

A Physics-Informed Neural Network Approach for Solving the Monge-Ampère Partial Differential Equation

Christopher Ogbonna¹, Maliki Olaniyi S², Nkem Ogbonna³

^{1,2,3}Michael Okpara University of Agriculture Umudike, Abia State, Nigeria.

DOI: <https://doi.org/10.51584/IJRIAS.2026.11090019>

Received: 30 July 2026; Accepted: 04 August 2026; Published: 02 October 2026

ABSTRACT

This paper presents a comparative study of classical numerical and machine learning techniques for solving the nonlinear Monge–Ampère Equation (MAE). A finite difference method (FDM) implemented in MATLAB is first validated using a quadratic exact solution on a square domain. Two neural network approaches are then investigated: a supervised neural network trained on analytical solution data, and a Physics-Informed Neural Network (PINN) implemented in Python using a [Streamlit](#) interface. The PINN solves the MAE through an unsupervised learning strategy based on minimizing PDE residuals and boundary losses. The effects of tanh and sigmoid activation functions on convergence and accuracy are also examined. Numerical results show that while FDM provides a reliable benchmark, PINNs offer an effective mesh-free alternative. Compared with classical methods, PINNs trade algorithmic rigor for adaptability and dimensional scalability, offering a modern, mesh-free perspective on PDE approximation. The ReLU/Tanh networks employed here is capable of representing both convex and concave solutions, however the gradient descent procedure can get trapped in the concave branch with the hyperbolic tangent (tanh) activation function, as we discovered in our case. Overall, our study demonstrates the potential of PINNs for solving fully nonlinear partial differential equations.

Keywords: Physics-Informed Neural Networks, Monge-Ampère PDE, Scientific Machine Learning, Neural PDE Solvers.

INTRODUCTION

The theory of the Monge-Ampère equation goes back to the work of Pogorelov [1], and Aleksandrov *et al.* [2]; which focused mostly on geometry of convex surfaces and on the problem of finding a closed convex surface in $\mathbb{R}^n$ with prescribed Gaussian curvature. Some important results regarding the Monge-Ampère equation include the notion of generalized solution; the notion of viscosity solution and its equivalence to the generalized solution, the Aleksandrov maximum principle, the classification of Monge-Ampère equations and, most notably, Caffarelli's regularity results; Caffarelli [3]. The Monge-Ampère equation has wide applications in geometry, particularly in the problem of prescribed Gaussian curvature Kutateladze [4], and in optimal control, particularly in the mass transfer problems.

The Monge-Ampere equation arises in several problems from analysis and geometry. In its classical form this equation is given by

$$\det D^2u = f(x, u, \nabla u) \quad \text{in } \Omega$$

where $\Omega \subset \mathbb{R}^n$ is some open set, $u: \Omega \rightarrow \mathbb{R}$ is a convex function and $f: \Omega \times \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^+$ is given. In other words, the Monge-Ampere equation prescribes the product of the eigenvalues of the Hessian of u , in contrast with the “model” elliptic equation $\Delta u = f$ which prescribes their sum. We remark here that the convexity of the solution u is a necessary condition to make the equation degenerate elliptic, and this in turn will hopefully lead to regularity results.

BACKGROUND AND MOTIVATION

Nonlinear partial differential equations play a fundamental role in modeling complex phenomena in science, engineering, and applied mathematics. Unlike linear PDEs, nonlinear PDEs can describe interactions, instabilities, shocks, and other complicated behaviors observed in real-world systems. They arise naturally in areas such as fluid dynamics, quantum mechanics, elasticity, biological systems, meteorology, and financial mathematics. Nonlinear PDEs are often more difficult to analyze and solve because their solutions may exhibit singularities, multiple solutions, or instability. As a result, advanced numerical techniques and modern computational methods, including machine learning and Physics-Informed Neural Networks (PINNs), are increasingly being developed to study these equations efficiently.

The Monge-Ampère equation is a fully nonlinear second-order PDE with significant applications in several scientific and engineering fields. We mention a few.

2.1 Optimal Transport

One of the most important applications of the Monge-Ampère equation is in optimal transport theory, where the objective is to determine the most efficient way of transporting mass from one distribution to another while minimizing transportation cost. The equation arises naturally in the characterization of optimal transport maps and has applications in economics, logistics, meteorology, fluid mechanics, and machine learning.

2.2 Geometry

In differential geometry, the Monge-Ampère equation is used in the study of convex surfaces, curvature problems, and geometric optics. It appears in the construction of surfaces with prescribed Gaussian curvature and in problems involving affine and Kähler geometry. The equation also plays a central role in geometric analysis and the theory of convex functions.

2.3 Image Processing

The Monge-Ampère equation has important applications in image processing and computer vision. It is used in image registration, image warping, mesh generation, shape reconstruction, and reflector design. In optimal transport-based imaging methods, the equation helps map one image distribution onto another while preserving important structural features, thereby improving image reconstruction and enhancement techniques.

The development of the Monge-Ampère equation and optimal transport theory is closely associated with the pioneering contributions of several mathematicians.

- Gaspard Monge first introduced the optimal transport problem in 1781 while studying the most efficient method of moving soil masses in engineering applications. His work laid the foundation for modern optimal transport theory and the Monge-Ampère equation.
- André-Marie Ampère later investigated a class of nonlinear second-order partial differential equations that became known as the Monge-Ampère equations. His contributions helped establish the analytical structure of these fully nonlinear PDEs.
- Yann Brenier made fundamental advances in optimal transport by proving the existence and uniqueness of optimal transport maps under quadratic cost functions. Brenier established the deep connection between convex analysis, optimal transport, and the Monge-Ampère equation through what is now known as Brenier's theorem.
- Luis Caffarelli made groundbreaking contributions to the regularity theory of the Monge-Ampère equation and optimal transport problems. His work on weak and viscosity solutions provided important analytical results concerning existence, uniqueness, smoothness, and stability of solutions to fully nonlinear PDEs.

These foundational works have significantly influenced modern research in nonlinear PDEs, geometry, optimal transport, and scientific machine learning.

NUMERICAL SOLUTIONS

In the last several years, a number of methods have been proposed for the numerical solution of the Monge-Ampère equation. We make a distinction between methods for the PDE itself and methods that address the transport boundary condition, which has received much less attention from a numerical standpoint. We also distinguish between methods that provably converge to a weak solution of the equation, and other methods that provide no guarantee of convergence. An early work by Oliker and Prussner [5] introduced a discretization based on a geometric interpretation of the solutions. In two dimensions, this method converges to the Aleksandrov solution and the extension to practical three-dimensional schemes were not introduced until some 20 years later Brenner and Neilan [6], Feng and Neilan [7], Froese and Oberman[8] provides a unique monotone finite difference approach to the Monge-Ampère issue. Its primary achievement is the development of a method that, by combining wide stencils with a rigorous discretization, is scientifically convergent.

