Nonlinear Dynamics and Systems Theory, **22** (4) (2022) 432–446

A Neural Network Approximation for a Model of Micromagnetism

M. Moumni [*] and M. Tilioua

MAIS Laboratory, MAMCS Group, FST Errachidia, Moulay Ismail University of Meknès, P.O. Box: 509 Boutalamine, 52000 Errachidia, Morocco.

Received: January 24, 2022; Revised: September 13, 2022

Abstract: Micromagnetics is a continuum theory describing magnetization patterns inside ferromagnetic media. The dynamics of a ferromagnetic material are governed by the Landau-Lifshitz equation (LL). This equation is highly nonlinear and has a non-convex constraint. In this work, we propose two new algorithms for solving this equation in high dimension, by using deep neural networks. Numerical and comparative tests using TensorFlow illustrate the performance of our algorithms.

Keywords: *data-driven scientific computing; machine learning; numerical analysis; micromagnetism; Landau-Lifshitz equation.*

Mathematics Subject Classification (2010): 78A25, 35Q60, 35B40, 93-10, 70K75.

1 Introduction

Differential equations, including ordinary differential equations (ODEs) and partial differential equations (PDEs), formalize the description of the dynamical nature of the world around us. However, solving these equations is a challenge due to extreme computational cost and because most PDEs do not have an analytical solution, their solution can be approximated using classical numerical methods (which are based on a discretization of the domain) [17], [18], [11]. These methods are particularly efficient for low-dimensional problems on regular geometries; however, finding an appropriate discretization for a complex geometry can be as difficult as solving the partial differential equation itself. This problem is particularly severe if the space dimension is large as there is no straightforward way to discretize irregular domains in space dimensions larger than three. Solving equations is a high-level human intelligence work and a crucial step towards general

[*] Corresponding author: `mailto:md.moumni@gmail.com`

artificial intelligence (AI). Therefore, the obstacle of extreme computational cost in numerical solution may be bypassed by using general AI techniques such as deep learning and reinforcement learning, which were rapidly developed during the last decades. Data used to train the network is randomly sampled within the entire solution domain in each training batch, including initial conditions and boundary conditions.

Recently, deep learning has revolutionized many scientific fields [4], [7], [5]. Including the solution of the differential equation PDEs, R. Maziar [14] proposes a deep learning approach for discovering nonlinear partial differential equations from scattered and potentially noisy observations in space and time. Beck and *et al.* [6] propose a new method for solving high-dimensional fully nonlinear second-order PDEs. The Deep Galerkin Method uses a deep neural network instead of a linear combination of basis functions. The deep neural network is trained to satisfy the differential operator, initial condition, and boundary conditions using stochastic gradient descent at randomly sampled spatial points. In the stochastic framework, Weinan *et al.* [23] propose a new algorithm for solving PDEs and backward stochastic differential equations in high dimension. There are many other works on solving differential equations using the neural network [22], [9], [16]. The ODEs and PDEs are equations which impose relationships between the different partial derivatives of a multivariable function. We ask the following question: if there exists a neural network capable of simultaneously and uniformly approaching a function and its partial derivatives. The answer to this question and the mathematical theory of physics-informed neural networks is already treated by Allan Pinkus in [20]. In this work, we will solve the LL equation in high dimension, using the artificial neural network by two different methods as will be described below. One of the major difficulties that exists for solving this equation by the classical methods is the non-covex contraint, see [2, 3, 17]. We will see that this constraint is not a problem for the neural network approach because we add this constraint in the loss function.

The rest of the paper is structured as follows. In the next section, we present the LL equation arising in micromagnetism and which will be the subject of our investigation. In Section 3, we explore the use of deep learning for solving the PDEs under consideration in micromagnetism in high dimension. To this end, it is necessary to formulate the PDEs as a learning problem. We put forth two distinct classes algorithms of deep learning, and highlight their performance through the lens of different benchmark problems. In fact, we use a deep neural network to approximate the PDE solution with this parameterization, a loss function is set up, and then we train so that the *Loss* function becomes very small. For the training data, the network uses points randomly sampled from the region where the function is defined, and the optimization is performed using gradient descent. We conclude the paper in Section 4 by giving some comments and a direction for future work.

2 The Model Statement

In this paper, we consider the simplified LL equation in which we neglect magnetostatic, anistotropy, and a Zeeman field. To describe the model equations, we consider $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}^*$, a bounded and regular open set. We assume that a ferromagnetic material occupies the domain Ω. The magnetization field of the ferromagnetic material is denoted by $M(x,t)$, where x and t mean the position and time, respectively. Then, the LL model in $Q = \Omega \times (0,T)$ is described by

$$\partial_t M(x,t) = -M(x,t) \times \triangle M(x,t) - \mu M(x,t) \times M(x,t) \times \triangle M(x,t) \text{ in } Q \qquad (1)$$

subject to the initial conditions

$$M(x,0) = M_0(x), \quad |M_0(x)| = 1 \text{ in } \Omega \tag{2}$$

and a periodic boundary condition. Here, $\times$ denotes the exterior product, $M(x,t) = (m_1(x,t), m_2(x,t), m_3(x,t)) \in \mathbb{R}^3$, and $\mu \geqslant 0$ is the damping parameter. Next, we consider the energy-structure

$$E(M(t)) = \|\nabla M(t)\|_{\mathbb{L}^2(\Omega)}.$$

By integration on the equation (1), we obtain the following energy equation:

$$E(M(t)) = E(M(0)) - 2\mu \int_0^t \int_\Omega \| M(x,s) \times \triangle M(x,s) \|^2 \, dxds. \tag{3}$$

For any $t \geqslant 0$, equation (3) implies that the problem has the energy dissipation property for the case $\mu > 0$ and the energy conservation property for the case $\mu = 0$.

