East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.2023-084.050923

Vol. **xx**, No. x, pp. 1-32
xxx 2024

Machine Learning Algorithm for the Monge-Ampère Equation with Transport Boundary Conditions

Hongtao Chen and Tong Wang*

School of Mathematical Sciences, Fujian Provincial Key Laboratory of Mathematical Modeling and High-Performance Scientific Computing, Xiamen University, Xiamen 361005, China.

Received 16 March 2023; Accepted (in revised version) 5 September 2023.

Abstract. In this article we introduce a novel numerical method to solve the problem of optimal transport and the related elliptic Monge-Ampère equation using neural networks. It is one of the few numerical algorithms capable of solving this problem efficiently with the proper transport boundary condition. Unlike the traditional deep learning solution of partial differential equations (PDEs) attributed to an optimization problem, in this paper we adopt a relaxation algorithm to split the problem into three suboptimization problems, making each subproblem easy to solve. The algorithm not only obtains the mapping that solves the optimal mass transport problem, but also can find the unique convex solution of the related elliptic Monge-Ampère equation from the mapping using deep input convex neural networks, where second-order partial derivatives can be avoided. It can be solved for high-dimensional problems, and has the additional advantage that the target domain may be non-convex. We present the method and several numerical experiments.

AMS subject classifications: 65N25, 65B99, 68T07, 65N99

Key words: Relaxation algorithm, Monge-Ampère equation, optimal transport, machine learning.

1. Introduction

The motivation for this work is the problem of optimal mass transport. In 1781, Monge proposed a study on optimal transport (OT) when considering the best way to rearrange a pile of materials from one configuration to another. Compared to its core theories in partial differential equations (PDEs), probability, analysis are mature enough cf. [12, 41], numerical methods for the OT problem remain underdeveloped.

The optimal mass transport problem can be stated as follows. Suppose we are given two probability densities:

1) f, a probability density supported on X,

2) g, a probability density supported on Y,

*Corresponding author. *Email addresses:* chenht@xmu.edu.cn (H. Chen), 847719457@qq.com (T. Wang)

where $X, Y \subset \mathbb{R}^d$, $d \geq 2$ are bounded and compact. The problem is to find a mapping $m : X \to Y$ which minimizes the transportation cost

$$m(x) = \underset{m \in \mathbb{M}}{\operatorname{argmin}} \int_X |x - m(x)|^p f(x) dx,$$

where

$$\mathbb{M} = \left\{ m : X \to Y \;\middle|\; \int_X h\big(m(x)\big) f(x) dx = \int_Y h(p) g(p) dp \right. \\ \left. \text{for all continuous test functions } h \right\}. \tag{1.1}$$

In the original problem Monge took $p = 1$; the case $p = 2$ corresponds to the Kantorovich (or Wasserstein) distance. Same as in the most commonly studied cases, we consider the case of $p = 2$ for the remainder of this paper.

An important theorem by Brenier [5, 15] states that the unique optimal mapping is the (almost everywhere) unique gradient of a convex function, which is denoted by ∇u. By using a change of variables and coordinates, we obtain

$$\det(D^2 u) = \frac{f}{g(\nabla u)}, \tag{MA}$$

along with the restriction

$$u \text{ is convex.} \tag{C}$$

The accompanying boundary condition is derived from the condition that m maps X to Y and reads

$$\nabla u(\partial X) = \partial Y. \tag{BV2}$$

OT problem is equivalent to Monge-Ampère equation with conditions (BV2).

OT is used in a wide range of fields, including computational fluid dynamics, color transfer between multiple images or deformation in the context of image processing, interpolation schemes in computer graphics, and economics, via matching and equilibrium problems. In addition, optimal transport has recently attracted the attention of biomedical-related scholars and is widely used as a data enhancement tool for guiding differentiation during single-cell RNA development as well as for improving cellular observables, thereby improving the accuracy and stability of various downstream sub-tasks. Among the many applications, we focus on mesh generation and illumination optics. In our numerical results, we will give two examples from each of these two applications.

Until now, the numerical solution of the Monge-Ampère equation is still being developed. However effective algorithms with transport conditions are still rare. An early numerical method for the related Monge-Ampère equation introduced by Oliker and Prussner [33] used a discretization based on the geometric interpretation of the solutions. Another recent method developed by Benamou *et al.* [3] introduced a new discretization wide-stencil

scheme of the determinant operator, but it had the restriction that the target domain is convex. Using the augmented Lagrangian and least-squares methods discussed by Dean and Glowinski *et al.* [17], one had been able to compute least-squares solutions, despite owing to the lack of the data, this problem has no classical solutions (solution in $\mathbf{H}^2(\Omega)$). Based on this, two least-squares methods were presented by Caboussat *et al.* [7] and Prins *et al.* [36]. In recent years, Lévy [25] has proposed a semi-discrete method to routinely handle three-dimensional problems containing millions of unknowns, and Cuturi [11] has proposed the Sinkhorn approach, which can handle arbitrary transport cost functions, and can be made very efficient in the case of quadratic cost functions.

Inspired by Prins *et al.*, in this article we advocate a relaxation algorithm to solve a well-chosen least-squares variant of the combined problem (MA, BV2, C) in combination with the machining learning method. With such an algorithm, we are able to decouple the treatment of the differential operators from the treatment of the nonlinearities.

Our algorithm differs from the algorithm of Prins *et al.* in three ways. First, we use machine learning to do least squares, which allows us to solve higher dimensional problems. Secondly, instead of using experiments to test which α (Relates to the ratio of the boundary to the interior, seeing (3.26) and (3.27) for details) is better, we can choose the α adaptively based on a machine learning strategy. Thirdly, we use a convex neural network to compute u which guarantees the convexity of u. It is worth noting that our algorithm has the limitation of only being able to deal with the problem where the support of the target density is simply connected.

The remainder of this paper is as follows. In Section 2, we introduce the numerical method and give the framework of the relaxation algorithm. In the outer iteration of the algorithm, there are three sub-optimization problems that need to be solved. In Section 3, we briefly discuss neural networks and how each suboptimization problem constructs a neural network to train. In Section 4, we briefly introduce the input convex neural network, explain how to calculate the solution u of the Monge-Ampère equation from the OT mapping m, and summarize the algorithm. In Section 5, we present numerical examples. The paper ends with a brief section devoted to a summary and conclusions.

2. Relaxation Algorithm for a Least Square Problem

In this section, we introduce our novel numerical method to solve optimal mass transport with quadratic cost function. Let $f : X \to [0, \infty)$ and $g : Y \to (0, \infty)$ be bounded functions denoting (mass) densities with bounded compact supports $X \subset R^d$ and $Y \subset R^d$. The problem is to find a mapping $m : X \to Y$, minimizing the transportation cost

$$C[m] = \int_X |x - m(x)|^2 f(x) dx.$$

According to (1.1) the source density f and target density g satisfy the condition

$$\int_X f(x) dx = \int_Y g(p) dp.$$

Due to key results by Brenier, the optimal transport plan can be characterised as the (sub) gradient of a convex function u. We transform the optimal transport problem into the combined problem (MA,BV2,C).

To this end, we require that the Jacobi matrix of m, given by

$$Dm = \begin{pmatrix} \frac{\partial m_1}{\partial x_1} & \cdots & \frac{\partial m_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial m_d}{\partial x_1} & \cdots & \frac{\partial m_d}{\partial x_d} \end{pmatrix}$$

equals a real symmetric positive semidefinite matrix P, and satisfies

$$\det(P) = \frac{f}{g(m)}.$$

Since P is symmetric, we have $\partial m_i/\partial x_j = \partial m_j/\partial x_i$. This implies that m is a conservative vector field and thus the gradient of a function. Introducing $V = [C^2]^d$, which is the set of d-dimensional, twice continuously differentiable vector fields, we enforce the equality $Dm = P$ by minimizing the following functional over $V \times Q_{f/g}$:

$$\begin{aligned} J_I(m,P) &= \frac{1}{2}\int_X \|Dm - P\|_F^2 dx, \\ Q_{f/g(m)} &= \left\{ q \in Q, \det(q) = \frac{f}{g(m)} \right\}, \\ Q &= \left\{ q \in L^2(\Omega)^{d\times d},\ q = q^t \right\}, \end{aligned} \tag{2.1}$$

where $\|\cdot\|_F$ is the Fröbenius norm. In addition to mass conservation, we require that the mapping satisfies

$$m(X) = Y, \quad m(\partial X) = \partial Y.$$

This boundary condition is nonstandard, nonlinear, and requires special attention. We address the boundary condition by minimizing a second functional

$$J_b(m,b) = \frac{1}{2}\int_{\partial X} |m-b|^2 ds.$$

We minimize this functional over b from the set

$$\mathbb{B} = \left\{ b \in \left[C^1(\partial X)\right]^d \mid b(x) \in \partial Y, \forall x \in \partial X \right\}.$$

We combine the functionals J_I for the interior and J_b for the boundary by a weighted average

$$J(m,P,b) = (1-\alpha)J_b(m,b) + \alpha J_I(m,P).$$

Under ideal conditions that is P, b and m are in continuous space, we calculate the minimizers by a relaxation algorithm, that is repeatedly minimizing over the three sets $V, Q_{f/g(m)}$,

and $\mathbb{B}$ separately. Due to the fact that the optimal mass transport problem has a solution, there is a set of parameters P, b and m such that $J(m, b, P) = 0$.

In practice, by using three neural networks we calculate the minimizers by a relaxation algorithm, that is repeatedly minimizing over the three sets $\widetilde{V}$, $\widetilde{Q}_{f/g(m)}$, and $\widetilde{\mathbb{B}}$ separately. $\widetilde{V}$ is a finite set of points of V, $\widetilde{Q}_{f/g(m)}$ is a finite set of points of $Q_{f/g(m)}$, and $\widetilde{\mathbb{B}}$ is a finite set of points of $\mathbb{B}$.

The initialization strategy is as follows: Without loss of generality, assume that both X and Y are d-dimensional spaces, and Y is simply connected, because the target density function $g(p)$ must be nonzero. Let $[a_1, b_1]\times[a_2, b_2]\times\cdots\times[a_d, b_d]$ be the smallest bounding box of X that are parallel to the axes. Also let $[c_1, d_1]\times[c_2, d_2]\times\cdots\times[c_d, d_d]\supset Y$ be the smallest bounding box of Y that are parallel to the axes. The initial guess m^0 is given by

$$m_i^0 = \frac{x_i - a_i}{b_i - a_i} d_i + \frac{b_i - x_i}{b_i - a_i} c_i, \tag{2.2}$$

where $i = 1, 2, 3, \ldots, d$, and m_i^0 is the i-th component of the initial guess m^0.

Subsequently, we perform the iteration

$$b^{n+1} = \operatorname*{argmin}_{b\in\widetilde{\mathbb{B}}} J_b(m^n, b), \tag{2.3a}$$

$$P^{n+1} = \operatorname*{argmin}_{P\in\widetilde{Q}_{f/g(m^n)}} J_I(m^n, P), \tag{2.3b}$$

$$m^{n+1} = \operatorname*{argmin}_{m\in\widetilde{V}} J(m, b^{n+1}, P^{n+1}). \tag{2.3c}$$

This procedure is repeated until the value of $J(m^n, b^n, P^n)$ stabilizes.

