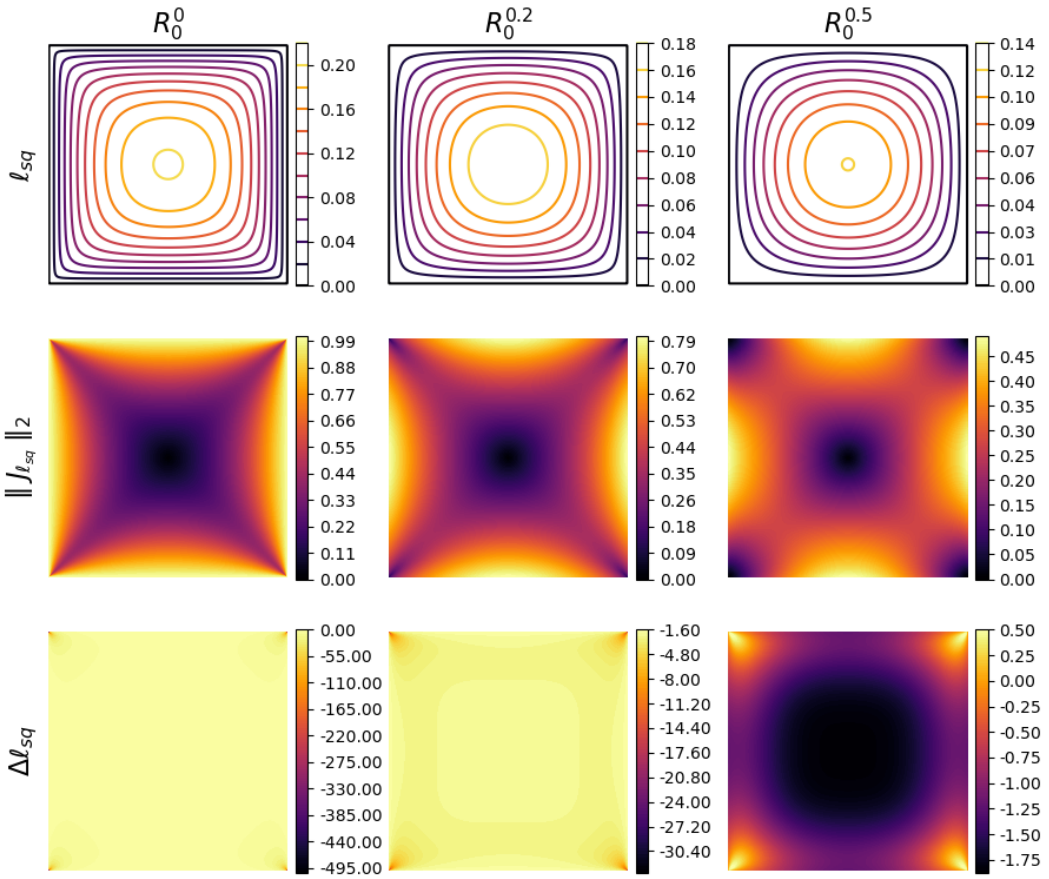


Figure 4.4: ADF of a square constructed with the associative R_0^m-equivalence operation, together with the Jacobian norm $\|J_{\ell_{sq}}\|_2$ and the Laplace $\Delta \ell_{sq}$ for different values of m (left $m = 0$, middle $m = 0.2$, right $m = 0.5$).

Another possibility is to keep the ADF normalized, but to change the geometry. Therefore, a blending system of R-functions can be used. For instance [82]

$$B_\rho : \quad \ell_1 + \ell_2 \pm \sqrt{\ell_1^2 + \ell_2^2 + \frac{1}{8\rho^2} s_\rho (s_\rho - |s_\rho|)}, \tag{4.14}$$

with $s_\rho = \ell_1^2 + \ell_2^2 - \rho^2$ will round the corners of the ADF but preserve the normalization. Figure 4.5 shows again the ADF, together with the Jacobian norm and the Laplace, of a square, but now constructed with the equivalence operation using the system B_ρ. It can be seen that the corners are rounded and normalization is preserved. Also, the rounding of the corners reduces the curvature.

Another system, as a generalization of the R_0-system, is given by [84]

$$R_p : \quad \ell_1 + \ell_2 \pm (\ell_1^p + \ell_2^p)^{\frac{1}{p}}, \tag{4.15}$$

for some even positive integer p. This comes from restating the triangle inequality with the L_p-norm. The system R_p preserves normalization up to the $(p-1)^{\text{th}}$ order. It is differentiable everywhere, except the corner points.

4.2 Sampled indicator functions

In order to construct an indicator function ℓ even for complex geometries, one can use the method described in [87]. For that purpose, the boundary $\partial\Omega$ of the domain is partitioned into K segments $\{\partial\Omega_1, \dots, \partial\Omega_K\}$, such that for each segment $\partial\Omega_k$, there is a non-adjacent segment $\partial\Omega_{k_o}$. For each segment, a spline function is computed which satisfies

$$\begin{cases} l_k(\boldsymbol{x}) = 0 & \text{if } \boldsymbol{x} \in \partial\Omega_k, \\ l_k(\boldsymbol{x}) = 1 & \text{if } \boldsymbol{x} \in \partial\Omega_{k_o}, \\ 0 < l_k(\boldsymbol{x}) < 1 & \text{otherwise.} \end{cases} \tag{4.16}$$

According to [87], the indicator function is then given by

$$\ell(\boldsymbol{x}) = \prod_{k=1}^{K} 1 - (1 - l_k(\boldsymbol{x}))^\mu, \tag{4.17}$$

where the hyperparameter $\mu \geq 1$ can adjust the shape of ℓ. In [87], it is also suggested to set $\mu = K$. The composition (4.17) could be problematic in some instances, since it might not satisfy the third condition of Theorem 1. Therefore, it might be better to use some system of R-functions in combination with (4.9) to assure a normalized distance function.

To compute l_k, an inverse multiquadric radial basis interpolation [97, 87] is applied. For N_k sample points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_{N_k}\} \subset \partial\Omega_k \cup \partial\Omega_{k_o}$, l_k is given by

$$l_k(\boldsymbol{x}) = \sum_{i=1}^{N_k} a_i \frac{1}{\sqrt{(e^2 + \|\boldsymbol{x} - \boldsymbol{x}_i\|^2)}} + \boldsymbol{b}^T \boldsymbol{x} + c. \tag{4.18}$$

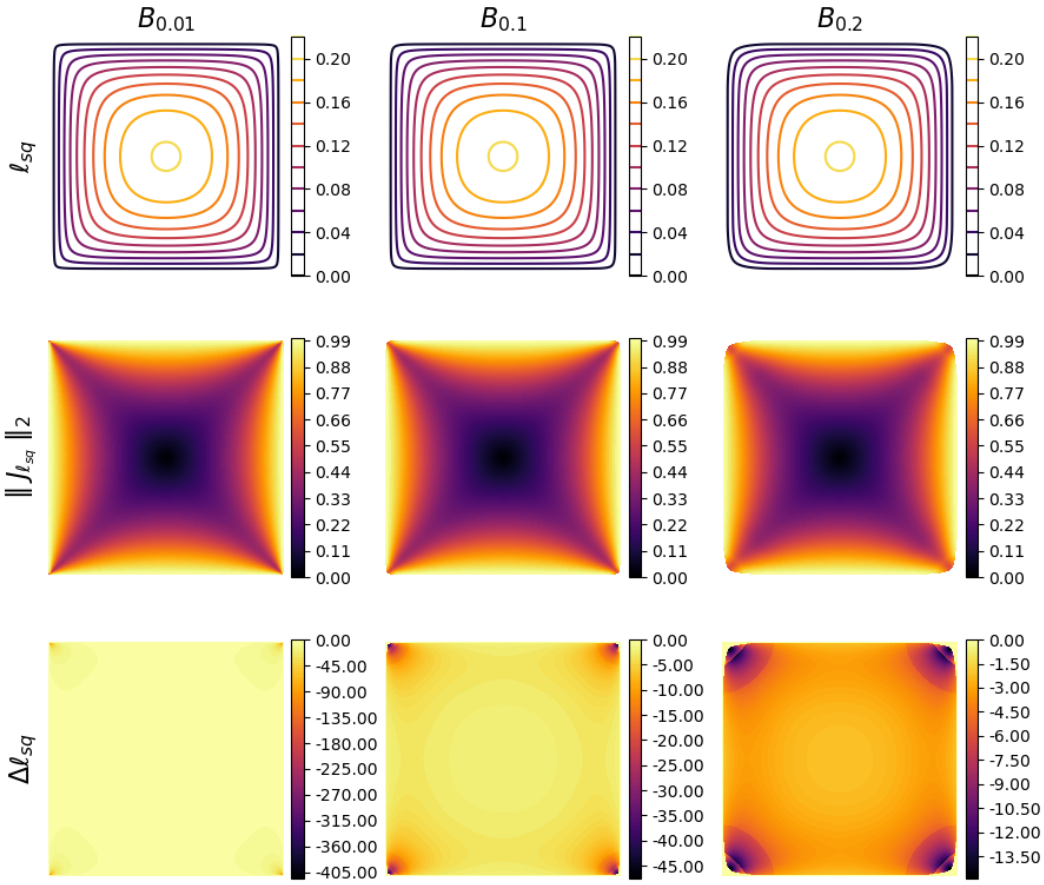


Figure 4.5: ADF of a square constructed with the associative B_ρ-equivalence operation, together with the Jacobian norm $\|J_{\ell_{sq}}\|_2$ and the Laplace $\Delta\ell_{sq}$ for different values of m (left $\rho = 0.01$, middle $\rho = 0.1$, right $\rho = 0.2$). Values corresponding to negative values of the ADF were removed.

The parameter e is used to adjust the shape of the basis function. In [87] it is recommended to set $e = 1.25R/\sqrt{N_k}$ to maintain scale invariance, where R is the radius of the minimal circle which encloses all sample points. The coefficients can be determined by solving the following linear system

$$\begin{aligned} &\sum_{i=1}^{N_k} a_i \boldsymbol{x}_i = \mathbf{0}, \\ &\sum_{i=1}^{N_k} a_i = 0, \\ &l_k(\boldsymbol{x}_i) = \begin{cases} 0 & \text{if } \boldsymbol{x}_i \in \partial\Omega_k \\ 1 & \text{if } \boldsymbol{x}_i \in \partial\Omega_{k_o} \end{cases}, \; i = 1, \ldots, N_k, \end{aligned} \tag{4.19}$$

which has a unique solution if all sample points are distinct. This construction of the indicator function is applicable even for complex geometries and only requires a point cloud representation of the respective surface of the domain.

The constructive modeling approach is a powerful tool for the description of computational domains and, as seen in Chapter 3, can be used to build highly complex ansatz functions which are capable to capture essential, natural or mixed boundary conditions and to approximate the solution to PDEs on those domains. They can be optimized in a physics-informed setting. In the forthcoming chapters, these implicit functions are used to describe simple domains for micromagnetic modeling to solve particular problems solely with a data-driven PINN approach.

5 Magnetostatics with PINNs

Parts of this chapter were previously published or submitted for publication as [79]. Permission for use has been granted by the co-authors.

Magnetostatics focuses on the study of magnetic fields in systems with steady currents. The fundamental equations governing magnetostatics are derived from Maxwell's equations and describe how magnetic fields are generated by steady currents and magnetic materials. Solving these equations typically involves complex boundary conditions and interactions between magnetic domains, which has traditionally required numerical methods such as the finite difference method (FDM) [58] or finite element method (FEM) [81].

Continuum mechanics provides a framework for describing the behavior of materials by considering them as continuous media. In the context of magnetostatics, continuum mechanics principles are applied to model the continuous distribution of magnetization within a material. This approach facilitates the understanding of how magnetic fields interact with materials at a macroscopic level, accounting for phenomena such as stress and strain in magnetic materials.

Recently, the advent of PINNs has introduced a novel approach to solving magnetostatic problems. PINNs can be used to model continuous magnetization configurations. This approach not only simplifies the computational process but also offers flexibility in incorporating additional conditional parameters such as exchange length. Moreover, as described in this chapter, they can efficiently handle the computationally intensive stray field problem by employing techniques like ELMs and splitting methods, further enhancing their applicability to magnetostatic simulations.

Overall, the integration of PINNs into magnetostatic modeling is an important research topic, which could allow researchers and engineers to tackle complex magnetic problems with improved efficiency.

5.1 The magnetostatic self-energy

An important aspect of magnetostatics is the calculation of magnetostatic self-energy, which represents the energy stored in the magnetic field due to the magnetization of the material. This self-energy is a crucial factor in determining the stability and equilibrium configurations of magnetic systems, influencing how magnetic domains form and interact.

The magnetostatic self-energy or demagnetizing energy E_d of a magnet $\Omega \subset \mathbb{R}^3$

is usually computed via the stray field $\boldsymbol{H}_d = M_s \boldsymbol{h}_d$ through [11]

$$E_d(\boldsymbol{m}) = -\frac{\mu_0 M_s^2}{2} \int_\Omega \boldsymbol{m} \cdot \boldsymbol{h}_d \, \mathrm{d}\boldsymbol{x} = \frac{\mu_0 M_s^2}{2} \, e_d(\boldsymbol{m}), \tag{5.1}$$

where $\boldsymbol{m} \in L^2(\mathbb{R}^3)^3$ is the magnetization configuration, which is non-zero in the bounded domain Ω and zero in the exterior domain $\overline{\Omega}^c = \mathbb{R}^3 \setminus \overline{\Omega}$, M_s the saturation magnetization, μ_0 the vacuum permeability, $\boldsymbol{h}_d \in L^2(\mathbb{R}^3)^3$ the dimensionless stray field and e_d the reduced self-energy in dimensions of volume.

Accurate computation of the stray field is crucial for many numerical algorithms in micromagnetism. This step enables precise mathematical modeling of magnetic system behaviors, which is essential for designing highly efficient devices or permanent magnets.

The stray field $\boldsymbol{h}_d = -\nabla \phi_d$ is the solution of the whole space Poisson problem

$$\Delta \phi_d = \nabla \cdot \boldsymbol{m} \quad \text{in } \mathbb{R}^3 \tag{5.2}$$

for the scalar potential ϕ_d. The Poisson equation (5.2) follows from the magnetostatic Maxwell's equations,

$$\begin{aligned} \boldsymbol{b}_d &= \boldsymbol{m} + \boldsymbol{h}_d \\ \nabla \cdot \boldsymbol{b}_d &= 0 \\ \nabla \times \boldsymbol{h}_d &= \boldsymbol{0}, \end{aligned} \tag{5.3}$$

where $\boldsymbol{b}_d = \boldsymbol{B}_d/(M_s \mu_0)$ is the (dimensionless) magnetic induction (or magnetic flux density) $\boldsymbol{B}_d$. The fields $\boldsymbol{h}_d = \boldsymbol{h}_d(\boldsymbol{m})$ and $\boldsymbol{b}_d = \boldsymbol{b}_d(\boldsymbol{m})$ depend (linearly) on $\boldsymbol{m}$, but we omit this dependency for the sake of simpler notation. The components $\boldsymbol{b}_d \cdot \boldsymbol{n}$ and $\boldsymbol{h}_d \times \boldsymbol{n}$, with $\boldsymbol{n}$ denoting the outward unit normal vector, are continuous across the surface of Ω [10]. The fields $\boldsymbol{h}_d$ and $\boldsymbol{b}_d$ form a Helmholtz decomposition of the magnetization $\boldsymbol{m}$ in whole space into a sum of an irrotational and solenoidal part [10], which are L^2-orthogonal in whole space, that is,

$$\int_{\mathbb{R}^3} \boldsymbol{h}_d \cdot \boldsymbol{b}_d \, \mathrm{d}\boldsymbol{x} = 0. \tag{5.4}$$

Using this orthogonality and the relations $\boldsymbol{b}_d = \boldsymbol{m} + \boldsymbol{h}_d$ in (5.3) one can express the demagnetizing energy (5.1) in the equivalent forms

$$\begin{aligned} e_d(\boldsymbol{m}) &= -\int_\Omega \boldsymbol{m} \cdot \boldsymbol{h}_d \, \mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^3} \|\boldsymbol{h}_d\|^2 \, \mathrm{d}\boldsymbol{x} \\ &= \int_\Omega \|\boldsymbol{m}\|^2 - \boldsymbol{m} \cdot \boldsymbol{b}_d \, \mathrm{d}\boldsymbol{x} = \int_\Omega \|\boldsymbol{m}\|^2 \, \mathrm{d}\boldsymbol{x} - \int_{\mathbb{R}^3} \|\boldsymbol{b}_d\|^2 \, \mathrm{d}\boldsymbol{x}, \end{aligned} \tag{5.5}$$

where integration involving $\boldsymbol{m}$ is over Ω only and $\int_\Omega \|\boldsymbol{m}\|^2 \, dx = V$ the volume of Ω. Opposed to $\boldsymbol{h}_d$ being the solution of (5.2), the magnetic induction $\boldsymbol{b}_d = \nabla \times \boldsymbol{A}_d$ is the solution of the whole space vector Poisson equation

$$\nabla \times (\nabla \times \boldsymbol{A}_d) = \nabla \times \boldsymbol{m} \quad \text{in } \mathbb{R}^3 \tag{5.6}$$

for the vector potential $\boldsymbol{A}_d$. There is freedom of gauge for the vector potential, since one can add an arbitrary gradient field, i.e., $\boldsymbol{b}_d = \nabla \times (\boldsymbol{A}_d + \nabla\Psi)$. Usually the *Coulomb gauge*

$$\nabla \cdot \boldsymbol{A}_d = 0 \tag{5.7}$$

is chosen, which reduces (5.6) to

$$\Delta \boldsymbol{A}_d = -\nabla \times \boldsymbol{m} \quad \text{in } \mathbb{R}^3. \tag{5.8}$$

Splitting the interior and the exterior part of the scalar potential, $\phi_d = \phi_d^{int}$ in Ω and $\phi_d = \phi_d^{ext}$ in $\overline{\Omega}^c$, (5.2) can be formulated as a transmission problem [14, 21]

$$\begin{aligned}
-\Delta\phi_d &= -\nabla \cdot \boldsymbol{m} && \text{in } \Omega \subset \mathbb{R}^3,\\
-\Delta\phi_d &= 0 && \text{in } \overline{\Omega}^c,\\
[\phi_d] &= 0 && \text{on } \partial\Omega,\\
[D_{\boldsymbol{n}}\phi_d] &= -\boldsymbol{m} \cdot \boldsymbol{n} && \text{on } \partial\Omega,\\
\phi_d(\boldsymbol{x}) &= \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) && \text{as } |\boldsymbol{x}| \to \infty,
\end{aligned} \tag{5.9}$$

The jump conditions $[\,\cdot\,]$ describe the difference of the exterior and interior behavior at the boundary of the magnet, i.e.,

$$\begin{aligned}
[\phi_d]_{\partial\Omega} &= (\phi_d^{ext} - \phi_d^{int})\big|_{\partial\Omega}\\
[D_{\boldsymbol{n}}\phi_d]_{\partial\Omega} &= (D_{\boldsymbol{n}}\phi_d^{ext} - D_{\boldsymbol{n}}\phi_d^{int})\big|_{\partial\Omega}.
\end{aligned} \tag{5.10}$$

Further, the normal derivative, with outward unit normal vector $\boldsymbol{n}$, is denoted by $D_{\boldsymbol{n}}$. For (5.9) to be well-posed, the boundary $\partial\Omega$ needs to be Lipschitz continuous. For grain structure modeling, this is usually true. In the same way, the Poisson equation of the vector potential (5.8) gives [21]

$$\begin{aligned}
\Delta \boldsymbol{A}_d &= -\nabla \times \boldsymbol{m} && \text{in } \Omega,\\
\Delta \boldsymbol{A}_d &= \boldsymbol{0} && \text{in } \overline{\Omega}^c,\\
[\boldsymbol{A}_d] &= \boldsymbol{0} && \text{on } \partial\Omega,\\
[D_{\boldsymbol{n}}\boldsymbol{A}_d] &= -\boldsymbol{m} \times \boldsymbol{n} && \text{on } \partial\Omega,\\
(\boldsymbol{A}_d)_j(\boldsymbol{x}) &= \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) && \text{as } \|\boldsymbol{x}\| \to \infty,\ j = 1, 2, 3.
\end{aligned} \tag{5.11}$$

The transmission problems (5.9) or (5.11) can be solved in different ways numerically. Finite element methods usually solve the weak formulation for certain splitting of the potentials [32, 34, 81, 28], finite difference methods approximate the integral representations of the solution [58],

$$\phi_d(\boldsymbol{x}) = -\frac{1}{4\pi}\left(\int_\Omega \frac{\nabla \cdot \boldsymbol{m}(\boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|}\mathrm{d}\boldsymbol{y} - \int_{\partial\Omega} \frac{\boldsymbol{m}(\boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|}\mathrm{d}s(\boldsymbol{y})\right). \tag{5.12}$$

However, there are significant challenges in implementing this solution. Firstly, the computation of the convolution integral across the entire domain is highly resource-intensive. Secondly, obtaining the gradient of (5.12) is typically done with numerical differentiation, which can lead to discretization and round-off errors. Recent neural network methods take a mixed approach [78, 76]. In whatever way the solution is obtained, the computation of the $\boldsymbol{h}_d$-field (or $\boldsymbol{b}_d$-field) is usually the most time-consuming part in micromagnetic modelling due to its non-local nature [2].

5.2 Brown's energy bounds

In the 60s, W.F. Brown suggested simple sharp bounds for the magnetostatic energy only in terms of magnetostatic fields with no dependency on $\boldsymbol{m}$, free from long-range interactions [10, 11]. For any irrotational field $\boldsymbol{h} \in L^2(\mathbb{R}^3)^3$ and solenoidal field $\boldsymbol{b} \in L^2(\mathbb{R}^3)^3$ holds

$$\int_\Omega \|\boldsymbol{m}\|^2 \,\mathrm{d}\boldsymbol{x} - \int_{\mathbb{R}^3} \|\boldsymbol{h}+\boldsymbol{m}\|^2 \,\mathrm{d}\boldsymbol{x} \;\le\; e_d(\boldsymbol{m}) \;\le\; \int_{\mathbb{R}^3} \|\boldsymbol{b}-\boldsymbol{m}\|^2 \,\mathrm{d}\boldsymbol{x}. \tag{5.13}$$

The upper bound is derived by adding $\int_{\mathbb{R}^3} \|\boldsymbol{b}_d - \boldsymbol{b}\|^2 \, dx$ to the energy, while the lower bound results from subtracting $\int_{\mathbb{R}^3} \|\boldsymbol{h}_d - \boldsymbol{h}\|^2 \, dx$ from the energy [3] and using $\boldsymbol{b}_d = \boldsymbol{m} + \boldsymbol{h}_d$ as well as the orthogonality of irrotational and solenoidal fields (5.4). In order to compute the energy as well as the respective stray field $\boldsymbol{h}_d$, one can maximize the lower bound with respect to a scalar potential ϕ for the ansatz $\boldsymbol{h} = -\nabla\phi$. Similarly, minimizing the upper bound with $\boldsymbol{b} = \nabla \times \boldsymbol{A}$ with a vector potential $\boldsymbol{A}$ gives the respective magnetic induction $\boldsymbol{b}_d$ in the optimum.

While those energy bounds have been shown to be applicable in a physics-informed setting [47] in 2d, in practice however, the integration over whole space in the bounds (5.13) causes problems. We will come back to this issue in Section 5.4. For now, we will use those bounds to derive some basic results.

