StableMDS: A Novel Gradient Descent-Based Method for Stabilizing and Accelerating Weighted Multidimensional Scaling

Zhongxi Fang[1] **Xun Su**[1] **Tomohisa Tabuchi**[1] **Jianming Huang**[1] **Hiroyuki Kasai**[1,2]
[1]Department of Computer Science and Communications Engineering, FSE Graduate School, Waseda University
[2]Department of Communications and Computer Engineering, FSE School, Waseda University

Abstract

Multidimensional Scaling (MDS) is an essential technique in multivariate analysis, with Weighted MDS (WMDS) commonly employed for tasks such as dimensionality reduction and graph drawing. However, the optimization of WMDS poses significant challenges due to the highly non-convex nature of its objective function. Stress Majorization, a method classified under the Majorization-Minimization algorithm, is among the most widely used solvers for this problem because it guarantees non-increasing loss values during optimization, even with a non-convex objective function. Despite this advantage, Stress Majorization suffers from high computational complexity, specifically $\mathcal{O}(\max(n^3, n^2p))$ per iteration, where n denotes the number of data points, and p represents the projection dimension, rendering it impractical for large-scale data analysis. To mitigate the computational challenge, we introduce *StableMDS*, a novel gradient descent-based method that reduces the computational complexity to $\mathcal{O}(n^2p)$ per iteration. StableMDS achieves this computational efficiency by applying gradient descent independently to each point, thereby eliminating the need for costly matrix operations inherent in Stress Majorization. Furthermore, we theoretically ensure non-increasing loss values and optimization stability akin to Stress Majorization. These advancements not only enhance computational efficiency but also maintain stability, thereby broadening the applicability of WMDS to larger datasets.

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

1 Introduction

Understanding relationships based on similarity, distance, or rank order among data points is important in many disciplines. Multidimensional Scaling (MDS) is a method used to generate embeddings in a coordinate space while preserving the relationships among data points. Unlike traditional dimensionality reduction techniques, which typically aim to project high-dimensional data into lower-dimensional spaces, MDS retains the integrity of the given relational information by solving an optimization problem, regardless of the dimensionality of the input or output space. This method has practical applications in fields such as psychology and social sciences (Kruskal and Wish, 1978), bioinformatics (Balanzá-Martínez et al., 2021), and network analysis, including tasks like sensor network localization (Saeed et al., 2019).

MDS can be categorized into several variants depending on the relationships it seeks to preserve. Classical MDS (CMDS), also referred to as Principal Coordinate Analysis (Sonthalia et al., 2021), operates on similarity measures to compute embeddings that maintain likenesses. In contrast, Metric MDS employs distance information between data points to generate embeddings, aiming to preserve these distances (Samet, 2006), with its cost function known as the stress function (Mardia, 1978). Non-metric MDS focuses on preserving the rank order of dissimilarities, ensuring that the rank order of points is maintained (Agarwal et al., 2007). These variants allow MDS to address a wide range of scenarios, from preserving similarities or distances to maintaining rank order, such as those in ranking problems.

Within the category of Metric MDS, several formulations address more complex scenarios. One of the most versatile approaches is Weighted MDS (WMDS), which incorporates weights into the formulation to adjust the influence of different point pairs on the embedding. This flexibility makes WMDS particularly suitable for applications where the significance of cer-

tain data points or relationships outweighs others.

WMDS takes input from a distance matrix and a weighting matrix. The distance matrix is defined as $\mathbf{D} \coloneqq [d_{ij}]_{n\times n} \in \mathbb{R}^{n\times n}$, where d_{ij} represents the distance between each pair of data points i and j. The weighting matrix is defined as $\mathbf{W} \coloneqq [w_{ij}]_{n\times n} \in \mathbb{R}^{n\times n}$, where w_{ij} indicates the significance assigned to each pair of data points. WMDS aims to optimally position the data points in a p-dimensional space so that both the distances between data points and their corresponding significance are accurately represented. Letting $\mathbf{y}_i \in \mathbb{R}^p$ denote the coordinates of data point i, the task is to determine the matrix $\mathbf{Y} \coloneqq [\mathbf{y}_1, \ldots, \mathbf{y}_n]^\top \in \mathbb{R}^{n\times p}$ comprising these coordinates. This objective is achieved by solving an optimization problem defined by the following stress function $S(\mathbf{Y}) : \mathbb{R}^{n\times p} \to \mathbb{R}$:

$$\min_{\mathbf{Y}} \left\{ S(\mathbf{Y}) \coloneqq \sum_{i<j} w_{ij} \left(\|\mathbf{y}_i - \mathbf{y}_j\| - d_{ij} \right)^2 \right\}.$$