3. Minimizing Procedures for b, P, m

In this section, firstly we briefly introduce fully connected feedforward neural network in Section 3.1. Secondly, by using the initial guess m^0, we start the iteration process. Each iteration consists of three steps, which are described one by one in Sections 3.2-3.4.

3.1. Fully connected feedforward neural network

An increasing number of algorithms have emerged in recent years for solving PDEs using neural networks [9, 18, 20, 21, 27, 29, 44]. Our algorithm also uses neural networks to solve the problem. In this paper, we consider two kinds of neural networks: standard fully connected feedforward neural network (FNN) and input convex neural networks. In this section we describe the standard fully connected feedforward neural networks, which can approximate arbitrarily well a large class of input-output maps, i.e., they are universal approximators, cf. [10,35]. The core of our algorithm is to build three FNN(Net_i) to update b, P, m with $M_i \geq 3$ layers, of which $(M_i - 2)$ are hidden layers, $i = 1, 2, 3$. Since b^{n+1} of (2.3a), P^{n+1} of (2.3b), m^{n+1} of (2.3c) are our approximation goal, we will consider Net_1 and

Net_3 as a $R^d \to R^d$ map, Net_2 as a $R^d \to R^{(d^2+d)/2}$ map ($R^{(d^2+d)/2}$ denotes the elements of the upper triangle of the symmetric matrix P). Let n_j^i, $j = 1, \ldots, M_i$, be the number of neurons in each layer of Net_i, we then have $n_1^1 = n_{M_1}^1 = n_1^3 = n_{M_3}^3 = n_1^2 = d$ and $n_{M_2}^2 = (d^2+d)/2$.

We can consider Net_i as an operator. For any input $\mathbf{y}^{in} \in R^d$, the output of the network Net_i is

$$\mathbf{y}^{\text{out}} = \mathbf{Net_i}(\mathbf{y}^{in}; \Theta),$$

where Θ is the parameter set including all the parameters in the network. The operator is a composition of the following operators:

$$\text{Net}_i(\cdot; \Theta) = (\sigma_{M_i} \circ \mathbf{W}_{M_i - 1}) \circ \cdots \circ (\sigma_2 \circ \mathbf{W}_1),$$

where $\mathbf{W}_j$ is a matrix for the weight parameters connecting the neurons from j-th layer to $(j+1)$-th layer, after using the standard method of adding biases to the weights. The so-called activation functions $\sigma_j : R \to R$ in the j-th layer are often defined by applying some non-affine function such as sigmoid functions, ReLU (rectified linear unit), etc. In this paper, we use $\tanh(x)$ as the activation function in all layers i.e. $\sigma_j(x) = \tanh(x)$, except at the output layer $\sigma_{M_i}(x) = x$. This is one of the common choices for FNN.

3.2. Minimizing procedure for b

In this section, we construct the neural network $\text{Net}_1$to perform the minimization step (2.3a). We minimize the functional $J_B(m, b)$ over $b \in \widetilde{\mathbb{B}}$. As we said before $m = m^n$ is fixed in this step. If $n = 1$, m is the initialization choice determined by (2.2), if $n > 1$, m is the output of Net_3. The update in this step is only done on the boundary ∂X, so we choose $\{x_i\}_{i=1}^{M_b}$ as a set of some sufficiently densely chosen collocation points on the boundary ∂X. The input of Net_1 is the coordinates of the boundary sampling points $\{x_i\}_{i=1}^{M_b}$, and the output is the coordinates of b. Because no derivative of b with respect to either direction in the functional J_b, the minimization can be done point-by-point. So the minimization step (2.3) equals to

$$\min_{b_i \in \partial Y} |m_i - b_i|, \tag{3.1}$$

where m_i indicates the mapping value for each point $x_i \in \partial X$. In other words, m_i is the output value of Net_3 with input x_i. In general, we assume that the boundary of the target domain Y can be expressed in terms of the elementary function. Supposing the analytical expression is $B(x)$ (for example, boundary on a circle: $x^2 + y^2 - 1 = 0$). Multiply the constraints on the boundary by a sufficiently large penalty factor Λ_1 as part of the optimization problem (3.1). Thus, the loss function corresponding to the minimal value problem takes the form

$$E_1 := \frac{1}{M_b} \sum_{i=1}^{M_b} \left(|m_i - b_i|^2 + \Lambda_1 B^2(b_i) \right). \tag{3.2}$$

For complex boundary cases such as polygons, we discretize the boundary of Y by employing points $z_k \in \partial Y$, $k = 1, 2, \ldots, N_b$ with increasing index along the boundary, and define

$z_{N_b+1} = z_1$. We connect neighboring points using line segments (z_k, z_{k+1}) and determine the closest point to $m_{i,j}$ on all line segments (z_k, z_{k+1}).

First we determine for a given point $m_{i,j}$ and a given line segment (z_k, z_{k+1}), the projection m_i^k of m_i on the line through z_k and z_{k+1}. This projection is given by

$$m_i^k = z_k + t_k(z_{k+1} - z_k),$$
$$t_k = \frac{(m_i - z_k)\cdot(z_{k+1} - z_k)}{|z_k - z_{k+1}|}.$$

If $t_k \in [0,1]$, the projected point is on the line segment, and the nearest point to $m_{i,j}$ is given by $m_{i,j}^k$. If $t_k < 0$, the projected point is not on the line segment and the nearest point is given by z_k, and if $t_k > 1$, the nearest point is given by z_{k+1}. Thus, the nearest point on the line segment (z_k, z_{k+1}) is given by

$$\tilde{m}_i^k = z_k + \min(1, \max(0, t_k))(z_{k+1} - z_k).$$

For each point $x_{i,j}$ on ∂X, obviously we can consider the nearest point $b_{i,j}$ on all the line segments by

$$b_i = \underset{\tilde{m}_i^k}{\operatorname{argmin}} \left|\tilde{m}_i^k - m_i\right|^2.$$

For high-dimensional problems, we only consider two simple cases, one in which the boundary function can be expressed analytically, and the other is the unit cube.

For the all sampling points $\sum_{i=1}^{M_b} x_i$ on ∂X, the loss function is given by

$$E_1 := \frac{1}{M_b}\sum_{i=1}^{M_b} |b_i - m_i|^2. \tag{3.3}$$

It is worth noting that, although E_1 contains the output of Net_3, we only optimize the parameters of Net_1 by cutting off backpropagation with respect to the Net_3 parameters.

3.3. Minimizing procedure for P

In this section, we construct the neural network Net_2 to perform the minimization step (2.3b). We minimize $J_I(m,P)$ defined in (2.1) over the matrices $P \in \widetilde{Q}_{f/g(m^n)}$. Thus, we require that P is symmetric and satisfies

$$\det(P) = \frac{f}{g(m)}.$$

We assume that $m = m^n$ is given in this step. Same as minimizing procedure for b, if $n = 1$, m is the initialization choice determined by (2.2), and if $n > 1$, m is the output of Net_3. The update in this step is performed only in region X (not on the boundary), so we choose $\{x_i\}_{i=1}^{M_a}$ as a set of some sufficiently densely chosen collocation points in the domain X.

Since the integrand of $J_I(m,P)$ does not contain derivatives of P, the minimization can be carried out pointwise; thus for each $x_i \in X$ we have to minimize $\|D-P\|_F$, where D is the Jacobi matrix of m. For ease of presentation, we introduce the notation

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{pmatrix}, \quad D = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}$$

with $d_{11}, d_{12}, d_{21}, d_{22}$ can be obtained by back propagation of Net_3, respectively. The input of Net_2 is the coordinates of the internal sampling points $\{x_i\}_{i=1}^{M_a}$, and the output is elements (p_{11}, p_{12}, p_{22}) of the triangle on matrix P. The same can be deduced when the dimension is higher than 2. Next, we introduce the function

$$H(p_{11}, p_{12}, p_{22}) = \frac{1}{2}\|D-P\|_F^2.$$

Thus, for each $x_i \in X$, we have the following problem:

$$\begin{aligned} &\text{minimize} \quad H(p_{11}, p_{12}, p_{22}), \\ &\text{satisfying} \quad p_{11}p_{22} - p_{12}^2 = \frac{f}{g}, \end{aligned} \tag{3.4}$$

where f/g is shorthand notation for $f(x_i)/g(m(x_i))$. If the dimension is higher than 2, this problem can not be solved analytically. Naturally we construct the loss function of Net_2

$$E_2 := \frac{1}{M_a}\left(\|D-P\|_F^2 + \Lambda_2\left(\det(P) - \frac{f}{g}\right)\right), \tag{3.5}$$

where Λ_2 is a suitably large constant.

The following analysis and results in this section are from Prins *et al.* [36], whose results we use to construct neural networks to solve the problem, and to which we add proof details such as the discriminant of a quadratic equation in one variable with no real roots.

We study the situation when the space dimension is 2, i.e. $d = 2$. For each fixed $x_i \in X$, there is a unique set (p_{11}, p_{12}, p_{22}) corresponding to it by solving (3.4) analytically. Therefore E_2 is different from (3.5). First, we slightly simplify the minimization problem. Let D_S be the symmetric part of D, i.e.,

$$D_S = \begin{pmatrix} d_{11} & d_S \\ d_S & d_{22} \end{pmatrix}$$

with $d_S = (d_{12} + d_{21})/2$. The new function to be minimized reads

$$H_S(p_{11}, p_{12}, p_{22}) = \frac{1}{2}\|D_S - P\|^2 = H(p_{11}, p_{12}, p_{22}) - \frac{1}{4}(d_{12} - d_{21})^2.$$

Because $(d_{12}-d_{21})^2$ is constant with respect to p_{11}, p_{22}, and p_{12}, and we are only interested in the minimizer p_{11}, p_{12}, p_{22} and not in its value $H(p_{11}, p_{12}, p_{22})$, the minimization problem (3.4) is equivalent to

$$\begin{aligned} &\text{minimize} \quad H_S(p_{11}, p_{12}, p_{22}), \\ &\text{satisfying} \quad p_{11}p_{22} - p_{12}^2 = \frac{f}{g}. \end{aligned} \tag{3.6}$$

Given d_{11}, d_{22}, d_S, and f/g, we distinguish several cases, with at least one and at most four possible solutions p_{11}, p_{12}, p_{22} that minimize $H_S(p_{11}, p_{12}, p_{22})$ and satisfy the nonlinear constraint $p_{11}p_{22} - p_{12}^2 = f/g$, which are (local) minima, maxima, or saddle points. We will show that this is always possible and refer to these solutions as possible minimizers. Finally, we determine the global minimizer by substituting each possible minimizer in the function $H_S(p_{11}, p_{12}, p_{22})$.

The possible minimizers of (3.6) are critical points of the Lagrangian function $L = L(p_{11}, p_{12}, p_{22}, \lambda)$ defined by

$$L(p_{11}, p_{12}, p_{22}, \lambda) = \frac{1}{2}\|D_S - P\|^2 + \lambda\left(\det(P) - \frac{f}{g}\right), \tag{3.7}$$

where λ is the Lagrange multiplier. From the Lagrange's multiplier theorem, we obtain the following algebraic system:

$$p_{11} + \lambda p_{22} = d_{11}, \tag{3.8a}$$
$$\lambda p_{11} + p_{22} = d_{22}, \tag{3.8b}$$
$$(1-\lambda)p_{12} = d_S, \tag{3.8c}$$
$$p_{11}p_{22} - p_{12}^2 = \frac{f}{g}. \tag{3.8d}$$

It is obvious that if $\lambda \neq \pm 1$, the coefficient matrix of the system of linear equations consisting of (3.8a), (3.8b), (3.8c) about p_{11}, p_{22}, p_{12} is invertible.