5.2.1 Weak formulation

A variational form of the transmission problem (5.9) involving the scalar potential is derived by utilizing partial integration and the jump condition $[D_{\boldsymbol{n}}\phi_d] = -\boldsymbol{m}\cdot\boldsymbol{n}$ at the boundary:

$$\begin{aligned} &\text{For } \boldsymbol{m} \in \left(H^1(\Omega)\right)^3, \text{ find } \phi_d \in X_\phi \text{ such that for all } \phi \in X_\phi \text{ there holds} \\ &\int_{\mathbb{R}^3} \nabla\phi_d \cdot \nabla\phi \,\mathrm{d}\boldsymbol{x} = \int_\Omega \boldsymbol{m}\cdot\nabla\phi \,\mathrm{d}\boldsymbol{x}, \end{aligned} \tag{5.14}$$

where X_ϕ is a suitable weighted Sobolev space [75] which incorporates the continuity across the boundary and the decay behavior. There exists a unique solution of (5.14) in X_ϕ [75]. Solving the weak form (5.14) is equivalent to maximizing the following functional over X_ϕ

$$e_\phi(\phi;\boldsymbol{m}) = -\int_{\mathbb{R}^3} \|\nabla\phi\|^2 \,\mathrm{d}\boldsymbol{x} + 2\int_\Omega \boldsymbol{m}\cdot\nabla\phi \,\mathrm{d}\boldsymbol{x}. \tag{5.15}$$

The variational formulation for the vector potential is derived similarly, using the jump condition $[D_{\boldsymbol{n}}\boldsymbol{A}_d] = -\boldsymbol{m}\times\boldsymbol{n}$:

$$\begin{aligned}&\text{For } \boldsymbol{m}\in\left(H^1(\Omega)\right)^3, \text{ find } \boldsymbol{A}_d\in X_{\boldsymbol{A}} \text{ such that for all } \boldsymbol{A}\in X_{\boldsymbol{A}} \text{ there holds}\\ &\int_{\mathbb{R}^3}\nabla\boldsymbol{A}_d:\nabla\boldsymbol{A}\,\mathrm{d}\boldsymbol{x}=\int_{\Omega}\boldsymbol{m}\cdot(\nabla\times\boldsymbol{A})\,\mathrm{d}\boldsymbol{x},\end{aligned}\tag{5.16}$$

where $\nabla\boldsymbol{A}_d:\nabla\boldsymbol{A}=Tr[\nabla(\boldsymbol{A}_d)^T\nabla\boldsymbol{A}]$ denotes double contraction. Solving (5.16) is equivalent to minimizing

$$e_{\boldsymbol{A}}(\boldsymbol{A};\,\boldsymbol{m})=\int_{\Omega}\|\boldsymbol{m}\|^2\,\mathrm{d}\boldsymbol{x}+\int_{\mathbb{R}^3}\|\nabla\boldsymbol{A}\|^2\,dx-2\int_{\Omega}\boldsymbol{m}\cdot(\nabla\times\boldsymbol{A})\,\mathrm{d}\boldsymbol{x}\tag{5.17}$$

over $X_{\boldsymbol{A}}$, where $\int_{\Omega}\|\boldsymbol{m}\|^2\,\mathrm{d}\boldsymbol{x}$ is a constant and $\|\nabla\boldsymbol{A}\|^2=\nabla\boldsymbol{A}:\nabla\boldsymbol{A}$.

The energy functionals (5.15) and (5.17) correspond to the lower and upper bound of (5.13)

$$e_{\phi}(\phi;\,\boldsymbol{m})\le e_d(\boldsymbol{m})\le e_{\boldsymbol{A}}(\boldsymbol{A};\,\boldsymbol{m}).\tag{5.18}$$

In numerical computations, we do not actually solve the potential equations in the infinite dimensional spaces above, but rather assume a (finite dimensional) approximation space $\widetilde{X}_{\phi}\subset X_{\phi}$. Hence, assuming an exact solution $\widetilde{\phi}_d\in\widetilde{X}_{\phi}$ with $\widetilde{\boldsymbol{h}}_d=-\nabla\widetilde{\phi}_d$ of the corresponding weak form (5.14), that is, we solve the (Galerkin) problem:

$$\begin{aligned}&\text{for } \boldsymbol{m}\in\left(H^1(\Omega)\right)^3, \text{ find } \widetilde{\phi}_d\in\widetilde{X}_{\phi} \text{ such that for all } \phi\in\widetilde{X}_{\phi} \text{ there holds}\\ &\int_{\mathbb{R}^3}\nabla\widetilde{\phi}_d\cdot\nabla\phi\,\mathrm{d}\boldsymbol{x}=\int_{\Omega}\boldsymbol{m}\cdot\nabla\phi\,\mathrm{d}\boldsymbol{x}.\end{aligned}\tag{5.19}$$

Or, equivalently maximize the energy functional (5.15) over $\widetilde{X}_{\phi}$,

$$\widetilde{\phi}_d=\arg\max_{\phi\in\widetilde{X}_{\phi}}e_{\phi}(\phi;\,\boldsymbol{m})\tag{5.20}$$

Applying this solution to (5.15) gives

$$e_{\phi}(\widetilde{\phi}_d;\,\boldsymbol{m})=-\int_{\Omega}\boldsymbol{m}\cdot\widetilde{\boldsymbol{h}}_d\,\mathrm{d}\boldsymbol{x}.\tag{5.21}$$

Therefore, by using L^2-orthogonality (5.4) and the first relation in (5.3)

$$|e_d(\boldsymbol{m})-e_{\phi}(\widetilde{\phi}_d;\,\boldsymbol{m})|=|\int_{\Omega}\boldsymbol{m}\cdot(\boldsymbol{h}_d-\widetilde{\boldsymbol{h}}_d)\,\mathrm{d}\boldsymbol{x}|=|\int_{\mathbb{R}^3}\boldsymbol{h}_d\cdot(\boldsymbol{h}_d-\widetilde{\boldsymbol{h}}_d)\,\mathrm{d}\boldsymbol{x}|\tag{5.22}$$

and the Cauchy-Schwarz inequality, the error of the self energy can be bound by

$$\frac{|e_d(\boldsymbol{m})-e_{\phi}(\widetilde{\phi}_d;\,\boldsymbol{m})|}{\sqrt{e_d(\boldsymbol{m})}}\le\left(\int_{\mathbb{R}^3}\|\boldsymbol{h}_d-\widetilde{\boldsymbol{h}}_d\|^2\,\mathrm{d}\boldsymbol{x}\right)^{\frac{1}{2}}.\tag{5.23}$$

With the same line of reasoning, for an exact solution $\widetilde{\boldsymbol{A}}_d$ with $\widetilde{\boldsymbol{b}}_d = \nabla \times \widetilde{\boldsymbol{A}}_d$ of (5.16) on a finite dimensional subspace $\widetilde{X}_{\boldsymbol{A}} \subset X_{\boldsymbol{A}}$, one finds

$$\frac{|e_d(\boldsymbol{m}) - e_{\boldsymbol{A}}(\widetilde{\boldsymbol{A}}_d;\, \boldsymbol{m})|}{\sqrt{\int_\Omega |\boldsymbol{m}|^2 \,\mathrm{d}\boldsymbol{x} - e_d(\boldsymbol{m})}} \leq \left(\int_{\mathbb{R}^3} \|\boldsymbol{b}_d - \widetilde{\boldsymbol{b}}_d\|^2 \,\mathrm{d}\boldsymbol{x} \right)^{\frac{1}{2}}. \tag{5.24}$$

Therefore, the deviation of the energy values can be estimated by the L^2-error resulting from the respective field computation.

Further, since the exact solution on an approximate solution space of the weak form is in the infinite-dimensional spaces X_ϕ and $X_{\boldsymbol{A}}$ respectively, the self-energy for the scalar potential is underestimated while the energy for the vector potential is overestimated, i.e.,

$$-\int_\Omega \boldsymbol{m} \cdot \widetilde{\boldsymbol{h}}_d \,\mathrm{d}\boldsymbol{x} \leq e_\phi(\phi_d;\, \boldsymbol{m}) = e_d(\boldsymbol{m}) = e_{\boldsymbol{A}}(\boldsymbol{A}_d;\, \boldsymbol{m}) \leq \int_\Omega \|\boldsymbol{m}\|^2 - \boldsymbol{m} \cdot \widetilde{\boldsymbol{b}}_d \,\mathrm{d}\boldsymbol{x} \tag{5.25}$$

However, our assumption of an exact solution of the approximate formulation is not realistic in numerical computations in general, where the whole space problem (5.19) will hardly hold exactly, and likewise (5.21). Hence, (5.25) might be violated from both sides. Nevertheless, in this case where we assume only an approximation $\widetilde{\phi} \approx \widetilde{\phi}_d$ and $\widetilde{\boldsymbol{A}} \approx \widetilde{\boldsymbol{A}}_d$, the lower and upper bounds are still guaranteed to hold,

$$e_\phi(\widetilde{\phi};\, \boldsymbol{m}) \leq e_d(\boldsymbol{m}) \leq e_{\boldsymbol{A}}(\widetilde{\boldsymbol{A}};\, \boldsymbol{m}). \tag{5.26}$$

For the evaluation of the self-energy, it is necessary to compute the stray field or the magnetic induction within the magnetic domain. However, this requires a solution to the whole space problem (5.9). One could for instance use a truncation approach and minimize the upper bound in (5.13). In [47] this was done in 2d. In 3d, however, it turned out to be much more challenging, and we were not able to verify this approach. This could be due to the jump conditions which are not automatically satisfied. Especially the jump in the normal derivative can not be satisfied with a simple neural network ansatz. It might be possible if those conditions are incorporated into the minimization problem. Even though the resulting objective would not be a lower or upper bound, the optimization of this functional would still give the same result. An easier and more flexible approach is described in the following section.

5.3 Splitting ansatz of Garcia-Cervera and Roma

To find a solution to the whole space transmission problem (5.9), it can be expressed as two easier sub-problems which only require the solution within the magnetic domain. Therefore, the splitting ansatz of Garcia-Cervera and Roma is employed. It can be used for the scalar and the vector potential alike. For the scalar potential, it is split into two parts $\phi = \phi_1 + \phi_2$ with

$$\phi_1 = \begin{cases} \phi_1^{int} & \text{in } \Omega \\ \phi_1^{ext} & \text{in } \overline{\Omega}^c \end{cases}, \quad \phi_2 = \begin{cases} \phi_2^{int} & \text{in } \Omega \\ \phi_2^{ext} & \text{in } \overline{\Omega}^c \end{cases},$$

and $\phi_1^{int},\ \phi_2^{int} \in H^1(\Omega)$ and $\phi_1^{ext},\ \phi_2^{ext} \in H^1_{loc}(\overline{\Omega}^c)$.

The component ϕ_1 satisfies the homogeneous Dirichlet Poisson problem, i.e.

$$\begin{aligned} -\Delta\phi_1^{int} &= -\nabla\cdot\boldsymbol{m} && \text{in } \Omega, \\ \phi_1^{int} &= 0 && \text{on } \partial\Omega \\ \phi_1^{ext} &= 0 && \text{in } \overline{\Omega}^c, \end{aligned} \tag{5.27}$$

whereas the second component ϕ_1 fulfills the Laplace equation

$$\begin{aligned} -\Delta\phi_2^{int} &= 0 && \text{in } \Omega, \\ -\Delta\phi_2^{ext} &= 0 && \text{in } \overline{\Omega}^c, \\ [\phi_2] &= 0 && \text{on } \partial\Omega, \\ [D_{\boldsymbol{n}}\phi_2] &= -\boldsymbol{m}\cdot\boldsymbol{n} + D_{\boldsymbol{n}}\phi_1^{int} && \text{on } \partial\Omega, \\ \phi_2^{ext}(\boldsymbol{x}) &= \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) && \text{as } \|\boldsymbol{x}\| \to \infty. \end{aligned} \tag{5.28}$$

It is easy to verify that a solution to (5.27) and (5.28) satisfies the jump conditions in (5.9) since $[\phi_1] = 0$ and $[D_{\boldsymbol{n}}\phi_1] = -D_{\boldsymbol{n}}\phi_1^{int}$.

The solution of (5.28) can be expressed with the *single layer potential*

$$\phi_2^{ext}(\boldsymbol{x}) = \phi_\nu(\boldsymbol{x}; \phi_1, \boldsymbol{m}) := \frac{1}{4\pi}\int_{\partial\Omega} \frac{(\boldsymbol{m}\cdot\boldsymbol{n} - D_{\boldsymbol{n}}\phi_1^{int})(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}s(\boldsymbol{y}). \tag{5.29}$$

Likewise, the vector potential can be split in $\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2$ with

$$\begin{aligned} \Delta\boldsymbol{A}_1^{int} &= -\nabla\times\boldsymbol{m} && \text{in } \Omega, \\ \boldsymbol{A}_1^{int} &= 0 && \text{on } \partial\Omega \\ \boldsymbol{A}_1^{ext} &= 0 && \text{in } \overline{\Omega}^c, \end{aligned} \tag{5.30}$$

and

$$\begin{aligned} \Delta\boldsymbol{A}_2^{int} &= 0 && \text{in } \Omega, \\ \Delta\boldsymbol{A}_2^{ext} &= 0 && \text{in } \overline{\Omega}^c, \\ [\boldsymbol{A}_2] &= 0 && \text{on } \partial\Omega, \\ [D_{\boldsymbol{n}}\boldsymbol{A}_2] &= -\boldsymbol{m}\times\boldsymbol{n} + D_{\boldsymbol{n}}\boldsymbol{A}_1^{int} && \text{on } \partial\Omega, \\ (\boldsymbol{A}_2^{ext})_j(\boldsymbol{x}) &= \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right),\ j=1,2,3, && \text{as } \|\boldsymbol{x}\| \to \infty. \end{aligned} \tag{5.31}$$

The solution of $\boldsymbol{A}_2^{ext}$ is then given by

$$\boldsymbol{A}_2^{ext}(\boldsymbol{x}) = \boldsymbol{A}_\nu(\boldsymbol{x}; \boldsymbol{A}_1^{int}, \boldsymbol{m}) := \frac{1}{4\pi}\int_{\partial\Omega} \frac{(\boldsymbol{m}\times\boldsymbol{n} - D_{\boldsymbol{n}}\boldsymbol{A}_1^{int})(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}s(\boldsymbol{y}). \tag{5.32}$$

With the single layer potential, the solution of the exterior domain can be projected onto the boundary. Only the interior domain Ω needs to be considered and therefor, from now on, we dismiss the superscript *int* and *ext*.

To compute a solution to the stray field, one could proceed as follows. First, compute a solution ϕ_1 to (5.27). Second, use the retrieved solution to compute ϕ_2 with (5.29) and third, compute $\boldsymbol{h}_d = -\nabla(\phi_1 + \phi_2)$. The computation of the stray field with a hard constrained ELM ansatz is summarized in Algorithm 1. To

Algorithm 1 Computation of the stray field with a hard constrained ELM ansatz

Require: Magnetization $\boldsymbol{m}$, ADF $\ell : \mathbb{R}^3 \to \mathbb{R}$, ELM embedding $\boldsymbol{z} : \Omega \to \mathbb{R}^M$, collocation points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subset \Omega$ and Ridge regularization parameter μ

Phase 1 – Precomputation:
$A_{ij} = -\Delta\left(\ell(\boldsymbol{x}_i)\boldsymbol{z}(\boldsymbol{x}_i)_j\right)$ for $i = 1, \dots, N$ and $j = 1, \dots, M$
$USV^T = A$ ▷ compute truncated SVD
$A^\dagger = VS(S^2 + \mu I)^{-1}U^T$ ▷ compute solution operator for ϕ_1

Phase 2 – Computation of ϕ_1:
$\boldsymbol{b}_i = -\nabla \cdot \boldsymbol{m}(\boldsymbol{x}_i)$ for $i = 1, \dots, N$ ▷ compute right hand side
$\boldsymbol{\beta} = A^\dagger \boldsymbol{b}$
$\widetilde{\phi}_1(\,\cdot\,) = \boldsymbol{\beta}^T \ell(\,\cdot\,)\boldsymbol{z}(\,\cdot\,)$ ▷ approximate solution to ϕ_1

Phase 3 – Computation of ϕ_2:
$\widetilde{\phi}_2(\,\cdot\,) = \phi_\nu(\,\cdot\,; \widetilde{\phi}_1, \boldsymbol{m})$ ▷ ϕ_2 is given by the single layer potential (5.29)

Phase 4 – Automatic Differentiation:
$\widetilde{\boldsymbol{h}}_d = -\nabla(\widetilde{\phi}_1 + \widetilde{\phi}_2)$ ▷ compute stray field with forward mode AD

compute the magnetic induction, one would use (5.30) and (5.32) instead. If one uses a more complicated network structure than an ELM, the computation of ϕ_1 needs to be replaced with the respective minimization problem. Alternatively to solving the single layer potential, one could compute the Dirichlet boundary data to train a PINN or an ELM to solve the respective Laplace problem (5.28).

Section 3.2 explains in detail how to solve a problem like (5.27) with PINNs. Especially the hard constraint formulation $\phi_1(\boldsymbol{x}) = \ell(\boldsymbol{x})\hat{\phi}_1(\boldsymbol{x})$ (see section 3.2.2) is of interest. Here, ℓ is an Approximate Distance Function, normalized to first order, i.e., $-\nabla\ell|_{\partial\Omega} = \boldsymbol{n}|_{\partial\Omega}$ (see chapter 4.1), and $\hat{\phi}_1$ is a neural network model. With this ansatz, equation (5.29) simplifies to

$$\phi_2(\boldsymbol{x}) = \frac{1}{4\pi} \int_{\partial\Omega} \frac{(\boldsymbol{m} \cdot \boldsymbol{n} + \hat{\phi}_1)(\boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s(\boldsymbol{y}). \tag{5.33}$$

In this work, this integral is solved numerically. Numerical integration can either be performed with a Monte Carlo approach, as in [78], or with integration over surface elements forming a partition [76]. The latter method has the advantage of being more cost-efficient and accurate. Also, it allows for a pre-computation step, which significantly reduces computational work during the actual computation. The surface is partitioned into K surface elements $\partial\Omega_i$ for $i = 1, \dots, K$ and the surface charges are approximated with a Taylor expansion of order p around a point

$\bar{\boldsymbol{y}}_i \in \partial\Omega_i$, for each element $i = 1, \ldots, K$, i.e.,

$$f(\boldsymbol{x}; \bar{\boldsymbol{y}}_i) = \sum_{|\alpha| \leq p} \frac{D^\alpha f(\bar{\boldsymbol{y}}_i)}{\alpha!} (\boldsymbol{x} - \bar{\boldsymbol{y}}_i)^\alpha + \mathcal{O}(\|\boldsymbol{x} - \bar{\boldsymbol{y}}_i\|^{p+1}), \tag{5.34}$$

where α denotes the multi-index with $|\alpha| = \alpha_1 + \alpha_2 + \alpha_3$, $\alpha! = \alpha_1! \, \alpha_2! \, \alpha_3!$ and mixed derivatives

$$D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial \boldsymbol{x}_1^{\alpha_1} \partial \boldsymbol{x}_2^{\alpha_2} \partial \boldsymbol{x}_3^{\alpha_3}}. \tag{5.35}$$

Using this expansion for the single layer potential (5.33) with charge $f(\boldsymbol{y}) = \boldsymbol{m}(\boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y}) + \tilde{\phi}_1(\boldsymbol{y})$ results in

$$\phi_2(\boldsymbol{x}) = \int_{\partial\Omega} \frac{f(\boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s(\boldsymbol{y}) \approx \sum_{i=1}^{N} \sum_{|\alpha| \leq p} \frac{D^\alpha f(\bar{\boldsymbol{y}}_i)}{\alpha!} \int_{\partial\Omega_i} \frac{(\boldsymbol{y} - \bar{\boldsymbol{y}}_i)^\alpha}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s(\boldsymbol{y}) = \boldsymbol{F} : \boldsymbol{Z}(\boldsymbol{x}). \tag{5.36}$$

Let us refer to $\boldsymbol{F}$ as the *charge tensor* and $\boldsymbol{Z}$ as the *surface tensor*.

Both have shape of $K \times T$, where T is the number of terms in the Taylor series. For a fixed geometry, this representation has the advantage that $\boldsymbol{Z}$ can be pre-computed for each sample $\boldsymbol{x} \in \mathcal{S}$. The result is given by double tensor contraction, which can then be done in $\mathcal{O}(K)$. Since the stray field requires the computation of the gradient of ϕ_2, we can also pre-compute the Jacobian of the surface tensors $J\boldsymbol{Z}$ of size $(N \times T \times 3)$. The gradient is then given by $\nabla\phi_2(\boldsymbol{x}) = \boldsymbol{F} : J\boldsymbol{Z}(\boldsymbol{x})$. This can be done either analytically, but the more convenient way is to use automatic differentiation, which is of course also used to compute the partial derivatives of $\boldsymbol{F}$. In our implementation, we only use a second order Taylor series approximation, but higher order approximations can be derived the same way. The computational effort of repeated automatic differentiation grows exponentially $\mathcal{O}(2^p)$. Therefore, for higher order approximations, one can use Faà di Bruno's formula [16] to automatically compute higher order derivatives efficiently using `JAX`'s experimental `jet` feature [8]. This would reduce the computational complexity to $\mathcal{O}(p^2)$. However, for second order approximation, this does not lead to major improvements. The quality of the approximation depends on the accuracy of the Taylor series within each surface element, as well as the solution to ϕ_1. Storing this tensor may require a lot of memory, but handling large datasets is of key importance to machine learning applications and if memory is of concern, one can use out-of-core execution.

If the dataset $\mathcal{S}$ has a specific structure, e.g., lies on a tensor grid, this computation can be further accelerated by means of FFT or for non-rectangular geometries by *Non-Uniform-FFT* (NUFFT) [28]. Also, for rectangular geometries, one can use exponential sum approximation to reduce the computational cost [9, 20]. In our case, where the dataset is scattered across the domain, such accelerations are not easily available, although recent advances with hypernetworks [44] could be very

significant. With parallel computing, however, on modern GPUs, the evaluation is still very efficient if done naively. Especially, in the course of SGD it is only required to compute the stray field for a mini batch of the dataset, hence computational complexity is $\mathcal{O}(NB)$, where B is the batch size.

5.4 Evaluation of the energy bounds on the finite domain

For our numerical experiments, we are interested in the evaluation of the lower energy bound (5.15) and the upper energy bound (5.17). In principle, a truncation approach could be used to evaluate the respective whole space integrals, but this still requires a very big domain outside the magnet and further introduces a truncation error. In the previous section, we used the splitting ansatz of Garcia-Cervera and Roma to derive a solution to the whole space transmission problems (5.9) and (5.11). This splitting ansatz can also be used for the computation of the energy bounds without considering the external domain $\overline{\Omega}^c$.

Two important properties can be used to derive a version of the lower bound (5.13) on the finite domain:

Lemma 3 ([21]). *Let $\phi_1 \in H_0^1(\Omega)$ and $\phi_2 \in \left(H^1(\Omega) \times H^1_{loc}(\overline{\Omega}^c)\right)$ solving the Laplace equation. Then we have*
(i) The gradients $\nabla\phi_1$ and $\nabla\phi_2$ are L^2-orthogonal, that is,

$$\int_\Omega \nabla\phi_1 \cdot \nabla\phi_2 \,\mathrm{d}\boldsymbol{x} = 0. \tag{5.37}$$

(ii) Let ϕ_2 be as in (5.29), then the L^2-norm of $\nabla\phi_2$ can be expressed as a double surface integral

$$\int_{\mathbb{R}^3} \|\nabla\phi_2\|^2 \,\mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} (\boldsymbol{m}\cdot\boldsymbol{n} - D_{\boldsymbol{n}}\phi_1)\phi_2 \,\mathrm{d}s(\boldsymbol{x}). \tag{5.38}$$

□

Using the splitting of *Garcia-Cervera and Roma* $\phi = \phi_1 + \phi_2$ with $\phi_1 \in H_0^1(\Omega)$ and $\phi_2 = \phi_\nu(\boldsymbol{x}; \phi_1, \boldsymbol{m})$ being the single layer potential (5.29), as well as property (i) and (ii) from Lemma 3 and applying it to the energy functional (5.15) yields

$$e_\phi(\phi_1; \boldsymbol{m}) = -\int_\Omega \|\nabla\phi_1\|^2 \,\mathrm{d}\boldsymbol{x} + 2\int_\Omega \boldsymbol{m}\cdot(\nabla\phi_1 + \nabla\phi_2)\,\mathrm{d}\boldsymbol{x} - \int_{\partial\Omega}(\boldsymbol{m}\cdot\boldsymbol{n} - D_{\boldsymbol{n}}\phi_1)\cdot\phi_2 \,\mathrm{d}s(\boldsymbol{x}). \tag{5.39}$$

Note that ϕ_1 does not satisfy the Dirichlet Poisson problem (5.27) here, but is a free function. The surface integral can be transformed into a volume integral using the divergence theorem, together with property (i),

$$\int_{\partial\Omega}(\boldsymbol{m}\cdot\boldsymbol{n} - D_{\boldsymbol{n}}\phi_1)\cdot\phi_2 \,\mathrm{d}s(\boldsymbol{x}) = \int_\Omega (\nabla\cdot\boldsymbol{m})\phi_2 + \boldsymbol{m}\cdot\nabla\phi_2 - (\Delta\phi_1)\phi_2 \,\mathrm{d}\boldsymbol{x} \tag{5.40}$$

Lemma 3 can be restated for the vector potential, consequently, the upper bound (5.17), with $\boldsymbol{A}_2 = \boldsymbol{A}_\nu(\boldsymbol{x};\, \boldsymbol{A}_1, \boldsymbol{m})$ as in (5.32), can be written as

$$\begin{aligned} e_{\boldsymbol{A}}(\boldsymbol{A}_1; \boldsymbol{m}) \;=& \int_\Omega \|\boldsymbol{m}\|^2 \,\mathrm{d}\boldsymbol{x} + \int_\Omega |\nabla \boldsymbol{A}_1|^2 \,\mathrm{d}\boldsymbol{x} - 2\int_\Omega \boldsymbol{m} \cdot (\nabla \times (\boldsymbol{A}_1 + \boldsymbol{A}_2))\,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\partial\Omega} (\boldsymbol{m} \times \boldsymbol{n} - D_{\boldsymbol{n}} \boldsymbol{A}_1) \cdot \boldsymbol{A}_2 \,\mathrm{d}s(\boldsymbol{x}). \end{aligned} \tag{5.41}$$

Again, it is possible to write the surface integral as volume integral like

$$\int_{\partial\Omega} (\boldsymbol{m}\times\boldsymbol{n} - D_{\boldsymbol{n}}\boldsymbol{A}_1)\cdot\boldsymbol{A}_2 \,\mathrm{d}s(\boldsymbol{x}) = -\int_\Omega (\nabla\times\boldsymbol{m})\cdot\boldsymbol{A}_2 - \boldsymbol{m}\cdot\nabla\times\boldsymbol{A}_2 + (\Delta\boldsymbol{A}_1)\cdot\boldsymbol{A}_2 \,\mathrm{d}\boldsymbol{x}. \tag{5.42}$$

Especially for numerical integration of the single layer potential it is useful not to integrate over the surface, but the volume instead since singularities are easier to control.