If we multiply the LL equation (1) by $M(x,t)$, we obtain the important hypothesis of micromagnetism is that the local magnetization vector must be constant in magnitude

$$|M(x,t)| = |M_0(x)| = 1 \quad \text{for any} \quad t > 0. \tag{4}$$

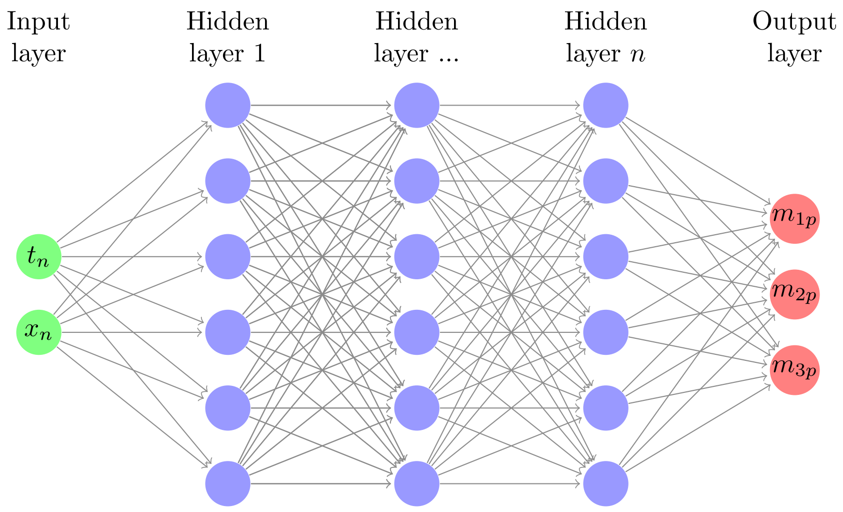
Ferromagnetic materials are very important in industry and modern technology and have been used for fundamental studies and in many everyday applications such as sensors, electric motors, generators, hard disk media, and most recently spintronic memories.

3 Deep Learning Algorithm

The goal is to approximate the solution $M = (m_1, m_2, m_3)$ for the equation (1) by a deep neural network with parameter set {weights,biases}. For this, we will work on two different cases. In the first case, suppose we know the solution at some random points in Q. In the second case, assume we only know the solution at some random points in ∂Q and some random points in $\Omega \times \{0\}$.

3.1 First case

We consider a neuron network composed by several layers such that the first layer represents the inputs (the random points (t^i, x^i) of Q), while the last layer represents the output solution $M_p = (m_{1p}, m_{2p}, m_{3p})$ at the random points (t^i, x^i) linked by the parameters weights and biases, the other layers are hidden layers.



The first step in approximating the solution M at all points of Q is to calculate the objective function *Loss*.

Let N be the number of the random points $(x^i, t^i) \in Q$ at which we know the exact solution. We define

$$(f_{m_1}, f_{m_2}, f_{m_3}) = \frac{\partial M_p(x,t)}{\partial t} + M_p(x,t) \times \triangle M_p(x,t) + \mu M_p(x,t) \times M_p(x,t) \times \triangle M_p(x,t),$$

$$MSE_f = MSE_{f_{m_1}} + MSE_{f_{m_2}} + MSE_{f_{m_3}},$$

$$MSE_M = MSE_{m_1} + MSE_{m_2} + MSE_{m_3}$$

and

$$MSE_{Contraint} = \frac{1}{N}\sum_{i=1}^{N} |(m_1(x^i,t^i))^2 + (m_2(x^i,t^i))^2 + (m_3(x^i,t^i))^2 - 1|^2$$

with

$$MSE_{f_{m_r}} = \frac{1}{N}\sum_{i=1}^{N} |f_{m_r}(x^i,t^i)|^2, \quad MSE_{m_r} = \frac{1}{N}\sum_{i=1}^{N} |m_r(x^i,t^i) - m_{rp}(x^i,t^i)|^2.$$

for $r = 1, 2, 3$. The objective function *Loss* is given by

$$Loss = MSE_f + MSE_M + MSE_{Contraint}.$$

In the next step, we will deduce an iterative gradient algorithms designed to minimize *Loss*. This minimization is achieved by an adequate weight configuration. For this, we will use the limited-memory quasi-Newton code for unconstrained optimization L-BFGS, developed at the Optimization Center, a joint venture of Argonne National Laboratory and Northwestern University by Liu and Nocedal [13]. Numerical and comparative tests using TensorFlow [1] illustrate the performance of our algorithm. More specifically, we apply the following algorithm.

1. Initialize the parameter set {weights,biases}.
2. Generate random samples (t^i, x^i) from Q.
3. Calculate the *Loss* functional for the current mini-batch $s^i = \{(t^i, x^i)\}$.
4. We choose {weights,biases} randomly such that *Loss* becomes minimal.
5. Repeat steps (3)-(4) until *Loss* is very small.

When the *Loss* function becomes small enough, we say that the neuron network has become trained and in this case, we can determine the solution of the equation (1) at any point of Q.