Brittany and Oberman[9], developed a finite difference solver for the Monge-Ampère equation which converges even for singular solutions.

3.1 Neural Network as Function Approximators

As universal Function approximators, neural networks can be trained on input-output examples to learn to represent almost any complicated function from data. Neural networks (NNs) were first used in important literature such as Lagaris *et al* [10] to solve PDEs by minimizing the residual loss. This was expanded to high dimensional PDEs using stochastic gradient descent by Han *et al* [11]. Deep Galerkin Method. Physics-informed Neural Networks (PINNs) Raissi *et al* [12], a framework for solving partial differential equations (PDEs) by embedding the governing physics directly into the neural network's training objective.

3.2 Neural Network-Based Supervised Solvers for Monge-Ampère PDE

A neural network approach for solving the Monge-Ampère PDE with transport boundary condition was investigated by Hacking *et al* [13], they introduce a multilayer perceptron (MLP) trained via *supervised learning*, minimizing a composite loss combining PDE residual Transport boundary loss and Convexity loss. Chen *et al* [14], proposes a *relaxation-based algorithm* splitting optimal transport PDE into three sub-optimization stages.

MATHEMATICAL FORMULATION

In this section, we consider numerical solutions to a Monge-Ampère type boundary value problem of the form;

$$Lu(x, y) = f(x, y) \quad \text{in } \Omega \quad (1)$$

$$u = g, \quad u \text{ convex} \quad \text{on } \partial\Omega \quad (2)$$

Where $Lu = u_{xx}u_{yy} - (u_{xy})^2$, and $\Omega = \{(x, y) : a \leq x \leq b, a \leq y \leq b\}$ as shown in fig. 1.

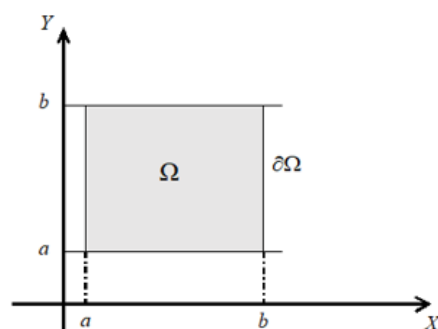


Fig.1 The domain Ω

Specifically, we are interested in solving the following problem

$$\begin{aligned} Lu &= (1 + x^2 + y^2) \exp(x^2 + y^2) & \text{in } \Omega \\ u &= \exp\left(\frac{1}{2}(x^2 + y^2)\right) & \text{on } \partial\Omega \end{aligned} \quad (3)$$

where Ω is the square domain $[-0.5, 0.5] \times [-0.5, 0.5]$, and L is the Monge-Ampere differential operator. An exact solution to the given problem is;

$$u(x, y) = \exp\left(\frac{1}{2}(x^2 + y^2)\right) \quad (4)$$

4.1 An Explicit Finite Difference Method

The method we adopt involves simply discretizing the second derivatives in equation (1) using standard central differences on a uniform Cartesian grid. The result is

$$(D_{xx}^2 u_{ij})(D_{yy}^2 u_{ij}) - (D_{xy}^2 u_{ij})^2 = f_{i,j} \quad (5)$$

where

$$\begin{aligned} D_{xx}^2 u_{ij} &= \frac{1}{h^2} (u_{i+1,j} + u_{i-1,j} - 2u_{i,j}), \quad D_{yy}^2 u_{ij} = \frac{1}{h^2} (u_{i,j+1} + u_{i,j-1} - 2u_{i,j}) \\ D_{xy}^2 u_{ij} &= \frac{1}{4h^2} (u_{i+1,j+1} + u_{i-1,j-1} - u_{i-1,j+1} - u_{i+1,j-1}) \end{aligned} \quad (6)$$

Since we are using centered differences, this discretization is consistent with (6), and it is second order accurate if the solution is smooth.

4.1.1 Remark

The above discretization is valid in the interior of the domain. Naturally, in a general (non-rectangular) domain it may be necessary to interpolate the boundary values to obtain a first order scheme at the boundary.

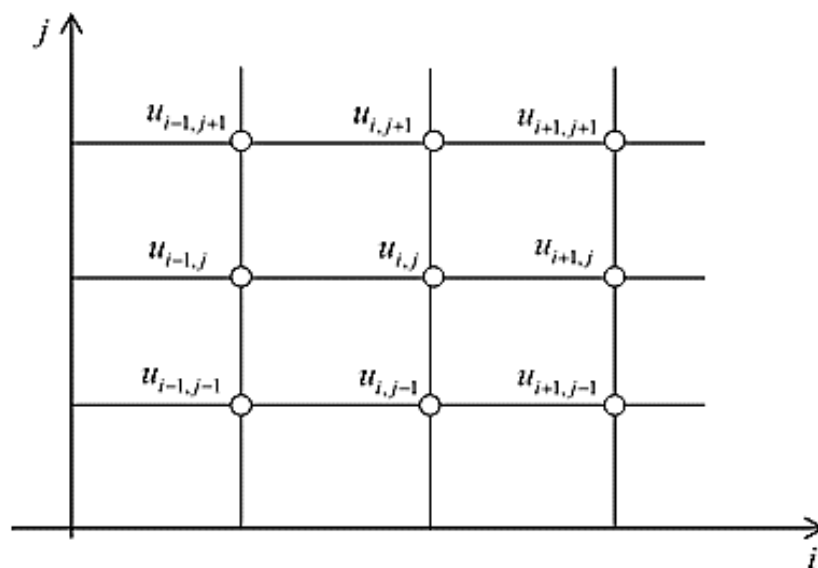


Fig.2 Discretization over a rectangular domain

Next, introduce the notation

$$a_1 = \frac{1}{2}(u_{i+1,j} + u_{i-1,j}), \quad a_2 = \frac{1}{2}(u_{i,j+1} + u_{i,j-1}), \quad a_3 = \frac{1}{2}(u_{i+1,j+1} + u_{i-1,j-1}), \quad a_4 = \frac{1}{2}(u_{i-1,j+1} + u_{i+1,j-1}) \quad (7)$$

and rewrite (4.16) as a quadratic equation for $u_{i,j}$.

$$4(a_1 - u_{i,j})(a_2 - u_{i,j}) - \frac{1}{4}(a_3 - a_4)^2 = h^4 f_{i,j}$$

Solving for $u_{i,j}$ and selecting the smaller root (in order to select the locally convex solution), we have

$$u_{i,j} = \frac{1}{2}(a_1 + a_2) - \frac{1}{2}\sqrt{(a_1 - a_2)^2 + \frac{1}{4}(a_3 - a_4)^2 + h^4 f_{i,j}} \quad (8)$$

We can now use Gauss-Seidel iteration to find the fixed point of (8). The Dirichlet boundary conditions (2) are enforced at boundary grid points. The convexity constraint is not enforced beyond the selection of the positive root in (8).