Since the value of the stress function is typically large, it can be normalized using input distances as follows: $S_n(\mathbf{Y}) \coloneqq \sqrt{\frac{S(\mathbf{Y})}{\sum_{i<j} d_{ij}^2}}$, where the subscript n denotes the normalized version of the stress function.

Depending on the configurations of $\mathbf{D}$ and $\mathbf{W}$, WMDS can be transformed into several specific methods. For instance, if $\mathbf{D}$ is computed using geodesic distances (Wang et al., 2021) and $\mathbf{W}$ is configured such that the off-diagonal elements are set to 1 whereas the diagonal elements are set to 0, WMDS corresponds to Isomap (Tenenbaum et al., 2000). If $\mathbf{D}$ is computed using Euclidean distances and $\mathbf{W}$ is configured similarly, WMDS reduces to CMDS (Sonthalia et al., 2021). Additionally, if $\mathbf{D}$ is computed using Euclidean distances and $\mathbf{W}$ is specified as $w_{ij} = \frac{1}{d_{ij}}$ with a normalization factor of $\frac{1}{\sum_{i<j} d_{ij}}$, WMDS translates to Sammon Mapping (Lerner et al., 1998).

As previously mentioned, WMDS has essential applications in data science, with one notable application being dimensionality reduction. In this context, a dataset $\mathbf{X} \coloneqq [\mathbf{x}_1, \cdots, \mathbf{x}_n]^\top \in \mathbb{R}^{n\times m}$ is embedded into a lower-dimensional space based on a distance matrix $\mathbf{D}$. Typically, the projection dimension ($p < m$) is chosen to be 2 or 3, which aids in the intuitive visualization and interpretation of high-dimensional data. Another crucial application is graph drawing. In this scenario, $\mathbf{D}$ is derived from the shortest-path distances between nodes in the graph. By setting the projection dimension p to 2, the graph can be visualized in the plane, thereby providing a clearer understanding of complex network structures.

There are three primary approaches to solving WMDS: (i) conversion to CMDS, (ii) gradient-based methods, and (iii) Stress Majorization. When all weights are set to 1, except for the diagonal elements which are set to 0, and the distances in $\mathbf{D}$ are Euclidean, WMDS reduces to CMDS. To reformulate the problem into CMDS, the distance matrix is initially transformed into a similarity matrix through a process called double-centering. The optimization problem can be regarded as an eigenvalue problem, which is solvable through eigenvalue decomposition with a computational complexity of $\mathcal{O}(n^3)$. Consequently, CMDS using the Euclidean distance is equivalent to Principal Component Analysis, providing a closed-form solution (Ghojogh et al., 2023). However, in general cases, WMDS is NP-hard (Demaine et al., 2021). In such cases, heuristic methods such as gradient-based approaches or Stress Majorization are more suitable.

Gradient-based approaches encompass both first-order and second-order methods. First-order methods refer to algorithms that utilize at least one first-derivative or gradient, such as the gradient descent method. The gradient descent method is straightforward and effective, typically employing line search algorithms to select appropriate step sizes, thereby preventing increases in loss values during each iteration (Nesterov, 2018). However, in MDS, the complexity of calculating the loss is $\mathcal{O}(n^2)$, rendering the use of line search computationally expensive, as it necessitates multiple evaluations of the loss function at each step. Second-order methods, such as the Kamada-Kawai algorithm, apply Newton's method (or quasi-Newton methods) (Nesterov, 2018) to iteratively minimize the stress function. These methods generally exhibit greater efficiency in locating stationary points. Nonetheless, the Kamada-Kawai algorithm lacks theoretical guarantees, as it does not necessarily ensure a non-increasing loss value at each iteration and may fail to converge if the initial position is improper.