Numerical simulation. For our purpose we consider non-trivial exact solutions [10] for LL equation (1) on Ω. Here, let $\alpha \in \mathbb{R}$, $l \in \mathbb{Z}$ and $k = l\pi$. The exact solution in one-dimensional space is given by

$$M(t,x) = (m_1(t,x), m_2(t,x), m_3(t,x)),$$

where

$$\begin{aligned} m_1(t,x) &= \frac{\sin\alpha\ \cos(kx - \phi(x,t,\alpha,k,\mu))}{d(t,\alpha,k,\mu)}, \\ m_2(t,x) &= \frac{\sin\alpha\ \sin(kx - \phi(x,t,\alpha,k,\mu)}{d(t,\alpha,k,\mu)}, \\ m_3(t,x) &= \frac{\exp(k^2\mu t)\cos\alpha}{d(t,\alpha,k,\mu)}, \end{aligned}$$

with

$$d(t,\alpha,k,\mu) = \sqrt{\sin^2\alpha + \exp(2k^2\mu t)\cos^2\alpha}$$

and

$$\phi(x,t,\alpha,k,\mu) = \frac{1}{\mu}\log\Big(\frac{d(t,\alpha,k,\mu) + \exp(k^2\mu t)\cos\alpha}{1+\cos\alpha}\Big).$$

The exact solution in two-dimensional space is given by

$$M(t,x) = (m_1(t,x,y), m_2(t,x,y), m_3(t,x,y)),$$

where

$$\begin{aligned} m_1(t,x,y) &= \frac{\sin\alpha\ \cos(k(x+y) - \phi(x,t,\alpha,k,\mu)}{d(t,\alpha,k,\mu)}, \\ m_2(t,x,y) &= \frac{\sin\alpha\ \sin(k(x+y) - \phi(x,t,\alpha,k,\mu)}{d(t,\alpha,k,\mu)}, \\ m_3(t,x,y) &= \frac{\exp(2k^2\mu t)\cos\alpha}{d(t,\alpha,k,\mu)}, \end{aligned}$$

with

$$d(t,\alpha,k,\mu) = \sqrt{\sin^2\alpha + \exp(4k^2\mu t)\cos^2\alpha}$$

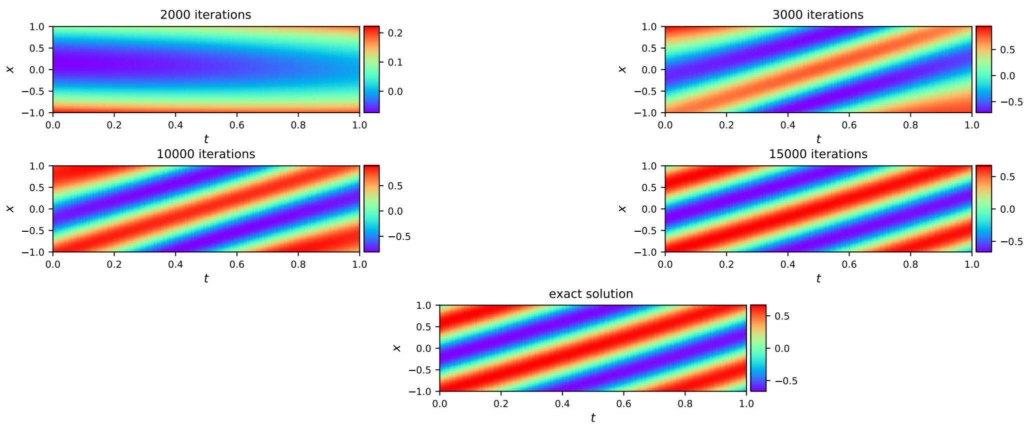


Figure 1: A comparison between the exact solution m_1 and the calculated solution m_{1p}, for different number of iterations and $d = 1$.

and

$$\phi(x, t, \alpha, k, \mu) = \frac{1}{\mu} \log\Big(\frac{d(t, \alpha, k, \mu) + \exp(2k^2 \mu t) \cos\alpha}{1 + \cos\alpha}\Big).$$

We trained the neural network on 2000 random points; which took approximately 5000 and 25000 iterations for the one-dimensional space and two-dimensional space, respectively. After the training we gave to neural network 100×100 for $d = 1$, and $100\times100\times100$ for $d = 2$ we compared the outputs m_{1p}, m_{2p} and m_{3p} with the exact solutions m_1, m_2 and m_3. We observed that they were almost equal, indicating that the neural network has become well trained to find the value of m_1, m_2 and m_3 at each point of the domain Q. The following figures illustrate everything we have said. Figures 1, 2 and 3 propose a comparison between the exact solutions m_1, m_2 and m_3 and the calculated solutions m_{1p}, m_{2p} and m_{3p}, respectively, for different number of iterations and $d = 1$. Figure 4 proposes a comparison between the exact solution m_1 and the calculated solution m_{1p}, for different number of iterations, $t = 0.5$ and $d = 1$. Figures 5 and 6 propose a comparison between the exact solutions m_1 and m_2 and the calculated solutions m_{1p} and m_{2p}, respectively, for different number of iterations, $t = 0.5$ and $d = 2$. We use the parameters $\mu = 0.01$, $\alpha = \pi/3$ and $k = 4$. This data-set is then used to train a 5-layer deep neural network with 200 neurons per a hidden layer by minimizing the mean square error loss of using the L-BFGS optimizer.