Assuming that $\lambda \neq \pm 1$, we obtain that

$$\begin{aligned} p_{11} &= \frac{\lambda d_{22} - d_{11}}{\lambda^2 - 1}, \\ p_{22} &= \frac{\lambda d_{11} - d_{22}}{\lambda^2 - 1}, \\ p_{12} &= \frac{\lambda d_S}{1-\lambda^2}. \end{aligned} \tag{3.9}$$

Substituting these expressions in (3.8d) gives the quartic equation

$$a_4\lambda^4 + a_2\lambda^2 + a_1\lambda + a_0 = 0 \tag{3.10}$$

with coefficients given by

$$\begin{aligned} a_4 &= \frac{f}{g}, \\ a_2 &= -2\frac{f}{g} - \det(D_S), \\ a_1 &= \|D_S\|^2 \geq 0, \\ a_0 &= \frac{f}{g} - \det(D_S). \end{aligned} \tag{3.11}$$

Substituting the solution in (3.8) yields the possible minimizers of $H_S(p_{11}, p_{12}, p_{22})$.

Next, we will discuss the solution of (3.10) in three situations under the assumption that $\lambda \neq \pm 1$.

The first situation is when $f > 0$. In this situation we can rewrite the quartic equation as two quadratic equations using Ferrari's method [39]. Because of $f > 0$, $a_4 > 0$. We rewrite (3.10) by dividing a_4

$$\left(\lambda^2 + \frac{a_2}{2a_4}\right)^2 = -\frac{a_1}{a_4}\lambda - \frac{a_0}{a_4} + \left(\frac{a_2}{2a_4}\right)^2.$$

Adding an arbitrary variable y to the square on the left side and the consequent additional term to the right side yields

$$\left(\lambda^2 + \frac{a_2}{2a_4} + y\right)^2 = 2y\lambda^2 - \frac{a_1}{a_4}\lambda - \frac{a_0}{a_4} + \left(\frac{a_2}{2a_4}\right)^2 + \frac{a_2}{a_4}y + y^2. \tag{3.12}$$

Attempting to represent the right-hand side as a perfect square yields the following equation:

$$\left(\lambda^2 + \frac{a_2}{2a_4} + y\right)^2 = \left(\sqrt{2y}\lambda - \frac{a_1}{2a_4\sqrt{2y}}\right)^2. \tag{3.13}$$

By equating the right-hand sides of (3.12) and (3.13), we see that this is only conceivable, if y is the solution of the following cubic equation:

$$y^3 + b_2 y^2 + b_1 y + b_0 = 0 \tag{3.14}$$

with coefficients given by

$$b_2 = \frac{a_2}{a_4}, \quad b_1 = \frac{1}{4}\left(\frac{a_2}{a_4}\right)^2 - \frac{a_0}{a_4}, \quad b_0 = -\frac{1}{8}\left(\frac{a_1}{a_4}\right)^2.$$

A real solution for y can be obtained by Cardano's method [39]

$$\begin{aligned} G &= \frac{3b_1 - b_2^2}{3}, \quad K = \frac{27b_0 - 9b_2b_1 + 2b_2^2}{27}, \\ z &= \sqrt[3]{-\frac{K}{2} + \sqrt{\left(\frac{K}{2}\right)^2 + \left(\frac{G}{3}\right)^3}} + \sqrt[3]{-\frac{K}{2} - \sqrt{\left(\frac{K}{2}\right)^2 + \left(\frac{G}{3}\right)^3}}, \\ y &= z - \frac{b_2}{3}. \end{aligned} \tag{3.15}$$

We obtain from (3.12) using the value of y given by (3.15)

$$\lambda^2 + \frac{a_2}{2a_4} + y = \pm\left(\sqrt{2y}\lambda - \frac{a_1}{2a_4\sqrt{2y}}\right),$$

where λ is represented by two quadratic equations. By solving both, the following four roots of the quartic equation are obtained:

$$\begin{aligned}
\lambda_1 &= -\sqrt{\frac{y}{2}} + \sqrt{-\frac{y}{2} - \frac{a_2}{2a_4} + \frac{a_1}{2a_4\sqrt{2y}}},\\
\lambda_2 &= -\sqrt{\frac{y}{2}} - \sqrt{-\frac{y}{2} - \frac{a_2}{2a_4} + \frac{a_1}{2a_4\sqrt{2y}}},\\
\lambda_3 &= \sqrt{\frac{y}{2}} + \sqrt{-\frac{y}{2} - \frac{a_2}{2a_4} - \frac{a_1}{2a_4\sqrt{2y}}},\\
\lambda_4 &= \sqrt{\frac{y}{2}} - \sqrt{-\frac{y}{2} - \frac{a_2}{2a_4} - \frac{a_1}{2a_4\sqrt{2y}}}.
\end{aligned} \tag{3.16}$$

But there may be two problems. First, if $y = 0$, division will be divided by zero. It is not allowed, because when $y = 0$, from (3.14) we can obtain $b_0 = 0$, therefore $a_1 = \|D_S\|^2 = 0$ which contradicts our assumptions $\lambda \neq \pm 1$ ($\lambda = \pm 1$ case will be discussed later). Second, a quartic equation may have no real roots. Fortunately this will not happen again in our algorithm. By using [14, Theorem 5], we can obtain the necessary condition for the quartic equation to have no real roots

$$\begin{aligned}
&\Delta_4 \geq 0,\\
&(H \geq 0) \quad \text{or} \quad (F \leq 0),
\end{aligned} \tag{3.17}$$

where Δ_4 is the discriminant of the quartic equation. Δ_4, F and H are given by

$$\begin{aligned}
\Delta_4 &= 256a_4^3a_0^3 - 128a_4^2a_2^2a_0^2 + 144a_4^2a_2a_1^2a_0\\
&\quad - 27a_4^2a_1^4 + 16a_4a_2^4a_0 - 4a_4a_2^3a_1^2,\\
H &= \frac{8a_2}{a_4}, \quad F = \frac{16a_2^2}{a_4^2} - \frac{64a_0}{a_4}.
\end{aligned}$$

For ease of expression, we define

$$V = \frac{\det D_s}{f/g}, \quad r = \frac{\|D_s\|^2}{f/g} \geq 0. \tag{3.18}$$

From (3.17) we find

$$(V \leq -2) \quad \text{or} \quad (-8 \leq V \leq 0). \tag{3.19}$$

We rewrite the discriminant

$$\Delta_4 = (2V - r)(2V + r)(27r^2 + 256 - 192V - 60V^2 - 4V^3).$$

From (3.19) we have

$$256 - 192V - 60V^2 - 4V^3 \geq 0.$$

Thus, the following is a required condition for the quartic equation to have no actual roots:

$$2V \leq -r.$$

Substituting (3.18) and simplifying the inequality we find

$$(d_{11} + d_{22})^2 \leq 0.$$

This means $d_{11} = -d_{22}$, which contradicts with the hypothesis $\lambda \neq -1$. So the real roots of the quartic equation given by (3.16) are substituted in (3.9) and (3.4), yielding at least two and at most four critical points of the Lagrangian $L(p_{11}, p_{12}, p_{22}, \lambda)$ given by (3.7), and thus at least two and at most four possible minimizers of (3.6).

The second situation is when $f = 0$. That means the source density is zero. Because $a_4 = 0$, (3.10) reduces to a quadratic equation. And discriminant of this quadratic equation is

$$\Delta = a_1^2 - 4a_2a_0 = \left(d_{11}^2 - d_{22}^2\right)^2 + 4d_S^2 (d_{11} + d_{22})^2 > 0.$$

Always the two distinct real roots are given by

$$\lambda = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2a_0}}{2a_2}.$$

Substituting these values of λ in (3.9), we can find two possible minimizers.

The third situation is when $f = 0$ and $a_2 = 0$. Substituting $f = 0$ and $a_2 = 0$ in (3.11), we can find $\|D_S\| = 0$, i.e. $d_{11}d_{22} - ds^2 = 0$ and $a_0 = 0$. Therefore, the (3.10) reduces to a linear equation with the coefficient of the constant term to be 0. Obviously, the only root is $\lambda = 0$. Again, the values of p_{11}, p_{12}, and p_{22} are determined using (3.9), yielding one potential minimizer $p_{11} = d_{11}$, $p_{12} = d_S$ and $p_{22} = d_{22}$.

Next, we discuss the two situations when $\lambda = \pm 1$.

The fourth situation is when $\lambda = 1$. Substituting $\lambda = 1$ into (3.8) gives us $d_{11} = d_{22}$ and $d_s = 0$. At this point we cannot calculate (p_{11}, p_{12}, p_{22}) by (3.9). Consequently, we compute the minimizer of $H(p_{11}, p_{12}, p_{22}, \lambda)$ using a different method. First of all, we simplify (3.4) by using $d_{11} = d_{22}$ and $d_s = 0$

$$\begin{aligned} &\text{minimize} \quad H(p_{11}, p_{12}, p_{22}) = \frac{1}{2}\left((p_{11} - d_{11})^2 + 2p_{12}^2 + (p_{22} - d_{11})^2\right), \\ &\text{satisfying} \quad p_{11}p_{22} - p_{12}^2 = \frac{f}{g}. \end{aligned}$$

Secondly, using the constraint $p_{11}p_{22} - p_{12}^2 = f/g$, we transform the optimization problem over R^3 into a problem over R^2 restricted to the domain, where $p_{12} = \pm\sqrt{p_{11}p_{22} - f/g}$ is real,

$$\underset{(p_{11}, p_{22})}{\operatorname{argmin}} \frac{1}{2}\left((p_{11} - d_{11})^2 + 2\left(p_{11}p_{22} - \frac{f}{g}\right) + (p_{22} - d_{11})^2\right). \tag{3.20}$$

The minimizer can be found in the interior of this domain or on the boundary. If the minimizer is in the interior, by setting the derivatives with respect to p_{11} and p_{22} to 0, we will obtain a critical line

$$p_{11} + p_{22} = d_{11}. \tag{3.21}$$

Since p_{12} must be a real number, from $p_{12} = \pm\sqrt{p_{11}p_{22} - f/g}$, we deduce that $p_{11}p_{22} - f/g \geq 0$. Combining $p_{11}p_{22} - f/g \geq 0$ with $p_{11} + p_{22} = d_{11}$ yields

$$p_{11}^2 - d_{11}p_{11} + \frac{f}{g} \leq 0.$$

In the same way because p_{11} is real, the discriminant of the quadratic equation on the left hand side greater than or equal to 0, i.e. $\Delta = d_{11}^2 - 4f/g \geq 0$. Then, the part of the line corresponding to real values of p_{11} is given by

$$\frac{d_{11} - \sqrt{d_{11}^2 - 4f/g}}{2} \leq p_{11} \leq \frac{d_{11} + \sqrt{d_{11}^2 - 4f/g}}{2}. \tag{3.22}$$

When (3.21) holds and p_{11} satisfies (3.22), we can get more than one set $(p_{11}, p_{12}, p_{22}) \in R^3$ such that (3.20) is minimized. Bringing (3.21) to $H(p_{11}, p_{12}, p_{22})$ yields the minimum value

$$H_{interior} = \frac{1}{2}d_{11}^2 - \frac{f}{g},$$

where $H_{interior}$ is the shorthand notation for $H(p_{11}, p_{12}, p_{22})$ when minimizer is in the interior of the domain.