For the purpose of implementing the above discretization scheme in Matlab, we rewrite equation (8) more explicitly as;

$$u_{i,j} = \frac{1}{2} \left(\frac{1}{2}(u_{i+1,j} + u_{i-1,j}) + \frac{1}{2}(u_{i,j+1} + u_{i,j-1}) \right) - \frac{1}{2} \sqrt{\left(\frac{1}{2}(u_{i+1,j} + u_{i-1,j}) - \frac{1}{2}(u_{i,j+1} + u_{i,j-1}) \right)^2 + \frac{1}{4} \left(\frac{1}{2}(u_{i+1,j+1} + u_{i-1,j-1}) - \frac{1}{2}(u_{i-1,j+1} + u_{i+1,j-1}) \right)^2 + h^4 f_{i,j}} \quad (9)$$

or

$$u_{i,j} = \frac{1}{4} (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) - \frac{1}{8} \sqrt{4(u_{i+1,j} + u_{i-1,j} - u_{i,j+1} - u_{i,j-1})^2 + (u_{i+1,j+1} + u_{i-1,j-1} - u_{i-1,j+1} - u_{i+1,j-1})^2 + 16h^4 f_{i,j}} \quad (10)$$

with $x = ih$, $y = jh$ and taking $h = 1$, we have;

4.2 Numerical Results for discretization of the MA PDE

In this section, we apply the proposed discretization method for the Monge-Ampere PDE (3). The computations were performed on a HP laptop with 64 based processor and 4GB RAM, using MATLAB running in 64 bit OS.

4.2.1 Solution procedure

The discrete counterpart of the boundary conditions corresponding to

$$f(x, y) = (1 + x^2 + y^2) \exp(x^2 + y^2), \quad \forall (x, y) \in \Omega = [-0.5, 0.5] \times [-0.5, 0.5] \quad (11)$$

is computed as follows. The solutions are computed on an $N \times N$ grid and iteration is continued until the maximum difference between two subsequent iterates is less than 10^{-14} . By convention, h is the spatial discretization parameter $h = L/N$ where L is the length of one side of the square domain. Here we choose the square $\Omega = [-0.5, 0.5] \times [-0.5, 0.5]$, so that $h = 1/N$. Since $x = i, y = j$ from above, it then follows that on the square Ω , we have;

$$u(x, y) = \exp\left(\frac{x^2 + y^2}{2}\right) \text{ on } \partial\Omega, \quad (12)$$

4.2.2 Error Analysis

Let $u_{i,j}$ be the numerical value calculated by formulae (10), an approximate value of the theoretical solution $u(x, y)$ at the mesh point $(x, y) = (x_i, y_i)$. The maximum absolute error

$$MAE(u) = \max_{1 \leq i, j \leq N} |u(x_i, y_i) - u_{i,j}| \quad (13)$$

are easily computed for different value of h , the mesh size. The stopping criterion for iteration can be taken as, either error of order 10^{-8} or number of iterations 10^3 .

ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) are central to the present research and a brief introduction is necessary here. An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks, have been developed as generalizations of mathematical models of human cognition or neural biology [15], based on the assumptions that:

- Information processing occurs at many simple connections called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function to its net input to determine its output signal.

The basic component of an artificial neural network is the artificial neuron, similar to biological neuron [15]. in biological neural network. A biological neuron may be modeled artificially to perform computation and then the model is termed as artificial neuron.

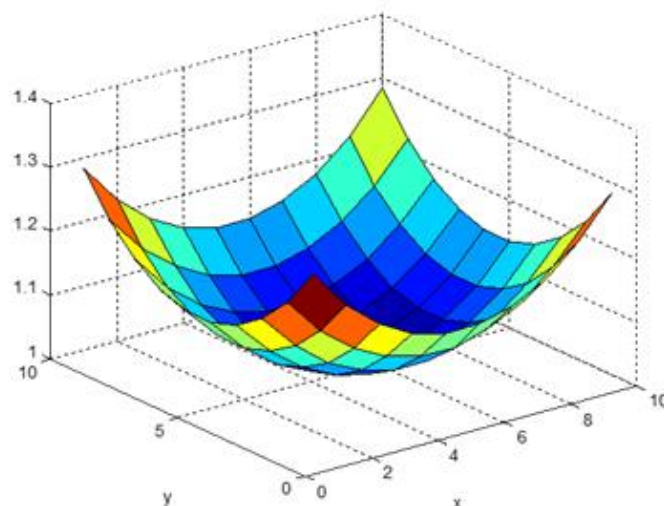


Fig. 3 Exact solution of MA PDE

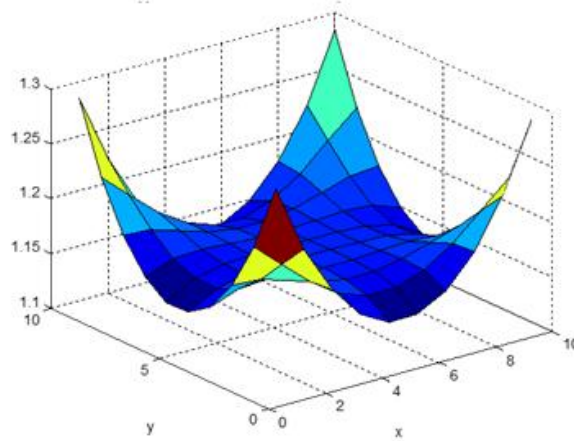


Fig. 4 Approximate solution of the MA PDE with 250 iterations in Matlab

5.1 Activation functions

Activation functions are very important in Machine Learning because they incorporate *nonlinearity* into our ANN [15]. This enables the neural network to learn complicated tasks, without it the neural network can only perform *linear regression*. The most popular hidden neuron activation functions are the sigmoid, arctangent and the hyperbolic tangent functions given respectively by;

$$\varphi_1(x) = \frac{1}{1+e^{-x}}, \quad \varphi_2(x) = \frac{2}{\pi} \arctan x \quad \text{and} \quad \varphi_3(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x) \quad (14)$$