Further, for the unique solution of (5.27), $\phi_d = \phi_1 + \phi_2$ defines the unique solution of (5.9). The field $-\nabla(\phi_1 + \phi_2)$ defines the stray field $\boldsymbol{h}_d$, where the energy $e_d(\boldsymbol{m}) = -\int_\Omega \boldsymbol{m} \cdot \boldsymbol{h}_d \,\mathrm{d}\boldsymbol{x}$ can also be expressed as [21]

$$e_d(\boldsymbol{m}) = \int_\Omega \|\nabla\phi_1\|^2 \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} (\boldsymbol{m}\cdot\boldsymbol{n} - D_{\boldsymbol{n}}\phi_1)\cdot\phi_2 \,\mathrm{d}s(\boldsymbol{y}). \tag{5.43}$$

5.5 Results

This section includes results of scalar and vector potential for simple domains, and also computes the energy bounds (5.15) and (5.17), with the respective formulas, (5.39) and (5.41), introduced in this section. Algorithm 1 is used for the computation of the scalar potential and the respective stray field. An analogous version of the algorithm is used for the vector potential. Further, in Section 5.5.2, ϕ_1 and $\boldsymbol{A}_1$ are also modelled via PINNs to have a comparison to the ELM solution.

5.5.1 Outward magnetized sphere

As a first test for the computation of the micromagnetic self energy and Algorithm 1, we considered a spherical magnetic particle with radius $R = 1$ and outward facing magnetization $\boldsymbol{m}(\boldsymbol{x}) = \boldsymbol{x}/\|\boldsymbol{x}\|$ centered at the origin. The exact solution to the scalar potential is given by

$$\phi_d = \begin{cases} \|\boldsymbol{x}\| - 1 & \text{in } \Omega \\ 0 & \text{in } \overline{\Omega}^c. \end{cases} \tag{5.44}$$

For this example, the scalar potential is zero at the boundary, hence $\phi_d = \bar{\phi}_1$ and $\bar{\phi}_2 = 0$. The energy is therefore $e_d = \frac{4}{3}\pi$. However, ϕ_2 was also included in the numerical test since errors in ϕ_1 will propagate. The solution of ϕ_1 was computed with a hard constraint ELM with the ADF $\ell(\boldsymbol{x}) = (R^2 - \|\boldsymbol{x}\|_2^2)/2R)$, i.e.,

$$\widetilde{\phi}_1(\boldsymbol{x};\, \boldsymbol{\beta}) = \ell(\boldsymbol{x})\hat{\phi}_1(\boldsymbol{x};\, \boldsymbol{\beta}), \tag{5.45}$$

where $\hat{\phi}_1$ is an ELM model. Therefore, the full model for the scalar potential is given by

$$\phi_{ELM}(\boldsymbol{x};\, \boldsymbol{\beta}) = \widetilde{\phi}_1(\boldsymbol{x};\, \boldsymbol{\beta}) + \phi_\nu(\boldsymbol{x}\ \widetilde{\phi}_1(\,\cdot\,;\, \boldsymbol{\beta}), \boldsymbol{m}) \tag{5.46}$$

The stray field was then computed with AD in forward mode

$$\boldsymbol{h}_{ELM}(\boldsymbol{x};\, \boldsymbol{\beta}) = -\nabla \phi_{ELM}(\boldsymbol{x};\, \boldsymbol{\beta}). \tag{5.47}$$

From this PINN solution and the exact solution, the error of the scalar potential and the stray field was computed.

To compute the single layer potential, the surface of the sphere was parameterized by

$$s(\varphi, \vartheta) = R \begin{pmatrix} \sin(\varphi)\cos(\vartheta) \\ \sin(\varphi)\sin(\vartheta) \\ \cos(\varphi) \end{pmatrix} \quad \varphi \in [0, \pi],\ \vartheta \in [0, 2\pi]. \tag{5.48}$$

The input space $[0, \pi] \times [0, 2\pi]$ was partitioned into 8×8 equal spaced tiles and mapped onto the surface of the sphere. For each sample point of the domain, the *surface tensor* was pre-computed with a 4^{th} order Gauss–Legendre quadrature and the *charge tensor* was evaluated for the center points of the tiles in the parameter space $[0, \pi] \times [0, 2\pi]$. A second order Taylor expansion was used for the computation of the *source tensor*. The training data was drawn from a Halton sequence [36]. In total, there were 4096 collocation points which are uniformly drawn from $[0, 1]^3$ and uniformly mapped onto the sphere with the parametrization

$$z(r, \varphi, \vartheta) = r \begin{pmatrix} \cos(\vartheta)\sqrt{1-\varphi^2} \\ \sin(\vartheta)\sqrt{1-\varphi^2} \\ \varphi \end{pmatrix}, \quad \varphi = 2\boldsymbol{x}_1 - 1,\ \vartheta = 2\pi\boldsymbol{x}_2,\ r = R\sqrt[3]{\boldsymbol{x}_3},\ \boldsymbol{x} \in [0, 1]^3. \tag{5.49}$$

The ELM solution was computed for a variable hidden layer size $M \in \{8, 16, 32, 64, 128, 256, 512, 1024\}$. The ELM weights were drawn with a Halton sequence from $[-2, 2]^4$, where the 4^{th} dimension stems from the bias term of the ELM. A `gelu` activation function was used for all ELMs. Figure 5.1 shows the solution of the scalar potential as well as the absolute error of an ELM with 256 hidden nodes. It can be seen that the ELM has problems to approximate the solution at the center, since the solution is not smooth at the origin, but the model is. This is a general issue of the continuous PINN ansatz. The error plots for a varying number of hidden nodes are shown in Figure 5.2. Also, the relative error of the energy is shown. The magnetostatic energy was computed as in (5.5) with

$$e_d^{\phi}(\boldsymbol{\beta}) = -\int_\Omega \boldsymbol{m}(\boldsymbol{x}) \cdot \boldsymbol{h}_{ELM}(x;\, \boldsymbol{\beta})\, \mathrm{d}\boldsymbol{x}, \tag{5.50}$$

on an independent validation data set of the same size as the training data set. It was drawn from a Halton Sequence as well. It can be seen that the error estimates (5.23) hold true.

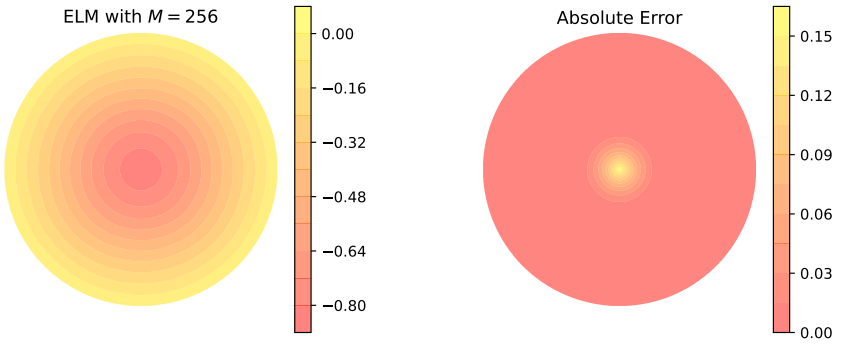


Figure 5.1: Solution in the $x-y$ plane of the scalar potential for an outward magnetized sphere, computed with an ELM with 256 hidden nodes and `gelu` activation function, together with the absolute error of the solution.

5.5.2 Flower state

The second test for the computation of the scalar potential, as well as the vector potential, was a flower state in a unit cube $[-0.5, 0.5]^3$. The magnetization was given by the normalized version of

$$\boldsymbol{m}(\boldsymbol{x}) = \left(\boldsymbol{x}_1\boldsymbol{x}_3, \boldsymbol{x}_2\boldsymbol{x}_3 + (\frac{1}{2}\boldsymbol{x}_2\boldsymbol{x}_3)^3, 1\right)^T \tag{5.51}$$

For the hard constraint model, the ADF

$$\ell(\boldsymbol{x}) = \hat{\ell}(\boldsymbol{x}_1) \sim_0 \hat{\ell}(\boldsymbol{x}_2) \sim_0 \hat{\ell}(\boldsymbol{x}_3), \quad \hat{\ell}(x) = (x + 0.5) \sim_0 (0.5 - x) \tag{5.52}$$

was used, where $\sim_0$ is the R_0-equivalence operation (see Equation 4.13 for comparison). Again, a training set with $N = 2^{12}$ was drawn from a Halton sequence. The ELM weights were also drawn from a Halton sequence from $[-2, 2]^4$ and a `tanh` activation function was used for the hidden layer. Further, an ℓ_2-regularization term of $\mu = 10^{-3}$ was added to the ELM (see section 3.3). Each face of the surface is discretized with 6×6 equal sized tiles, and a 10^{th} order Gauss–Legendre quadrature rule was used to evaluate the single layer potential. The Taylor expansion was again of second order. By *Monte Carlo* integration, the magnetostatic energy (5.5) was computed with an independent validation set with 2^{13} samples. Also, the lower bounds (5.39) and upper bounds (5.41) were computed. The double surface integrals were evaluated with (5.40) and (5.42). The results are shown in Figure 5.3. The computed solution of the energy with respect to the scalar potential is denoted with e_d^{ϕ} and with respect to the vector potential with $e_d^{\boldsymbol{A}}$. The reference solution is $0.1528[\mu_0 M s^2]$, which was computed with the demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2]. The upper bound is slightly lower than the reference solution. This error originates from numerical integration. It can be seen

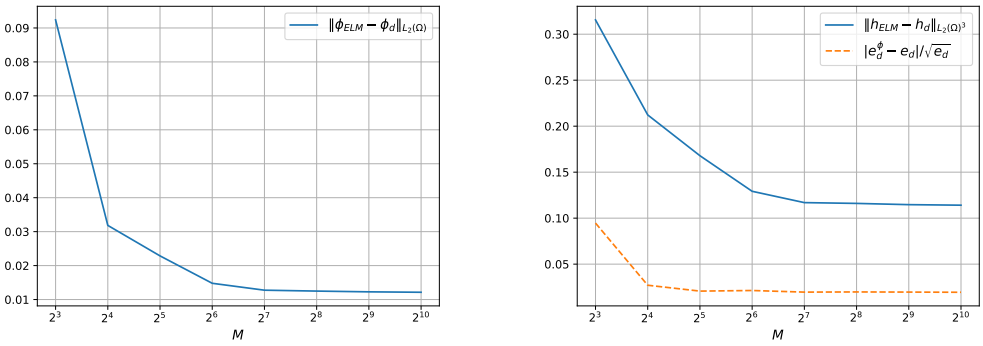


Figure 5.2: Error plots of the scalar potential for an outward magnetized sphere, computed with an ELM with `gelu` activation function for a varying number of hidden nodes.

that the bounds and the respective solution converge to the same value, hence the bounds, especially the lower bound, might be used as a validation metric for the model. If the difference between the bound and the energy is too large, the model performance is likely to be bad.

As a further test, the solution was also approximated with a hard constraint PINN ansatz for the scalar potential as well as for the vector potential. The network structure given in Table 5.1. Figure 5.4 shows the energies and energy bounds during

Layer	Activation	Input shape	Output shape
Dense	`gelu`	3	16
Dense	`gelu`	16	16
Dense	`identity`	16	t

Table 5.1: PINN architecture for the flower state model. The output dimension was $t = 1$ for the scalar potential and $t = 3$ for the vector potential.

the training process. A trust region optimizer with Steihaug-CG as a subproblem solver was used for training (see Appendix B.

Table 5.2 shows the computed self energies and energy bounds of the final model of the PINNs and the largest ELMs with $M = 1024$, i.e., the rightmost values of Figure 5.3 and Figure 5.4. It can be seen that the ELM and PINN both compute the solution very accurately, but training time for the ELM was only a fraction of a second, while for the PINN it takes minutes. However, this strongly depends on the optimizer and the required accuracy.

5.5.3 Vortex state

For this test, a vortex state within a cubic domain is solved via vector potential. The scalar potential is not considered since it has a trivial solution $\phi_1 = 0$. The

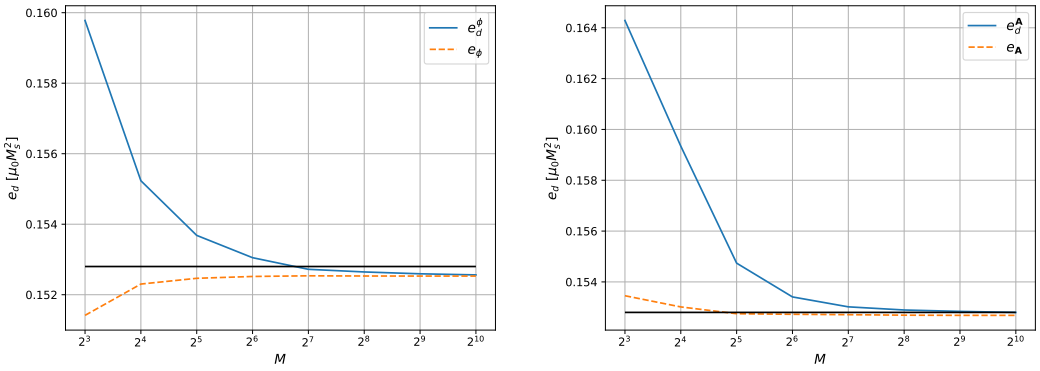


Figure 5.3: Magnetostatic energy and energy bounds (5.39) and (5.41) for a flower state computed with an ELM with a variable number of hidden nodes, and `tanh` activation. The energy was computed with the scalar potential (left) and with the vector potential (right). The horizontal line marks the reference solution of $0.1528[\mu_0 M_s^2]$ computed with the demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2].

	e_d^ϕ	$e_d^{\boldsymbol{A}}$	e_ϕ	$e_{\boldsymbol{A}}$
ELM $M = 1024$	0.152 56	0.152 90	0.152 53	0.152 68
PINN	0.152 49	0.152 87	0.152 56	0.152 73

Table 5.2: Computed self energies and energy bounds of a flower state in a cube for a hard constraint ELM and a hard constraint PINN model. The reference solution is $0.1528[\mu_0 M_s^2]$, computed with a demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2].

magnetization is given by the normalized version of

$$\boldsymbol{m}_v(\boldsymbol{x}) = \left(-\frac{\boldsymbol{x}_3}{r}\sqrt{1-\exp\left(-4\frac{r^2}{r_c^2}\right)}, \exp\left(-2\frac{r^2}{r_c^2}\right), \frac{\boldsymbol{x}_1}{r}\sqrt{1-\exp\left(-4\frac{r^2}{r_c^2}\right)} \right)^T, \quad (5.53)$$

with $r = \sqrt{\boldsymbol{x}_1^2 + \boldsymbol{x}_3^2}$ and $r_c = 0.14$. The domain, ADF, evaluation of the single layer potential and the sample set is the same as for the flower state. A hard constrained ELM with `rbf` activation $\sigma_i(\boldsymbol{x}) = \exp(-\gamma\|\boldsymbol{x} - \boldsymbol{w}_i\|^2)$ with $\boldsymbol{w}_i \in \Omega$ and $\gamma = 10$ was solved for a variable number of hidden nodes. The reference solution is $0.0219[\mu_0 M s^2]$, which was computed with the demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2]. Figure 5.5 shows the result of this computation. The energy for the ELM with 2^{10} hidden nodes was $e_d^{\boldsymbol{A}} = 0.0237[\mu_0 M_s^2]$ and the upper bound $e_{\boldsymbol{A}} = 0.0229[\mu_0 M_s^2]$. This problem is quite difficult to solve and we were not able to solve this problem with other ELM activations. This indicates that `rbf` activations are very powerful function approximators, however, for easy functions other choices also work increadible well. Previous results for energy minimization in

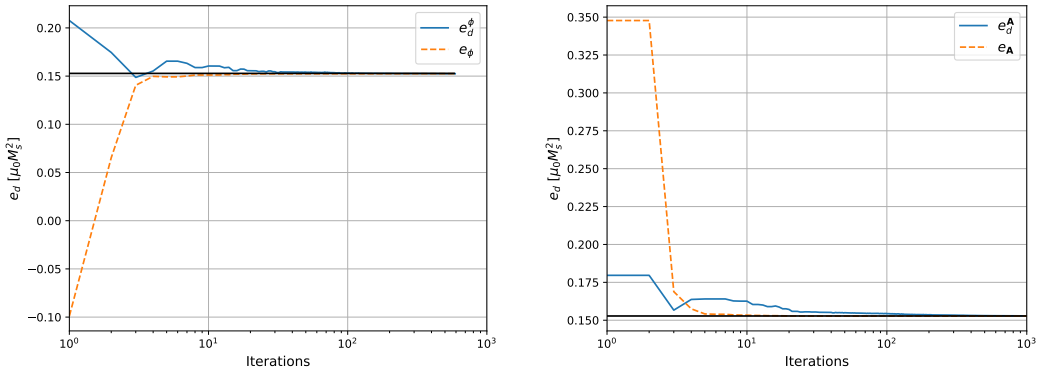


Figure 5.4: Magnetostatic energy and energy bounds (5.39) and (5.41) for a flower state computed with a hard constraint PINN. A trust region optimizer was used, and the curves display the energies and energy bounds during the training process. The network architecture is given in Table 5.1. The energy was computed with the scalar potential (left) and with the vector potential (right). The horizontal line marks the reference solution of $0.1528[\mu_0 M_s^2]$ computed with the demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2].

Chapter 6 were still achieved with `tanh` activations.

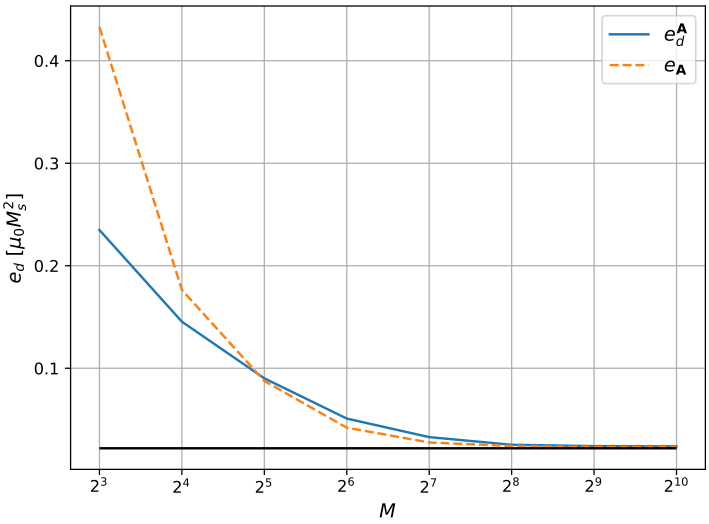


Figure 5.5: Magnetostatic energy and the upper energy bound (5.41) computed with an ELM via vector potential with a variable number of hidden nodes, and `RBF` activation, where the kernel parameter $\gamma = 10$ was used. The horizontal line marks the reference solution of $0.0219[\mu_0 M_s^2]$ computed with the demagnetization tensor method with a discretization of $40 \times 40 \times 40$ cells [2]. The energy for the ELM with 2^{10} hidden nodes was $e_d^{\boldsymbol{A}} = 0.0237[\mu_0 M_s^2]$ and the upper bound $e_{\boldsymbol{A}} = 0.0229[\mu_0 M_s^2]$.

6 Micromagnetic Energy Minimization

Parts of this chapter were previously published or submitted for publication as [78, 76, 79]. Permission for use has been granted by the co-authors.

Micromagnetic energy minimization aims to minimize the Gibbs free energy [29] of the magnet $\Omega \subset \mathbb{R}^3$ which is given in reduced units as

$$e = \frac{1}{2} e_d + \underbrace{\int_\Omega -\boldsymbol{m} \cdot \boldsymbol{h}_{ext} \mathrm{d}\boldsymbol{x}}_{e_{zee}} + \underbrace{\int_\Omega Q(1 - (\boldsymbol{m} \cdot \boldsymbol{a})^2) \mathrm{d}\boldsymbol{x}}_{e_a} + \underbrace{\int_\Omega \widetilde{A}_{ex} \|\nabla \boldsymbol{m}\|_F^2 \mathrm{d}\boldsymbol{x}}_{e_{ex}}, \tag{6.1}$$

with the four fundamental energy terms: magnetostatic energy e_d as in (5.5), Zeeman energy e_{zee}, anisotropy energy e_a and exchange energy e_{ex}. The reduced energy has units $[e] = \mathrm{m}^3$. The energy functional in reduced units is derived by dividing the Gibbs free energy with the energy density $K_m := \mu_0 M_s^2$ ($[\mu_0 M_s^2] = \mathrm{J/m^3}$). The quantity $\mu_0 = 4\pi 10^{-7}\,\mathrm{Tm/A}$ is the vacuum permeability and M_s (in units of A/m) is the saturation magnetization. Therefore, the reduced uniaxial anisotropy constant is denoted with $Q := K_u / K_m$ with K_u being the dimensionful uniaxial anisotropy constant and $\boldsymbol{a}$ is the easy axis of the magnet. Further, $\widetilde{A}_{ex} := A_{ex}/Km$ is the reduced exchange stiffness constant. One can for instance apply *LaBonte's method* to minimize the energy and perform a curvilinear steepest descent optimization with the projected gradient of the energy to find a local minimum [23], or variants of nonlinear conjugate gradient methods [30, 24]. For a nonlinear PINN ansatz, optimization is more intricate and one encounters similar problems as described in Section 3.4.

6.1 Optimization of the Gibbs free energy functional

Let the magnetization be parameterized by the model $\widetilde{\boldsymbol{m}}(\,\cdot\,;\boldsymbol{\theta})$, it is possible to use AD to compute the gradient $\nabla_{\boldsymbol{\theta}} e$ of the energy functional with respect to the model parameters $\boldsymbol{\theta}$ and use (stochastic) gradient descent to update the model parameters. Also higher order methods can be applied. However, two major problems arise. The first problem is the satisfaction of the unit norm constraint, $\|\boldsymbol{m}\|_2 = 1$, which will be explained later on. The second one arises from the non-local interaction due to the stray field.