Next, we need to find possible minimizers on the boundary of the domain where p_{12} is real. The boundary is an hyperbola, given by $p_{12} = 0$

$$p_{11}p_{22} = \frac{f}{g}.$$

Bringing into (3.20) gives

$$\underset{p_{11}\in\mathbb{R}}{\operatorname{argmin}} \frac{1}{2}\left((p_{11} - d_{11})^2 + \left(\frac{f}{gp_{11}} - d_{11}\right)^2\right). \tag{3.23}$$

The minimizer satisfies the derivative of (3.23) with respect to p_{11}

$$p_{11} - d_{11} + \frac{f\,d_{11}}{gp_{11}^2} - \frac{f^2}{g^2p_{11}^3} = 0.$$

Multiplying both sides of the equation by p_{11}^3 and then factoring gives

$$\left(p_{11}^2 - \frac{f}{g}\right)\left(p_{11}^2 - d_{11}p_{11} + \frac{f}{g}\right) = 0.$$

The first two solutions are given by

$$p_{11} = \frac{d_{11} \pm \sqrt{d_{11}^2 - 4f/g}}{2},$$

correspond either to the ends of the line segment specified by (3.22), or are complex value. If they are complex value, they can be ignored obviously. If they correspond to the segment's borders, they will provide the same $H(p_{11}, p_{12}, p_{22})$ value as the case where minimizer is in the interior of the domain, and can therefore be ignored as well.

The other two solutions are given by

$$p_{11} = \pm\sqrt{\frac{f}{g}},$$

correspond to the following two possible minimizers:

$$p_{11} = p_{22} = \pm\sqrt{\frac{f}{g}}, \quad p_{12} = 0.$$

Substituting these expressions in $H(p_{11}, p_{12}, p_{22})$ yields the minimum value in boundary of the domain

$$H_{boundary} = \frac{f}{g} \pm 2d_{11}\sqrt{\frac{f}{g}} + d_{11}^2,$$

where $H_{boundary}$ is the shorthand notation for $H(p_{11}, p_{12}, p_{22})$ when minimizer is on the boundary of the domain. Subtracting $H_{interior}$ from $H_{boundary}$ yields

$$H_{boundary} - H_{interior} = \frac{1}{2}d_{11}^2 + 2\frac{f}{g} \pm 2d_{11}\sqrt{\frac{f}{g}} \geq 0,$$

therefore we only need to consider the case where the minimizer is in the interior when $d_{11}^2 - 4f/g \geq 0$. For the sake of simplicity, we select p_{11} in the middle of the line segment.

To summarize the above, we can determine (p_{11}, p_{12}, p_{22}) provided by

$$\begin{cases} p_{11} = p_{22} = \dfrac{d_{11}}{2}, \quad p_{12} = \pm\sqrt{\dfrac{d_{11}^2}{4} - \dfrac{f}{g}}, & \text{if} \quad d_{11}^2 - 4\dfrac{f}{g} \geq 0, \\ p_{11} = p_{22} = \pm\sqrt{\dfrac{f}{g}}, \quad p_{12} = 0, & \text{otherwise.} \end{cases}$$

The fifth situation is when $\lambda = -1$. Substituting $\lambda = -1$ into (3.8) gives us $d_{11} = -d_{22}$. Now we cannot calculate (p_{11}, p_{12}, p_{22}) by (3.9). Consequently, we compute the minimizer of $H(p_{11}, p_{12}, p_{22}, \lambda)$ using a different method. We find from (3.8a) and (3.8c)

$$p_{11} - p_{22} = d_{11}, \quad p_{12} = \frac{d_s}{2}.$$

Substituting these in (3.8d) we can obtain a quadratic equation

$$p_{11}^2 - d_{11}p_{11} - \frac{d_S^2}{4} - \frac{f}{g} = 0.$$

Solving for p_{11} we find two solutions

$$\begin{aligned} p_{11} &= \frac{d_{11}}{2} \pm \frac{\sqrt{d_{11}^2 + 4f/g + d_S^2}}{2}, \\ p_{12} &= \frac{d_S}{2}, \\ p_{22} &= -\frac{d_{11}}{2} \pm \frac{\sqrt{d_{11}^2 + 4f/g + d_S^2}}{2}, \end{aligned}$$

which are always real.

Summary. The following is the process for minimizing over P. When the space dimension is 2, for each point x_i which is in the interior of domain X, we calculate the value of $f(x_i)/g(m(x_i))$, where $m(x_i)$ is the output of the Net_3 with input x_i. We determine which of the five situations discussed above applies and find at least one and at most four possible minimizers $(p_{11,i}, p_{12,i}, p_{22,i})$ for fixed x_i. From this and (3.5), we can summarize the modified loss function of Net_2

$$E_2 := \begin{cases} \dfrac{1}{M_a}\displaystyle\sum_{i=1}^{M_a}\left(\|D-P\|_F^2 + \Lambda_2\left(\det(P) - \frac{f}{g}\right)\right), & d > 2, \\ \dfrac{1}{M_a}\displaystyle\sum_{i=1}^{M_a}\left((p_{11}-p_{11,i})^2 + (p_{12}-p_{12,i})^2 + (p_{22}-p_{22,i})^2\right), & d = 2. \end{cases} \tag{3.24}$$

In practice, when $d > 2$ the $\|D-P\|_F^2$ part of E_2 can be modified to

$$\frac{1}{M_a}\sum_{i=1}^{M_a}\sum_{m\leqslant n}\frac{|P_{mn}-d_{mn}|^2}{d_{mn}^2}$$

to prevent optimization towards one component while ignoring other components. Similar to the minimizing procedure for b, it is worth noting that, although E_2 contains the output of Net_3, we only optimize the parameters of Net_2 by detaching the output of Net_3.

3.4. Minimizing procedure for m

In this section, we construct the neural network Net_3 to perform the minimization step (2.3c). We assume P and b are given by Net_1 and Net_2 and minimize $J(m, P, b)$ over the functions $m \in \widetilde{V}$. For ease of notation, we omit the indices n and $n+1$. In contrast to the other two minimization steps, this step cannot be performed pointwise. The update

in this step is not only done on the boundary ∂X, but also in the interior of domain X, so collocation points are $\{x_i\}_{i=1}^{M_b+M_a}$. At this point the loss function of Net_3 is

$$E_3 := (1-\alpha)E_1 + \alpha E_2. \tag{3.25}$$

It is worth noting that, although E_3 contains the output of Net_1 and Net_2, we only optimize the parameters of Net_3 by detaching the output of Net_1 and Net_2.

As can be seen from Fig. 3, the choice of α has a large impact on the performance of the algorithm (speed of convergence, degree of fit of the solution at the boundary, etc.) Next we give two strategies for α selection:

1. Frequency principle [45]. It means Deep Neural Networks (DNNs) often fit target functions from low to high frequencies during the training process. For this algorithm, fitting the boundary is more difficult. In order to make the boundary fit better, α should be taken relatively small, such as 0.2, 0.1, 0.05, etc.

2. α adaptive [26]. We train Net_3 with minimax architecture, which changes the minimization E_3 to

$$\min_x \max_{\mathbf{w}} E_3(\mathbf{w}, x) = \alpha_1(\mathbf{w})E_1(x) + \alpha_2(\mathbf{w})E_2(x), \tag{3.26}$$

where the weights of different losses α_1, α_2 are now functions of parameters $\boldsymbol{w} = (w_1, w_2)$.

In this article, the weights of different losses are defined as the softmax functions

$$\alpha_i(\mathbf{w}) = \frac{\exp(w_i)}{\exp(w_1) + \exp(w_2)}, \qquad i = 1, 2. \tag{3.27}$$

After applying softmax functions, the range of the weights of different losses $\alpha_i(\mathbf{w})$ will be in the interval $[0, 1]$, and they will add up to one, similar to (3.25). Consider the updates of a gradient descent/ascent approach to this problem:

$$\begin{aligned} x^{k+1} &= x^k - \eta_k \nabla_x E_3(\mathbf{w}, x), \\ w_1^{k+1} &= {w_1}^k + \rho_1^k \nabla_{w_1} E_3(\mathbf{w}, x), \\ w_2^{k+1} &= {w_2}^k + \rho_2^k \nabla_{w_2} E_3(\mathbf{w}, x), \end{aligned}$$

where $\eta_k > 0$ is the learning rate for the neural network weights x^k at step k, $\rho_s^k > 0$ is a separate learning rate for the self-adaption weights, for $s = 1, 2$. Combining two strategies, we can initialize the Net_3 parameters w such that $\alpha = 0.2$ or a sufficiently small value in the interval $[0, 1]$, and train it with minimax architecture. At this point usually the algorithm has good performance.

4. Computation of u and Algorithm Summary

The minimization steps from Section 3 are repeated until convergence. As we are interested in the solution of the Monge-Ampère equation, we derive u from the mapping m by $\nabla u = m$, i.e., we calculate u such that $\nabla u = m$. This u will be the approximate solution of the Monge-Ampère equation (MA) with boundary condition (BV2). In the ideal situation, $Dm = P$ and thus $d_{ij} = d_{ji}$. In this case there exists a function u such that $\nabla u = m$, because m is a conservative vector field [30]. However, we will most likely not be in this ideal situation. Therefore we look for a function which has m as gradient in the least-squares sense, i.e.,

$$u = \operatorname*{argmin}_{\phi} I(\phi), \quad I(\phi) = \frac{1}{2}\int_X |\nabla\phi - m|^2. \tag{4.1}$$

4.1. Fully input convex neural networks

According to the previously described Breniers' theorem, the optimal transport m is the gradient of a convex function u, which is precisely what we seek. Consequently a fully input convex neural network (FICNN) [1] which explicitly constraints the function approximated by the network to be convex naturally lends itself to our proposed requirement. Generally we consider a k-layer FICNN, shown in Fig. 1.

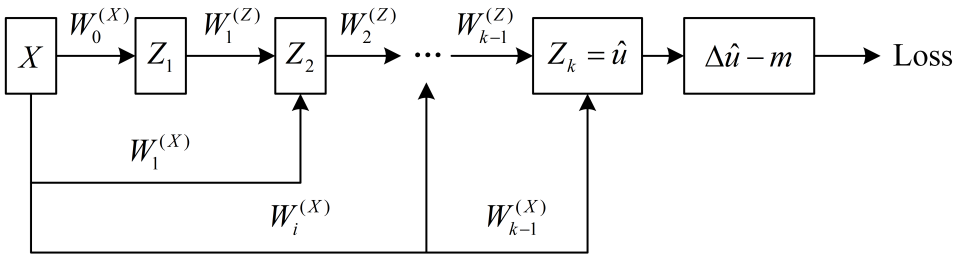


Figure 1: A fully input convex neural network for Monge-Ampère equation.