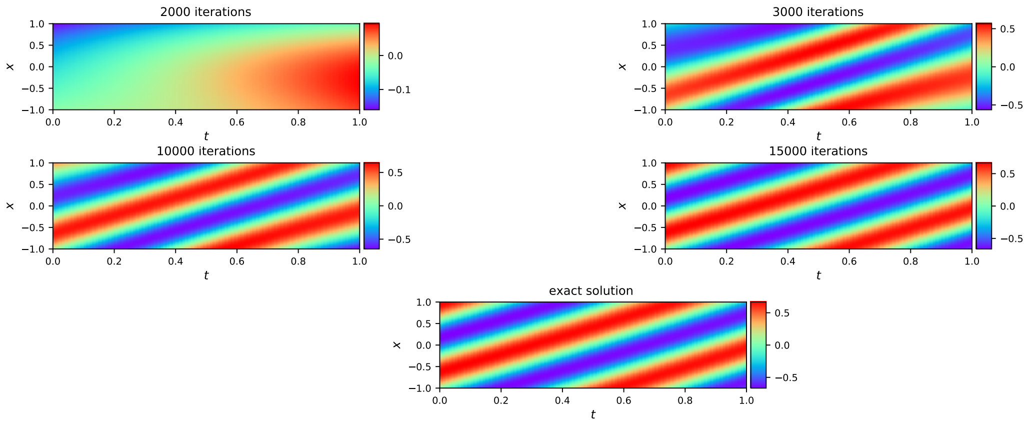


Figure 2: A comparison between the exact solution m_2 and the calculated solution m_{2p}, for different number of iterations and $d = 1$.

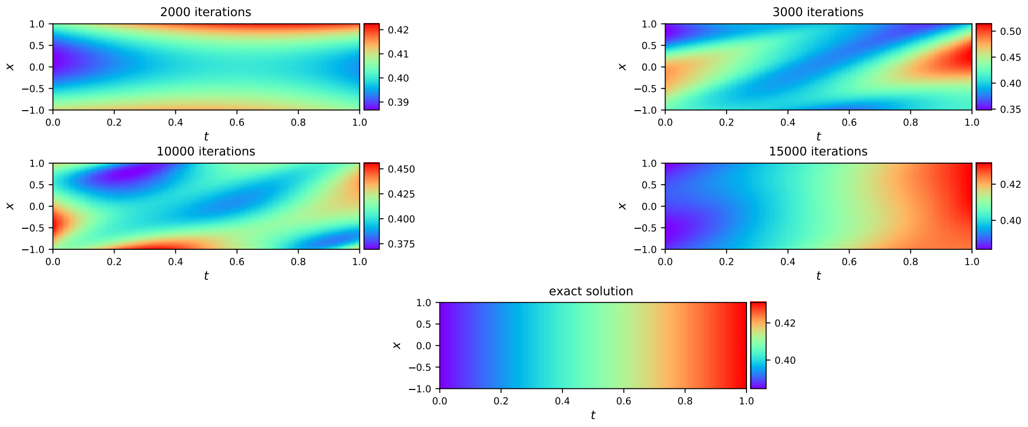


Figure 3: A comparison between the exact solution m_3 and the calculated solution m_{3p}, for different number of iterations and $d = 1$.

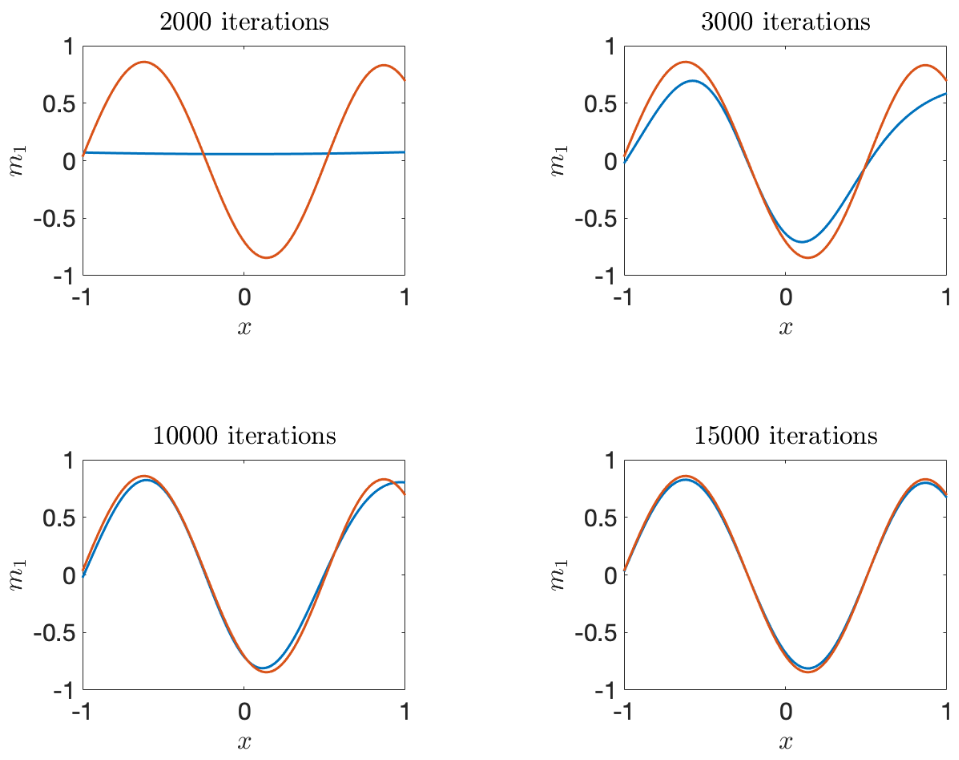


Figure 4: A comparison between the exact solution represented by the red color and the approximate solution represented by the blue color for $t = 0.50$, $d = 1$ and for different iterations.