When approximating the integral in (6.1), the integrand needs to be computed pointwise. The stray field is a function of the magnetization, $\boldsymbol{h}_d = \boldsymbol{h}_d(\,\cdot\,;\widetilde{\boldsymbol{m}},\boldsymbol{\theta})$,

thus, computing the self energy for instance with

$$e_d(\boldsymbol{\theta}) = -\int_\Omega \widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta}) \cdot \boldsymbol{h}_d(\boldsymbol{x};\widetilde{\boldsymbol{m}},\boldsymbol{\theta})\mathrm{d}\boldsymbol{x} \tag{6.2}$$

requires AD through the stray field solver. This can be done with ease with modern AD libraries but comes with a big computational burden. Using the splitting ansatz of Garcia-Cervera and Roma in Section 5.3, $\boldsymbol{h}_d = -\nabla(\phi_1 + \phi_2)$, and for instance a PINN to compute the approximate solution to $\widetilde{\phi}_1$ and the single layer potential (5.29) $\widetilde{\phi}_2 = \phi_\nu(\,\cdot\,;\widetilde{\phi}_1,\widetilde{\boldsymbol{m}})$ makes AD very expensive and prohibitive. If $\widetilde{\phi}_1$ is computed via an optimization process, it would require implicit AD [7] of the solver. Further, AD of the single layer potential requires AD of the integral operator, which is expensive for large domains or grain structure models. To our luck, it is not necessary to compute the gradient of the stray field with respect to the model parameters $\boldsymbol{\theta}$, since the stray field is a linear function of the magnetization, $\boldsymbol{h}_d(\,\cdot\,;\widetilde{\boldsymbol{m}},\boldsymbol{\theta}) = \boldsymbol{D}(\,\cdot\,)\widetilde{\boldsymbol{m}}(\,\cdot\,;\boldsymbol{\theta})$, with $\boldsymbol{D}$ being the constant demagnetization tensor which is only dependent on the geometry. Hence, the partial derivative of e_d with respect to some parameter of the model $\boldsymbol{\theta}_i$ is given by

$$\frac{\partial e_d}{\partial \boldsymbol{\theta}_i} = -\frac{\partial}{\partial \boldsymbol{\theta}_i}\int_\Omega \widetilde{\boldsymbol{m}} \cdot \boldsymbol{D}\widetilde{\boldsymbol{m}}\,\mathrm{d}\boldsymbol{x} = -2\int_\Omega \boldsymbol{h}_d \frac{\partial \widetilde{\boldsymbol{m}}}{\partial \boldsymbol{\theta}_i}\,\mathrm{d}\boldsymbol{x}. \tag{6.3}$$

Using a first order optimization method such as SGD, only the gradient is required to update the model parameters. Therefore, one can compute the stray field pointwise, before evaluating the gradient $\nabla_{\boldsymbol{\theta}}(e_d + e_{zee} + e_a + e_{ex})$. Note that the factor $\frac{1}{2}$ is missing before the self energy term.This approach results in Algorithm 2, where

$$q(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\theta}) = -\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\cdot\boldsymbol{h} - \widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\cdot\boldsymbol{h}_{ext}(\boldsymbol{x}) + Q(1-(\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\cdot\boldsymbol{a})^2) + \widetilde{A}_{ex}\|\nabla\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\|_F^2 \tag{6.4}$$

is the corresponding integrand of $e_d + e_{zee} + e_a + e_{ex}$, and $\boldsymbol{h} \in \mathbb{R}^3$ is the pointwise stray field. All other terms of (6.1) can be computed pointwise using automatic differentiation. All this also holds for the vector potential and the magnetic flux density $\boldsymbol{b}_d$. The algorithm can be formulated analogous.

This algorithm also perfectly fits into a conditional framework, where the magnetization model includes additional conditional parameters. They can be incorporated into the model by appending them to the input vector of the neural network [48]. In that case, one can draw a full batch or minibatch of the domain Ω as well as a minibatch of the conditional domain and compute the stray field for each conditional sample. The ADF which is used for computation of the scalar potential can also be parameterized, accounting for different geometries. However, one should note that the sample space grows exponentially with the dimension and therefore, the computational load of evaluating (6.1) over all conditional parameters also grows exponentially. Note that the computation of the magnetostatic fields can also be done with any other method. For instance, *MagTense* [6] is one such framework.

Algorithm 2 Energy minimization with first order methods

Require: Iteration limit $N_{\boldsymbol{m}}$

for $1, \ldots, N_{\boldsymbol{m}}$ **do**

1. Draw batch $\boldsymbol{X} \in \Omega$ from the computational domain
2. Compute $\boldsymbol{h}_{d,i} = \boldsymbol{h}_d(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}$. ▷ e.g. with Algorithm 1
3. Compute updates using AD

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{|\boldsymbol{X}|} \sum_{\boldsymbol{x}_i \in \boldsymbol{X}} q(\boldsymbol{x}_i; \boldsymbol{h}_{d,i}, \boldsymbol{\theta}) \right)$$

4. Update the parameters $\boldsymbol{\theta}$ with $\boldsymbol{g}$ using some first order optimizer

end for

The remaining problem of Algorithm 2 is that it does not deal with the satisfaction of the unit norm constraint. In the following, two different approaches are introduced. The first one uses a penalty framework and therefore changes the algorithmic framework, while the second one imposes the constraints on the model level and can apply Algorithm 2 as given.

6.1.1 Penalty framework

A penalty framework can be used to account for the unit norm constraint [78], i.e.,

$$e_\alpha(\boldsymbol{\theta}) = e(\boldsymbol{\theta}) + \alpha \int_\theta (\|\boldsymbol{m}(\boldsymbol{x}; \boldsymbol{\theta})\| - 1)^2 \, \mathrm{d}\boldsymbol{x} \to \min_{\boldsymbol{\theta}} \tag{6.5}$$

with the penalty parameter α. Starting with a low value for the penalty parameter, after minimization of the objective, it is increased for each consecutive optimization. The constrained is fulfilled if $\alpha \to \infty$. However, this caused the problem to be more ill-conditioned [61]. Therefore, one needs to stop the algorithm if the penalty parameter becomes prohibitively large. Algorithm 3 gives a pseudocode for such a framework. It builds on Algorithm 2 and adds an outer loop to control the size of the penalty parameter.

6.1.2 Cayley transform

Another possibility to account for the unit norm constraint is the utilization of the Cayley-transform [50]. This assures that the model output is given on the Lie algebra of the $SO(3)$ rotation group, and this rotation can be applied to some initial magnetization $\boldsymbol{m}_0$ [76]. More formally, consider the isomorphism between $\mathbb{R}^3$ and the Lie algebra $so(3)$ of 3×3 skew symmetric matrices

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \mapsto \hat{\boldsymbol{p}} = \begin{pmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{pmatrix}, \tag{6.6}$$

Algorithm 3 Penalty framework for Gibbs free energy minimization

Require: Iteration limits N_α, $N_{\boldsymbol{m}}$; initial penalty α_0; increase factor $\zeta > 1$

$\alpha \leftarrow \alpha_0$

for $1, \dots, N_\alpha$ **do**

for $1, \dots, N_{\boldsymbol{m}}$ **do**

1. Draw batch $\boldsymbol{X} \in \Omega$ from the computational domain
2. Compute $\boldsymbol{h}_{d,i} = \boldsymbol{h}_d(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}$. ▷ e.g. with Algorithm 1
3. Compute updates of penalized objective using AD

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{|\boldsymbol{X}|} \sum_{\boldsymbol{x}_i \in \boldsymbol{X}} q(\boldsymbol{x}_i; \boldsymbol{h}_{d,i}, \boldsymbol{\theta}) + \alpha (\|\widetilde{\boldsymbol{m}}(\boldsymbol{x}_i; \boldsymbol{\theta})\| - 1)^2 \right)$$

4. Update the parameters $\boldsymbol{\theta}$ with $\boldsymbol{g}$ using some first order optimizer.

end for

$\alpha \leftarrow \zeta \alpha$ ▷ increase penalty parameter

end for

and use the Cayley transform

$$\operatorname{cay}(\boldsymbol{p}) = (I - \hat{\boldsymbol{p}})^{-1}(I + \hat{\boldsymbol{p}}) \tag{6.7}$$

to get the respective rotation. Given some input $\boldsymbol{x} \in \Omega$, a neural network $\mathcal{N} : \Omega \times \mathbb{R}^M \to \mathbb{R}^3, (\boldsymbol{x}, \boldsymbol{\theta}) \mapsto \mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^M$ and some initial magnetization $\boldsymbol{m}_0$, the hard constraint model takes the form

$$\widetilde{\boldsymbol{m}}(\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{cay}\left(\mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta})\right) \boldsymbol{m}_0(\boldsymbol{x}). \tag{6.8}$$

This model architecture will preserve the magnitude of the initial magnetization and allows energy minimization of (6.1) using Algorithm 2 without the need of any framework for constrained optimization.

6.2 Optimization of the energy bounds on the finite domain

Up to this point, only first order optimization method were considered. If using second order optimizers, not only the gradient but also the objective value is important, and they need to be consistent. This can be problematic for implementation of second order methods, since one need to consider the objective $e = \frac{1}{2}e_d + e_{zee} + e_a + e_{ex}$ but with gradients given by $\nabla_{\boldsymbol{\theta}}(e_d + e_{zee} + e_a + e_{ex})$, assuming the stray field is independent of the magnetization and the gradient of the self energy is given by (6.3). In a first version, it was thought that the objective $e_d + e_{zee} + e_a + e_{ex}$ can replace e and AD of the stray field can simply be prevented on a code level with the `JAX` directive `stop_gradient`. However, this does not work for second order methods, since the gradient corresponds to e but the value does not. This issue can probably easily be resolved on a code level, but the study on energy bounds opened

a new viewpoint. It turns out, that the minimization of $\bar{e} = e_d + e_{zee} + e_a + e_{ex}$, without the factor $\frac{1}{2}$ for the self energy, corresponds to minimization of the lower bound e_ϕ in (5.15) with respect to the magnetization. In essence, this means that one can consider the value of $\bar{e}$ instead of e. Therefore, in this section, the joint optimization of $\boldsymbol{m}$ and the energy bounds is explained and results in a sophisticated alternating minimization algorithm.

Equation (5.39) and (5.41) give a computable formulation of the energy bounds on the finite domain and could be tackled, for instance, with Monte Carlo integration. However, we were not able to use those bounds directly for energy minimization in a physics-informed setting. It would require AD over the single layer potential, and minimization of the energy functional with PINNs does not seem to be numerically sound.

When training a physics-informed model like a PINN, in essence, we are only interested in the updates of the model parameters. Considering again the lower bound (5.15) of the self energy

$$e_\phi(\phi;\, \boldsymbol{m}) = -\int_{\mathbb{R}^3} \|\nabla\phi\|^2 \,\mathrm{d}\boldsymbol{x} + 2\int_\Omega \boldsymbol{m}\cdot\nabla\phi \,\mathrm{d}\boldsymbol{x}.$$

The goal is the maximization of this functional with respect to the scalar potential ϕ, but minimize it with respect to the magnetization $\boldsymbol{m}$. Hence, this leads to a min-max problem. Computing the variational derivative with a variation $\delta\phi$ with $[\delta\phi] = 0$ across the boundary $\partial\Omega$ yields

$$\begin{aligned}\mathrm{d}_\phi e_\phi(\phi;\, \boldsymbol{m}) = 2\Big(&\int_{\mathbb{R}^3} (\Delta\phi)\delta\phi \,\mathrm{d}\boldsymbol{x} - \int_\Omega (\nabla\cdot\boldsymbol{m})\delta\phi \,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\partial\Omega} ([D_n\phi] + \boldsymbol{m}\cdot\boldsymbol{n})\delta\phi \,\mathrm{d}s(\boldsymbol{x})\Big).\end{aligned} \tag{6.9}$$

The Fréchet derivatives, and hence the local updates, are given by

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 2\,(\Delta\phi - \nabla\cdot\boldsymbol{m}) \qquad \text{in } \Omega, \tag{6.10}$$

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 2\,\Delta\phi \qquad \text{in } \overline{\Omega}^c, \tag{6.11}$$

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 2\,([D_n\phi] + \boldsymbol{m}\cdot\boldsymbol{n}) \qquad \text{on } \partial\Omega, \tag{6.12}$$

and variation of $\boldsymbol{m}$ yields

$$\frac{\partial}{\partial\boldsymbol{m}} e_\phi(\phi;\boldsymbol{m}) = 2\,(\nabla\phi) \qquad \text{in } \Omega. \tag{6.13}$$

Decomposing ϕ into $\phi = \phi_1 + \phi_2$ with ϕ_1 zero at the boundary and extended with zero in $\overline{\Omega}^c$ and ϕ_2 satisfying Laplace equation in $\Omega \cup \overline{\Omega}^c$, continuous across $\partial\Omega$ and fulfilling the jump in the normal derivative as $[D_n\phi_2] = D_n\phi_0 - \boldsymbol{m}\cdot\boldsymbol{n}$, we arrive at the following gradients

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 2\left(\Delta\phi_1 - \nabla\cdot\boldsymbol{m}\right) \qquad \text{in } \Omega, \tag{6.14}$$

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 0 \qquad \text{in } \overline{\Omega}^c, \tag{6.15}$$

$$\frac{\partial}{\partial\phi} e_\phi(\phi;\boldsymbol{m}) = 0 \qquad \text{on } \partial\Omega. \tag{6.16}$$

Therefore, only ϕ_1 but not ϕ_2 needs to be updated to maximize the energy functional with respect to ϕ_1. Let $\phi_\nu(\,\cdot\,;\phi_1,\boldsymbol{m})$ be the single layer potential (5.29). If ϕ_1 is parameterized by a hard constraint PINN $\widetilde{\phi}_1(\,\cdot\,;\boldsymbol{\omega})$ (see section 3.2.2) with model parameters $\boldsymbol{\omega}$ and $\boldsymbol{m}$ is parameterized by the model $\widetilde{\boldsymbol{m}}(\,\cdot\,;\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, we can consider the following objectives

$$\mathcal{L}_{\phi_1}(\boldsymbol{\omega},\boldsymbol{\theta}) = \int_\Omega |\Delta\widetilde{\phi}_1(\boldsymbol{x};\boldsymbol{\omega}) - \nabla\cdot\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})|^2 \mathrm{d}\boldsymbol{x} \tag{6.17}$$

$$e_\phi(\boldsymbol{\omega},\boldsymbol{\theta},\bar{\boldsymbol{\theta}}) = -\int_{\mathbb{R}^3} \|\nabla\widetilde{\phi}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}})\|^2 \,\mathrm{d}\boldsymbol{x} + 2\int_\Omega \nabla\widetilde{\phi}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}})\cdot\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{x}, \tag{6.18}$$

with $\widetilde{\phi}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}}) = \widetilde{\phi}_1(\boldsymbol{x};\boldsymbol{\omega}) + \phi_\nu(\boldsymbol{x};\widetilde{\phi}_1(\,\cdot\,;\boldsymbol{\omega}),\widetilde{\boldsymbol{m}}(\,\cdot\,;\bar{\boldsymbol{\theta}}))$. The update for ϕ_1 is replaced by the update resulting from the strong residual of (5.27) and $\bar{\boldsymbol{\theta}}$ is independent of $\boldsymbol{\theta}$. The derivatives with respect to some model parameters $\boldsymbol{\omega}_i$ and $\boldsymbol{\theta}_j$ are given by

$$\frac{\partial}{\partial\boldsymbol{\omega}_i}\mathcal{L}_{\phi_1}(\boldsymbol{\omega},\boldsymbol{\theta}) = 2\int_\Omega \left(\Delta\widetilde{\phi}_1(\boldsymbol{x};\boldsymbol{\omega}) - \nabla\cdot\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\right)\frac{\partial\Delta\widetilde{\phi}_1(\boldsymbol{x};\boldsymbol{\omega})}{\partial\boldsymbol{\omega}_i}\mathrm{d}\boldsymbol{x} \tag{6.19}$$

$$\frac{\partial}{\partial\boldsymbol{\theta}_j} e_\phi(\boldsymbol{\omega},\boldsymbol{\theta},\bar{\boldsymbol{\theta}}) = 2\int_\Omega \nabla\widetilde{\phi}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}})\cdot\frac{\partial\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_j}\,\mathrm{d}\boldsymbol{x}. \tag{6.20}$$

Both (6.17) and (6.18) need to be minimized. Exactly the same can be done for the vector potential $\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2$ with parametrization $\widetilde{\boldsymbol{A}}_1(\,\cdot\,;\boldsymbol{\omega})$ with the objectives

$$\mathcal{L}_{\boldsymbol{A}_1}(\boldsymbol{\omega},\boldsymbol{\theta}) = \int_\Omega \|\Delta\widetilde{\boldsymbol{A}}_1(\boldsymbol{x};\boldsymbol{\omega}) + \nabla\times\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\|^2\mathrm{d}\boldsymbol{x} \tag{6.21}$$

$$\begin{aligned} e_{\boldsymbol{A}}(\boldsymbol{\omega},\boldsymbol{\theta},\bar{\boldsymbol{\theta}}) = {} & \int_\Omega \|\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\|^2\,\mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^3}\|\nabla\widetilde{\boldsymbol{A}}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}})\|^2\,dx \\ & - 2\int_\Omega \left(\nabla\times\widetilde{\boldsymbol{A}}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}})\right)\cdot\widetilde{\boldsymbol{m}}(\boldsymbol{x};\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{x}, \end{aligned} \tag{6.22}$$

with $\widetilde{\boldsymbol{A}}(\boldsymbol{x};\boldsymbol{\omega},\bar{\boldsymbol{\theta}}) = \widetilde{\boldsymbol{A}}_1(\boldsymbol{x};\boldsymbol{\omega}) + \boldsymbol{A}_\nu\left(\boldsymbol{x};\widetilde{\boldsymbol{A}}_1(\,\cdot\,;\boldsymbol{\omega}),\widetilde{\boldsymbol{m}}(\,\cdot\,;\bar{\boldsymbol{\theta}})\right)$ and $\boldsymbol{A}_\nu$ as in (5.32). Again, only the last term is relevant for the optimization of $\widetilde{\boldsymbol{m}}$.

Comparing (6.20) with (6.3) and setting $h_d = -\nabla\widetilde{\phi}$, we see that the gradients match. The magnetostatic energy e_d in (6.1) can be replaced by (6.18) or (6.22) which yields the objectives

$$\hat{e}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) = \frac{1}{2} e_\phi(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) + e_{zee}(\boldsymbol{\theta}) + e_a(\boldsymbol{\theta}) + e_{ex}(\boldsymbol{\theta}) \tag{6.23}$$

$$\check{e}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) = \frac{1}{2} e_{\boldsymbol{A}}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) + e_{zee}(\boldsymbol{\theta}) + e_a(\boldsymbol{\theta}) + e_{ex}(\boldsymbol{\theta}) \tag{6.24}$$

Those formulations still include a whole space integral. The evaluation on the finite domain is explained in Section 5.4. However, since e_ϕ and $e_{\boldsymbol{A}}$ are minimized with respect to $\boldsymbol{\theta}$ but not $\bar{\boldsymbol{\theta}}$, the whole space integral can be treated as constant and be removed from the objective. Algorithm 4 shows the minimization of the energy functional using an arbitrary first or second order optimizer, assuming that the unit norm constraint is satisfied via hard constraint formulation. This algorithm is a generalization of Algorithm 2, where the stray field is not computed explicitly.

Algorithm 4 Energy minimization with lower energy bound for the self-energy

Require: $\hat{e}$, $\boldsymbol{\omega}$, $\boldsymbol{\theta}$, tolerance $\epsilon_{\boldsymbol{m}}$
Ensure: $\|\nabla_{\boldsymbol{\theta}} \hat{e}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}})\| < \epsilon_{\boldsymbol{m}}$
  $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$
  loop
    1. Find approximate solution $\widetilde{\phi}_1$ with parameters $\boldsymbol{\omega}$ ▷ ELM or PINN solution
    2. Perform minimization step with objective $\hat{e}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}})$ with respect to $\boldsymbol{\theta}$
    $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$
  end loop

When using a hard constraint PINN ansatz for $\widetilde{\phi}_1$ and $\widetilde{\boldsymbol{m}}$, an alternating optimization scheme can be used for energy minimization. First, the solution to $\widetilde{\phi}_1$ is replaced with the respective PINN minimization problem, and second, we observe that more than one steps can be performed for each iteration. This leads to Algorithm 5.

All the introduced algorithms can equally be applied with the vector potential and the upper bound instead. Minimization of the vector potential can be performed with the same algorithm by just replacing $\mathcal{L}_{\phi_1}$ with $\mathcal{L}_{\boldsymbol{A}_1}$ and by using $\check{e}$. If ϕ_1 or $\boldsymbol{A}_1$ is modeled with a hard constrained ELM ansatz, with a precomputed solution operator, the inner loop to update $\boldsymbol{\omega}$ can be replaced with the solution of the ELM. Further, if the unit norm constraint is modelled with a penalty framework, one needs to add the respective penalty term and gradually increase the penalty parameter during optimization.

The proposed energy minimization method can be applied for the computation of a hysteresis curve (see Section 2.3). The hysteresis depends on the history of previous magnetization states as a function of the external field. Incorporating the external field as conditional parameter can therefore be difficult and might lead to an optimization problem which is not well-posed. Moreover, a simple continuous neural network model cannot approximate discontinuities well and the energy functional

Algorithm 5 Alternating scheme for energy minimization

Require: $\hat{e}$, $\mathcal{L}_{\phi_1}$, $\boldsymbol{\omega}$, $\boldsymbol{\theta}$, tolerances $\epsilon_{\boldsymbol{m}}$ and ϵ_{ϕ_1}, iteration limits $N_{\boldsymbol{m}}$ and N_{ϕ_1}
Ensure: $\|\nabla_{\boldsymbol{\omega}} \mathcal{L}_{\phi_1}(\boldsymbol{\omega}, \boldsymbol{\theta})\| < \epsilon_{\phi_1}$ and $\|\nabla_{\boldsymbol{\theta}} \hat{e}(\boldsymbol{\omega}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}})\| < \epsilon_{\boldsymbol{m}}$

```
θ̄ ← θ
loop
    for 1, ..., N_{φ1} do
        ω ← Perform minimization step for L_{φ1}(ω, θ) w.r.t ω
        if ‖∇_ω L_{φ1}(ω, θ)‖ < ε_{φ1} then
            break
        end if
    end for
    for 1, ..., N_m do
        θ ← Perform minimization step for ê(ω, θ, θ̄) w.r.t θ
        if ‖∇_θ ê(ω, θ, θ̄)‖ < ε_m then
            break
        end if
    end for
    θ̄ ← θ
end loop
```

needs to be properly minimized for each external field. Therefore, a better choice of optimization might be the application of a full batch second order optimizer such as the *Trust Region* method and increase/decrease the external field step-by-step.

To minimize grain structure models, one could use a separate small scale PINN to model the magnetization for each grain. Further, and ELM or PINN for each grain would be required to model scalar or vector potential. The training for the scalar or vector potential model can be performed in parallel on many computational units. The same holds true for the magnetization. Only the charge tensor $\boldsymbol{F}$ in (5.36) needs to be shared and synchronized. Therefore, better evaluation strategies for the single layer potential are of particular importance.

6.3 Results

The algorithms described above are applied for full 3d micromagnetic energy minimization. We performed energy minimization to solve the NIST μMAG Standard Problem #3 [43] with stochastic first order methods and the computation of a demagnetization loop for a spherical and cubic domain using second order full batch optimization schemes.

- In Section 6.3.1.1, the NIST μMAG Standard Problem #3 is solved with a penalty framework and Algorithm 3, where both parts, ϕ_1 and ϕ_2, of the scalar potential are modeled via hard constraint ELMs.

- In Section 6.3.1.2, the NIST μMAG Standard Problem #3 is solved with the hard constraint ansatz in Section 6.1.2 and Algorithm 2, where the stray field is computed with Algorithm 1 adapted for the vector potential and upper bound.
- In Section 6.3.2, the demagnetization curve of a hard magnetic sphere is computed with a hard constraint PINN ansatz for $\boldsymbol{A}_1$ and the hard constraint ansatz in Section 6.1.2 for the magnetization. Energy minimization is performed with Algorithm 5.
- In Section 6.3.3, the demagnetization curve of a hard magnetic cube is computed and compares the scalar and vector potential approach for the computation of the self energy. The magnetization is modelled with the hard constraint ansatz in Section 6.1.2. First, in Section 6.3.3.1, ϕ_1 and $\boldsymbol{A}_1$ are modeled via a hard constraint ELM ansatz and minimization is performed with Algorithm 4. Optimization is performed once with a trust region framework and once with L-BFGS. Second, in Section 6.3.3.3, ϕ_1 and $\boldsymbol{A}_1$ are modeled via a hard constraint PINN ansatz and minimization is performed with Algorithm 5. The algorithms are adapted for the vector potential and the upper bound.