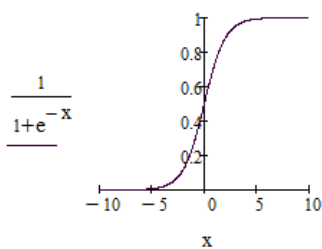
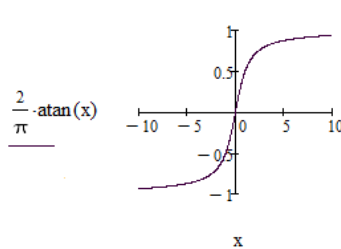
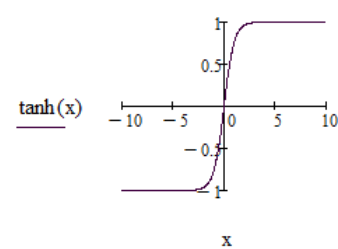


Fig. 5(a) $\varphi_1(x)$



(b) $\varphi_2(x)$



(c) $\varphi_3(x)$

5.1.1 Universal Approximation Theorem

The universal approximation theorem [16] for MLP is stated as follows. Let I_n represent an n -dimensional unit cube containing all possible input samples $\mathbf{x} = (x_1, x_2, \dots, x_n)$ with $x_i \in [0, 1]$, $i = 1, 2, \dots, n$. Let $C(I_n)$ be the space of continuous functions on I_n , given a continuous sigmoid function $\sigma(\cdot)$, then universal approximation theorem states that the finite sums of the form

$$y_k = y_k(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N_2} w_{ki}^3 \sigma \left(\sum_{j=0}^n w_{ki}^2 x_j \right) \quad k = 1, 2, \dots, m \quad (15)$$

are dense in $C(I_n)$. This simply means that given any function $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum $y(\mathbf{x}, \mathbf{w})$ of the above form that satisfies

$$|y(\mathbf{x}, \mathbf{w}) - f(\mathbf{x})| < \varepsilon \quad \forall \mathbf{x} \in I_n \quad (16)$$

5.1.2 Remark

The universal approximation theorem expressed in equation (15), simply states that there always exists a 3-layer perceptron that can approximate an arbitrary nonlinear, continuous, multidimensional function f with any desired accuracy. However, the theorem does not state how many neurons are needed by the three-layer MLP to approximate the given function, as such, failure to develop an accurate model can be attributed to an inadequate number of hidden neurons, inadequate learning or training, or perhaps the presence of a stochastic rather than a deterministic relation between inputs and outputs.

5.1.3 Model of a Neuron

The computational structure of a neuron (fig. 6) consists of a set of synapses each of which is characterized by

- A weight or strength of its own.
- An adder, an activation function and a bias.

In mathematical terms, a neuron k can be described by:

$$v_k = \sum_{j=1}^m w_{kj} x_j \quad \text{and} \quad y_k = \varphi(u_k + b_k) \quad (17)$$

where u_k is the linear combiner output due to input signals. Moreover, $v_k = u_k + b_k$.

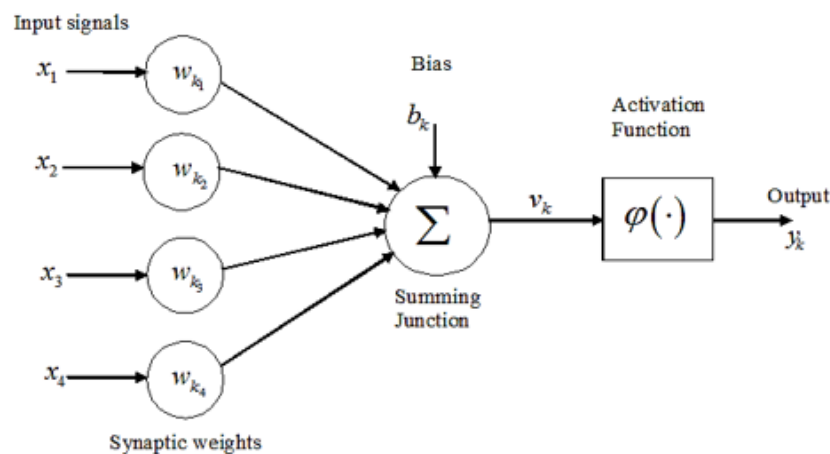


Fig.6 Computational structure of a neuron

5.2 Training

All neural networks must be trained, and the common training algorithm involves backpropagation described below. We note that it is necessary to train the neural network so that it can learn the task given to it. This process involves minimizing an objective, which is the loss function.

5.3 Mathematical Analysis of Backpropagation

Backpropagation actually stands for *backward propagation of errors*, and is commonly used technique for training neural network, Millidge *et al* [17]. It is a mechanism employed to update the weights using gradient descent. It computes the gradient of the error function with respect to the neural network's weights, Hinton *et al* [18]. The calculation proceeds backwards through the network. We note that gradient descent is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize the error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

NEURAL NETWORKS FOR SOLVING PDES

In this section we will consider a neural network to be given by a compositional function,

$$u(\mathbf{x}, \Theta) = f(\dots f(f(\mathbf{x}W^1)W^2)\dots W^N) \quad (18)$$

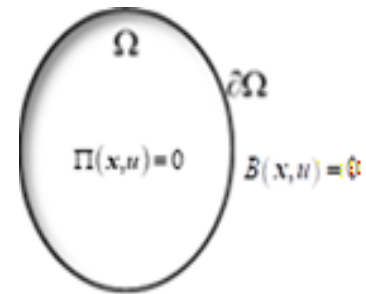
where $\mathbf{x}$ is the input vector, $\Theta = (W^1, \dots, W^N)$ is the set of model parameters, $f(\cdot)$ is the non-linear activation function and N is the number of layers. The optimization function is given by;

$$\min_{\Theta} L(\Theta) \quad (19)$$

where the *loss* $L(\Theta)$ is some function of $u(\mathbf{x}, \Theta)$. This will be studied in detail subsequently.

In order to solve a PDE with the above defined neural network, let us consider the following general PDE for the function $u(\mathbf{x})$;

$$\begin{cases} \Pi(\mathbf{x}, u) = 0 & \text{in } \Omega \\ B(\mathbf{x}, u) = 0 & \text{on } \partial\Omega \end{cases}$$



In the above;

- $\Pi(\cdot)$ is a differential operator in the interior of the domain.
- $B(\cdot)$ is a differential operator on the boundary of the domain.