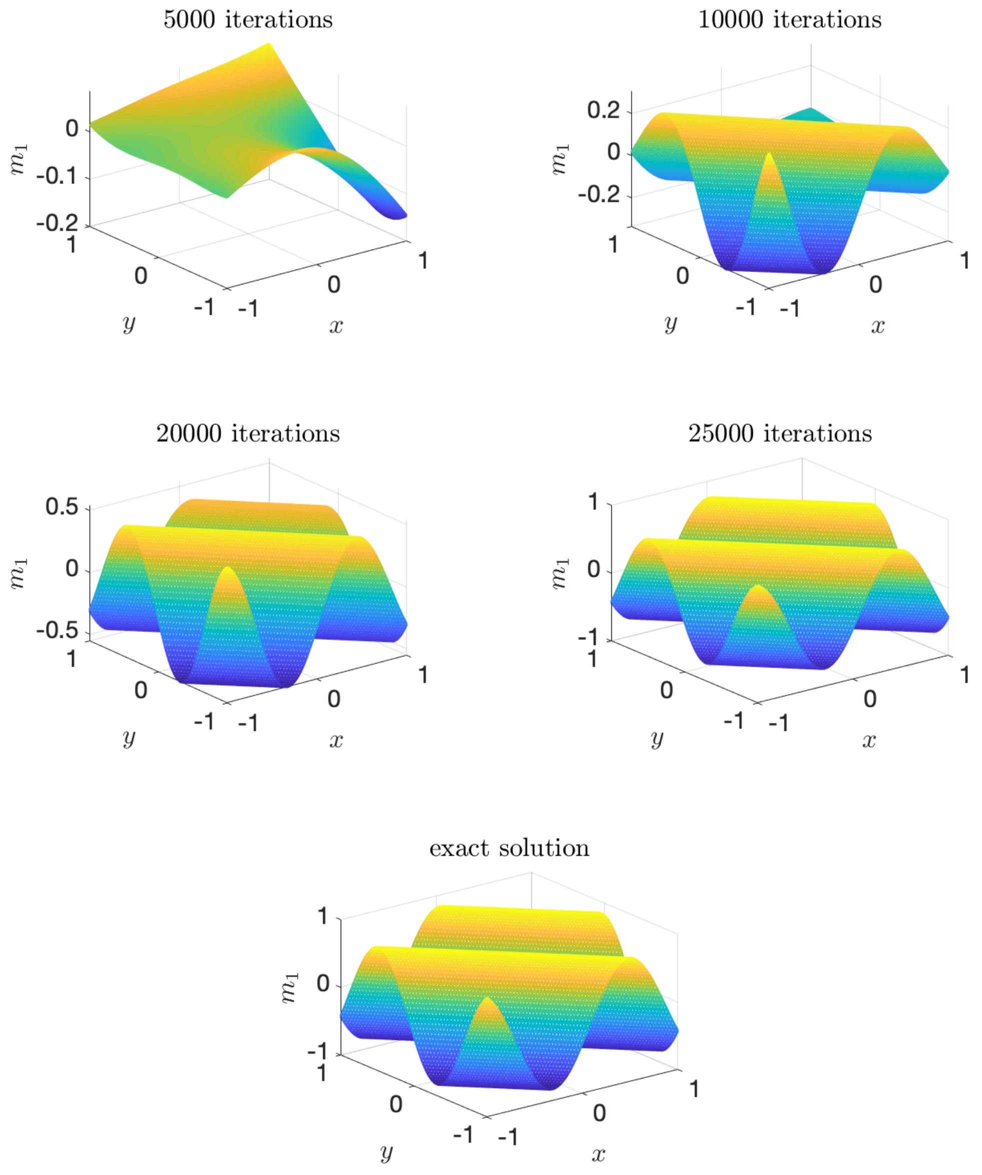


Figure 5: A comparison between the exact solution m_1 and the approximate solution m_{1p} for $t = 0.50$, $d = 2$ and for different iterations.