Stress Majorization (De Leeuw, 1988; Gansner et al., 2004), classified under the Majorization-Minimization algorithm (Sun et al., 2016), ensures non-increasing loss values without the requirement for line search, enhancing stability. Nevertheless, its efficiency diminishes as the dataset size increases, with a per-iteration complexity of $\mathcal{O}(\max(n^3, n^2 p))$, rendering it less practical for large-scale problems. Consequently, computational complexity remains a significant concern.

To address the computational problem, a modified stochastic gradient descent method has also been proposed (Zheng et al., 2017). A critical issue with this method is the choice of step size, which significantly affects the stability and efficiency of minimizing the stress function. Step sizes are typically chosen heuristically; however, our theoretical analysis from Inequality (3) suggests that as the number of point pairs in

the stress function increases, smaller step sizes might be required for stable optimization.

Based on the aforementioned context, it can be observed that the optimization of the stress function presents two primary challenges: **the stability of the optimization process** and **the computational complexity of the optimization method**. In this paper, we propose methods to address these issues. Our contributions are as follows:

- We introduce *StableMDS*, a novel gradient descent-based method. To enhance the stability of the optimization process, *StableMDS* provides theoretical guarantees that the loss values will not increase and that the optimization will converge to stationary points. Additionally, *StableMDS* reduces computational complexity compared to the Kamada-Kawai algorithm and Stress Majorization, making it more efficient.
- Furthermore, we present *FastMDS*, an accelerated variant of *StableMDS*. Although *FastMDS* lacks formal convergence guarantees, it empirically demonstrates robust convergence behavior and is particularly suitable for larger datasets.

2 Background & Related Work

2.1 Preliminaries

The set of real numbers is denoted by $\mathbb{R}$, with $\mathbb{R}^n$ representing the n-dimensional real-valued vector space and $\mathbb{R}^{n\times m}$ representing the space of $n \times m$ real-valued matrices. Column vectors are denoted by bold lowercase letters (e.g., $\mathbf{a}$), and matrices by bold uppercase letters (e.g., $\mathbf{A}$), while sets are denoted using curly braces $\{\cdot\}$ or calligraphic letters (e.g., $\mathcal{A}$). A column vector $\mathbf{a} \in \mathbb{R}^m$ is written as $\mathbf{a} \coloneqq [a_1, \ldots, a_m]^\top$. A matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ is defined as $\mathbf{A} \coloneqq [a_{ij}]_{n\times m}$, where a_{ij} is the (i, j)-element of $\mathbf{A}$. Alternatively, it can be written as $\mathbf{A} \coloneqq [\mathbf{a}_1, \ldots, \mathbf{a}_n]^\top$, where each $\mathbf{a}_i \in \mathbb{R}^m$ is the i-th row of $\mathbf{A}$ in column-vector form. The trace of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, denoted by $\mathrm{Tr}(\mathbf{A})$, is the sum of its diagonal elements: $\mathrm{Tr}(\mathbf{A}) \coloneqq \sum_{i=1}^n a_{ii}$. Let $g : \mathbb{R}^n \to \mathbb{R}$ be a differentiable real-valued function. Its partial derivative with respect to x_i is denoted by $\frac{\partial g}{\partial x_i}$, and $\left.\frac{\partial g}{\partial x_i}\right|_{x_i=x_0}$ is its value at $x_i = x_0$. The gradient of g is $\nabla g \coloneqq \frac{\partial g}{\partial \mathbf{x}} = \left[\frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n}\right]^\top$. We use $\|\cdot\|$ to denote the ℓ_2-norm of a vector, and write $\mathbf{A} \preceq \mathbf{B}$ to indicate that $\mathbf{B} - \mathbf{A}$ is positive semidefinite (Loewner order). Finally, $\max(\cdot)$ extracts the largest element from a set, so for real values $a_1, \ldots, a_n$ we write $\max(a_1, \ldots, a_n)$, and for a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, we interpret the entries of $\mathbf{A}$ as a set and write $\max \mathbf{A} = \max\{a_{ij} \mid 1 \le i \le n, 1 \le j \le m\}$.