6.1 A Supervised Approach

In this approach we generate a dataset of solutions, i.e. dataset consisting of $\mathbf{x}$, $u(\mathbf{x})$. We then train a neural network to learn the mapping $\mathbf{x} \rightarrow u(\mathbf{x})$. We then hope that the trained network generalizes to new inputs $\mathbf{x}$.

6.2 An Unsupervised Approach

In this approach we select interior and boundary collocation points $\mathbf{x}_i^L$ and $\mathbf{x}_i^B$.

We now solve the PDE by minimizing a single loss function which includes all constraints;

$$L(\Theta) = \frac{1}{n_L} \sum_{i=1}^{n_L} \Pi(\mathbf{x}_i^L, \hat{u}(\Theta))^2 + \frac{1}{n_B} \sum_{i=1}^{n_B} B(\mathbf{x}_i^B, \hat{u}(\Theta))^2 \quad (20)$$

The trained neural network $u(\Theta, \mathbf{x})$ is hopefully close to the PDE solution $\hat{u}(\mathbf{x})$.

SUPERVISED LEARNING OF THE MONGE-AMPERE SOLUTION

Traditional numerical method for example Finite element, Finite difference often struggle with high dimensional cases or complex boundary conditions. Recently, supervised machine learning (ML) approaches have been explored to approximate solutions efficiently. Supervised learning involves training a neural network to learn a mapping between input and output data. We use a neural network to learn the solution to the Monge-Ampere equation (1,2) and evaluate its performance on a test set. We also investigate the effect of different network architectures, activation functions and training algorithms on the accuracy of the solution. Furthermore, we discuss the key points in supervised learning methods for the Monge Ampere PDE. We begin with the exact

solution matrix computed from equation (12). We require our Neural Network to produce an approximation of the exact solution, in other words we want it to learn the provided solution (supervised learning). For this purpose we produce the following table which is just a cross section of the complete table given in the Appendix. Here X and Y are the independent variables, while $U(X,Y)$ represents the exact MA solution and $UP(X,Y)$ is the predicted or approximate solution. UP is obtained using the MLP-Neural Network built into SPSS23.

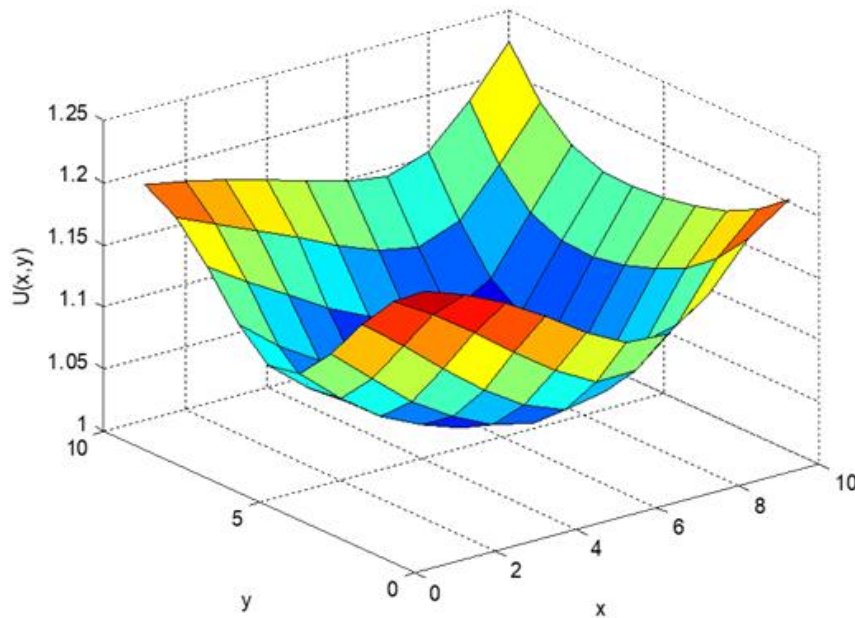
7.1 ANN approximation of the MA solution

In the above, Table-1 in Fig 7(a) presents the table of values for the predicted solution of the MA PDE with supervised learning. These values are then transcribed into Table-2 in Fig 7(b) for Matlab to give the plot shown in Fig. 8. We observe from the graph that the approximation is quite good.

Moreover, this is also evident in two dimensions as depicted in the graphical profile of Fig. 9.

PHYSICS-INFORMED NEURAL NETWORKS (PINNS) SOLUTION OF THE MA PDE

This section is concerned with unsupervised learning and we will show how a PINN computes the required partial derivatives for the Monge–Ampère equation using *automatic differentiation* (AD), specifically *reverse-mode* AD (backpropagation).



Fig, 8 Predicted solution of MA PDE with supervised learning

We want to solve the 2D elliptic Monge–Ampère equation:

$$\det(D^2u) = u_{xx}u_{yy} - (u_{xy})^2 = f(x, y) \quad \text{in } \Omega \in \mathbb{R}^2 \quad (21)$$

with boundary conditions $u = g(x, y)$ on $\partial\Omega$, u is convex.

The PINN approximates the solution by a neural network:

$$\hat{u}(x, y; \theta) = \mathfrak{N}(x, y; \theta) \quad (22)$$

where θ are the network weights and biases.

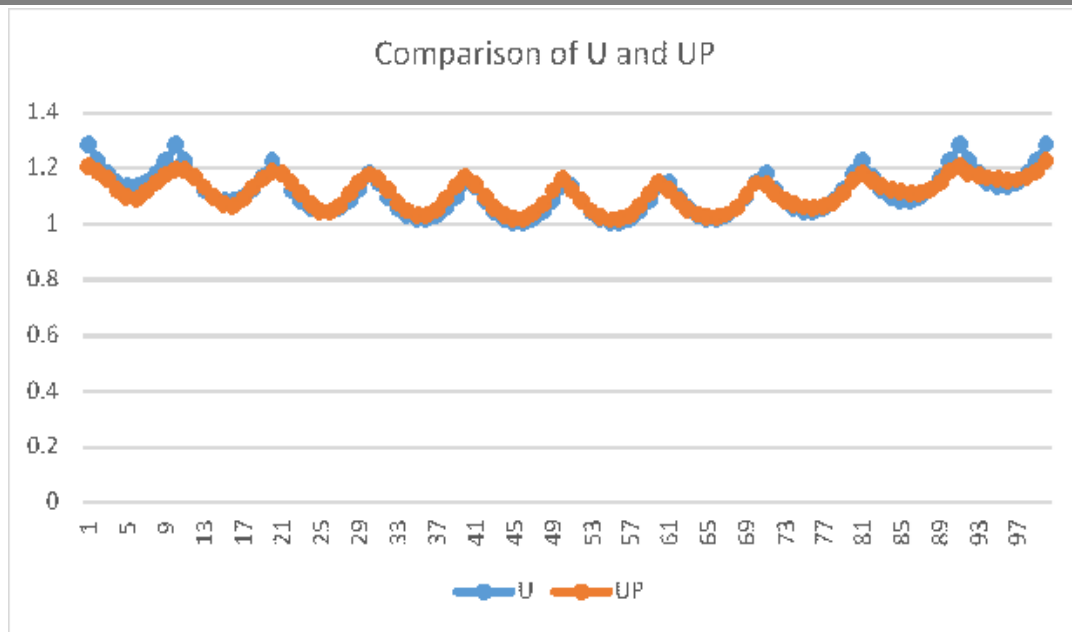


Fig 9 Predicted solution of MA PDE with supervised learning

Step 1: Forward Pass – Compute the Network Output

Here we define a fully connected network with L hidden layers, each with n_k neurons, and activation function φ (e.g., tanh).