This model defines a neural network over the input x. The update rule and final output are

$$\begin{aligned} z_{i+1} &= g_i\left(W_i^{(z)} z_i + W_i^{(x)} x + b_i\right), \quad i = 0, \ldots, k-1, \\ \hat{u}(x;\theta) &= z_k. \end{aligned}$$

Here, in the FICNN, the activation function g is a non-decreasing convex function and the weights $W_i^{(z)}$ are constrained to be non-negative. There is no constraints for weights $W_i^{(x)}$ (with $z_0, W_0^{(z)} \equiv 0$). Here, θ refers to all the parameters $\{W_{0:k-1}^{(y)}, W_{1:k-1}^{(z)}, b_{0:k-1}\}$. The following are the facts for deriving the convexity of the network:

1. A linear combination of convex functions with positive coefficients is convex.

2. If g is a convex function, f is a non-decreasing convex function and $R(g) \subset D(f)$, the composition $f \circ g$ will be convex.

Now there is a lack of theoretical proofs to show whether any convex function can be approximated (to arbitrary accuracy) by FICNN, but numerical experiments have shown good results in using FICNN to solve the Dirichlet problem for the Monge-Ampère equation [32], to make structured predictions [1], and so on. Therefore we can construct Net_4(FICNN) to solve u. From (4.1), the loss function used in this case is

$$E_4 := \frac{1}{M_a + M_b} \sum_{i=1}^{M_a+M_b} \left(|\nabla u - m|^2 \right).$$

Noting that problem (4.1), which obviously has a non-unique solution due to the fact that it contains only the derivative of u, and is accurate to an additive constant. We add the constraint

$$u(x_i) = 0,$$

for some arbitrarily chosen i to make the solution unique. In this case $u(x)$ is expressed as

$$u(x) = \mathrm{Net}_4(x) + u(x_i) - \mathrm{Net}_4(x_i).$$

4.2. Algorithm summary

The numerical algorithm is summarized as follows. We discretize the source domain X and the boundary ∂X. The initial guess m^0 is given by (2.2). Subsequently, we construct three neural networks to repeatedly perform the steps given by (2.3a)-(2.3c). The first step is a minimization for b and is to update the parameters of Net_1 by minimization loss function E_1 about all the boundary points. The second step is a minimization procedure for P and is to update the parameters of Net_2 by minimization loss function E_2 about all the interior points. The third and last step is a minimization procedure for m and is to update the parameters of Net_3 by minimization loss function E_3. The three steps are repeated until the functional $J(m, b, P)$ is no longer decreasing. If needed, the convex function u is computed from m by updating the parameters of Net_4 (FICNN).

5. Numerical Examples

In this section, we apply the algorithm introduced before to six concrete examples: in the first test problem, we map a square with uniform density into a circle with uniform density; in the second, a circle to a square; in the third, a square to target-distributions on non-convex domains; in the fourth, we challenge our algorithm to design a special lens mapping a uniform, square parallel beam of light to a projection of a famous Netherlands painting on a wall; in the fifth, we use our algorithm to move the grid adaptively under a given monitoring function M; in the sixth, we map a cube of uniform density into a ball of uniform density as an illustration of the potential of the algorithm to expand to higher dimensions.

In our numerical calculations, all derivatives (our algorithm contains only first-order derivatives) and the gradient of the loss functions with respect to the parameters (weights and biases) has been calculated using automatic differentiation in Pytorch [34]. We train these networks, i.e. minimize the loss functions, using the Adam algorithm [23] from Pytorch.

The number of outer iterations of the algorithm is denoted as T. The number of inner iterations for updating the parameters of Net_i is denoted as t_i, $i = 1, 2, 3, 4$. For a fixed T, Adam needs an initial learning rate lr_0. Inspired by the Learning Rate Annealing [43] (Noam Decay [40], Cosine Annealing Decay [28]), we can set the lr_0 as a decay function with respect to T, for example

$$lr_0(T) = lr_0(T-1) * d_{out}^{-0.5} * \min\Big(T^{-0.5}, T * T_{warmup}^{-1.5}\Big), \tag{5.1}$$

where d_{out} is the output dimension of the neural networks, T_{warmup} is the number of pre-heating steps which is equal to 10 in this article, and $lr_0(T)$ is the initial learning rate of the neural networks at T outer iterations.

The activation functions are all chosen as tanh. Initial values for weights are chosen by Xavier uniform [16] and initial values for biases are zero. In the case of FICNN, some parameters w need to remain nonnegative. This issue has been resolved by expressing these parameters in the form $w = v^2$ and we train with respect to v rather than w.

The scattering methods in this paper all use the Quasi-Monte Carlo method based on Sobol sequences [22] from the SciPy library [42]. In contrast to common pseudo-random numbers, it is a Low Discrepancy Sequence [24] and more uniformly distributed in high-dimensional space.

The value of α taken during training affects the performance of the algorithm. It is not good to compare J in the case of different α. Therefore, to unify the evaluation criteria with different α, J of all the graphs in the following examples are denoted as:

$$J(m, P, b) = \frac{1}{2}\big(J_b(m, b) + J_I(m, P)\big).$$

5.1. From a square to a circle

In our first test problem, the source domain is given by $X = [-1, 1]^2$, and the target domain by $Y = \{(p, q) \in \mathbb{R}^2 \mid p^2 + q^2 \le 1\}$. We have $f(x, y) = 1/4$ and $g(p, q) = 1/\pi$.

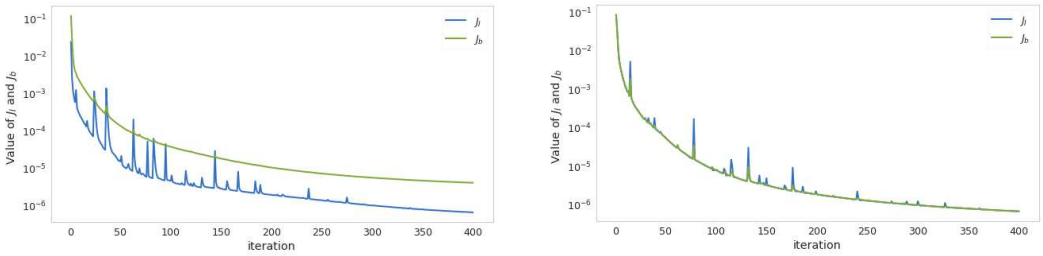
In our numerical illustrations we use 4096 randomly distributed collocation points inside the domain, and 512 uniformly distributed points on the boundary. We construct Net_1 with 6 hidden layer and 50 hidden nodes, Net_2 with 8 hidden layer and 50 hidden nodes and Net_3 with 12 hidden layer and 50 hidden nodes. Because the boundaries of the target domain have smooth analytic expressions, the E_1 function is taken as (3.2), and we take the factor $\Lambda_1 = 10^2$ in front of the term which penalizes the degree of deviation from the boundary.

We calculate the mapping on the collocation points using $T = 400$ outer iterations in (2.3). In each outer iteration, the neural networks corresponding to each update step

(2.3a)-(2.3c) will be trained with Adam for $t_i = 400$ inner iterations. The initial learning rate of each neural network at each outer iteration is given by (5.1), where $lr_0(0) = 10^{-3}$. When Net_3 is training with minimax architecture (3.26), at each outer iteration the initial learning rates ρ_2^1, ρ_1^1 for $\boldsymbol{w} = (w_1, w_2)$ are set to 10^{-3}.

Subsequently we test the algorithm with different α. The influence of α on the convergence of the algorithm is shown in Fig. 3. It is obvious that the choice of α strongly influences the performance of the algorithm. If α is a fixed constant value, experiments are the only method to determine a suitable value for α. The value of α between 0.1 and 0.5 results in good performance. Fortunately, one of the best choice of α can be adaptively determined by the minmax structure (3.26). From Fig. 2(a), the slower training at $\alpha = 0.5$ may be due to the issue of gradient pathologies arising from imbalanced loss terms [43]. And the adaptive modification of α makes it simple to train the two components J_I and J_b by balancing them.

For α adaptive case, we selected 101×101 grids as a test set. After 400 outer iterations, the value of J, J_I, J_b are $1.31 \cdot 10^{-6}$, $1.42 \cdot 10^{-6}$, $1.19 \cdot 10^{-6}$ respectively. The distribution of the test set after mapping is shown in Fig. 4(a). The error plot on the test set with respect to J is shown in Fig. 4(b).



(a) $\alpha = 0.5$ (b) α adaptive

Figure 2: Values of J_I and J_B for the square to a circle problem as function of the iteration number.

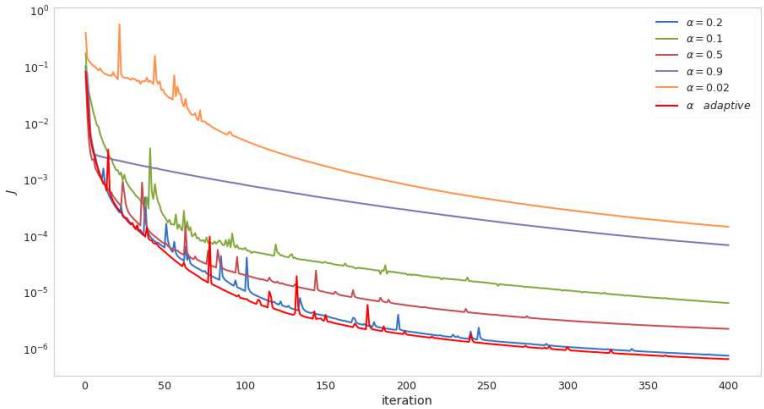
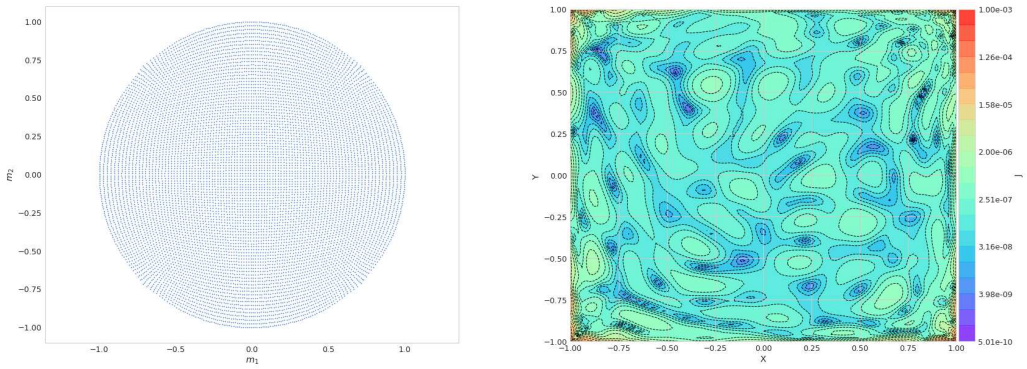


Figure 3: Value of J as a function of the number of iterations for mapping a square to a circle with various values of α.



(a) The mapping for the square to a circle problem after 400 iterations on the test set, and α adaptive

(b) Value of J for mapping a square to a circle on the test set, α adaptive

Figure 4: The distribution of the final mapping for α adaptive and the error map about J.

5.2. From a circle to a square

In the second test instance, a circle is mapped to a square. We have

$$f(x,y)=\begin{cases}\dfrac{1}{\pi}, & \text{if} \quad x^2+y^2\le 1,\\ 0, & \text{otherwise.}\end{cases}$$

And we have $g(p,q)=1/4$ for target domain $[-1,1]^2\in\mathbb{R}^2$.