2.2 Kamada-Kawai Algorithm

The Kamada-Kawai algorithm (Kamada and Kawai, 1989) generates graph layouts commonly known as Kamada-Kawai layout. It is a highly regarded force-directed method for graph drawing. The algorithm models the graph as a system of springs, where nodes repel each other and edges act as springs pulling connected nodes together. The objective is to minimize the system's overall energy to achieve an aesthetic arrangement of nodes. For a given graph, the algorithm seeks to minimize the following energy function $E(\mathbf{Y}) : \mathbb{R}^{n\times p} \to \mathbb{R}$, which is a specific instance of the stress function representing the positions of nodes in a two-dimensional plane:

$$E(\mathbf{Y}) \coloneqq \sum_{i<j} k_{ij}(\|\mathbf{y}_i - \mathbf{y}_j\| - l_{ij})^2,$$

where $k_{ij} = \frac{k_0}{d_{ij}^2}$ is the spring constant contingent on the shortest-path distance d_{ij} between nodes i and j, with k_0 being a constant scalar. The term $l_{ij} = \frac{l_0}{\max \mathbf{D}} d_{ij}$ represents the ideal spring length, where l_0 is also a constant scalar. Both k_0 and l_0 are generally set to 1. Newton's method (or quasi-Newton methods) iteratively minimizes this energy function. The update rule for node positions is:

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \mathbf{H}^{-1} \left.\frac{\partial E(\mathbf{Y})}{\partial \mathbf{y}_i}\right|_{\mathbf{y}_i=\mathbf{y}_i^{(t)}},$$

where $\mathbf{H} = \left.\frac{\partial^2 E(\mathbf{Y})}{\partial \mathbf{y}_i^2}\right|_{\mathbf{y}_i=\mathbf{y}_i^{(t)}} \in \mathbb{R}^{p\times p}$ is the Hessian matrix of $E(\mathbf{Y})$, evaluated at $\mathbf{y}_i$.

The computational complexity for each iteration is dominated by the calculation and inversion of the Hessian matrix, amounting to $\mathcal{O}(n^2p^3)$ in the general case and $\mathcal{O}(n^2p^2)$ in the two-dimensional case. This computational complexity implies that the Kamada-Kawai algorithm is not practical for high-dimensional cases.

2.3 Stress Majorization

Stress Majorization (De Leeuw, 1988; Gansner et al., 2004) is a commonly used solver for WMDS. It falls under the category of Majorization-Minimization algorithms (Sun et al., 2016), thereby ensuring non-increasing loss values during the optimization process. This method iteratively enhances the approximation of $S(\mathbf{Y})$ by employing surrogate convex functions that are easier to minimize. The associated loss function,

$F_z(\mathbf{Y}, \mathbf{Z}) : \mathbb{R}^{n\times p} \times \mathbb{R}^{n\times p} \to \mathbb{R}$, is defined as:

$$\begin{aligned} F_z(\mathbf{Y}, \mathbf{Z}) &\coloneqq \sum_{i<j} w_{ij} d_{ij}^2 + \mathrm{Tr}(\mathbf{Y}^\top \mathbf{L}^w \mathbf{Y}) - 2\mathrm{Tr}(\mathbf{Y}^\top \mathbf{L}^z \mathbf{Z}) \\ &\geq S(\mathbf{Y}), \end{aligned}$$

where $\mathbf{Z} \coloneqq [\mathbf{z}_1, \ldots, \mathbf{z}_n]^\top \in \mathbb{R}^{n\times p}$ is an arbitrary real-valued matrix. The Laplacian matrices $\mathbf{L}^w \coloneqq [l_{ij}^w]_{n\times n} \in \mathbb{R}^{n\times n}$ and $\mathbf{L}^z \coloneqq [l_{ij}^z]_{n\times n} \in \mathbb{R}^{n\times n}$ are defined as:

$$l_{ij}^w \coloneqq \begin{cases} -w_{ij} & \text{if } j \neq i \\ \sum_{k\neq i} w_{ik} & \text{if } i = j \end{cases}$$

and

$$l_{ij}^z \coloneqq \begin{cases} -w_{ij} \frac{d_{ij}}{\|\mathbf{z}_i - \mathbf{z}_j\|} & \text{if } j \neq i \\ \sum_{k\neq i} l_{ik}^z & \text{if } i = j. \end{cases}$$

The iterative update rule is

$$\mathbf{Y}^{(t+1)} = (\mathbf{L}^w)^\dagger \mathbf{L}^z \mathbf{Y}^{(t)},$$

where † denotes the pseudo-inverse. This iterative majorization algorithm is also known as the SMACOF (Scaling by MAjorizing a COmplicated Function).