6.3.1 NIST μMAG Standard Problem #3

The methodology described above gives us a powerful framework for micromagnetic energy minimization. We tested our algorithmic framework on the NIST μMAG Standard Problem #3 [43]. The primary goal is to find the single domain limit between the flower and the vortex state of the magnetization. This is the size of a cube L with equal Gibbs free energy (6.1) for both states, and is given in units of the intrinsic length $l_{ex} = \sqrt{2A_{ex}/K_m}$. The reduced functional e as in (6.1) was used as the target objective, but without an external field applied ($\boldsymbol{h}_{ext} = \boldsymbol{0}$) and L being a conditional parameter, i.e.,

$$e(L;\, \boldsymbol{\theta}) = \frac{1}{2} e_d(\boldsymbol{m}(\,\cdot\,; L, \boldsymbol{\theta})) + e_a(\boldsymbol{m}(\,\cdot\,; L, \boldsymbol{\theta})) + e_{ex}(L, \boldsymbol{m}(\,\cdot\,; L, \boldsymbol{\theta})). \tag{6.25}$$

However, results of the energy terms are given in units of $\frac{1}{2}K_m$ due to the example specification. The uniaxial anisotropy is $K_u = 0.1\,[\frac{1}{2}\mu_0 M_s^2]$ with the easy axis directed parallel to the x_3-axis of the cube.

Here, the mean over the conditional domain is denoted by $\langle\,\cdot\,\rangle$. Hence, $\langle e \rangle$ was minimized with respect to the network parameters $\boldsymbol{\theta}$ by variants of SGD. For this problem, the self energy was directly computed with ELMs and no alternating scheme is required.

6.3.1.1 Penalty framework

As a proof of concept, the problem was solved with a penalty framework to account for the unit norm constraint. The objective function (6.5) was minimized with a stochastic optimization framework with an increasing penalty parameter α.

The sample set included the spatial domain sample points as well as the lengths of the cube, i.e. $\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_{N_\Omega}\} \times \Lambda$, where the interval $\Lambda = (8, 9)$ was chosen. The spatial domain sample set $\boldsymbol{X}_{dom}$ includes $N_\Omega = 2^{13}$ samples which were drawn from a Sobol sequence [89].

The evaluation of the scalar potential was done naively. First, a hard constraint ELM was trained to solve for ϕ_1, where the indicator function ℓ was computed using a sampling method as described in section 4.2. The surface was partitioned by the 6 faces of the cube and in total 256 sampling points on $\partial\Omega$ were used to build the indicator function. Note that the indicator function is not an Approximate Distance Function since it is not normalized to first order. Second, the single layer potential (5.29) was evaluated for each point of another sample set $\boldsymbol{X}_{bnd} \subset \partial\Omega$ with 2^{11} samples (also drawn from a Sobol sequence and mapped onto $\partial\Omega$). The solution of the single layer potential was computed by Monte Carlo sampling with another 2^{14} uniform samples. This boundary data was used to fit a second ELM to approximate the boundary solution $\phi_2|_{\partial\Omega}$. Last, a final ELM was trained to approximate ϕ_2 as in the ansatz (3.22). All ELMs shared the same input weights with a hidden layer size $M = 256$, which were drawn from a Sobol sequence from a hypercube $[-4, 4]^4$. A `tanh` activation function was used. For fast evaluation, the solution operators to the linear least squares problems were precomputed using SVD.

A batch size of 50 sample points, together with a single instance $L \in \Lambda$ of the conditional parameter, which were drawn from a uniform distribution, was used. Therefore, for each batch, there was a single stray field computation required. Minimization was perfomed with Algorithm 3. A *RAdam* optimizer [53] with a constant learning rate of 1e−5 was used to train the model for $N_\alpha = 10$ cycles with the dataset $\boldsymbol{X}_{dom}$. After each 1000 epochs, the penalty parameter α was increased by a factor of $\zeta = 1.7$. The initial penalty parameter was $\alpha_0 = 10$. Note that we refer to an epoch as an iteration over all batches ($N_{\boldsymbol{m}} = 1000B$ for B batches).

The network architecture for the magnetization is shown in Figure 6.1. It is important to set an appropriate initial magnetization for the energy minimization procedure, since there exist multiple local minima. Therefore, before the actual training process, the model was trained to approximate an initial magnetization. For the initial magnetization, we used the normalized version of

$$\boldsymbol{m}_f(\boldsymbol{x}) = (\boldsymbol{x}_1\boldsymbol{x}_3, \boldsymbol{x}_2\boldsymbol{x}_3, 1)^T \tag{6.26}$$

for the flower state and

$$\boldsymbol{m}_v(r) = \left(-\frac{z}{r}\sqrt{1 - \exp\left(-4\frac{r^2}{r_c^2}\right)}, \exp\left(-2\frac{r^2}{r_c^2}\right), \frac{x}{r}\sqrt{1 - \exp\left(-4\frac{r^2}{r_c^2}\right)}\right)^T, \tag{6.27}$$

with $r = \sqrt{\boldsymbol{x}_1^2 + \boldsymbol{x}_3^2}$ and $r_c = 0.14$, for the vortex state.

Figure 6.2 shows the total energy with respect to the cube length $L\,[l_{ex}]$ for the optimized flower and the vortex state. Note that each line is evaluated from one single low-parametric neural network and the solutions are continuous. We found the single domain limit at $L = 8.4729$ with a total reduced energy of $e = 0.3038\ \left[\frac{1}{2}\mu_0 M_s^2\right]$.

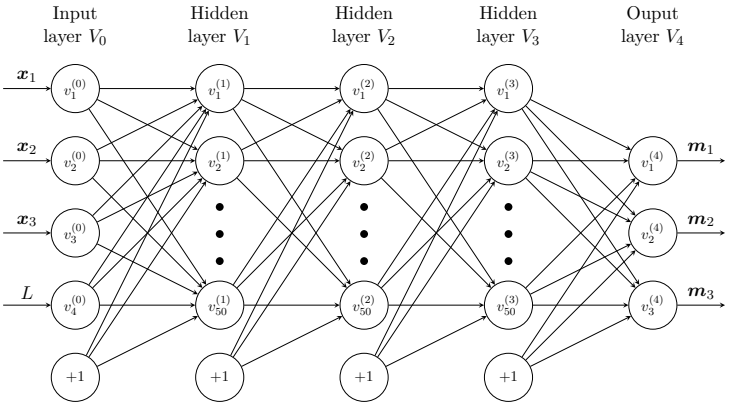


Figure 6.1: Neural network architecture for $\boldsymbol{m}$ with 3 hidden layers and 50 nodes per hidden layer. There were 5503 trainable parameters.

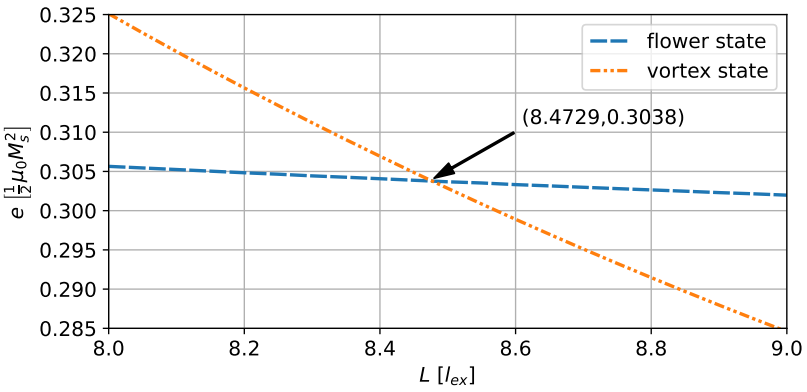


Figure 6.2: Energy crossing for the μMAG Standard Problem #3. The model was trained with a penalty framework. The Single Domain Limit is at $(8.4729, 0.3038)$. Each line is evaluated from a single low-parametric neural network that represents the local energy optimum at the respective exchange length.

	Flower State		Vortex State	
e_a	0.0052	(0.0052)	0.0519	(0.0522)
e_{ex}	0.0159	(0.0158)	0.1724	(0.1696)
e_s	0.2827	(0.2839)	0.0795	(0.0830)
$\langle m_1 \rangle$	0.0078	(-)	0.0107	(-)
$\langle m_2 \rangle$	0.0005	(-)	0.3461	(0.351)
$\langle m_3 \rangle$	0.9733	(0.973)	0.0086	(-)
$\| \operatorname{err}_c \|_{\max}$	0.0040	(-)	0.0062	(-)
$\operatorname{MSE}(\operatorname{err}_c)$	9.2e−07	(-)	1.4e−06	(-)

Table 6.1: Energies, mean magnetization and constraint violation in the single domain limit. The model was trained with a penalty framework. The values in brackets are here stated for reference and were previously published in [37].

Table 6.1 shows the reduced anisotropy energy e_a, the reduced exchange energy e_{ex}, the reduced magnetostatic energy e_s as well as the mean magnetization for the single domain limit. Here, $\operatorname{err}_c(\boldsymbol{x}) = \|\boldsymbol{m}(\boldsymbol{x})\| - 1$ is the norm constraint violation. The values are in excellent agreement with previously published results [37].

The training curves for the total energy (Figure 6.3) and the norm constraint violation (Figure 6.4) show the effect of the increasing penalty term in (6.5). As the penalty parameter is scaled up, the norm constraint violation is punished more severely and the energy increases. It is important that the initial penalty is not too small, since this would make it easier during training to jump out of a local minimum and converge to another state. We note that the total energy in Figure 6.3 is the stochastic mean over the domain of Λ.

6.3.1.2 Hard constraint PINN ansatz

Here, a hard constrained neural network ansatz with Cayley transform as in (6.8) was used to approximate the magnetization over the full conditional domain Λ,

$$\widetilde{\boldsymbol{m}}_{\boldsymbol{\theta}}(\boldsymbol{x}, L) = \operatorname{cay}\left(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}, L)\right) \boldsymbol{m}_0(\boldsymbol{x}). \tag{6.28}$$

To describe the domain $\Omega = [-0.5, 0.5]^3$, the ADF from (5.52) was used. A training set $\mathcal{S}$ of 2^{13} samples was drawn from this domain using a Halton sequence [36]. For computation of ϕ_1 an ELM with 512 hidden nodes, `tanh` activation function, and weights drawn from a Halton sequence within $[-2, 2]^4$, for the 3d input and the bias term respectively, was established. We note that `tanh` is more suitable as activation function for this ELM since it has non-vanishing second derivative for positive values, contrary to `elu` like activations. However, also `elu` works quite well but might require some ℓ_2 regularization. A second order Taylor series was used for the single layer potential in (5.36). Each face of the cube was equidistantly split into 4×4 squares, and the Jacobian of the surface tensor was precomputed for the full training

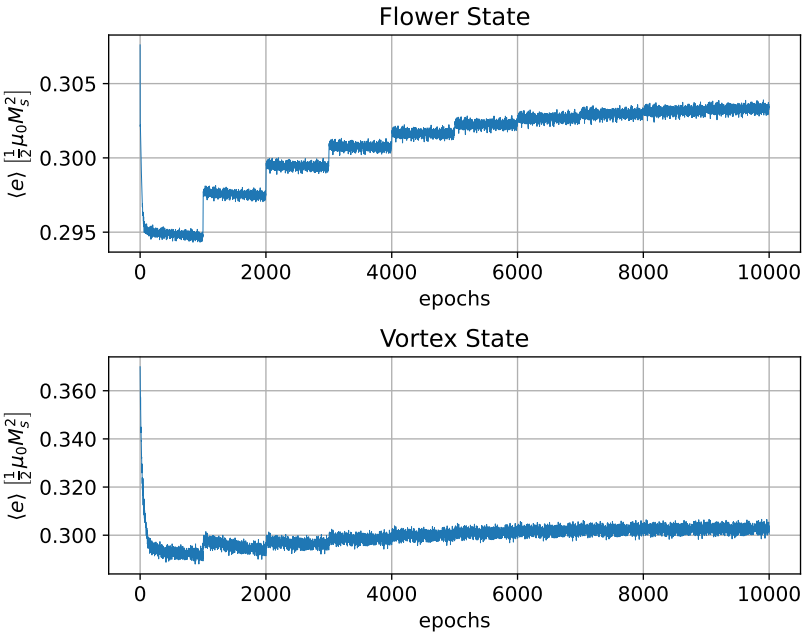


Figure 6.3: Learning curve of a penalty framework for the total Gibbs free energy for the flower and the vortex state.

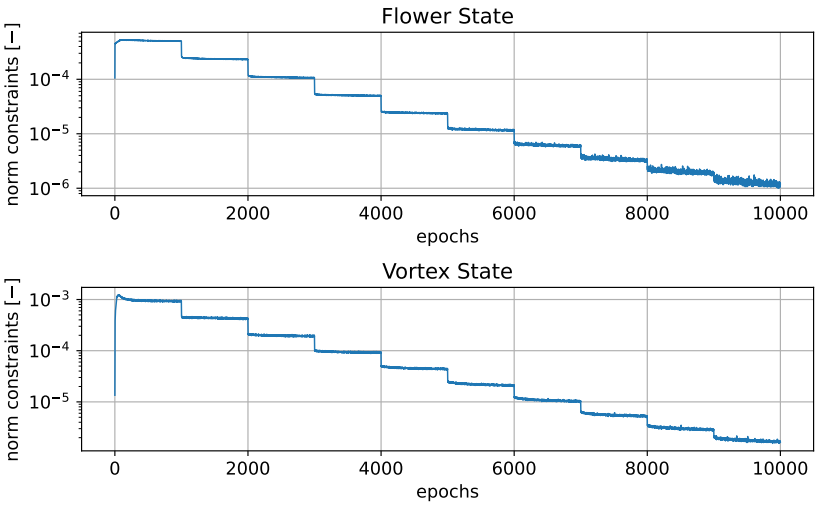


Figure 6.4: Norm constraint violation during training with a penalty framework for flower and vortex state.

Layer	Activation	Input shape	Output shape
Dense	`gelu`	4	50
Dense	`gelu`	50	50
Dense	`gelu`	50	50
Dense	`gelu`	50	50
Dense	`gelu`	50	50
Dense	`gelu`	50	50
Dense	`identity`	50	3

Table 6.2: Neural network architecture for the magnetization model in (6.8). The model had 13153 parameters.

	Flower	Vortex	Twisted Flower	Vortex
e_a	0.0058	0.0520	0.0203	0.0519
e_{ex}	0.0177	0.1738	0.0424	0.1723
e_d	0.2804	0.0781	0.2386	0.0771
$\langle m_1 \rangle$	-0.0008	-0.0005	-0.0014	0.0014
$\langle m_2 \rangle$	-0.0034	0.3464	-0.0002	0.3419
$\langle m_3 \rangle$	0.9700	-0.0068	0.8911	-0.0021

Table 6.3: Energies and mean magnetizations in the single domain limit for the hard constraint ansatz.

set $\mathcal{S}$. The surface integrals in (5.36) were computed with a Newton-Cotes formula and a 5^{th} order Gauss–Legendre quadrature. Each sample required the storage of a tensor of shape $16 \times 6 \times 6 \times 3$. Note that for a cuboid geometry, after removing zero terms and taking symmetry of second partial derivatives into account, each face has only 6 terms in the quadratic Taylor series approximation. This reduces the required storage space significantly at the expense of a more complicated implementation.

Table 6.2 shows the architecture which is used for the magnetization model (6.8). In total, the model had 13153 parameters. The model had three input nodes for a vector from the physical domain, as well as one input node for the conditional cube length parameter L from which the exchange constant could be computed. The cube length parameter is uniformly drawn from $\Lambda = [8, 9]$ during training. The weights of the network were initialized with the default initializer of `JAX`, which is the LeCun Normal initializer. The initial magnetization was as in (6.26) and (6.27).

Algorithm 2 was used for minimization with the *AdamW* optimizer [55] and an exponential learning rate decay of 0.1. For the vortex state, the initial learning rate was 10^{-4} and reached 10^{-6} by the end of training. For the flower state, a lower initial learning rate was used since the metastable flower state is very sensitive to a higher learning rate, which can cause faster transition to the more stable twisted flower state. Therefore, the initial rate was set to 10^{-5} and the learning rate reached 10^{-8} by the end of training. Both models were trained for 6000 epochs. A batch size of 200 samples and 5 samples for the conditional parameter was selected. For

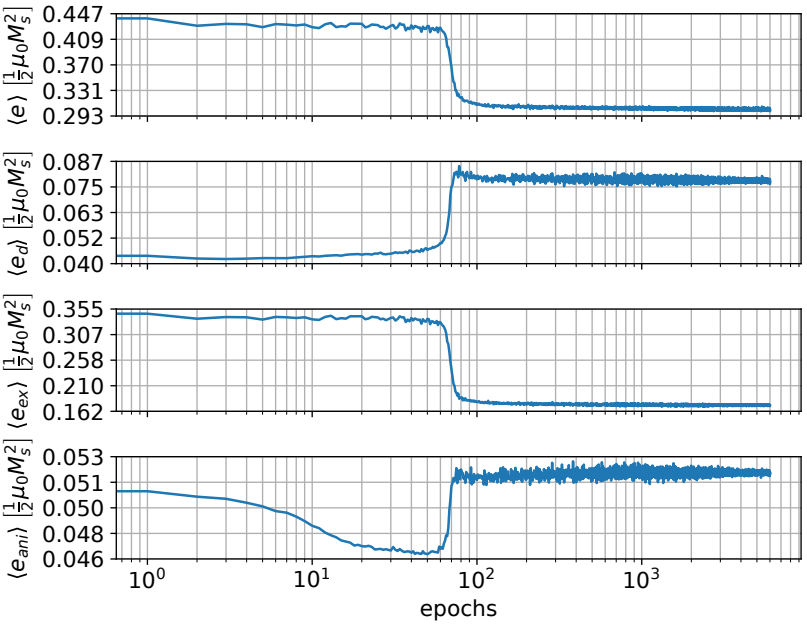


Figure 6.5: Evolution of different energy terms of (6.25) of the training dataset during the training process for the vortex state. The energy terms were averaged over the conditional domain $\Lambda = [8, 9]$. Note that this is only approximating (6.25) with minibatches of the dataset, and for each epoch the energies of the batches were averaged. The minimization was done with exponential learning rate decay. The initial learning rate was 1e−4, the final rate 1e−6, and the exponential decay rate was 0.1.

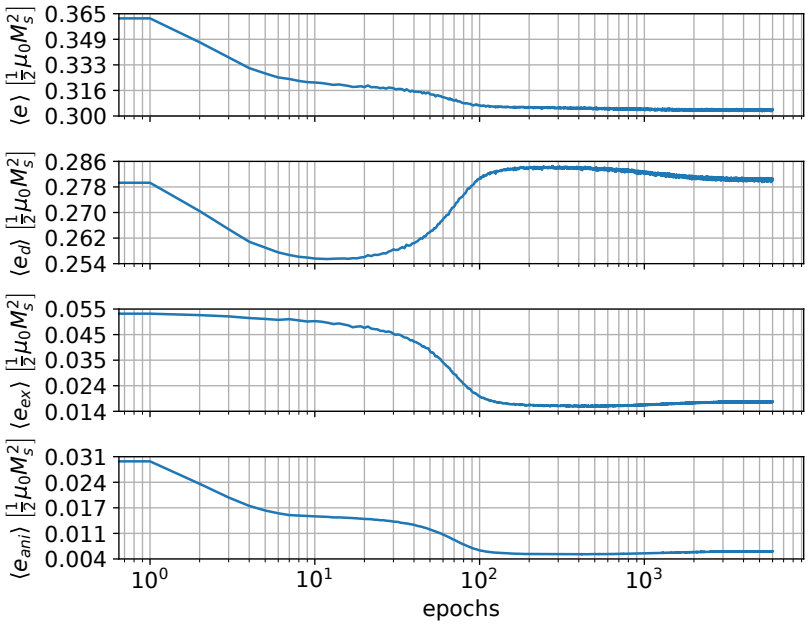


Figure 6.6: Evolution of different energy terms of (6.25) of the training dataset during the training process for the flower state. The energy terms were averaged over the conditional domain $\Lambda = [8, 9]$. Note that this is only approximating (6.25) with minibatches of the dataset and for each epoch the energies of the batches were averaged. The minimization was done with exponential learning rate decay. The initial learning rate was 1e−5, the final rate 1e−8, and the exponential decay rate was 0.1.

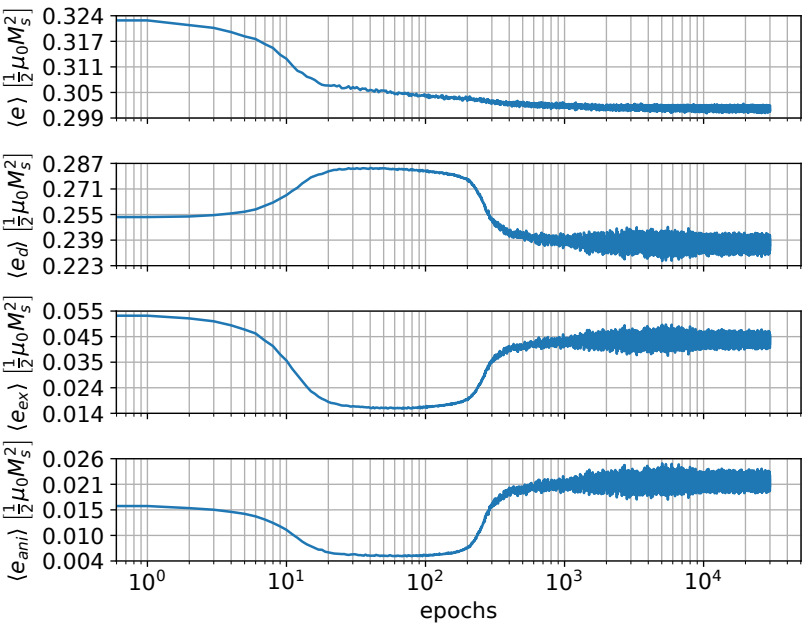


Figure 6.7: Evolution of different energy terms of (6.25) of the training dataset during the training process for the flower state. The energy terms were averaged over the conditional domain $\Lambda = [8, 9]$. Note that this is only approximating (6.25) with minibatches of the dataset and for each epoch the energies of the batches were averaged. The minimization was done with exponential learning rate decay. The initial learning rate was 1e−4, the final rate 1e−6, and the exponential decay rate was 0.1. After 200 training epochs the transition to the twisted flower state begins.

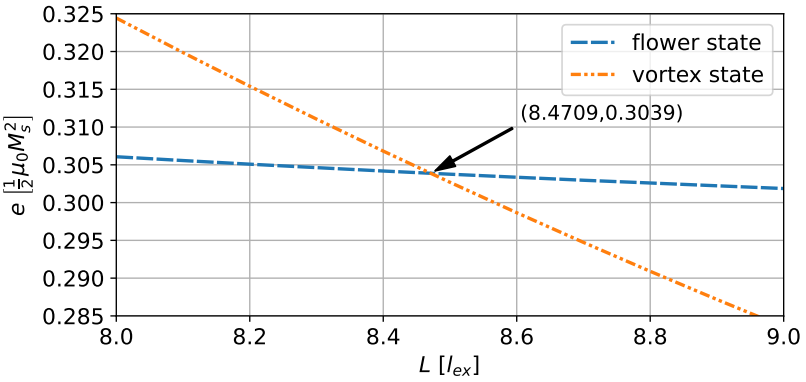


Figure 6.8: Energy crossing of the vortex and flower state for the μMAG Standard Problem #3. The Single Domain Limit was found at (8.4709, 0.3039).

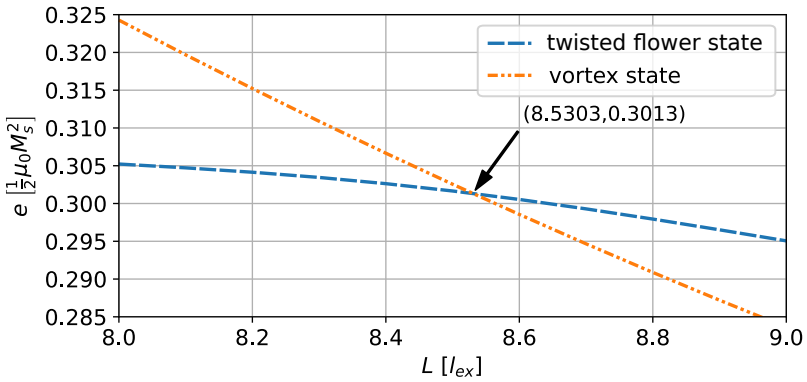


Figure 6.9: Energy crossing of the vortex and twisted flower state for the μMAG Standard Problem #3. The Single Domain Limit was found at (8.5303, 0.3013).

each conditional parameter, the mean over all 200 samples was computed, and the result was averaged to compute the stochastic gradient. Note that 5 evaluations of the stray field per batch were required, one for each conditional parameter. This is a more sophisticated evaluation strategy for the stray field, which allows a bigger batch size for the conditional parameter without using too much memory.