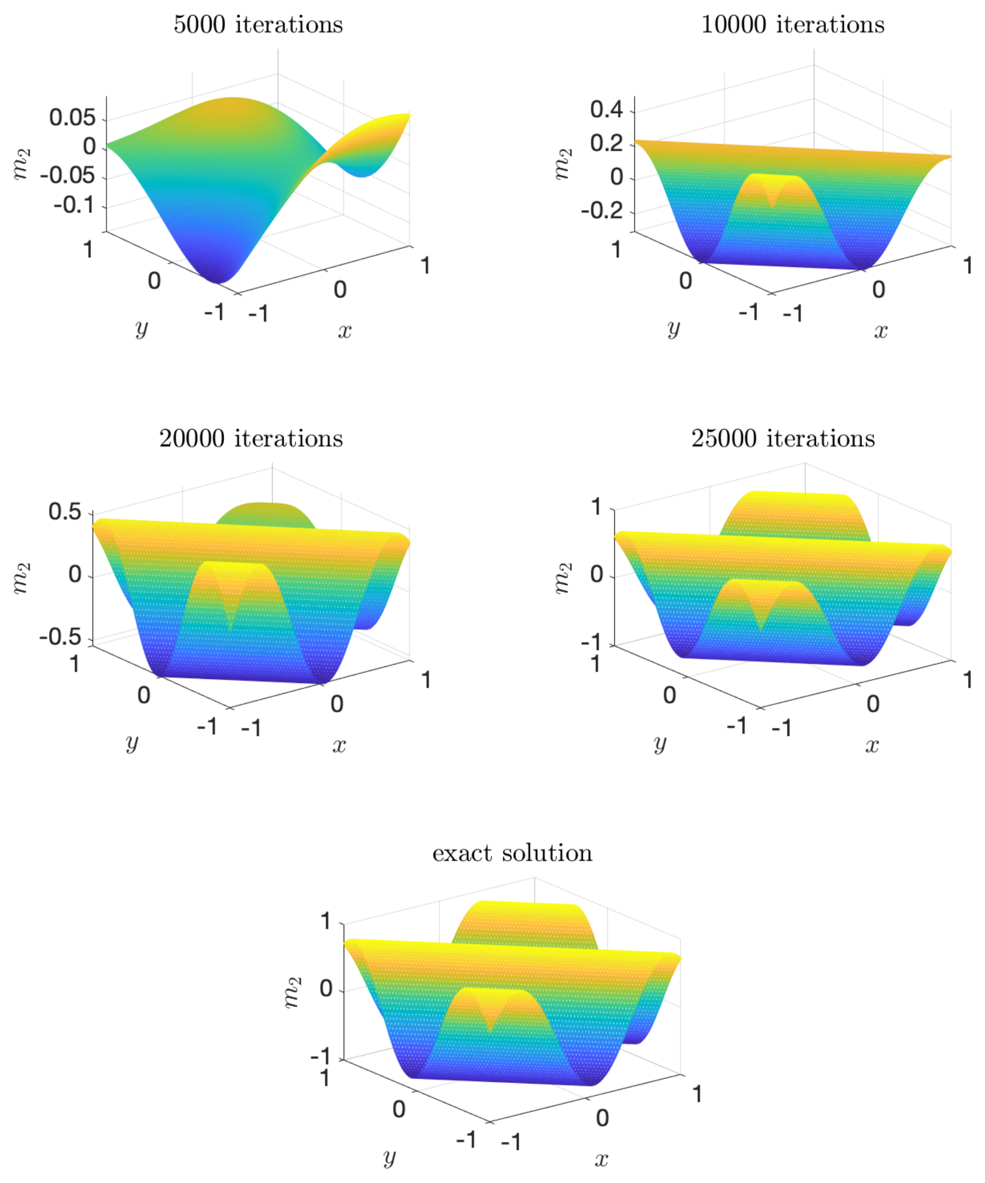
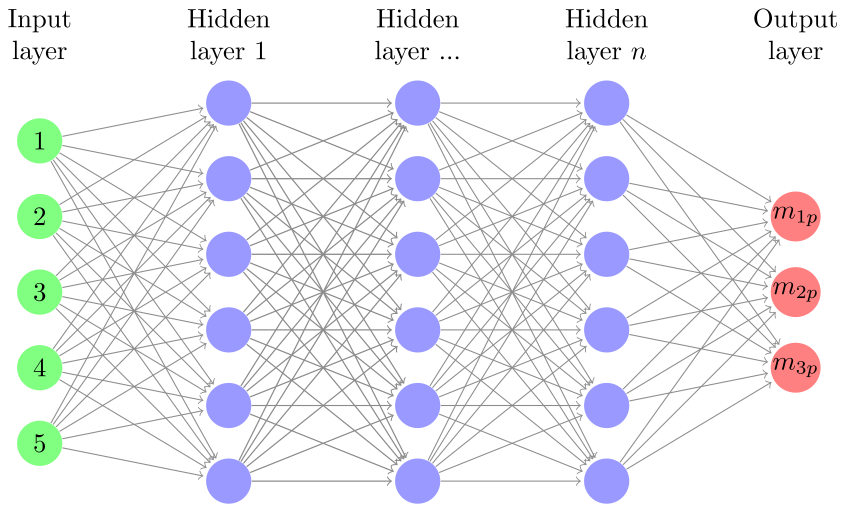


Figure 6: A comparison between the exact solution m_2 and the approximate solution m_{2p} for $t = 0.50$, $d = 2$ and for different iterations.

3.2 Second case

This situation differs from the first case in the input layer, but it is very similar in terms of the method and steps used in determining the solution of the equation at all points. Here we will consider a neuron network composed by several layers such that the first layer represents the inputs (the random points $s_n = \{(x_n, t_n), (\delta_n, v_n), w_n\}$ on Q, ∂Q and Ω), it is composed of five neurons, the first for x_n, the second for t_n, the third for δ_n, the fourth for v_n and the last for w_n, the last layer represents the output solution $M_p = (m_{1p}, m_{2p}, m_{3p})$ at the random points $\{(t_n, x_n), \delta_n, v_n, (w_n, 0)\}$ linked by the weights and biases parameters, while the other layers are hidden layers.