In our numerical illustrations we use 4000 randomly distributed collocation points inside the domain, and 500 uniformly distributed points on the boundary.

The structures of Net_1, Net_2, Net_3 are the same as Section 5.1. Because the boundary of the target domain is a square, the E_1 function is taken as (3.3). In addition to the $lr_0(0)=10^{-4}$, other initial parameters such as the number of outer and inner iterations, and the learning rate decay strategy are all the same as Section 5.1.

We also compared the performance of the algorithm for different choices of α. The results are shown in Fig. 5. When α is between 0.05 and 0.2, the algorithm have better performance. When the α adaptive strategy is chosen, the algorithm is very close to the best, although not the best performance. In the absence of a priori experience, it is usually a good approach to adopt the α adaptive strategy.

We selected 101×101 grids as a test set. After mapping, the distribution of the different values of α in the test set is shown in Fig. 6. Corners frequently presented difficulties with mapping convergence. Selecting the proper value of α lessens the severity of this issue.

5.3. A non-convex target

In the third test case, the source domain is a uniform square source distribution on the square $[-1,1]^2$ with $f(x,y)=1/4$. The target domain non-convex with uniform distribution, and the target boundary is defined by

$$\rho(\theta)=1+C\cos(3\theta), \tag{5.2}$$

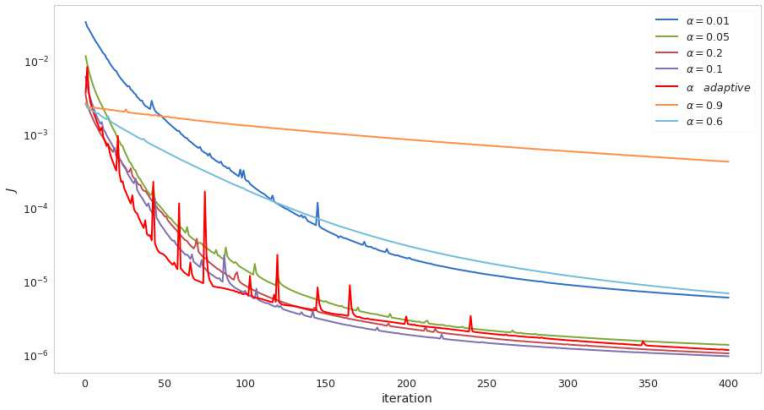
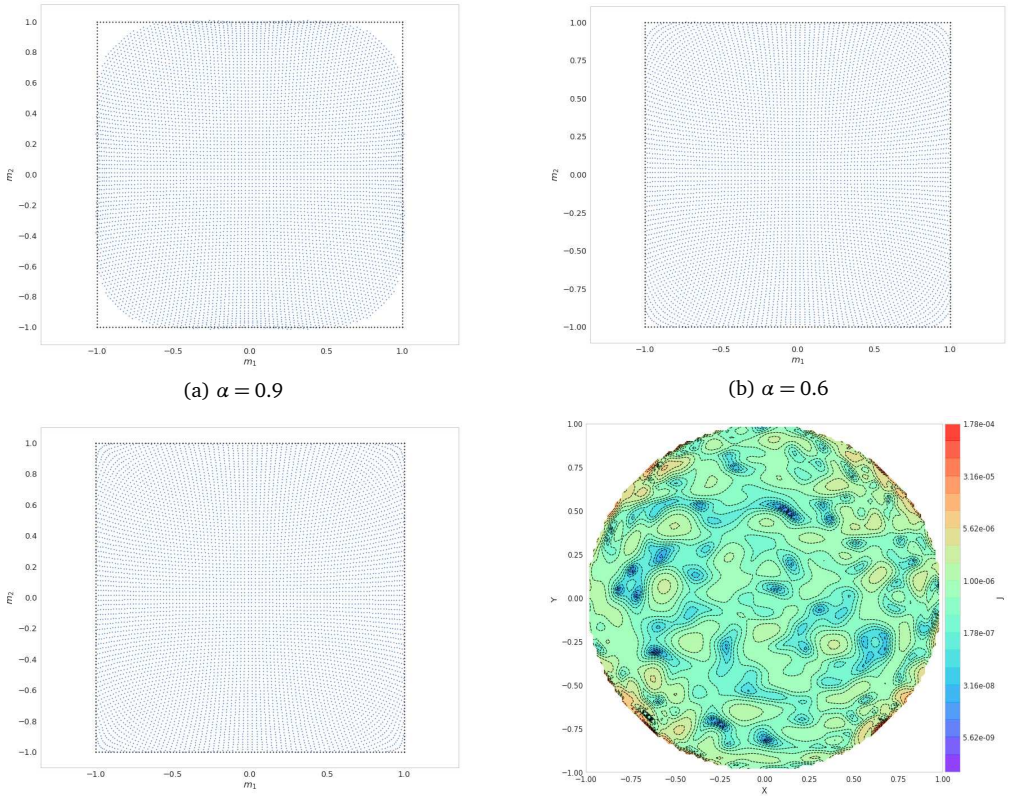


Figure 5: Value of J as a function of the number of iterations for mapping a circle to a square with various values of α.



(a) $\alpha = 0.9$

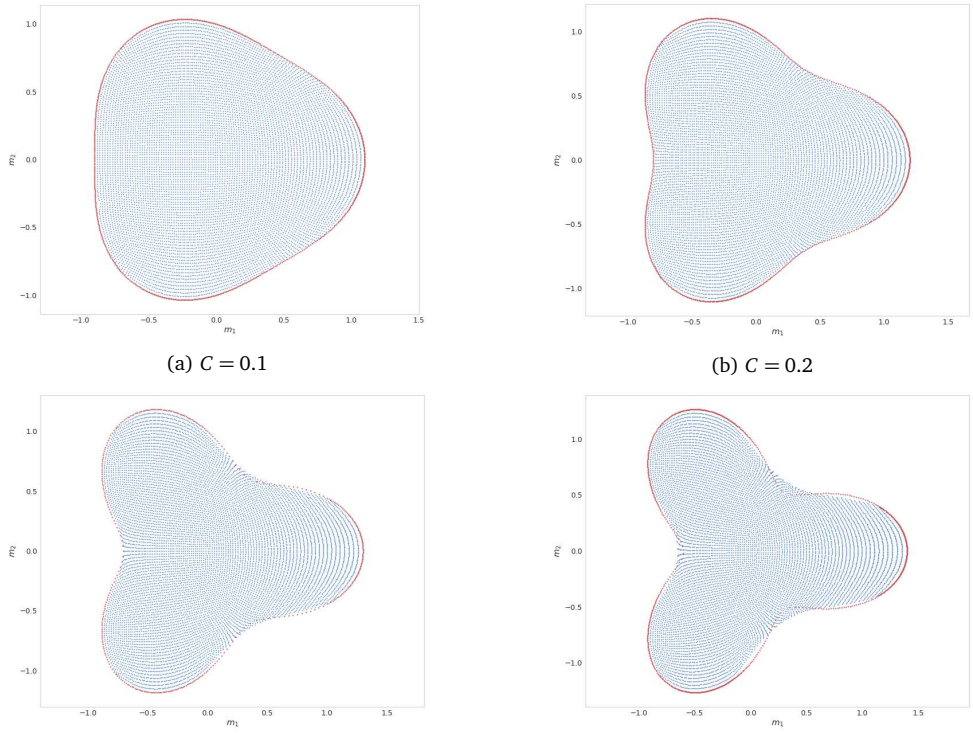
(b) $\alpha = 0.6$

(c) $\alpha = 0.1$

(d) $\alpha = 0.1$, value of J for final mapping on the test set

Figure 6: Final mapping after 400 iterations on a 101×101 grid for the second test case.

where the ρ coordinate represents the distance from the pole, and the θ coordinate represents the polar axis. The larger the C, the stronger the convexity. We test the algorithm for $C \in \{0.1, 0.2, 0.3, 0.4\}$.



(a) $C = 0.1$ (b) $C = 0.2$

(c) $C = 0.3$ (d) $C = 0.4$

Figure 7: Mapping after 200 iterations for target domains with boundary defined by (5.2).

In our numerical illustrations we use 10000 randomly distributed collocation points inside the domain, and 1000 uniformly distributed points on the boundary.

We construct Net_1 with 8 hidden layer and 50 hidden nodes, Net_2 with 10 hidden layer and 50 hidden nodes and Net_3 with 12 hidden layer and 50 hidden nodes. The other parameters settings are the same as in the second test case except that the t_1 (Net_1 iteration period) is increased appropriately to equal 600 or 800.

The value of J on the test set after 200 iterations are $3.28 \cdot 10^{-6}$, $3.71 \cdot 10^{-6}$, $1.47 \cdot 10^{-5}$, $6.32 \cdot 10^{-5}$ as shown in Fig. 8. It is shown in Fig. 7 that convergence issues exist for target domains that greatly deviate from a convex shape, but when the shape deviates just somewhat the technique performs satisfactorily.

5.4. Design of lens

Considering the problem of designing a free-form lens or free-form mirror that, when illuminated by a parallel beam of light with an arbitrary light distribution, produces a given illumination pattern on the target. Both problems can be modeled by strongly nonlinear second-order partial differential equations of Monge-Ampère type. Such optical systems have a wide range of technological applications, such as spotlights with predetermined lighting patterns used in street lamps or automotive headlights [2, 37]. In this article, we

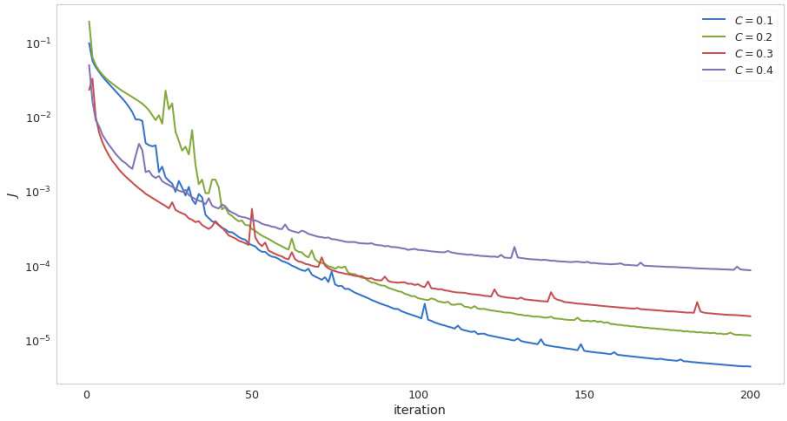


Figure 8: Value of J as function of the iteration number with different values of C for the third test case.

will push our algorithm to its limits by designing a lens that transforms a square uniform parallel beam of light into a target distribution comparable to a well-known Dutch artwork.