Figure 6.5 and Figure 6.6 show the energies for vortex and flower state during the training process. It can be seen that the vortex state converged very rapidly to a very good solution. Optimization for the flower state was more intricate, since a higher learning rate or even longer training may cause a transition to the more stable twisted flower state. For comparison, a second model for the flower state was trained with the same learning rate as for the vortex state but for 30000 epochs in order to see the long term evolution for the flower state. This is depicted in Figure 6.7. Figure 6.8 shows the energy crossing of vortex and flower state. The single domain limit was found at $(8.4709, 0.3039)$. The energy crossing for the twisted flower state is shown in Figure 6.9 and the single domain limit was found at $(8.5303, 0.3013)$. For the computation of the energies, a separate test set with 2^{14} samples, drawn from a Halton sequence, was used. Table 6.3 summarizes the results for the single domain limit. The computation took around one hour for the flower and vortex state and 5 hours for the twisted flower state respectively. All computations were done with single precision on a *Geforce RTX 2080 Ti*.

The computational cost to compute the solution of the ELM for ϕ_1 is $\mathcal{O}(|\mathcal{S}|MC)$, where M is the number of hidden nodes of the ELM and C denotes the batch size of the conditional parameter. Computation of ϕ_2 with a second order Taylor approximation requires $\mathcal{O}(BCR)$ operations, where B is the batch size regarding the dataset $\mathcal{S}$ of the physical domain and R is the number of surface elements. Energy minimization with SGD in total requires $\mathcal{O}(E(BC + |S|MC + BCR))$ for E epochs.

Also, the convergence with respect to the number of model parameters is of inter-

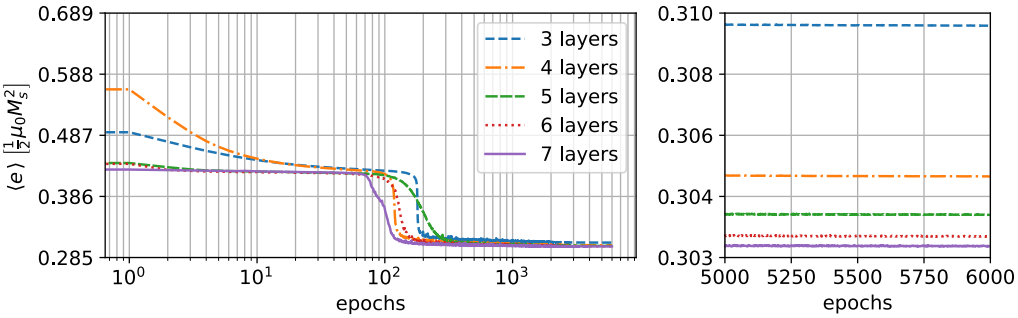


Figure 6.10: Training curves for the vortex state and different model architectures (left), converged values (right). Note that input and output layer were included in the number of layers. Each model had a constant width of 50 nodes per hidden layer, as in Table 6.2. The energy was computed with respect to a validation dataset. The minimization was done with exponential learning rate decay. The initial learning rate was 1e−4, the final rate 1e−6, and the exponential decay rate was 0.1.

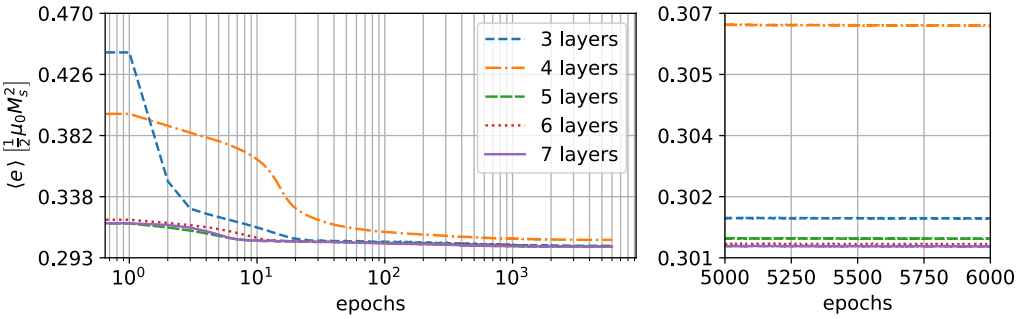


Figure 6.11: Training curves for the flower state and different model architectures (left), converged values (right). Note that input and output layer were included in the number of layers. Each model had a constant width of 50 nodes per hidden layer, as in Table 6.2. The energy was computed with respect to a validation dataset. The minimization was done with exponential learning rate decay. The initial learning rate was 1e−4, the final rate 1e−6, and the exponential decay rate was 0.1. Note that the model with 4 layers did not converge to the twisted flower state for some reason.

est. To that end, energy minimization for neural networks with increasing number of hidden layers and constant width of 50 nodes per layer, was performed. Figure 6.10 shows the training curves of the different models for the vortex state and Figure 6.11 for the flower state respectively. The curves show that the model performance increases for larger models. Hyperparameter validation for PINNs is still an important topic and, to our knowledge, there is no simple method. Ideally, one wants to select a model with enough parameters to learn the full solution space up to some error tolerance. However, since the solution is usually not known, this can not be achieved. A recent work [88] derives *a priori* and *a posterior* error estimates for PINNs which could be an important step forward for hyperparameter validation. On the other hand, overfitting does not seem to be a particular problem for PINNs, since one can always increase the sample size. Therefore, also a model much larger than required can be selected on expense of computational cost.

6.3.2 Demagnetization of a hard magnetic sphere

The second test involved the computation of the demagnetization process of a hard magnetic $Nd_2Fe_{14}B$ sphere with radius $R = 10\,\mathrm{nm}$. The main purpose was to test the capabilities of the hard constrained neural network model to capture the demagnetization and the switching of the magnetization. The sphere had uniaxial anisotropy in z-direction, with the anisotropy constant $K_u = 4.3 \times 10^6\,\mathrm{J/m}$. Saturation polarization for the hard magnetic material was $J_s = 1.61\,\mathrm{T}$ and the exchange stiffness for $Nd_2Fe_{14}B$ was selected to be $A_{ex} = 7.3 \times 10^{-12}\,\mathrm{J/m}$. Therefore, the reduced uniaxial anisotropy constant was $Q = K_u/K_m \approx 2.085$. The demagnetization was computed with an applied field along the anisotropy axis $(0, 0, 1)^T$ with the analytic value of the switching field given by $h_{sw} = 2Q$ and with the applied field in a 45° angle to the anisotropy axis with the analytic value of the switching field $h_{sw} = Q$ [51]. The micromagnetic self energy was computed with the vector potential, which is modeled with a PINN as given in Table 6.4. The ADF was given by $\ell(\boldsymbol{x}) = (R^2 - \|\boldsymbol{x}\|_2^2)/2R)$. A hard constrained ansatz, as in (6.8), was used to model the magnetization and the network architecture for $\mathcal{N}_{\boldsymbol{\theta}}$ was the same as in Table 6.4. The initial magnetization $\boldsymbol{m}_0$ was uniform along the anisotropy axis. The compu-

Layer	Activation	Input shape	Output shape
Dense	`gelu`	3	12
Dense	`gelu`	12	12
Dense	`identity`	12	3

Table 6.4: Neural network model for the vector potential of a hard magnetic $Nd_2Fe_{14}B$ particle.

tational domain was scaled to the unit sphere. The scaled exchange constant was given by $\widetilde{A}'_{ex} = \widetilde{A}_{ex}/R^2$. Energy minimization was performed with Algorithm 5 but with the vector potential and $\check{e}$. For the evaluation of the single layer potential, the parametrization (5.48) was used, with the input space partitioned into 6×6 equal

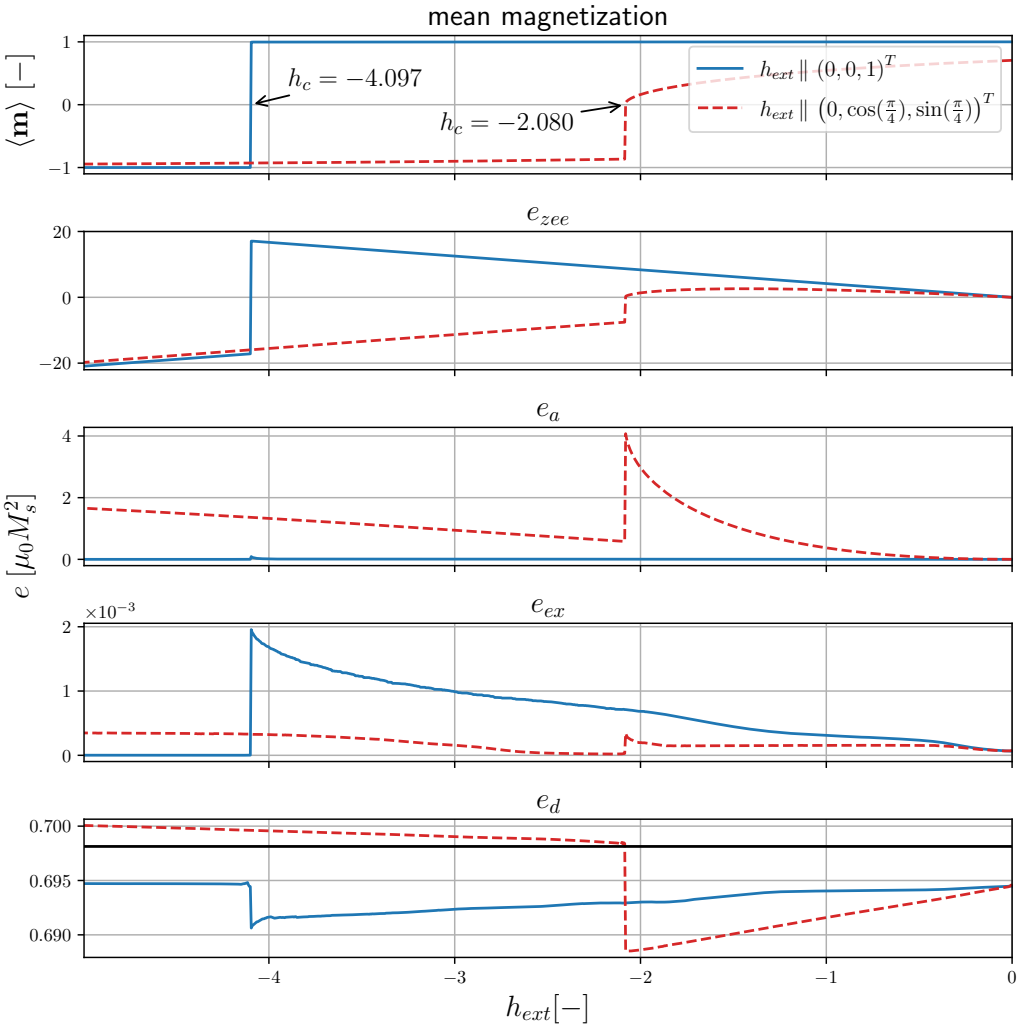


Figure 6.12: Demagnetization process of a hard magnetic sphere with the applied field along $(0,0,1)^T$ and $(0,\cos(\pi/4),\sin(\pi/4))^T$. The mean magnetization was computed by projection onto the axis of the applied field. The horizontal line in the last plot highlights the magnetostatic energy of a uniformly magnetized sphere.

Layer	Activation	Input shape	Output shape
Dense	`gelu`	3	16
Dense	`gelu`	16	16
Dense	`identity`	16	3

Table 6.5: Neural network model for the computation of a demagnetization process for a $Nd_2Fe_{14}B$ cube.

spaced tiles. Each tile was integrated with a 5^{th} order Gauss–Legendre quadrature rule. In total, there were 4096 collocation points, drawn from a Halton sequence [36], and are mapped to the domain with the parametrization (5.49). A Trust Region optimizer, with Steihaug-CG method as a sub-problem solver, was used for minimization and the parameters for Algorithm 5 were: $\epsilon_{\boldsymbol{A}_1} = 10^{-2}$, $\epsilon_{\boldsymbol{m}} = 10^{-2}$, $N_{\boldsymbol{m}} = 10$ and $N_{\boldsymbol{A}_1} = 20$. Also, a damping factor of 10^{-3} was applied to regularize the Hessian matrix. In the beginning of the demagnetization process, the particle was uniformly magnetized in the direction of the anisotropy axis and the applied field was set to zero. For each step, the applied field was reduced by $5e-3$.

Figure 6.12 shows the demagnetization and the corresponding energy terms for both trials. The self-energy of a uniformly magnetized sphere is $4\pi/18$ $[\mu_0 M_s^2]$. It can be seen from the figure that the magnetostatic energy is very close to this value. From this and the exchange energy, we can conclude that the model performs quite well in approximating the magnetization reversal, which is a homogeneous rotation of the uniform magnetization. The computation was performed with single precision on a *Geforce RTX 2080 Ti*.

6.3.3 Demagnetization of a hard magnetic cube

This test involved the computation of the demagnetization process of a hard magnetic $Nd_2Fe_{14}B$ cube with an edge length of $L = 70\,\text{nm}$. The material parameters and anisotropy axis were the same as for the sphere, and the scaled exchange constant was given by $\widetilde{A}'_{ex} = \widetilde{A}_{ex}/L^2$. With energy density $K_m = \mu_0 M_s^2$ J/m, we had an exchange length of $\ell_{ex} = \sqrt{2A_{ex}/K_m} \approx 2.6\,\text{nm}$, the material parameter $Q = K_u/K_m \approx 2.08$ and a wall width parameter of $\sqrt{A_{ex}/K_u} \approx 1.3\,\text{nm}$. If the exchange length is larger than the wall width, we are usually in the hard magnetic realm. The external field was applied along the vector $(1, 0, 10)^T$ and was nearly parallel to the anisotropy axis.

The neural network architecture of $\mathcal{N}_{\boldsymbol{\theta}}$ to model the magnetization is given in Table 6.5. The initial magnetization was uniform along the anisotropy axis. The training set contained 2^{12} samples from a Halton sequence. For a decreasing external field $h_{ext} = \|\boldsymbol{h}_{ext}\| \in [-3.5, 1]$, an equilibrium state was computed for each step using a trust region method. The constrained quadratic trust region problem was solved with Steihaug-CG method and a damping factor of 10^{-3} was used for Hessian regularization. At each step h_{ext} was reduced by $\Delta h_{ext} = 5 \times 10^{-3}$. The magnetostatic energy was modeled via scalar and vector potential.

6.3.3.1 ELM ansatz

In a first trial, an ELM with 512 hidden nodes, `tanh` activation and the input weights drawn from $[-2, 2]^4$, was applied to model ϕ_1 and $\boldsymbol{A}_1$. The solution operator was precomputed with a small Ridge regularization of 10^{-3} and was the same for scalar and vector potential. The ADF and the evaluation strategy for the single layer potential was the same as in Section 5.5.2. Algorithm 4 with $\epsilon_{\boldsymbol{m}} = 10^{-3}$ was used for energy minimization. Figure 6.13 shows the demagnetization process for the ELM ansatz for vector and scalar potential. Also, lower (5.39) and upper bounds (5.41) were computed, and it can be seen that energies and bounds were almost identical, which indicates that the model for computation of the self-energy was sufficient. The switching field was found at $h_c = -2.772$ for the scalar potential version and at $h_c = -2.767$ for the vector potential, which is the same as given in the reference [22]. The scalar potential missed the switching by a single step, which is still a very good result. One can see that the exchange energy of the vector potential simulation is higher than the one of scalar potential simulation, which is an artifact of the optimization process.

In our trials, we found that the improper evaluation of the single layer potential can have a negative effect on the computation and stability of the minimization, leading to early switching. Some validation metric of an accurate evaluation would therefore be of interest.

During the switching process, the energy landscape has a saddle point. Therefore, we assume that the quadratic form of the trust region method can capture this behavior very well during optimization. Figure 6.14 shows the total energy after each optimization step at the top and the minimum recorded curvature $\boldsymbol{p}^T H_{\boldsymbol{\theta}} e \boldsymbol{p}$ during the optimization at the bottom, $H_{\boldsymbol{\theta}} e$ is the Hessian matrix of e w.r.t. the model parameters $\boldsymbol{\theta}$ and $\boldsymbol{p}$ is the update direction. There is a strong negative curvature when the optimizer reaches the switching point.

The computational cost for the full batch trust region method is $\mathcal{O}(EI|\mathcal{S}||\boldsymbol{\theta}|^2)$, where I is the number of iterations for the Steihaug-CG method, E the number of total trust region iterations and $|\boldsymbol{\theta}|$ is the number of model parameters. The term $|\mathcal{S}||\boldsymbol{\theta}|^2$ arises from the Hessian vector products. This shows why an efficient implementation of those products is of importance, such that it is computed in parallel and can be treated as being of constant complexity. The computational cost for the stray field can be neglected here since in essence it is $\mathcal{O}(E|\mathcal{S}|)$, assuming constant cost for the ELM and Taylor approximation.

6.3.3.2 L-BFGS Optimizer

Although the trust region framework works reliably for energy minimization, the high computational cost to solve the subproblem is problematic. Also, if the parameters are already close to the optimum, the method struggles to find a good optimum in the flat energy landscape, leading to slow convergence. Therefore, line

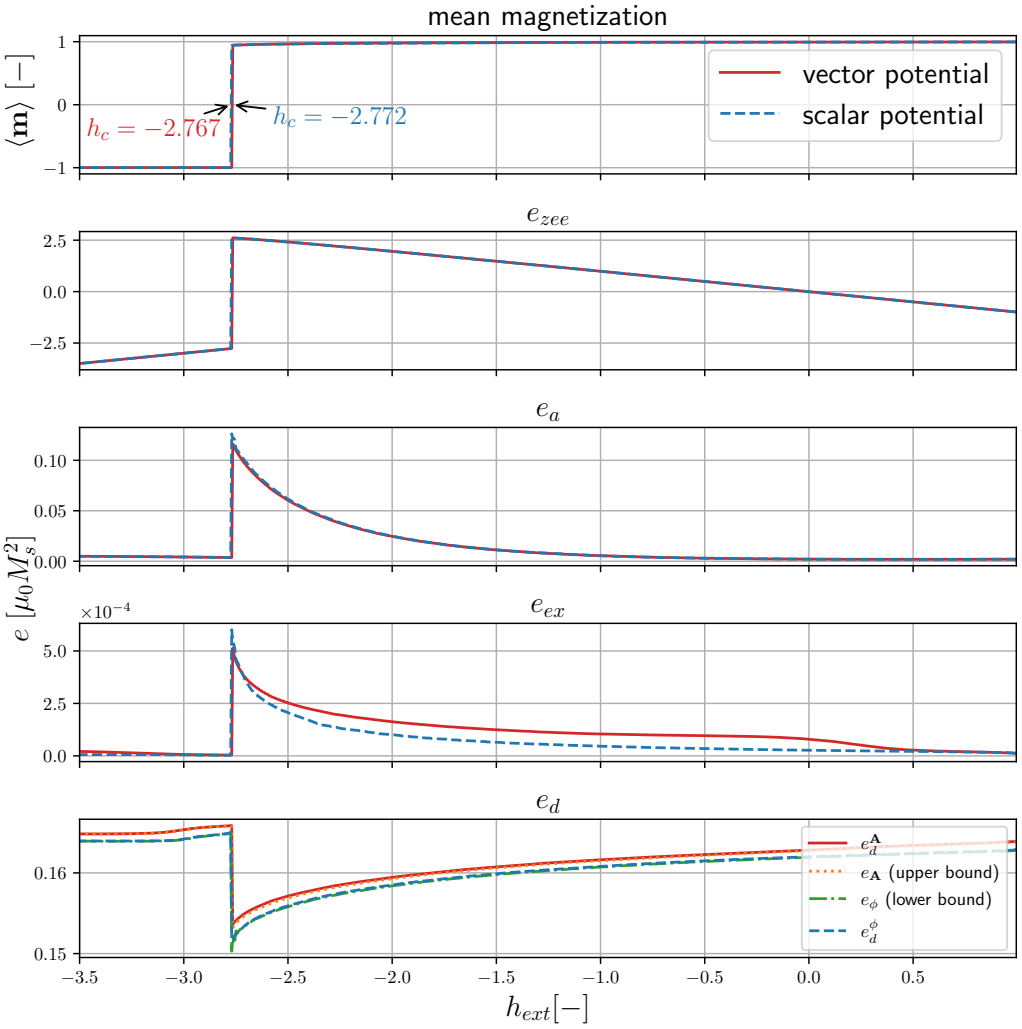


Figure 6.13: Demagnetization curve of a 70nm $Nd_2Fe_{14}B$ cube. The magnetization was projected onto the axis $(1, 0, 10)^T$ of the applied field. The computation was performed with a hard constrained ELM model for scalar and vector potential.

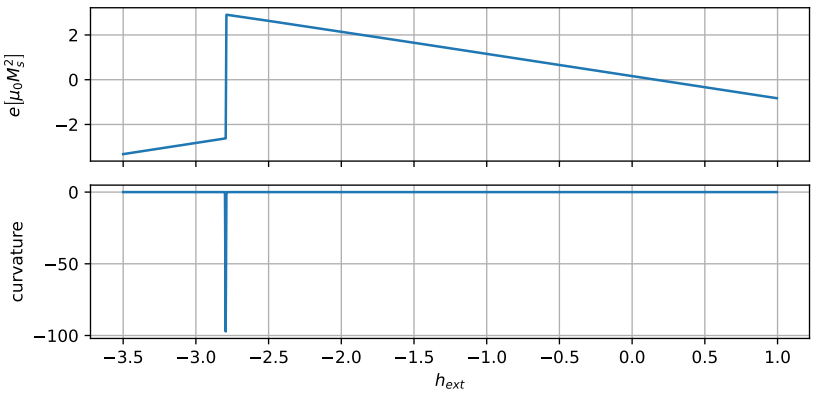


Figure 6.14: Total energy at the end of each optimization step at the top and recorded minimum curvature $\boldsymbol{p}^T H_{\boldsymbol{\theta}} e \boldsymbol{p}$ during the optimization at the bottom.

search methods, especially L-BFGS [61], seem to be more suitable for this task. The same test was performed with an L-BFGS optimizer and the same ELM ansatz for scalar and vector potential. While for the trust region method, the scalar and vector potential was evaluated every iteration, $N_{\boldsymbol{m}} = 20$ L-BFGS iterations are performed before re-evaluating the potential. Figure 6.15 shows the result of the computation. The switching occurred at $h_c = -2.772$ for the scalar potential ansatz and at $h_c = -2.802$ for the vector potential, while the computation took only about 90 s. This shows that also a positive definite Hessian approximation with line search can capture the switching. However, occasionally, the switching might not be completed after a single optimization step, which can cause the optimizer to stall since the history does not match. The information saved in the history of the previous iterates are a very poor approximation of the Hessian. This requires a restart of the optimizer, which is also necessary after the switching is completed. The trust region method seems to be the better choice to compute the switching field accurately, whereas L-BFGS outperforms the trust region method in terms of computation time. This suggests that both optimizers could be applied together. This could be done by switching to the trust region scheme close to the switching point. Another possibility would be the usage of a sampled version of L-BFGS as in [5] which should eliminate this issue. It might also be possible to intelligently sample the respective objective function during line search.

6.3.3.3 PINN ansatz

The same test was performed with a hard constrained PINN ansatz for scalar and vector potential. The model was the same as for the magnetization in Table 6.5 (the scalar potential has scalar output). Minimization was performed with Algorithm 5 for the scalar potential and the respective version for the vector potential and trust

region optimization. The parameters were $N_{\boldsymbol{m}} = 1$, $N_{\phi_1} = N_{\boldsymbol{A}_1} = 20$, $\epsilon_{\boldsymbol{m}} = 10^{-3}$ and $\epsilon_{\phi_1} = \epsilon_{\boldsymbol{A}_1} = 5 \times 10^{-2}$. Figure 6.16 shows the result for this version. The switching field is the same for the scalar potential as in the last trial, but is slightly off for the vector potential, $h_c = -2.782$. The plot for the self energy shows that the upper bound is slightly lower than the computed energy with the vector potential. This could explain why the result was worse. The vector potential might not be accurate enough and one would need to decrease $\epsilon_{\boldsymbol{A}_1}$ or use another network architecture to improve the result.