The objective function *Loss* in this case is given by

$$Loss = MSE_f + MSE_{init} + MSE_{bound} + MSE_{Contraint}$$

with

$$MSE_{init} = MSE_{m_{1init}} + MSE_{m_{2init}} + MSE_{m_{3init}},$$

and

$$MSE_{bound} = MSE_{m_{1bound}} + MSE_{m_{2bound}} + MSE_{m_{3bound}},$$

with

$$MSE_{m_{rinit}} = \frac{1}{N_{init}} \sum_{i=1}^{N_{init}} |m_r(\delta^i, v^i) - m_{rp}(\delta^i, v^i)|^2,$$

$$MSE_{m_{rbound}} = \frac{1}{N_{bound}} \sum_{i=1}^{N_{bound}} |m_r(w^i, 0) - m_{rp}((w^i, 0)|^2,$$

for $r = 1, 2, 3$, the numbers N_{init} and N_{bound} are, respectively, the numbers of the random points $(\delta^i, v^i) \in \partial Q$ and $(w^i, 0) \in \Omega \times \{0\}$ for which we know the solution of the equation (1). In the next step, we will deduce an iterative gradient algorithm designed to minimize *Loss*. We apply the following algorithm.

1. Initialize the parameter set {weights,biases}.
2. Generate random samples from the domain's and time spatial boundaries, *i.e.,*
   - Generate (t_n, x_n) from Q.
   - Generate (δ_n, v_n) from ∂Q.
   - Generate w_n from Ω.
3. Calculate the *Loss* functional for the current mini-batch
$$s_n = \{(x_n, t_n), (\delta_n, v_n), w_n\}.$$
4. We choose {weights,biases} randomly such that *Loss* becomes minimal.
5. Repeat steps (3)-(4) until *Loss* is very small.

Numerical simulation. At this stage, we solve the equation in the two -dimensional space only, for the numerical test, we apply an initial condition

$$M_0(x,y) = \begin{cases} \sin(\alpha)\cos(k(x+y)), \\ \sin(\alpha)\sin(k(x+y)), \\ \cos(\alpha). \end{cases} \tag{5}$$

We trained the neural network on $N = 2000$ random points in Q, $N_{init} = 100$ in $\Omega \times \{0\}$ and $N_{bound} = 200$ in ∂Q. We use the parameters $\mu = 3$, $\alpha = \pi/3$ and $k = 4$. This data-set is then used to train a 5-layer deep neural network with 50 neurons per a hidden layer. Figure 7 represent Magnetization component averages $< m_1 >$, $< m_2 >$, $< m_3 >$, $< norm >$ and the *Energy* versus time. Through this figure, we find that the solution obtained realizes all the properties of the equation, the decreasing energy, the conservation of the norm. Thus, we conclude that this method is effective in solving differential equation (1).

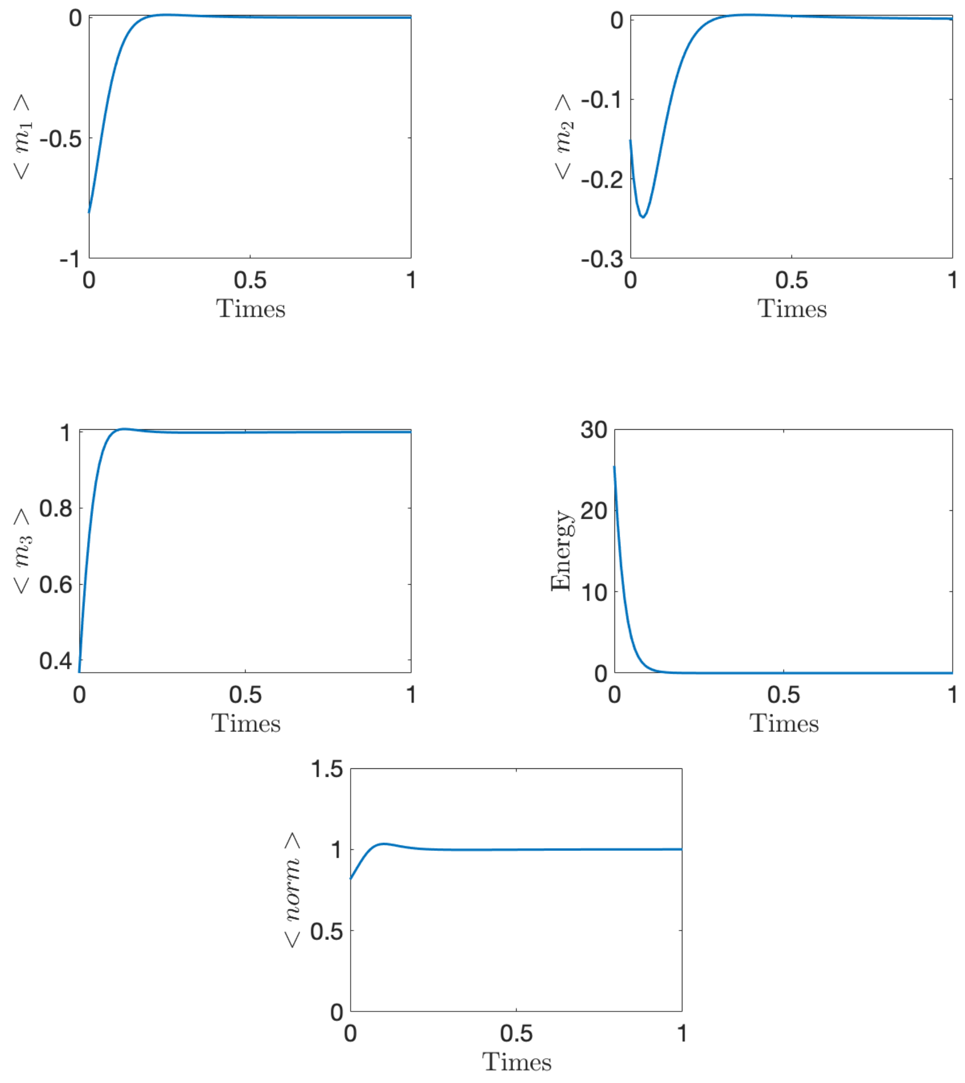


Figure 7: Magnetization component averages $< m_1 >, < m_2 >, < m_3 >, < norm >$ and *Energy* versus time for $\mu = 3$, $\alpha = \pi/3$ and $k = 4$.