The computational complexity of calculating the pseudo-inverse $(\mathbf{L}^w)^\dagger$ is $\mathcal{O}(n^3)$ (Golub and Kahan, 1965), and the complexity of matrix multiplication is $\mathcal{O}(\max(n^3, n^2 p))$. Consequently, the complexity of each iteration is $\mathcal{O}(\max(n^3, n^2 p))$. Due to this complexity, the algorithm is impractical for large datasets.

3 Methodology

In this section, we establish the theoretical foundation for the proposed method, followed by a detailed introduction to the technique. To ensure clarity in the subsequent discussion, we first restate the optimization problem as follows:

$$\min_{\mathbf{Y}} \left\{ S(\mathbf{Y}) \coloneqq \sum_{i<j} \underbrace{w_{ij} \left(\|\mathbf{y}_i - \mathbf{y}_j\| - d_{ij}\right)^2}_{f_{ij}(\mathbf{Y})} \right\},$$

where we define $f_{ij}(\mathbf{Y}) \coloneqq w_{ij} \left(\|\mathbf{y}_i - \mathbf{y}_j\| - d_{ij}\right)^2$ and note that $f_{ij}(\mathbf{Y}) = f_{ji}(\mathbf{Y})$.

3.1 Theoretical Foundation

The theoretical claim is as follows: *The stress function is non-convex, but the largest eigenvalue of its Hessian matrix at each point can be effectively upper bounded.* By applying the Descent Lemma to this property, it is ensured that the loss value does not increase when using gradient descent with an appropriate choice of step size derived from the magnitude of the upper bound.

We present two lemmas. The first lemma, the Descent Lemma, is typically derived under L-smoothness in the objective function. However, L-smoothness is merely a sufficient condition. In this study, we employ it in a broader context. Proofs of these lemmas are provided in the Supplementary Materials.

Lemma 1 (Descent Lemma)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function. Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{d} \in \mathbb{R}^n$ be vectors. If g satisfies the condition $\nabla^2 g(\mathbf{x}) \preceq L\mathbf{I}$ for any $\mathbf{x} \in \mathbb{R}^n$, where L is a non-negative constant and $\mathbf{I} \in \mathbb{R}^{n\times n}$ is the identity matrix, then the following inequality holds:*

$$g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) + \nabla g(\mathbf{x})^\top \mathbf{d} + \frac{L}{2}\|\mathbf{d}\|^2.$$

Lemma 2. *Consider the application of the gradient descent method to a function $g : \mathbb{R}^n \to \mathbb{R}$ that adheres to the Descent Lemma with a step size $\eta \in (0, \frac{2}{L})$. For any iteration t, the following inequality holds:*

$$g(\mathbf{x}^{(t+1)}) \leq g(\mathbf{x}^{(t)}) + \eta \left(\frac{L\eta}{2} - 1\right) \|\nabla g(\mathbf{x}^{(t)})\|^2.$$

Specifically, the function value decreases most rapidly when $\eta = \frac{1}{L}$, and the inequality simplifies to:

$$g(\mathbf{x}^{(t+1)}) \leq g(\mathbf{x}^{(t)}) - \frac{1}{2L}\|\nabla g(\mathbf{x}^{(t)})\|^2.$$

The gradient of $f_{ij}(\mathbf{Y})$ with respect to $\mathbf{y}_i$ is given by

$$\frac{\partial f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i} = 2w_{ij}(\mathbf{y}_i - \mathbf{y}_j)\left(1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}\right).$$