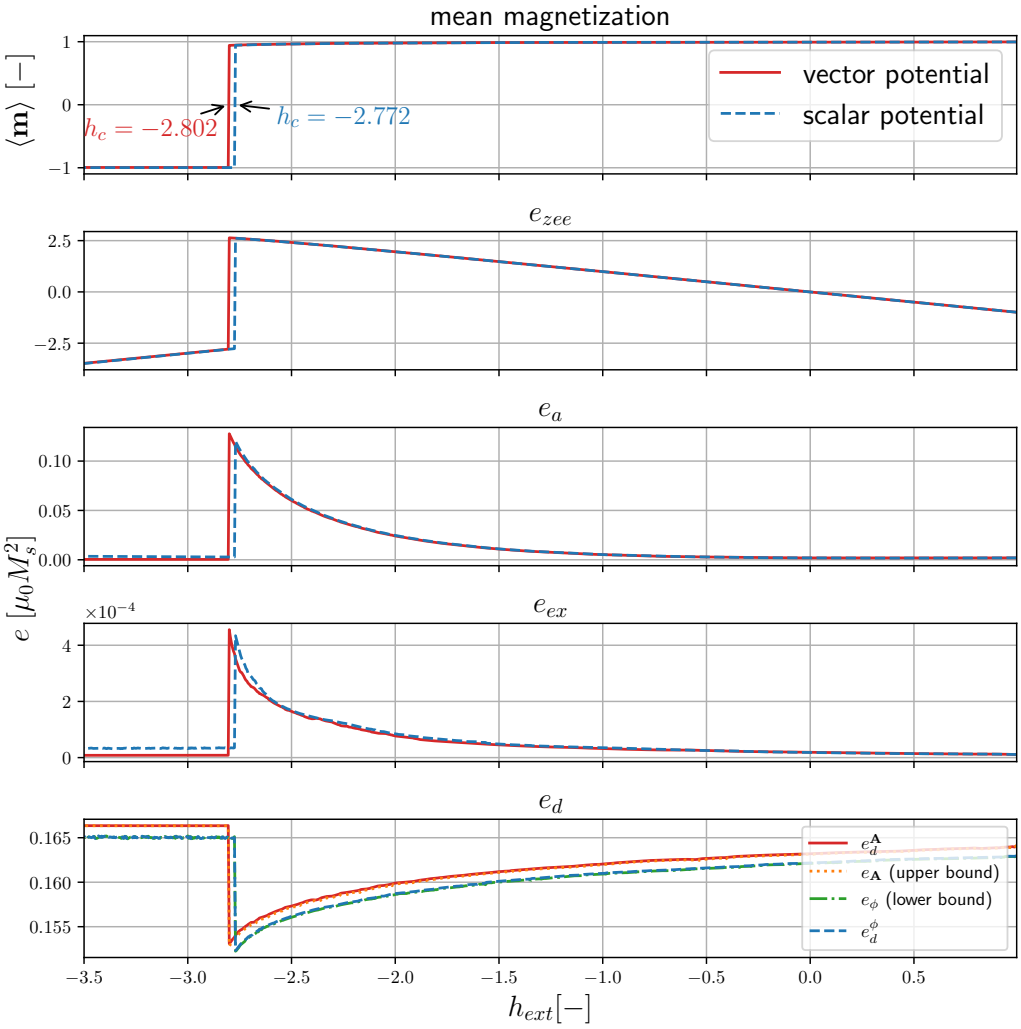


Figure 6.15: Demagnetization curve of a 70nm $Nd_2Fe_{14}B$ cube computed with an L-BFGS optimizer. The magnetization was projected onto the axis $(1, 0, 10)^T$ of the applied field. The computation was performed with a hard constrained ELM model for scalar and vector potential.

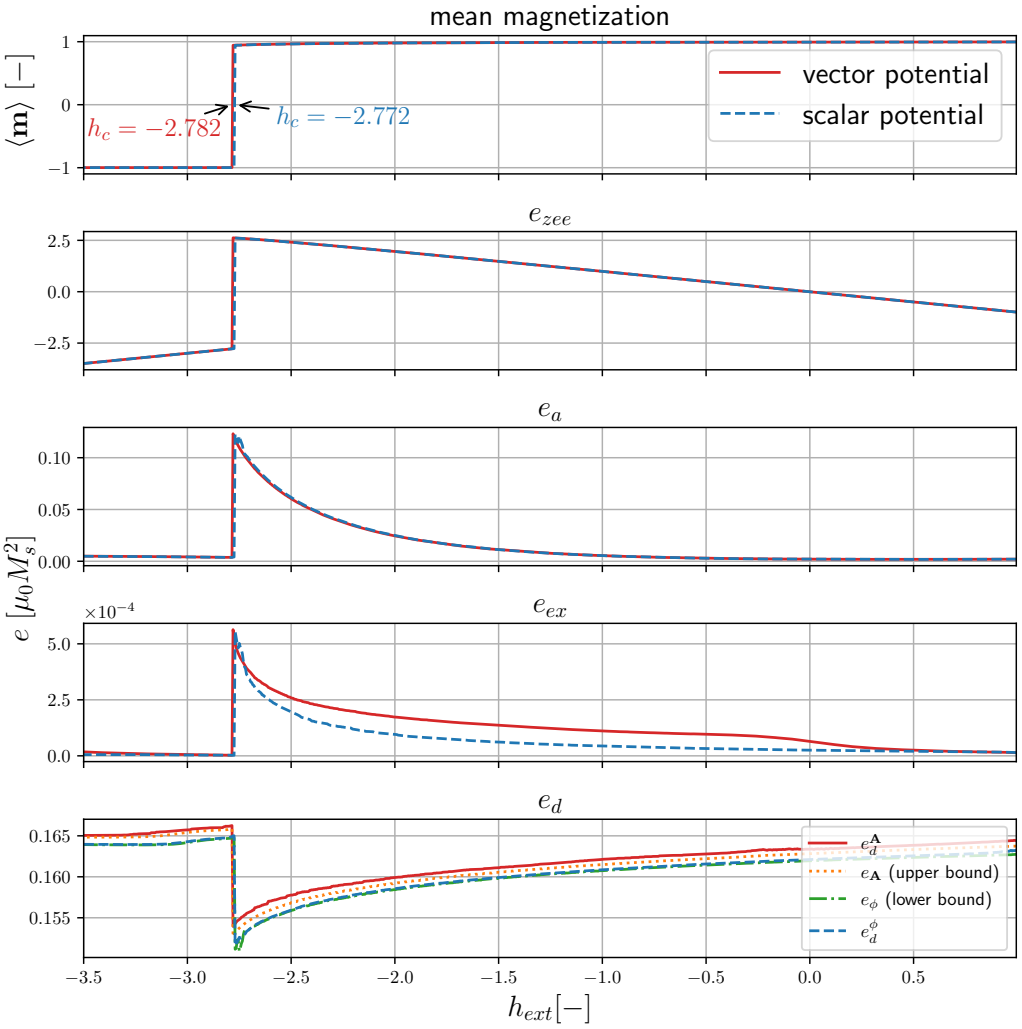


Figure 6.16: Demagnetization curve of a 70nm $Nd_2Fe_{14}B$ cube. The magnetization was projected onto the axis $(1, 0, 10)^T$ of the applied field. The computation was performed with a hard constrained PINN model for scalar and vector potential.

7 Conclusion

This thesis explores various neural network methodologies, focusing primarily on Physics Informed Neural Networks (PINNs). PINNs emerged as a powerful tool due to their ability to incorporate physical laws directly into the learning process, thus enabling efficient and accurate solutions without the need for extensive supervised data. In the beginning, the foundational concepts of neural networks are explored, and it is described how they can be tailored to solve certain time-independent PDEs such as the Poisson equation. This development is enhanced with hard constrained, PINNs which remove required constraints from the optimization problem. Therefore, optimization becomes easier, and it is possible to achieve much higher accuracies which can compete with traditional numerical solvers such as the finite element method (FEM) or the finite difference method (FDM). Further, optimization is performed mesh-free, without the need of any complex discretization scheme and hence, there is no restriction to a minimum mesh size. The non-linear models allow for a highly flexible optimization, which can also incorporate additional conditional parameters and represent the solution with a reduced order model. Traditional numerical models grow exponentially in size to represent the solution, but the reduced order models break this *curse of dimensionality*.

Extreme Learning Machines (ELMs) were introduced as a complementary technique, particularly useful in reducing the complexity of learning tasks by transforming them into linear least squares problems. Further, they also allow incorporation of domain knowledge via a hard constrained ELM ansatz, which further allows the pre-computation of the solution operator. This can be seen as a kind of operator network and also connections to Kernel Ridge regressions (KRRs) are drawn.

Constructive Solid Geometry (CSG) using R-functions is presented as a robust framework for modeling complex geometries within the context of hard constrained PINNs. This approach allows for the exact satisfaction of essential boundary conditions, which is critical for accurately simulating physical systems. By leveraging R-functions, we demonstrated how intricate geometrical constraints can be seamlessly integrated into the PINN framework, ensuring that the neural network outputs remain physically plausible and adhere to the specified boundaries. It is shown that the hard constrained PINN ansatz is complete, in the sense that the ansatz can approximate the solution of the PDE up to any accuracy, provided that the network has required capacity. Last but not least, other systems of R-functions, such as blending R-functions, are introduced, and it is explained how such a function can even be constructed by a sampling approach.

Since the accurate evaluation of the magnetostatic self-energy is crucial for numerical simulations, the thesis emphasizes the significance and challenges associated

with computation of magnetic fields. With the splitting ansatz proposed by Garcia-Cervera and Roma, energy bounds on the finite domain are derived. By decomposing the problem into manageable sub-tasks, the splitting approach facilitates the efficient calculation of the scalar or vector potential and the subsequent minimization of magnetostatic self-energy. The alternating scheme is developed which allows the joint optimization of the magnetization and the respective magnetic fields, making it a valuable technique for micromagnetic simulations.

The culmination of this work was the application of the hard constrained PINN ansatz to the full 3D minimization of Gibbs free energy. By incorporating the Cayley transform, we ensure that the neural network outputs remained on the Lie group of rotation matrices $SO(3)$, which is essential for maintaining the physical integrity of the magnetization configuration. This innovative approach enabled us to model continuous magnetization distributions accurately while minimizing the total Gibbs free energy, including contributions from exchange energy, anisotropy energy, and magnetostatic self-energy. An alternating scheme was applied for the computation of the solution of the *NIST μMAG Standard Problem #3*, where exchange length was included as a conditional parameter, and the PINN was able to learn and represent the magnetic configuration for the full conditional domain with a low-parametric model. Energy minimization was then utilized to perform the computation of a demagnetization process for a hard magnetic sphere and cube. ELMs and PINNs are used to model scalar or vector potential, showing the versatility of the approach.

This thesis makes several significant contributions to the field of computational magnetostatics. The utilization of Automatic Differentiation (AD) allows the training of highly efficient models. By integrating PINNs with hard constraints and advanced geometric modeling techniques, we developed a robust framework for solving complex physical problems. The application of the Cayley transform within the PINN framework represents a novel approach that ensures the physical plausibility of the solutions, thereby enhancing the reliability of the simulations. The methods and techniques developed in this work have broad applicability, extending beyond magnetostatics to other areas of physics and engineering where accurate and efficient modeling of physical systems is required.

The research presented in this thesis opens several avenues for future exploration. Further refinement of the hard constrained PINN approach could involve the integration of additional physical constraints, such as Neumann boundary conditions, and the exploration of more sophisticated neural network architectures. Expanding the application of R-functions to other types of boundary conditions and physical problems could provide new insights and enhance the versatility of the proposed framework. Additionally, the computational techniques developed for magnetostatics could be adapted and applied to other fields, such as fluid dynamics and elasticity, where similar challenges in modeling and computation are encountered. The main challenge lies in non-linear numerical optimization. Hence, novel optimization schemes, which allow fast, and accurate training are of particular interest.

In conclusion, this thesis demonstrates the power and potential of combining advanced neural network techniques with rigorous physical modeling frameworks. The

innovations presented here pave the way for more accurate, efficient, and versatile simulations of complex physical systems, contributing to the ongoing advancement of computational science and engineering.

Bibliography

[1] C. Abert, L. Exl, F. Bruckner, A. Drews, and D. Suess. "magnum. fe: A micromagnetic finite-element simulation code based on FEniCS". In: *Journal of Magnetism and Magnetic Materials* 345 (2013), pp. 29–35.

[2] C. Abert, L. Exl, G. Selke, A. Drews, and T. Schrefl. "Numerical methods for the stray-field calculation: A comparison of recently developed algorithms". In: *Journal of Magnetism and Magnetic Materials* 326 (2013), pp. 176–185.

[3] P. Asselin and A. Thiele. "On the field Lagrangians in micromagnetics". In: *IEEE transactions on magnetics* 22.6 (1986), pp. 1876–1880.

[4] M. Bashir, T. Schrefl, J. Dean, A. Goncharov, G. Hrkac, D. Allwood, and D. Suess. "Head and bit patterned media optimization at areal densities of 2.5 Tbit/in2 and beyond". In: *Journal of Magnetism and Magnetic Materials* 324.3 (2012), pp. 269–275.

[5] A. S. Berahas, M. Jahani, and M. Takác. "Quasi-Newton methods for deep learning: Forget the past, just sample". In: *arXiv preprint arXiv:1901.09997* 16 (2019).

[6] R. Bjørk, E. B. Poulsen, K. K. Nielsen, and A. R. Insinga. "MagTense: A micromagnetic framework using the analytical demagnetization tensor". In: *Journal of Magnetism and Magnetic Materials* 535 (2021), p. 168057.

[7] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert. "Efficient and modular implicit differentiation". In: *Advances in neural information processing systems* 35 (2022), pp. 5230–5242.

[8] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.

[9] D. Braess and W. Hackbusch. "On the efficient computation of high-dimensional integrals and the approximation by exponential sums". In: *Multiscale, Nonlinear and Adaptive Approximation: Dedicated to Wolfgang Dahmen on the Occasion of his 60th Birthday*. Springer, 2009, pp. 39–74.

[10] W. F. Brown. "Magnetostatic principles in ferromagnetism". In: *Amsterdam* 112 (1962).

[11] W. F. Brown. "Micromagnetics". In: *Interscience Publishers, Wiley & Sons, New York* (1963).

[12] J. Bu and A. Karpatne. “Quadratic residual networks: A new class of neural networks for solving forward and inverse problems in physics involving pdes”. In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 675–683.

[13] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. “Physics-informed neural networks (PINNs) for fluid mechanics: A review”. In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738.

[14] C. Carstensen and E. P. Stephan. “Adaptive coupling of boundary elements and finite elements”. In: *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique* 29.7 (1995), pp. 779–817.

[15] S. W. Cho, J. Y. Lee, and H. J. Hwang. *GraphDeepONet: Learning to simulate time-dependent partial differential equations using graph neural network and deep operator network*. 2024.

[16] G. Constantine and T. Savits. “A multivariate Faa di Bruno formula with applications”. In: *Transactions of the American Mathematical Society* 348.2 (1996), pp. 503–520.

[17] M. d’Aquino, C. Serpico, G. Bertotti, T. Schrefl, and I. Mayergoyz. “Spectral micromagnetic analysis of switching processes”. In: *Journal of Applied Physics* 105.7 (2009), p. 07D540.

[18] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in neural information processing systems* 27 (2014).

[19] C. Dyken and M. S. Floater. “Transfinite mean value interpolation”. In: *Computer Aided Geometric Design* 26.1 (2009), pp. 117–134.

[20] L. Exl, W. Auzinger, S. Bance, M. Gusenbauer, F. Reichel, and T. Schrefl. “Fast stray field computation on tensor grids”. In: *Journal of Computational Physics* 231.7 (2012), p. 2840.

[21] L. Exl. “A magnetostatic energy formula arising from the L2-orthogonal decomposition of the stray field”. In: *Journal of Mathematical Analysis and Applications* 467.1 (2018), pp. 230–237.

[22] L. Exl. “Tensor grid methods for micromagnetic simulations”. PhD thesis. Vienna University of Technology, 2014.

[23] L. Exl, S. Bance, F. Reichel, T. Schrefl, H. Peter Stimming, and N. J. Mauser. “LaBonte’s method revisited: An effective steepest descent method for micromagnetic energy minimization”. In: *Journal of Applied Physics* 115.17 (2014), p. 17D118.

[24] L. Exl, J. Fischbacher, A. Kovacs, H. Oezelt, M. Gusenbauer, and T. Schrefl. “Preconditioned nonlinear conjugate gradient method for micromagnetic energy minimization”. In: *Computer Physics Communications* 235 (2019), pp. 179–186.

[25] L. Exl, N. J. Mauser, S. Schaffer, T. Schrefl, and D. Suess. “Prediction of magnetization dynamics in a reduced dimensional feature space setting utilizing a low-rank kernel method”. In: *Journal of Computational Physics* 444 (2021), p. 110586.

[26] L. Exl, N. J. Mauser, T. Schrefl, and D. Suess. “Learning time-stepping by nonlinear dimensionality reduction to predict magnetization dynamics”. In: *Communications in Nonlinear Science and Numerical Simulation* 84 (2020), p. 105205.

[27] L. Exl, N. J. Mauser, T. Schrefl, and D. Suess. “The extrapolated explicit midpoint scheme for variable order and step size controlled integration of the Landau–Lifschitz–Gilbert equation”. In: *Journal of Computational Physics* 346 (2017), pp. 14–24.

[28] L. Exl and T. Schrefl. “Non-uniform FFT for the finite element computation of the micromagnetic scalar potential”. In: *Journal of Computational Physics* 270 (2014), pp. 490–505.

[29] L. Exl, D. Suess, and T. Schrefl. “Micromagnetism”. In: *Handbook of Magnetism and Magnetic Materials*. Ed. by M. Coey and S. Parkin. Cham: Springer International Publishing, 2020, pp. 1–44. ISBN: 978-3-030-63101-7. DOI: 10.1007/978-3-030-63101-7_7-1.

[30] J. Fischbacher, A. Kovacs, H. Oezelt, T. Schrefl, L. Exl, J. Fidler, D. Suess, N. Sakuma, M. Yano, A. Kato, et al. “Nonlinear conjugate gradient methods in micromagnetics”. In: *AIP Advances* 7.4 (2017), p. 045310.

[31] J. Fischbacher, A. Kovacs, M. Gusenbauer, H. Oezelt, L. Exl, S. Bance, and T. Schrefl. “Micromagnetics of rare-earth efficient permanent magnets”. In: *Journal of Physics D: Applied Physics* 51.19 (2018), p. 193002.

[32] D. R. Fredkin and T. R. Koehler. “Hybrid method for computing demagnetizing fields”. In: *IEEE Transactions on Magnetics* 26.2 (1990), pp. 415–417.

[33] P. Gangl, U. Langer, A. Laurain, H. Meftahi, and K. Sturm. “Shape optimization of an electric motor subject to nonlinear magnetostatics”. In: *SIAM Journal on Scientific Computing* 37.6 (2015), B1002–B1025.

[34] C. J. Garcia-Cervera and A. M. Roma. “Adaptive mesh refinement for micromagnetics simulations”. In: *IEEE transactions on magnetics* 42.6 (2006), pp. 1648–1654.

[35] R. Gower, D. Kovalev, F. Lieder, and P. Richtárik. “RSN: randomized subspace Newton”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[36] J. H. Halton. “On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals”. In: *Numerische Mathematik* 2 (1960), pp. 84–90.

[37] R. Hertel and H. Kronmüller. “Finite element calculations on the single-domain limit of a ferromagnetic cube—a solution to μMAG Standard Problem No. 3”. In: *Journal of magnetism and magnetic materials* 238.2-3 (2002), pp. 185–199.

[38] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[39] G.-B. Huang and L. Chen. “Convex incremental extreme learning machine”. In: *Neurocomputing* 70.16-18 (2007), pp. 3056–3062.

[40] G.-B. Huang, L. Chen, C. K. Siew, et al. “Universal approximation using incremental constructive feedforward networks with random hidden nodes”. In: *IEEE Trans. Neural Networks* 17.4 (2006), pp. 879–892.

[41] G.-B. Huang, D. H. Wang, and Y. Lan. “Extreme learning machines: a survey”. In: *International journal of machine learning and cybernetics* 2 (2011), pp. 107–122.

[42] C. Huber, C. Abert, F. Bruckner, M. Groenefeld, S. Schuschnigg, I. Teliban, C. Vogler, G. Wautischer, R. Windl, and D. Suess. “3D printing of polymer-bonded rare-earth magnets with a variable magnetic compound fraction for a predefined stray field”. In: *Scientific reports* 7.1 (2017), pp. 1–8.

[43] A. Hubert and R. McMichael. *μMAG Standard Problem #3*. https://www.ctcms.nist.gov/~rdm/mumag.org.html.

[44] B. James, S. Pollok, I. Peis, J. Frellsen, and R. Bjørk. “Scalable physical source-to-field inference with hypernetworks”. In: *arXiv preprint arXiv: 2405.05981* (2024).

[45] K. Kashinath, M. Mustafa, A. Albert, J. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, et al. “Physics-informed machine learning: case studies for weather and climate modelling”. In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200093.

[46] K. M. Koch, X. Papademetris, D. L. Rothman, and R. A. De Graaf. “Rapid calculations of susceptibility-induced magnetostatic field perturbations for in vivo magnetic resonance”. In: *Physics in Medicine & Biology* 51.24 (2006), p. 6381.

[47] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, D. Praetorius, D. Suess, et al. “Magnetostatics and micromagnetics with physics informed neural networks”. In: *Journal of Magnetism and Magnetic Materials* 548 (2022), p. 168951.

[48] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, et al. “Conditional physics informed neural networks”. In: *Communications in Nonlinear Science and Numerical Simulation* 104 (2022), p. 106041.

[49] A. Kovacs, J. Fischbacher, H. Oezelt, M. Gusenbauer, L. Exl, F. Bruckner, D. Suess, and T. Schrefl. “Learning magnetization dynamics”. In: *Journal of Magnetism and Magnetic Materials* 491 (2019), p. 165548.

[50] P. S. Krishnaprasad and X. Tan. “Cayley transforms in micromagnetics”. In: *Physica B: Condensed Matter* 306.1-4 (2001), pp. 195–199.

[51] H. Kronmüller. “General micromagnetic theory”. In: *Handbook of magnetism and advanced magnetic materials* (2007).

[52] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. “Neural operator: Graph kernel network for partial differential equations”. In: *arXiv preprint arXiv:2003.03485* (2020).

[53] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. “On the variance of the adaptive learning rate and beyond”. In: *arXiv preprint arXiv:1908.03265* (2019).

[54] W. E. Lorensen and H. E. Cline. “Marching cubes: a high resolution 3D surface construction algorithm”. In: *Seminal Graphics: Pioneering Efforts That Shaped the Field, Volume 1*. New York, NY, USA: Association for Computing Machinery, 1998, pp. 347–353. ISBN: 158113052X.

[55] I. Loshchilov and F. Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[56] L. Lu, P. Jin, and G. E. Karniadakis. “Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *arXiv preprint arXiv:1910.03193* (2019).

[57] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson. “Physics-informed neural networks with hard constraints for inverse design”. In: *SIAM Journal on Scientific Computing* 43.6 (2021), B1105–B1132.

[58] J. E. Miltat and M. J. Donahue. “Numerical micromagnetics: Finite difference methods”. In: *Handbook of magnetism and advanced magnetic materials* (2007).

[59] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[60] M. A. Nabian, R. J. Gladstone, and H. Meidani. “Efficient training of physics-informed neural networks via importance sampling”. In: *Computer-Aided Civil and Infrastructure Engineering* 36.8 (2021), pp. 962–977.

[61] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[62] OpenSCAD. *OpenSCAD*. https://openscad.org. Accessed: 2023-15-09.

[63] B. A. Pearlmutter. “Fast exact multiplication by the Hessian”. In: *Neural computation* 6.1 (1994), pp. 147–160.

[64] S. Perna, F. Bruckner, C. Serpico, D. Suess, and M. d’Aquino. “Computational micromagnetics based on normal modes: Bridging the gap between macrospin and full spatial discretization”. In: *Journal of Magnetism and Magnetic Materials* 546 (2022), p. 168683.

[65] S. Pollok, N. Olden-Jørgensen, P. S. Jørgensen, and R. Bjørk. “Magnetic field prediction using generative adversarial networks”. In: *Journal of Magnetism and Magnetic Materials* 571 (2023), p. 170556.