In this case, we have a parallel beam of light in the z-direction with radiant exitance $M(x,y)[\mathrm{W/m^2}]$ given by

$$M(x,y) = \begin{cases} 1, & \text{if} \quad (x,y) \in [-1,1]^2, \\ 0, & \text{otherwise}, \end{cases}$$

and an irradiance on the plane $\mathscr{P}$ given by $L(x,y)[\mathrm{W/m^2}]$. Because the source density is uniformly distributed, it is easy to derive that source density $f(x,y) = M(x,y)/4$. The lens that redistributes the parallel beam in the target region such that the irradiance of the target region is $L(x,y)[\mathrm{W/m^2}]$ can be expressed by $u(x,y)$ which is the convex solution of the Monge-Ampère equation (MA) with boundary condition (BV2). To get the target density $g(p,q)$ (shown in Fig. 9(a)) of the painting "Girl with a Pearl Earring" by the Dutch painter Johannes Vermeer, we convert it into grayscale values. The range of grayscale values is 0 to 255. In order to avoid $g(p,q)$ for some (p,q) which gives division by 0 in the Monge-Ampère equation, we increase the minimum illuminance of the picture to be 5% of the maximum illuminance. Images are made up of blocks of pixels, and the conversion of pixels to grayscale values means that the image can be viewed as a piecewise function to find its density distribution $g(p,q)$. The width of the image is 53.2 and the height is 63, so we put the target field in $[-0.532, 0.532] \times [-0.63, 0.63]$.

Because the picture details are many, in our numerical illustrations we use 40000 randomly distributed collocation points inside the domain, and 2000 uniformly distributed points on the boundary.

We construct Net_1 with 8 hidden layer and 80 hidden nodes, Net_2 with 12 hidden layer and 80 hidden nodes and Net_3 with 16 hidden layer and 80 hidden nodes. Except $t_i = 600$ inner iterations, other initial parameters are all the same as Section 5.2. The graphs in this subsection are all for $\alpha = 0.2$.

(a) The target distribution. Note that the target domain is not convex

(b) The mapping after the algorithm has converged

Figure 9: The target distribution and mapping for the lens.

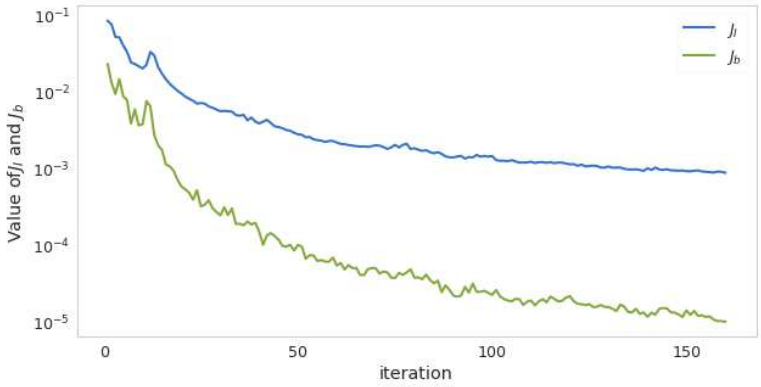


Figure 10: Values of J_I and J_B for the painting test case as function of the iteration number.

We stop the algorithm after 160 outer iterations, because J_I and J_b no longer appear to decline considerably. The values of J_I and J_b as function of the iteration number are plotted in Fig. 10. We select 401×401 grids as a test set. The resulting mapping on the test set is shown in Fig. 9(b). The higher the gray value of the area, the lighter the color and the greater the density of the corresponding test points.

In this problem we are more interested in $u(x, y)$. So we construct Net_4 with 8 hidden layer and 100 hidden nodes to compute $u(x, y)$. We first iterate 15000 times with the Adam optimizer with an initial learning rate of $8 \cdot 10^{-4}$, and then 3800 times with LBFGS optimizer with a learning rate of 1 and history size $= 20$. All parameter choices for the LBFGS are referenced in [31]. The function $u(x, y)$ representing the lens surface on the same test set is shown in Fig. 11.

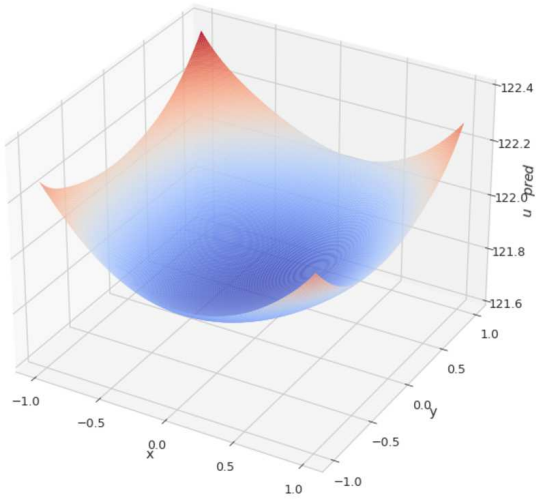


Figure 11: The function $u(x,y)$ representing the lens-surface.

5.5. Moving mesh generation

Our another interest in the Monge-Ampère equation comes from the field of a moving mesh which is suitable for the numerical solution of partial differential equations. One of the three approaches to mesh adaptivity [38], the r-adaptivity, aims to get the maximum possible accuracy for a given problem and number of mesh points (with fixed connectivity). The core of r-adaptivity is to find a mapping m from the computational domain X to the physical domain Y so that generated mesh in the physical domain can be close to one which equidistributes a suitable monitor M of the properties of the underlying solution, computational errors, and/or of the regularity of the grid.

But the mapping m which equidistributes the respective monitor function M is not unique. In [6], by adding constraints where the map m should minimize the distance that the uniformly distributed mesh points in the X have to move to the associated points in the Y, which not only uniquely determine the mapping m, but m is the optimal transport satisfying

$$\det(Dm) = \frac{\int_Y M(m(\xi,t),t)d\eta}{M(m(\xi,t),t)\int_X d\xi},$$
$$m(\partial X) = \partial Y,$$

where ξ and η are computational coordinate and physical coordinate.

In the fifth test case, the source domain is a uniform square source distribution on the square $[0,1]^2$ with $f(x,y)=1$ ($f = 1/\int_X d\xi$), the target domain is also the square $[0,1]^2$ with

$$g = \frac{M(m(\xi,t),t)}{\int_Y M(m(\xi,t),t)d\eta}.$$

The monitor function is given by

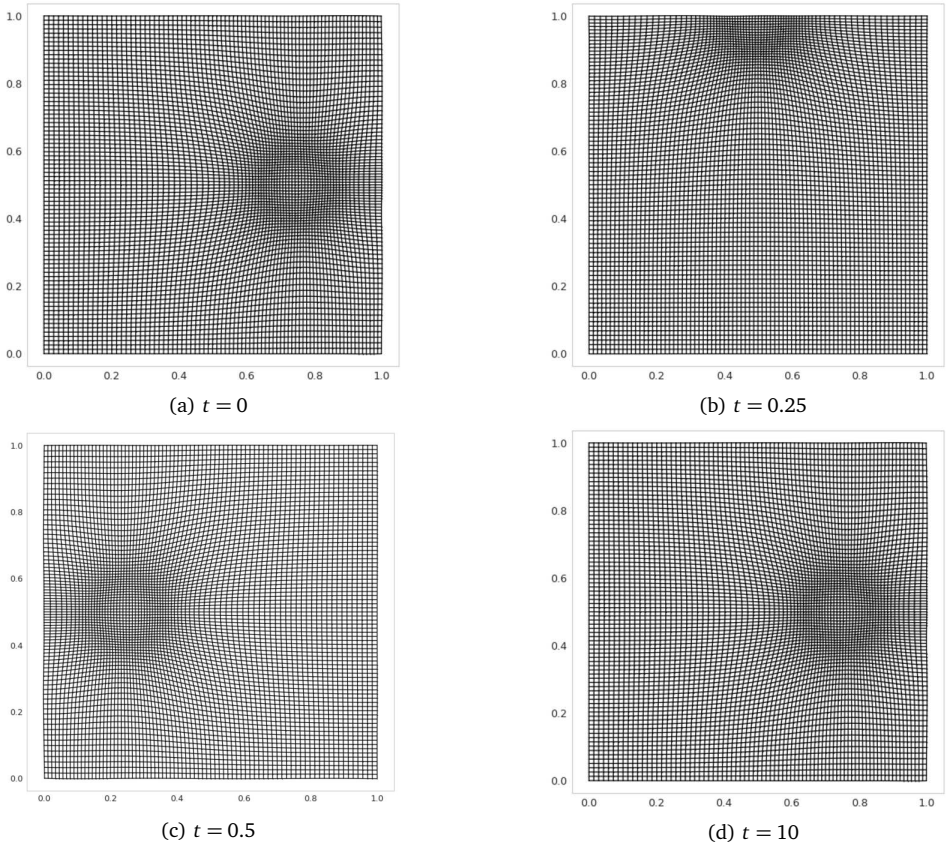
$$M_1(x,y,t) = 1 + 5\exp\left(-50\left|\left(x-\frac{1}{2}-\frac{\cos(2\pi t)}{4}\right)^2+\left(y-\frac{1}{2}-\frac{\sin(2\pi t)}{2}\right)^2-0.01\right|\right).$$

This is stated in [8] as a harsh test of a moving mesh approach, and the meshes produced using the (velocity-based) geometric conservation law (GCL) method [8] show a significant degree of skewness.

In our numerical illustrations, we use 80×80 grids as test set and training set, because we only care about the movement of the grid points.

We construct Net_1 with 8 hidden layer and 80 hidden nodes, Net_2 with 12 hidden layer and 80 hidden nodes and Net_3 with 15 hidden layer and 80 hidden nodes. The other parameters settings are the same as in the first test case except $T = 100$, because an accurate solution of the underlying PDE may not necessarily require an exact solution of the mesh equation.

From Fig. 12, it inherits the advantages of the original algorithm [6]. The mesh generated has excellent regularity and has no evidence of skewness. And the generated grid



(a) $t = 0$ (b) $t = 0.25$

(c) $t = 0.5$ (d) $t = 10$

Figure 12: The fifth test example, the distribution of 80×80 grid points in Y.

moves closely with the movement of the circle

$$M_1(x,y,t) = 1 + 5\exp\left(-50\left|\left(x - \frac{1}{2} - \frac{\cos(2\pi t)}{4}\right)^2 + \left(y - \frac{1}{2} - \frac{\sin(2\pi t)}{2}\right)^2 - 0.01\right|\right).$$

In addition, this algorithm also gives the possibility to generate mesh in higher dimensional spaces.

5.6. From a cube to a ball

In the sixth test instance, a cube is mapped to a unit ball. We have

$$g(p,q,s) = \begin{cases} \dfrac{3}{4\pi}, & \text{for} \quad p^2+q^2+s^2 \le 1, \\ 0, & \text{otherwise.} \end{cases}$$

And we have $f(x,y,z) = 1/8$ for target domain $[-1,1]^3 \in \mathbb{R}^3$.

In our numerical illustrations we use 2^{15} randomly distributed collocation points inside the domain, and 3072 uniformly distributed points on the boundary.

We construct Net_1 with 10 hidden layer and 80 hidden nodes, Net_2 with 15 hidden layer and 80 hidden nodes and Net_3 with 18 hidden layer and 80 hidden nodes. Other parameters are the same as Section 5.1 except that all the $t_i = 1000$. The E_1 function is taken as (3.2), and we take the factor $\Lambda_1 = 10^2$.

After 400 outer inter of training, we stop training. We select $40 \times 40 \times 40$ grids as a test set. On the test set the value of J, J_I, J_B are $4.11 \cdot 10^{-4}$, $6.90 \cdot 10^{-4}$, $1.32 \cdot 10^{-4}$. The resulting mapping on the test set is shown in Fig. 14. The values of J_I and J_b as function of the iteration number are plotted in Fig. 13.