Forward propagation:

Input layer:

$$a^{(0)} = (x, y) \quad (2\text{-dimensional input}) \quad (23)$$

Hidden layer 1:

$$z^{(1)} = W^{(1)}a^{(0)} + b^{(1)}, a^{(1)} = \varphi(z^{(1)}) \quad (24)$$

Hidden layer k ($k=2, \dots, L$):

$$z^{(k)} = W^{(k)}a^{(k-1)} + b^{(k)}, a^{(k)} = \varphi(z^{(k)}) \quad (25)$$

Output layer (linear, no activation):

$$\hat{u} = W^{(L+1)}a^{(L)} + b^{(L+1)} \quad (26)$$

Here:

- $W^{(k)}$ = weight matrix for layer k
- $b^{(k)}$ = bias vector
- φ = activation function (differentiable, e.g., tanh or sigmoid)

Thus for any input point (x_i, y_i) our result is $\hat{u}_i = \hat{u}(x_i, y_i; \theta)$.

Step 2: Automatic Differentiation (AD) – Computing First Derivatives

We need to compute $\partial\hat{u}/\partial x$ and $\partial\hat{u}/\partial y$. AD applies the chain rule through the network graph.

Chain rule for $\hat{u}_x$:

$$\frac{\partial\hat{u}}{\partial x} = \frac{\partial\hat{u}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdots \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial x} \quad (27)$$

In practice, deep learning frameworks (TensorFlow or PyTorch) do this automatically via a computational graph. For a manual illustration consider a single hidden layer network, suppose:

$$\hat{u} = W^{(2)} \varphi(W^{(1)}(x, y) + b^{(1)}) \cdot W_{[1]}^{(1)} \quad (28)$$

Then;

$$\frac{\partial\hat{u}}{\partial x} = W^{(2)} \varphi'(W^{(1)}(x, y) + b^{(1)}) \cdot W_{[1]}^{(1)} \quad (29)$$

where $W_{[1]}^{(1)}$ is the first column of $W^{(1)}$ (since derivative w.r.t. x).

Similarly for $\hat{u}_y$,

$$\frac{\partial\hat{u}}{\partial y} = W^{(2)} \varphi'(W^{(1)}(x, y) + b^{(1)}) \cdot W_{[2]}^{(1)} \quad (30)$$

Step 3: Computing Second Derivatives-Hessian Components

We need to compute:

$$u_{xx} = \frac{\partial^2\hat{u}}{\partial x^2}, u_{yy} = \frac{\partial^2\hat{u}}{\partial y^2}, u_{xy} = \frac{\partial^2\hat{u}}{\partial x\partial y} \quad (31)$$

Differentiate the computational graph again.

For u_{xx} , we apply AD to the expression for $\partial\hat{u}/\partial x$ as if it were a new function, i.e., let $v_x = \partial\hat{u}/\partial x$, then we compute $u_{xx} = \partial v_x / \partial x$ using the same chain rule, now applied to v_x .

Similarly for u_{xy} , first compute $v_x = \partial\hat{u}/\partial x$, then compute $u_{xy} = \partial v_x / \partial y$.

Let us consider a simple example with a single hidden layer with tanh activation function

Define:

$$h_1 = \varphi(w_{11}x_1 + w_{12}x_2 + b_1), h_2 = \varphi(w_{21}x_1 + w_{22}x_2 + b_2) \quad (32)$$

$$\hat{u} = v_1 h_1 + v_2 h_2 + c \quad (33)$$

The first derivatives are computed as

$$\hat{u}_x = v_1 \phi'(z_1) w_{11} + v_2 \phi'(z_2) w_{21} \quad (34)$$

where $z_1 = w_{11}x_1 + w_{12}x_2 + b_1$, $\phi'(z) = 1 - \tanh^2(z)$.

The second derivatives are given by;

$$u_{xx} = v_1 [\phi''(z_1) w_{11}^2] + v_2 [\phi''(z_2) w_{21}^2] \quad (35)$$

since $\phi''(z) = -2 \tanh(z) \cdot (1 - \tanh^2(z))$.

$$u_{xy} = v_1 \phi''(z_1) w_{11} w_{12} + v_2 \phi''(z_2) w_{21} w_{22} \quad (36)$$

And finally;

$$u_{yy} = v_1 [\phi''(z_1) w_{12}^2] + v_2 [\phi''(z_2) w_{22}^2] \quad (37)$$

Step 4: Computing the PDE Residual

Given u_{xx}, u_{yy}, u_{xy} from AD, compute the determinant:

$$r(x, y) = u_{xx} u_{yy} - (u_{xy})^2 - f(x, y) \quad (38)$$

This is the PDE residual. For a correct solution, $r(x, y) \approx 0 \forall (x, y) \in \Omega$.