[66] A. Radhakrishnan, D. Beaglehole, P. Pandit, and M. Belkin. “Mechanism of feature learning in deep fully connected networks and kernel machines that recursively learn features”. In: *arXiv preprint arXiv:2212.13881* (2022).

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

[68] M. Raissi, A. Yazdani, and G. E. Karniadakis. “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science* 367.6481 (2020), pp. 1026–1030.

[69] P. Rathore, W. Lei, Z. Frangella, L. Lu, and M. Udell. “Challenges in training PINNs: A loss landscape perspective”. In: *arXiv preprint arXiv:2402.01868* (2024).

[70] C. Rümenapp, B. Gleich, and A. Haase. “Magnetic nanoparticles in magnetic resonance imaging and diagnostics”. In: *Pharmaceutical research* 29 (2012), pp. 1165–1179.

[71] V. Rvachev. *Theory of R-functions and Some Applications*. 1982.

[72] V. L. Rvachev and T. I. Sheiko. *R-functions in boundary value problems in mechanics*. 1995.

[73] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov. “On completeness of RFM solution structures”. In: *Computational Mechanics* 25.2-3 (2000), pp. 305–317.

[74] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov. “Transfinite interpolation over implicitly defined sets”. In: *Computer aided geometric design* 18.3 (2001), pp. 195–220.

[75] S. A. Sauter and C. Schwab. *Boundary element methods*. Springer, 2011.

[76] S. Schaffer and L. Exl. “Constraint free physics-informed machine learning for micromagnetic energy minimization”. In: *Computer Physics Communications* 300 (2024), p. 109202.

[77] S. Schaffer, N. J. Mauser, T. Schrefl, D. Suess, and L. Exl. “Machine learning methods for the prediction of micromagnetic magnetization dynamics”. In: *IEEE Transactions on Magnetics* 58.2 (2021), pp. 1–6.

[78] S. Schaffer, T. Schrefl, H. Oezelt, A. Kovacs, L. Breth, N. J. Mauser, D. Suess, and L. Exl. “Physics-informed machine learning and stray field computation with application to micromagnetic energy minimization”. In: *Journal of Magnetism and Magnetic Materials* 576 (2023), p. 170761.

[79] S. Schaffer, T. Schrefl, H. Oezelt, N. J. Mauser, and L. Exl. “Physics aware machine learning for micromagnetic energy minimization: recent algorithmic developments”. In: *Computer Physics Communications, special issue: Advances in physics aware machine learning (submitted)* (2024).

[80] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, and D. Mortari. “Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations”. In: *Neurocomputing* 457 (2021), pp. 334–356.

[81] T. Schrefl, G. Hrkac, S. Bance, D. Suess, O. Ertl, and J. Fidler. “Numerical methods in micromagnetics (finite element method)”. In: *Handbook of magnetism and advanced magnetic materials* (2007).

[82] V. Shapiro. “Semi-analytic geometry with R-functions”. In: *ACTA numerica* 16 (2007), pp. 239–303.

[83] V. Shapiro. *Theory of R-functions and applications: A primer*. Tech. rep. Cornell University, 1991.

[84] V. Shapiro and I. Tsukanov. “Implicit functions with guaranteed differential properties”. In: *Proceedings of the fifth ACM symposium on Solid modeling and applications*. 1999, pp. 258–269.

[85] A. Sharma, S. Singh, and S. Ratna. “Graph neural network operators: a review”. In: *Multimedia Tools and Applications* 83.8 (2024), pp. 23413–23436.

[86] P. Sharma, L. Evans, M. Tindall, and P. Nithiarasu. “Hyperparameter selection for physics-informed neural networks (PINNs)–Application to discontinuous heat conduction problems”. In: *Numerical Heat Transfer, Part B: Fundamentals* (2023), pp. 1–15.

[87] H. Sheng and C. Yang. “PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries”. In: *Journal of Computational Physics* 428 (2021), p. 110085.

[88] Y. Shin, Z. Zhang, and G. E. Karniadakis. “Error estimates of residual minimization using neural networks for linear PDEs”. In: *Journal of Machine Learning for Modeling and Computing* 4.4 (2023).

[89] I. M. Sobol. “Uniformly distributed sequences with an additional uniform property”. In: *USSR Computational Mathematics and Mathematical Physics* 16.5 (1976), pp. 236–242.

[90] D. Suess, A. Bachleitner-Hofmann, A. Satz, H. Weitensfelder, C. Vogler, F. Bruckner, C. Abert, K. Prügl, J. Zimmer, C. Huber, et al. “Topologically protected vortex structures for low-noise magnetic sensors with high linear range”. In: *Nature Electronics* 1.6 (2018), pp. 362–370.

[91] N. Sukumar and A. Srivastava. “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333.

[92] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. “Fourier features let networks learn high frequency functions in low dimensional domains”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7537–7547.

[93] S. Wang, Y. Teng, and P. Perdikaris. “Understanding and mitigating gradient flow pathologies in physics-informed neural networks”. In: *SIAM Journal on Scientific Computing* 43.5 (2021), A3055–A3081.

[94] S. Wang, H. Wang, and P. Perdikaris. “On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 384 (2021), p. 113938.

[95] S. Wang, X. Yu, and P. Perdikaris. “When and why PINNs fail to train: A neural tangent kernel perspective”. In: *Journal of Computational Physics* 449 (2022), p. 110768.

[96] C. Williams and M. Seeger. “Using the Nyström method to speed up kernel machines”. In: *Advances in neural information processing systems* 13 (2000).

[97] G. B. Wright. *Radial basis function interpolation: numerical and analytical developments*. University of Colorado at Boulder, 2003.

[98] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. “A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), p. 115671.

[99] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[100] Z. Xiang, W. Peng, X. Liu, and W. Yao. “Self-adaptive loss balanced Physics-informed neural networks”. In: *Neurocomputing* 496 (2022), pp. 11–34.

[101] B. Yu et al. “The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems”. In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12.

[102] E. Zhang, M. Dao, G. E. Karniadakis, and S. Suresh. “Analyses of internal structures and defects in materials using physics-informed neural networks”. In: *Science advances* 8.7 (2022), eabk0644.

[103] S. Zhou, X. Liu, Q. Liu, S. Wang, C. Zhu, and J. Yin. “Random Fourier extreme learning machine with $\ell 2$, 1-norm regularization”. In: *Neurocomputing* 174 (2016), pp. 143–153.

A Automatic Differentiation

Automatic Differentiation (AD) is a computational technique to efficiently and accurately evaluate derivatives of programs or computational graphs. Unlike numerical differentiation, which introduces errors in the form of round-off errors, truncation errors, and discretization errors, AD exploits the chain rule to compute exact derivatives with high numerical precision. This makes AD invaluable in various fields such as optimization, machine learning, and scientific computing. By automating the process of calculating derivatives, AD enables faster and more reliable gradient-based optimization algorithms, leading to significant advancements in algorithmic efficiency and performance.

For a simple composition of three functions f, g, h

$$\begin{aligned} y = f(g(h(x))) = f(g(h(z_0))) = f(g(z_1)) = f(z_2) = z_3 \\ z_0 = x \\ z_1 = h(z_0) \\ z_2 = g(z_1) \\ z_3 = f(z_2) = y, \end{aligned} \tag{A.1}$$

the chain rule results in

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial x} = \frac{\partial f(z_2)}{\partial z_2}\frac{\partial g(z_1)}{\partial z_1}\frac{\partial h(z_1)}{\partial x}. \tag{A.2}$$

This decomposition can be traversed from the inside to the outside in forward-mode AD or from outside to inside, resulting in reverse-mode AD.

Forward-mode	Reverse-mode
$\frac{\partial f(z_2)}{\partial z_2}\left(\frac{\partial g(z_1)}{\partial z_1}\frac{\partial h(z_1)}{\partial x}\right)$	$\left(\frac{\partial f(z_2)}{\partial z_2}\frac{\partial g(z_1)}{\partial z_1}\right)\frac{\partial h(z_1)}{\partial x}$

During forward-mode AD, each variable z_i is computed together with its derivative $\dot{z}_i$ using the chain rule. A seed $(\dot{z}_1, \dot{z}_2)$ is required for the computation of the derivative, and the result is the directional derivative along $(\dot{z}_1, \dot{z}_2)$. Hence, to compute the gradient ∇f, two evaluations along $(1, 0)$ and $(0, 1)$ are required. Given the computational graph, the derivative of the output $\dot{z}_i$ of the i^{th} node is given by

$$\dot{z}_i = \sum_{j \in \{\text{predecessor of } i\}} \frac{\partial z_i}{\partial z_j}\dot{z}_j. \tag{A.3}$$

The products within the sum can be generalized with Jacobian-vector products. Forward-mode AD is particularly useful for functions $f : \mathbb{R}^m \to \mathbb{R}^n$ if $m \leq n$, since

there are only m evaluations required to compute the full Jacobian. Hence, each pass using forward AD recovers a single column of the Jacobian of f. The whole process can be seen as the Jacobian-vector product $J_f(\boldsymbol{x})\dot{\boldsymbol{x}}_0$ with the seed $\dot{\boldsymbol{x}}_0$.

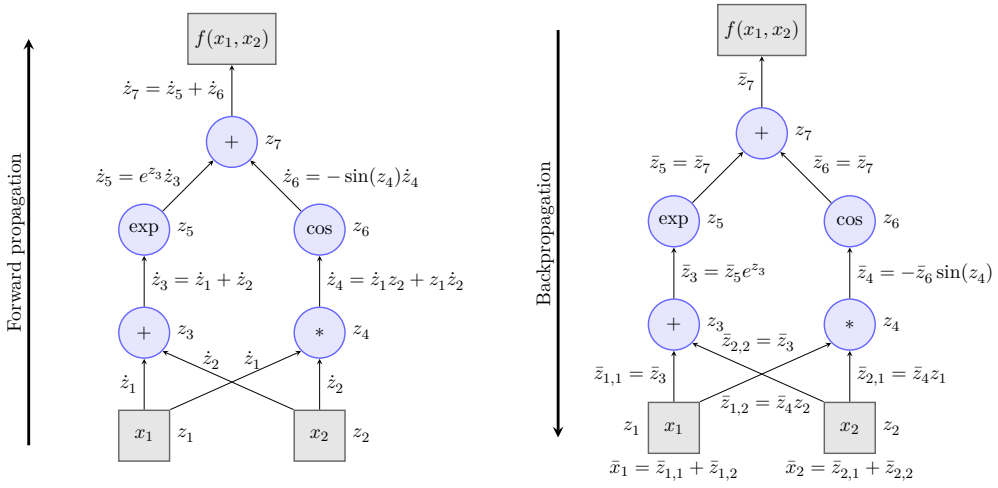
On the other hand, during reverse-mode AD, we require the computation of the *adjoint*, denoted with $\bar{z}_i$. To compute $\bar{z}_i$, of the i^{th} node, the successors of z_i in the computational graph are used

$$\bar{z}_i = \sum_{j \in \{\text{successors of } i\}} \bar{z}_j \frac{\partial z_i}{\partial z_j}. \tag{A.4}$$

Again, this can be generalized with vector-Jacobian products and for each pass one column of the Jacobian can be recovered. Reverse-mode AD computes the vector-Jacobian product $\bar{\boldsymbol{f}}^T J_f(\boldsymbol{x})$ with the seed $\bar{\boldsymbol{f}}$. This computation requires the specification of adjoint $\bar{f}$ of the output which is usually set to 1 for scalar valued functions.

While forward-mode AD can compute the value and the derivative concurrently, reverse-mode AD requires a forward propagation step to compute the value as well as a *backpropagation* step to compute the derivatives. Further, during the forward propagation, intermediate values are stored which are required by the backward propagation. This can make the process costly in regard to storage requirements. In general, for a function with output dimension n, there are n evaluations required to compute the full Jacobian. This is particularly useful for scalar valued functions, such as an objective function like (3.2).

Since both methods create a new computational graph, forward and reverse-mode can be arbitrarily combined to compute higher order derivatives.



(a) Forward mode (b) Reverse mode

Figure A.1: AD of equation (A.5).

Example A.1. *Consider the equation*

$$\begin{aligned} f(x_1, x_2) &= e^{x_1+x_2} + \cos(x_1 x_2) \\ &= e^{z_1+z_2} + \cos(z_1 z_2) \\ &= e^{z_3} + \cos(z_4) \\ &= z_5 + z_6 = z_7. \end{aligned} \tag{A.5}$$

Figure A.1a shows the computational graph to compute the value and derivative using forward-mode AD (compare with (A.3)) and Figure A.1b shows the same graph using reverse-mode AD (compare with (A.4)). Setting $(\dot{z}_1, \dot{z}_2) = (1, 0)$ *computes* $\frac{\partial f}{\partial x_1}$ *and* $(\dot{z}_1, \dot{z}_2) = (0, 1)$ *computes* $\frac{\partial f}{\partial x_2}$*, while for reverse mode,* $\bar{z}_7 = 1$*, gives both partial derivatives in one backpropagation step.*

A.1 Hessian-vector products

Hessian-vector products (HVPs) [63] are important for many optimization algorithms, particularly in second-order optimization methods like Newton's method and its variants.

Computing the entire Hessian matrix can be computationally expensive, especially for large-scale problems. HVPs offer a more efficient alternative by computing the product of the Hessian matrix with a given vector without explicitly constructing the entire matrix. Optimization routines using this approach are often called Hessian free second order methods.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable function with continuous derivatives, i.e., the Hessian matrix is symmetric. The Hessian at some input $\boldsymbol{x}$ is denoted as $H_f(\boldsymbol{x}) = J[J_f](\boldsymbol{x})$. For some vector $\boldsymbol{v} \in \mathbb{R}^n$, the HVP computes

$$\boldsymbol{v} \mapsto H_f(\boldsymbol{x})\boldsymbol{v}. \tag{A.6}$$

To evaluate this product without instantiating the full Hessian, AD of the Jacobian-vector product can be performed, i.e.,

$$H_f(\boldsymbol{x})\boldsymbol{v} = J[\boldsymbol{x} \mapsto J_f(\boldsymbol{x})\boldsymbol{v}]. \tag{A.7}$$

This approach offers several advantages. It reduces memory requirements and computational costs compared to computing the entire Hessian matrix, especially for high-dimensional problems. Further, seamlessly integrates with existing automatic differentiation frameworks, and the HVP can be computed using any combination of forward and reverse-mode AD. Listing A.1 shows the abstract implementation to compute the HVP using forward-over-reverse AD.

Listing A.1: Hessian-vector product in `JAX` using forward-over-reverse mode AD.

```
def hvp(f, primals, tangents):
    return jvp(grad(f), primals, tangents)[1]
```

A.2 Differential operators

Differential operators can be abstractly defined as HOF using forward-mode AD. Listing A.2 shows a basic implementation of the divergence operator. Only a single Jacobian-vector product is required to evaluate the divergence operator. Listing A.3, on the other hand, shows the implementation of the Laplace operator, where the diagonal of the Hessian is extracted using Hessian-vector products.

Listing A.2: Basic implementation of the divergence operator for a vector valued function f, such as the magnetization, using forward mode AD.

```
def divergence(f):
    def div(x, *args, **kwargs):
        g = lambda x: f(x, *args, **kwargs)
        return jvp(g, [x], [ones_like(x)])[1]
    return div
```

Listing A.3: Basic implementation of the vector Laplace operator as HOF using forward mode AD. The function f can have tensor output of arbitrary order (shape). The partial derivatives are extracted from the Hessian by repeated application of Hessian vector products to avoid instantiating the full Hessian matrix.

```
def hessian_diag(f, primals):
    primals = primals.ravel()
    vs = eye(primals.shape[0])
    comp = lambda v: dot(hvp(f, [primals], [v]), v)
    diag_entries = vmap(comp)(vs)
    return diag_entries

def laplace(f):
    def lap(x, *args, **kwargs):
        H_diag = hessian_diag(
            lambda x: f(x, *args, **kwargs), x)
        return jnp.sum(H_diag, axis=0)

    return lap
```

B Trust Region Optimization

Trust region optimization [61] is a powerful technique in nonlinear optimization that iteratively approximates the objective function within a local region around the current estimate, known as the "trust region". This method contrasts with line search methods by focusing on a neighborhood where the model is trusted to be an accurate representation of the objective function. The approach is particularly effective for handling complex, non-convex problems where global optimization methods may struggle.

The fundamental idea behind trust region methods is to construct a simpler model, typically a quadratic approximation of the objective function, and solve this approximate problem within the trust region. The size of the trust region is dynamically adjusted based on the agreement between the model and the actual objective function. If the model accurately predicts the behavior of the objective function, the trust region is expanded; otherwise, it is contracted.

Given an objective function $f(\mathbf{x})$, trust region methods iteratively solve the subproblem:

$$\begin{aligned} \min_{\mathbf{p}} m_k(\mathbf{p}) &= f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \nabla^2 f(\mathbf{x}_k)\mathbf{p} \\ \text{subject to:} \quad & \|\mathbf{p}\| \leq \Delta_k \end{aligned} \tag{B.1}$$

Here, $m_k(\mathbf{p})$ is the quadratic model of the objective function at iteration k, $\mathbf{x}_k$ is the current estimate of the solution, $\nabla f(\mathbf{x}_k)$ is the gradient of the objective function, $\nabla^2 f(\mathbf{x}_k)$ is the Hessian matrix, $\mathbf{p}$ is the step direction, and Δ_k is the radius of the trust region. Figure B.1 illustrates the minimization of the quadratic model in 2d within the trust region framework.

The success of the step $\mathbf{p}$ is evaluated by comparing the actual reduction in the objective function to the predicted reduction given by the model $m_k(\mathbf{p})$. This ratio ρ_k is defined as:

$$\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{p}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k)} \tag{B.2}$$

Based on ρ_k, the trust region radius Δ_k is updated as follows:

$$\Delta_{k+1} = \begin{cases} \gamma_1 \Delta_k & \text{if } \rho_k < \eta_1 \\ \Delta_k & \text{if } \eta_1 \leq \rho_k < \eta_2 \\ \min(\gamma_2 \Delta_k, \Delta_{\max}) & \text{if } \rho_k \geq \eta_2 \end{cases} \tag{B.3}$$

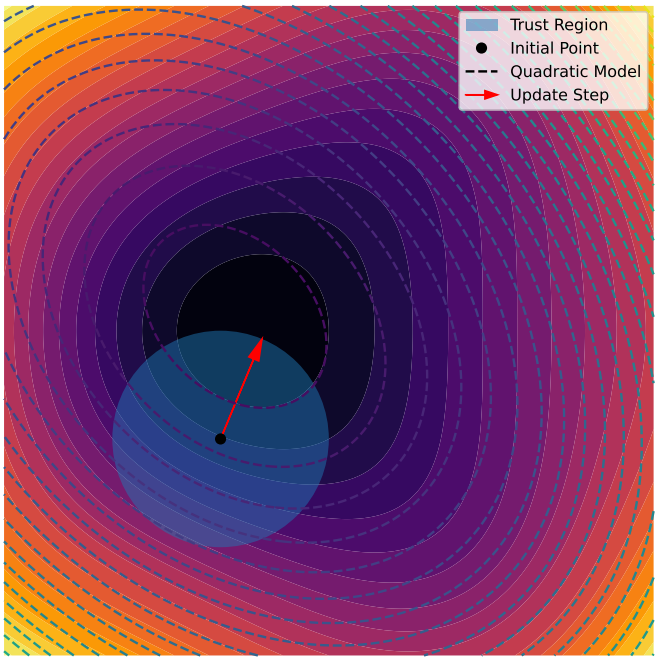


Figure B.1: Illustration of a trust region step.

where $0 < \eta_1 < \eta_2 < 1$, $0 < \gamma_1 < 1 < \gamma_2$, and $\Delta_{\max}$ is the maximum allowable trust region radius. In this work we used $\gamma_1 = \eta_1 = \frac{1}{4}$, $\gamma_2 = 2$, $\eta_2 = \frac{3}{4}$ and $\Delta_{\max} = 2$. A pseudocode for the Trust Region method is given in Algorithm 6.

Common algorithms within the trust region framework include the Cauchy point method, which considers a direction along the steepest descent, and the dogleg method, which combines the steepest descent and Newton's direction. Another notable method is the Steihaug Conjugate Gradient, which is particularly effective for large-scale optimization problems. The pseudocode for the Steihaug-CG method is given in Algorithm 7. For each iteration, an evaluation of the Hessian-vector product $\nabla^2 f(\mathbf{x}_k)\mathbf{d}_j$ is required. The computation of those products is explained in detail in Appendix A.1.

Trust region optimization provides a robust and powerful framework for solving nonlinear optimization problems by iteratively refining local models within a dynamically adjusted region. By balancing the accuracy of the model with the size of the trust region, these methods achieve reliable and efficient convergence for well-conditioned problems. In the case of ill-conditioning, as often present in PINN training, the convergence rate may suffer, but optimization is still very robust. To mitigate ill-conditioning, a damped trust region method can be applied. This is simply done by replacing $\nabla^2 f(\mathbf{x}_k)$ with $\nabla^2 f(\mathbf{x}_k) + \alpha I$, where α denotes the damping factor. However, in our experiments this does not really improve convergence, but can help to converge to a suboptimal solution which might be good enough for the

Algorithm 6 Trust Region Method

Require: Function $f : \mathbb{R}^n \to \mathbb{R}$
  Initialize $\mathbf{x}_0$, Δ_0, and tolerance ϵ
  for $k = 0, 1, 2, \ldots$ **do**
    Compute $\mathbf{p}_k$ by solving the trust region subproblem

$$\min_{\mathbf{p}} \quad m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \nabla^2 f(\mathbf{x}_k)\mathbf{p}$$
$$\text{subject to} \quad \|\mathbf{p}\| \leq \Delta_k$$

    Evaluate $\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{p}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k)}$
    if $\rho_k < \eta_1$ **then**
      $\Delta_{k+1} = \gamma_1 \Delta_k$
    else if $\eta_1 \leq \rho_k < \eta_2$ **then**
      $\Delta_{k+1} = \Delta_k$
    else
      $\Delta_{k+1} = \min(\gamma_2 \Delta_k, \Delta_{\max})$
    end if
    if $\rho_k > \eta_1$ **then**
      $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$
    else
      $\mathbf{x}_{k+1} = \mathbf{x}_k$
    end if
    if $\|\nabla f(\mathbf{x}_{k+1})\| < \epsilon$ **then**
      break
    end if
  end for
  return $\mathbf{x}_{k+1}$

problem setting.

Algorithm 7 Steihaug Conjugate Gradient Method

Require: Function $f : \mathbb{R}^n \to \mathbb{R}$, iterate $\mathbf{x}_k \in \mathbb{R}^n$ and tolerance ϵ
  Initialize $\mathbf{r}_0 = -\nabla f(\mathbf{x}_k)$, $\mathbf{d}_0 = \mathbf{r}_0$, and set $\mathbf{p} = \mathbf{0}$
  for $j = 0, 1, 2, \ldots$ **do**
    if $\mathbf{d}_j^\top \nabla^2 f(\mathbf{x}_k)\mathbf{d}_j \leq 0$ **then**
      Compute τ such that $\|\mathbf{p} + \tau \mathbf{d}_j\| = \Delta_k$
      $\mathbf{p} = \mathbf{p} + \tau \mathbf{d}_j$
      break
    end if
    $\alpha = \frac{\mathbf{r}_j^\top \mathbf{r}_j}{\mathbf{d}_j^\top \nabla^2 f(\mathbf{x}_k)\mathbf{d}_j}$
    if $\|\mathbf{p} + \alpha \mathbf{d}_j\| \geq \Delta_k$ **then**
      Compute τ such that $\|\mathbf{p} + \tau \mathbf{d}_j\| = \Delta_k$
      $\mathbf{p} = \mathbf{p} + \tau \mathbf{d}_j$
      break
    end if
    $\mathbf{p} = \mathbf{p} + \alpha \mathbf{d}_j$
    $\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha \nabla^2 f(\mathbf{x}_k)\mathbf{d}_j$
    if $\|\mathbf{r}_{j+1}\| < \epsilon$ **then**
      break
    end if
    $\beta = \frac{\mathbf{r}_{j+1}^\top \mathbf{r}_{j+1}}{\mathbf{r}_j^\top \mathbf{r}_j}$
    $\mathbf{d}_{j+1} = \mathbf{r}_{j+1} + \beta \mathbf{d}_j$
  end for
  return p