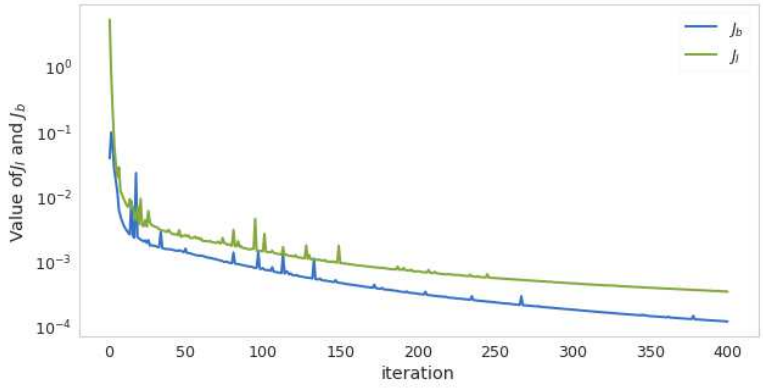


Figure 13: Values of J_I and J_B for the from a cube to a ball test case as function of the iteration number.

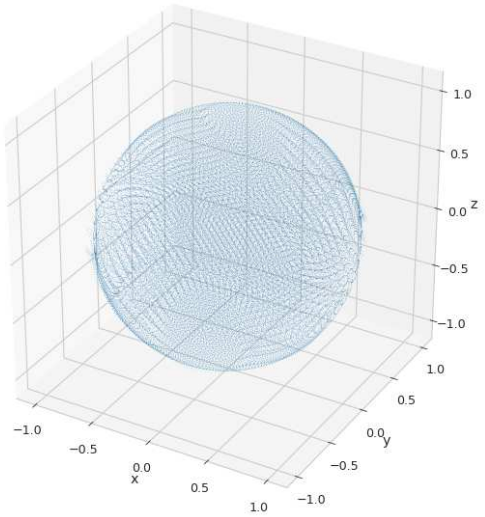


Figure 14: The final mapping for the cube to unit ball test case.

6. Conclusions

We design a new algorithm based on machine learning to solve the solution of optimal mass transport and the corresponding convex solution of the Monge-Ampère equation with the transport boundary conditions. The method builds on the work of Prins *et al.* [36], Glowinski *et al.* [7]. We use a relaxation approach to construct three neural networks to minimize b, P, m. It not only reduces the size of the problem but also makes each subproblem easy to solve. In particular, when the problem is a two-dimensional case, the pointwise minimization of P can be solved analytically so that we can construct the modified loss function of Net_2 (3.24) which can make Net_2 easy to train.

Unlike most algorithms that directly calculate the solution of the Monge-Ampère equation, we first find the optimal transport which is the gradient of the solution, and then compute the solution by input convex neural network so that the solution can maintain convexity. Selection of α for Net_3 affects the performance of the algorithm. We make it easy to select α based on the two strategies we have proposed rather than experimenting so that the algorithm often performs well.

There are extremely few numerical methods for the Monge-Ampère equation with transport boundary conditions, cf. [3,4,13,19]. Our algorithm has the ability to handle optimal transport problems where the target domain is non-convex. It is also able to solve high dimensional problems.

Acknowledgments

We are very grateful to the anonymous referees for their valuable suggestions which helped to improve the paper.

This paper is supported by the Natural Science Foundation of Fujian Province of China (No. 2021J01034), by the National Key R&D Program of China (No. 2022YFA1004500), by the National Natural Science Foundation of China (No. 12371372) and by the Fundamental Research Funds for the Central Universities (No. 20720220038).

References

[1] B. Amos, L. Xu and J.Z. Kolter, *Input convex neural networks*, in: *International Conference on Machine Learning*, pp. 146–155, PMLR (2017).

[2] A. Bäuerle, A. Bruneton, R. Wester, J. Stollenwerk and P. Loosen, *Algorithm for irradiance tailoring using multiple freeform optical surfaces*, Opt. Express **20(13)**, 14477–14485 (2012).

[3] J.D. Benamou, B.D. Froese and A.M. Oberman, *Numerical solution of the optimal transportation problem using the Monge-Ampère equation*, J. Comput. Phys. **260**, 107–126 (2014).

[4] J.D. Benamou, A. Oberman and F. Britanny, *Numerical solution of the second boundary value problem for the elliptic Monge-Ampère equation*, Ph.D. Thesis, INRIA (2012).

[5] Y. Brenier, *Polar factorization and monotone rearrangement of vector-valued functions*, Commun. Pure Appl. Math. **44(4)**, 375–417 (1991).

[6] C.J. Budd and J. Williams, *Moving mesh generation using the parabolic Monge-Ampère equation*, SIAM J. Sci. Comput. **31(5)**, 3438–3465 (2009).

[7] A. Caboussat, R. Glowinski and D.C. Sorensen, *A least-squares method for the numerical solution of the Dirichlet problem for the elliptic Monge-Ampère equation in dimension two*, ESAIM Control Optim. Calc. Var. **19(3)**, 780–810 (2013).

[8] W. Cao, W. Huang and R.D. Russell, *A moving mesh method based on the geometric conservation law*, SIAM J. Sci. Comput. **24(1)**, 118–142 (2002).

[9] Z. Chang, K. Li, X. Zou and X. Xiang, *High order deep neural network for solving high frequency partial differential equations*, Commun. Comput. Phys. **31**, 370–397 (2022).

[10] T. Chen and H. Chen, *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*, IEEE Trans. Neural Netw. **6(4)**, 911–917 (1995).

[11] M. Cuturi, *Lightspeed computation of optimal transport*, Adv. Neural Inf. Process. Syst. **26**, 2292–2300 (2013).

[12] L.C. Evans, *Partial differential equations and Monge-Kantorovich mass transfer*, Curr. Dev. Math. **1997(1)**, 65–126 (1997).

[13] B.D. Froese, *A numerical method for the elliptic Monge-Ampère equation with transport boundary conditions*, SIAM J. Sci. Comput. **34(3)**, A1432–A1459 (2012).

[14] A. Fuller, *Root location criteria for quartic equations*, IEEE Trans. Autom. Control **26(3)**, 777–782 (1981).

[15] W. Gangbo and A. Święch, *Optimal transport and large number of particles*, Discrete Contin. Dyn. Syst. **34(4)**, 1397 (2014).

[16] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings (2010).

[17] R. Glowinski, *An augmented Lagrangian approach to the numerical solution of the Dirichlet problem for the elliptic Monge-Ampère equation in two dimensions*, Electron. Trans. Numer. Anal. **22**, 71–96 (2006).

[18] H. Guo and X. Yang, *Deep unfitted Nitsche method for elliptic interface problems*, Commun. Comput. Phys. **31(4)**, 1162–1179 (2022).

[19] B.F. Hamfeldt, *Convergence framework for the second boundary value problem for the Monge-Ampère equation*, SIAM J. Numer. Anal. **57(2)**, 945–971 (2019).
[20] J. Huang, C. Wang and H. Wang, *A deep learning method for elliptic hemivariational inequalities*, East Asian J. Appl. Math. **12(3)**, 487–502 (2022).
[21] J. Huang, T. Zhou and H. Wang, *An augmented Lagrangian deep learning method for variational problems with essential boundary conditions*, Commun. Comput. Phys. **31(3)**, 966–986 (2022).
[22] S. Joe and F.Y. Kuo, *Constructing Sobol sequences with better two-dimensional projections*, SIAM J. Sci. Comput. **30(5)**, 2635–2654 (2008).
[23] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv:1412.6980 (2014).
[24] L. Kocis and W.J. Whiten, *Computational investigations of low-discrepancy sequences*, ACM Trans. Math. Softw. (TOMS) **23(2)**, 266–294 (1997).
[25] B. Lévy, *A numerical algorithm for L_2 semi-discrete optimal transport in 3D*, ESAIM: Math. Model. Numer. Anal. **49(6)**, 1693–1715 (2015).
[26] D. Liu and Y. Wang, *A dual-dimer method for training physics-constrained neural networks with minimax architecture*, Neural Netw. **136**, 112–125 (2021).
[27] L. Liu, B. Wang and W. Cai, *Linearized learning with multiscale deep neural networks for stationary Navier-Stokes equations with oscillatory solutions*, East Asian J. Appl. Math. **13(3)**, 740–758 (2023).
[28] I. Loshchilov and F. Hutter, *SGDR: Stochastic gradient descent with warm restarts*, arXiv:1608.03983 (2016).
[29] H. Ma, Y. Zhang, N. Thuerey, X. Hu and O.J. Haidn, *Physics-driven learning of the steady Navier-Stokes equations using deep convolutional neural networks*, Commun. Comput. Phys. **32(3)**, 715–736 (2022).
[30] J. Marsden and A. Tromba, *Vector Calculus*, W.H.Freeman and Company (1996).
[31] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer (1999).
[32] K. Nyström and M. Vestberg, *Solving the Dirichlet problem for the Monge-Ampère equation using neural networks*, J. Comput. Math. Data Sci. **8**, 100080 (2023).
[33] V.I. Oliker and L.D. Prussner, *On the numerical solution of the equation and its discretizations, I*, Numer. Math. **54(3)**, 271–293 (1989).
[34] A. Paszke et al., *PyTorch: An imperative style, high-performance deep learning library*, Adv. Neural Inf. Process. Syst. **32** (2019).
[35] A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta Numerica **8**, 143–195 (1999).
[36] C. Prins, R. Beltman, J. ten Thije Boonkkamp, W.L. IJzerman and T.W. Tukker, *A least-squares method for optimal transport using the Monge-Ampère equation*, SIAM J. Sci. Comput. **37(6)**, B937–B961 (2015).
[37] S. Seroka and S. Sertl, *Modeling of refractive freeform surfaces by a nonlinear PDE for the generation of a given target light distribution*, in: *International Light Simulation Symposium, Vol. 2012*, pp. 143–156, Steinbeis Transfer Center Applied Lighting Technology (2012).
[38] T. Tang, *Moving mesh methods for computational fluid dynamics*, Contemp. Math. **383(8)**, 141–173 (2005).
[39] J.P. Tignol, *Galois' Theory of Algebraic Equations*, World Scientific Publishing Company (2015).
[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser and I. Polosukhin, *Attention is All You Need*, in: *Advances in Neural Information Processing Systems, Vol. 30* (2017).
[41] C. Villani, *Topics in Optimal Transportation*, Graduate Studies in Mathematics, Vol. 58, AMS (2003).
[42] P. Virtanen et al., *Scipy 1.0: Fundamental algorithms for scientific computing in Python*, Nat.

Methods **17(3)**, 261–272 (2020).
[43] S. Wang, Y. Teng and P. Perdikaris, *Understanding and mitigating gradient flow pathologies in physics-informed neural networks*, SIAM J. Sci. Comput. **43(5)**, A3055–A3081 (2021).
[44] Y. Xu and T. Zeng, *Sparse deep neural network for nonlinear partial differential equations*, Numer. Math. Theory Methods Appl. **16(1)**, 58–78 (2023).
[45] Z.Q.J. Xu, *Frequency principle: Fourier analysis sheds light on deep neural networks*, Commun. Comput. Phys. **28(5)**, 1746–1767 (2020).