Step 5: Loss Function Construction

For N_f collocation points $\{(x_i, y_i)\}_{i=1}^{N_f} \in \Omega$:

$$L_{\text{PDE}} = \frac{1}{N_f} \sum_{i=1}^{N_f} [u_{xx}(x_i, y_i) u_{yy}(x_i, y_i) - (u_{xy}(x_i, y_i))^2 - f(x_i, y_i)]^2 \quad (39)$$

For N_b points on the boundary $\partial\Omega$:

$$L_{\text{BC}} = \frac{1}{N_b} \sum_{i=1}^{N_b} [\hat{u}(x_i, y_i) - g(x_i, y_i)]^2 \quad (40)$$

For convexity enforcement is optional but important for MAE:

Compute eigenvalues of the Hessian:

$$\lambda_{1,2} = \frac{u_{xx} + u_{yy} \pm \sqrt{(u_{xx} - u_{yy})^2 + 4u_{xy}^2}}{2} \quad (41)$$

Then we have;

$$L_{\text{convex}} = \frac{1}{N_f} \sum_{i=1}^{N_f} \left[\max(0, -\lambda_{1,i}) + \max(0, -\lambda_{2,i}) \right]^2 \quad (42)$$

The total loss is then;

$$L_{\text{Total}} = L_{\text{PDE}} + \beta_1 L_{\text{BC}} + \beta_2 L_{\text{convex}} \quad (43)$$

Here β_1 and β_2 are *hyperparameters* that balance the different components of the loss function. They are absolutely necessary for PINNs to work correctly. Without proper weighting, the network may:

- Satisfy boundary conditions perfectly but ignore the PDE (if β_1 too large)
- Satisfy the PDE but violate convexity (if $\beta_2 = 0$)
- Fail to converge if gradients from different loss components have vastly different scales.

It is customary to choose β_1 to be 10 because boundary conditions often need higher weight.

β_2 on the other hand can be set to the value of 1 for convexity penalty.

We note that the MA equation $\det(D^2 u) = f(x, y)$ with $f > 0$ has two families of solutions:

- Convex solutions which are physically and geometrically meaningful.
- Concave solutions which also satisfy the PDE.

Without L_{convex} , the PINN could converge to a concave solution, which is mathematically valid but often not the desired one. The convexity penalty guides the network to the correct branch.

8.1 Results of the PINN simulations

We present below the simulations using the Physics Informed Neural Network.

- The first prediction graphs fig (10) uses the sigmoid activation function in the neural network. We obtain very good approximation comparing with the exact solution.
- The second prediction graphs fig (11) uses the hyperbolic tangent activation function in the neural network. We obtain very good approximation comparing with the exact solution except that in this case the graph is inverted.

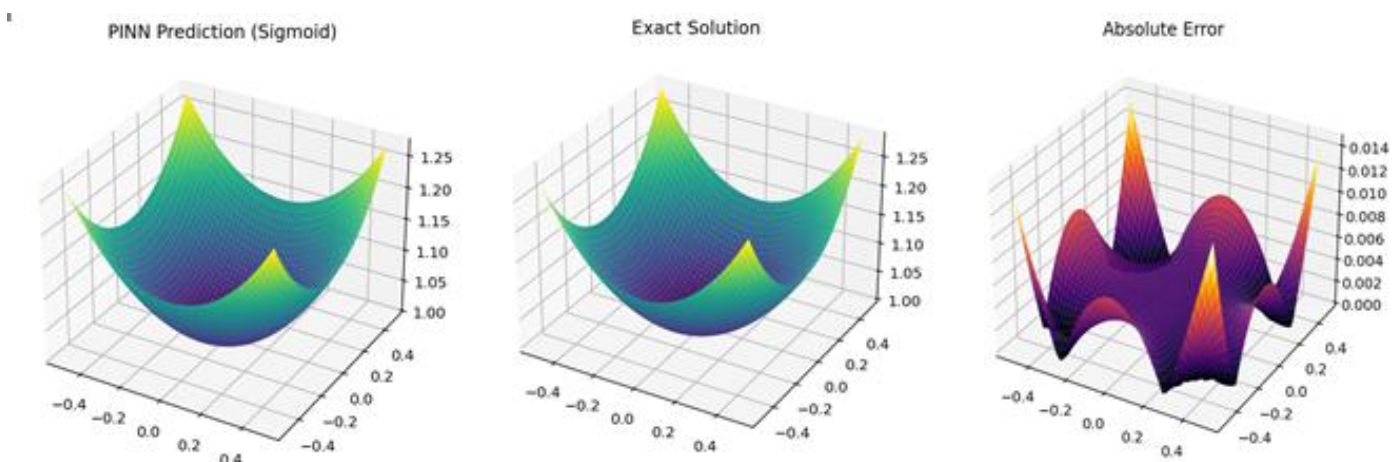


Fig 10 Predicted solution of MA PDE with unsupervised learning with sigmoid activation function.

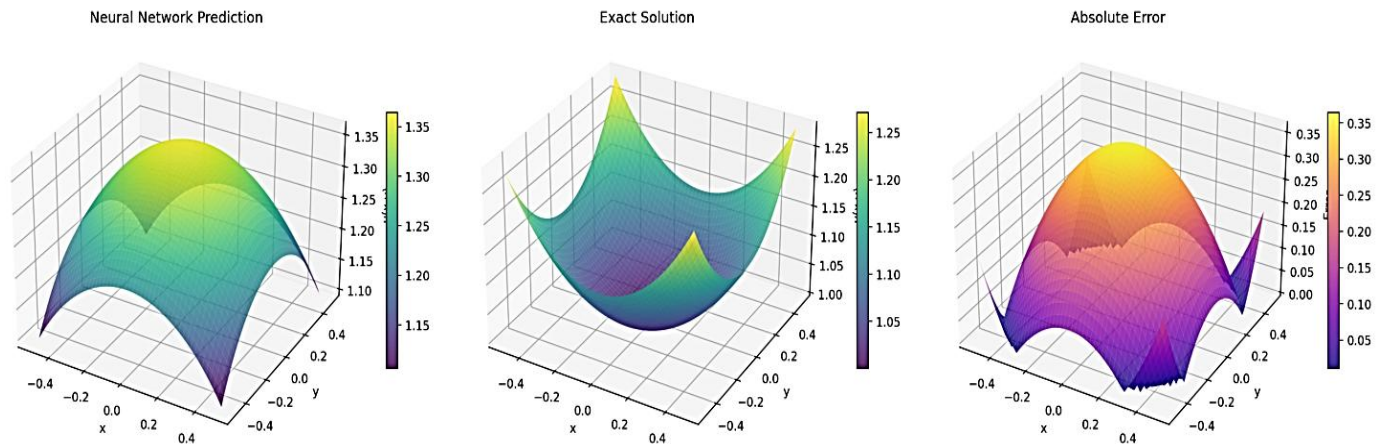


Fig 11 Predicted solution of MA PDE with unsupervised learning with tanh activation function, exact solution and error profile

8.2 Remark: In our simulation we recorded the following error losses.

- Final Total Loss = 0.006750
- Final PDE Loss = 0.000080
- Final BC Loss = 0.006670

8.3 Interpretation of tanh Activation Function MA PDE with Unsupervised Learning

The discrepancy between the neural network prediction (concave down) and the exact solution (concave up) with respect to the activation functions, reveals a fundamental issue in the PINN formulation. We state that the root cause is the missing *convexity constraint*.