4 Concluding Remarks

In this work, we proposed two essentially different approaches, but they are similar in steps for solving differential equation (1). The first method is based on knowledge of equation solutions at some random points of the field Q. The second depends on the knowledge of the solution at some random points of the fields $\Omega \times \{0\}$ and ∂Q. We obtain good results because we can find a solution for the equation using the two methods at each point of Q. Through these results, we conclude the following. The first method is ineffective because we often do not know the solution at some random points of Q, but they are good at identifying some variables in the differential equation, for example, looking for the correct PDEs, see [15], as we think, it will give good results in an inverse source problem. For the PDEs to be well posed, it is necessary to give the initial conditions and the conditions at the edge, which implies that it is possible to find the solutions of PDEs at random points of $\Omega \times \{0\}$ and ∂Q, this makes the second method more effective and more realistic in solving differential equations. In the next works, we will apply a deep learning approach to solve the model of magnetization dynamics with inertial effects [8] and compare the results we will get with the results obtained in [17]. Also, we will try to propose a new algorithm for solving PDE (1) in high dimension, by making an analogy between the backward stochastic differential equations and reinforcement learning with the gradient of the solution playing the role of the policy function.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean and M. Devin. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv: 1603.04467. 2016.

[2] F. Alouges. A new finite element scheme for Landau-Lifchitz equations. *Discrete Contin. Dyn. Syst. Ser. S* **1** (2) (2008) 187–196.

[3] F. Alouges, and P. Jaisson. Convergence of a finite element discretization for the Landau-Lifshitz equations in micromagnetism. *Math. Models Methods Appl. Sci.* **16** (2006) 299–316.

[4] C. Angermueller, T. Pärnamaa, L. Parts and O. Stegle. *Deep learning for computational biology Molecular systems biology* **12** (7) 2016.

[5] A. Bachouch, C. Huré, N. Langrené and H. Pham. Deep neural networks algorithms for stochastic control problems on finite horizon, Part 2. arXiv preprint arXiv:1812.05916. 2018.

[6] C. Beck, E. Weinan, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science* **29** (4) (2019) 1563–1619.

[7] F. Cabitza, R. Rasoini and G. F. Gensini. Unintended consequences of machine learning in medicine. *Jama* **318** (6) (2017) 517–518.

[8] M. Ciornei, J. Rubí and J. Wegrowe. Magnetization dynamics in the inertial regime: Nutation predicted at short time scales. *Physical Review B* **83** (2) (2011).

[9] T. Dockhorn. *A Discussion on Solving Partial Differential Equations using Neural Networks.* arXiv preprint arXiv:1904.07200. 2019.

[10] D. Jeong and J. Kim. An accurate and robust numerical method for micromagnetics simulations. *Current Applied Physics***14** (3) (2014) 476–483.

[11] M. Khumalo and A. Dlamini. The Finite Element Method for Nonlinear Nonstandard Volterra Integral Equations. *Nonlinear Dynamics and Systems Theory* **20** (2) (2020) 191–202.

[12] L. Landau and E. Lifshitz. On the theory of magnetic permeability in ferromagnetic bodies. *Phys', Zeitsch* **8** (1935) 153–169.

[13] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming* **45** (3)(1989) 503–528.

[14] R. Maziar. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research* **19** (1) (2018) 932–955.

[15] R. Maziar, P. Perdikaris and G. E. Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. arXiv preprint arXiv:1711.10566. 2017.

[16] M. Mellah, A. Ouannas, A. A. Khennaoui and G. Grassi. Fractional Discrete Neural Networks with Different Dimensions: Coexistence of Complete Synchronization, Antiphase Synchronization and Full State Hybrid Projective Synchronization. *Nonlinear Dynamics and Systems Theory* **21** (4) (2021) 410–419.

[17] M. Moumni and M.Tilioua. A finite-difference scheme for a model of magnetization dynamics with inertial effects. *Journal of Engineering Mathematics* **100** (1) (2016) 95–106.

[18] M. Moumni and M.Tilioua. A finite element approximation of a current-induced magnetization dynamics model. *Journal of Mathematical Modeling* **10** (1) (2022) 53–69.

[19] H. T. Nembach, J. M. Shaw, C. T. Boone and T. J. Silva. Mode-and size-dependent Landau-Lifshitz damping in magnetic nanostructures: evidence for nonlocal damping. *Physical review letters* **110** (11) (2013) 117201.

[20] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica* **8** (1999) 143–195.

[21] W. M. Saslow. Landau–Lifshitz or Gilbert damping? That is the question. *Journal of Applied Physics* **105** (7) (2009).

[22] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* **375** (2018) 1339–1364.

[23] E. Weinan, J. Han and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics* **5** (4) (2017) 349–380.