The Monge-Ampère equation written as:

$$u_{xx}u_{yy} - (u_{xy})^2 = \det(\text{Hessian}(u)) = f(x, y) \quad (44)$$

has a crucial property that our PINN formulation ignores.

For the elliptic MA PDE, the solution should be convex we have $\det(\text{Hessian}(u)) > 0$.

Our exact solution: $u(x, y) = \exp((x^2 + y^2)/2)$ has positive definite Hessian which implies a convex shape (concave up). However, our neural network searches for the concave solution for which $\det(\text{Hessian}(u)) < 0$, for the following reasons :

- Missing convexity regularization: The PDE $u_{xx}u_{yy} - (u_{xy})^2 = f(x, y)$ has two branches; one convex and one concave. Without convexity constraints, the optimizer can converge to either.
- Boundary conditions alone don't enforce convexity: Our BCs specify values but not the Hessian sign.
- Neural network bias: ReLU/Tanh networks can represent both convex and concave functions, but gradient descent might get trapped in the concave branch.

8.4 The Error Profile:

Characteristic Error Pattern Fig.12 :

- Larger errors near boundaries: The concave solution deviates more from the true convex solution near edges
- Maximum error at corners: Both convex and concave solutions must match BCs at boundaries, but their interior curvature differs greatly
- Symmetric error distribution: Due to problem symmetry

$$\text{Error} = |u_{\text{convex}}(x, y) - u_{\text{concave}}(x, y)| \quad (45)$$

This difference is substantial because:

- Convex function: curves upward
- Concave function: curves downward
- Both meet at boundaries but diverge inside



Fig 12 Training Loss history for the solution of MA PDE as well as boundary condition loss.

CONCLUSION

The Monge-Ampere equation studied here is a two dimensional elliptic PDE derived from the Navier-Stokes equation which is a fundamental partial differential equation in Fluid Mechanics and has a wide field of applications. The Monge-Ampere equation arises naturally in several problems in the areas of analysis and geometry, such as the prescribed Gaussian curvature equation, affine geometry, and optimal transportation.

As we have noted, *supervised learning* offers accuracy when labeled data exist, but lacks generality for real-world PDE problems. *Unsupervised PINN-based learning*, while computationally intensive, provides a flexible, data-free alternative that is particularly attractive for nonlinear PDEs like the MA equation where analytical or numerical solutions are difficult to obtain. Compared with classical methods, PINNs trade algorithmic rigor for adaptability and dimensional scalability, offering a modern, mesh-free perspective on PDE approximation. Supervised learning for rapid evaluation when data is plentiful; and unsupervised PINNs for new, data-scarce, or geometrically complex problems, although with higher computational cost and need for specialized tuning. This work provides a foundational comparison that underscores the complementary roles of classical numerics and modern machine learning in computational PDEs.

The future of MA PDE research lies in *hybrid approaches* that combine the stability and theoretical guarantees of classical schemes with the expressive power and generalization capability of physics-informed and convex neural architectures.

REFERENCES

- [1] Pogorelov, A., (1964); Monge-Ampere Equations of Elliptic Type, Published by Noordhoff, Groningen, The Netherlands.
- [2] Alexandrov, A. and Kutateladze, S., (2006); Selected Works Part II: Intrinsic Geometry of Convex Surfaces.
- [3] Caffarelli, L., Nirenberg, L. and Spruck, J., (1984); *The Dirichlet problem for nonlinear second-order elliptic equations I: Monge-Ampère equations*, Communications on Pure and Applied Mathematics Volume **37**, Issue 3, pp. 339–402.
- [4] Kutateladze S. S. (2006) Intrinsic Geometry of Convex Surfaces. ISBN: 978-0-415-29802-5.
- [5] Ollicker, V., and Prussner, L., (1989); On the numerical solution of the equation $z_{xx}z_{yy} - z_{xy}^2 = f$ and its discretization. I Numerical. Mathematics., 54, pp. 271-293
- [6] Brenner and Neilan (2012). Finite element approximations of the three dimensional Monge-Ampère equation. *ESAIM: Mathematical Modelling and Numerical Analysis*. 46(5), 979-1001.
- [7] Feng and Neilan (2009). Vanishing moment method and moment solutions for second order fully nonlinear partial differential equations. *Journal of Scientific Computing*. 38(1), 74-98.
- [8] Froese and Oberman (2011). Convergent finite difference solvers for viscosity solutions of the elliptic Monge-Ampère equation in dimensions two and higher. *SIAM Journal on Numerical Analysis*. 49(4), 1692-1714.
- [9] Britanny D. F. & Oberman A. M. (2011). Convergent finite difference solvers for viscosity solutions of the elliptic Monge-Ampère equation in dimensions two and higher, *SIAM Journal on Numerical Analysis*. Vol. 49.
- [10] Lagaris I. E., Likas A. C., and Fotiadis D.I., (1997). Artificial Neural Network for Solving Ordinary and Partial Differential Equations, arXiv:physics/9705023v1, Greece.
- [11] Han J., Jentzen A., Weinan E., (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*. 115(34). 8505–8510.
- [12] Raissi M., Perdikaris P., Karniadakis G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*. Vol. 378. 686–707.
- [13] Hacking, R., Kusch, L., Mitra, K., Anthonissen, M., & IJzerman, W. (2024). A neural network approach for solving the Monge-Ampère equation with transport boundary condition. DOI:10.48550/arXiv.2410.19496.
- [14] Chen H. and Wang T. (2024). Machine Learning Algorithm for the Monge-Ampère Equation with Transport Boundary Conditions. *East Asian Journal on Applied Mathematics*, 14(4), 788-819. <https://doi.org/10.4208/eajam.2023-084.050923>
- [15] Nielsen, Michael A. (2023). *Neural Networks and Deep Learning: A Textbook*. Springer. DOI: 10.1007/978-3-031-43780-5
- [16] Goodfellow I., Bengio, Y., & Courville A. (2020). *Deep Learning*. MIT Press. DOI: 10.7551/mitpress/11039.001.0001
- [17] Millidge B., Yuhang S., Tommaso S., Thomas L., Rafal B. (2023). Backpropagation at the Infinitesimal Inference Limit of Energy-Based Models: Unifying Predictive Coding, Equilibrium Propagation, and Contrastive Hebbian Learning. *International Conference on Learning Representations (ICLR)*.
- [18] Hinton, Geoffrey E., & Vinyals, Oriol (2021). *Deep Learning with Backpropagation: Progress and Challenges*. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5), 1011-1026. DOI: 10.1109/TNNLS.2020.2998540