The Hessian matrix of $f_{ij}(\mathbf{Y})$ with respect to $\mathbf{y}_i$ is given by

$$\begin{aligned} \frac{\partial^2 f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i^2} = 2w_{ij} \Bigg(&\left(1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}\right)\mathbf{I} \\ &+ \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|} \frac{(\mathbf{y}_i - \mathbf{y}_j)(\mathbf{y}_i - \mathbf{y}_j)^\top}{\|\mathbf{y}_i - \mathbf{y}_j\|^2} \Bigg). \end{aligned}$$

For simplicity, let us denote $a \coloneqq 1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}$, $b \coloneqq \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}$, and define the unit vector $\mathbf{v} \coloneqq \frac{\mathbf{y}_i - \mathbf{y}_j}{\|\mathbf{y}_i - \mathbf{y}_j\|}$, where we assume $\mathbf{y}_i \neq \mathbf{y}_j$. Therefore, the Hessian matrix can be rewritten as

$$\frac{\partial^2 f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i^2} = 2w_{ij}\left(a\mathbf{I} + b\mathbf{v}\mathbf{v}^\top\right). \tag{1}$$

Subsequently, we introduce Proposition 1, which is essential for comprehending the eigenstructure of matrices associated with Equation (1).

Algorithm 1 StableMDS

Input: Maximum number of iterations T, initial positions of embeddings $\mathbf{Y}^{(0)}$, distance matrix $\mathbf{D}$, weighting matrix $\mathbf{W}$, boolean variable `shuffle`.
Output: Updated embeddings $\mathbf{Y}^{(nT)}$

Initialize the indices list indices $= [1, \ldots, n]$, where n is the number of points.
for $t = 1$ to T **do**
  if `shuffle` **then**
    Shuffle indices to randomize the order of updates.
  end if
  for index **in** indices **do**
    Set i to the current index.
    Compute the step size: $\eta_i = \frac{1}{\sum_{j=1,j\neq i}^{n} w_{ij}}$.
    Compute the gradient:
    $\mathbf{g}_i = \sum_{j=1,j\neq i}^{n} w_{ij} \left(\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(u)}\right) \left(1 - \frac{d_{ij}}{\max(\epsilon, \|\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(u)}\|)}\right)$,
    where $\mathbf{y}_j^{(u)} = \begin{cases} \mathbf{y}_j^{(t+1)} & \text{if } \mathbf{y}_j \text{ has been updated} \\ \mathbf{y}_j^{(t)} & \text{otherwise.} \end{cases}$
    Update the embedding: $\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i \mathbf{g}_i$.
  end for
end for

Proposition 1. *Consider a matrix* $\mathbf{B} = \alpha\mathbf{I} + \beta\mathbf{u}\mathbf{u}^\top$, *where* $\mathbf{I}$ *is the identity matrix,* $\mathbf{u}$ *is a non-zero vector in* $\mathbb{R}^n$, *and* α *and* β *are scalars. The eigenvalues of* $\mathbf{B}$ *are* α *with multiplicity* $n-1$ *and* $\alpha + \beta\|\mathbf{u}\|^2$ *with multiplicity 1.*

Proposition 1 demonstrates that $2w_{ij}(a\mathbf{I} + b\mathbf{v}\mathbf{v}^\top)$ possesses *two distinct types of eigenvalues*:

$$\begin{cases} 2w_{ij} \text{ with multiplicity } 1 \\ 2w_{ij}\left(1 - \dfrac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}\right) \text{ with multiplicity } p-1. \end{cases}$$

It is evident that $2w_{ij} \geq 0$ and $1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|} \leq 1$, which implies $2w_{ij} \geq 2w_{ij}\left(1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}\right)$. From this, we deduce that

$$\frac{\partial^2 f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i^2} \preceq 2w_{ij}\mathbf{I}. \tag{2}$$

We advance our discussion by focusing on $\mathbf{y}_i$. Noting that $f_{ij}(\mathbf{Y}) = f_{ji}(\mathbf{Y})$, we decompose $S(\mathbf{Y})$ into two components, A and B, as follows:

$$S(\mathbf{Y}) = \underbrace{\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y})}_{\text{A}} + \underbrace{\sum_{k=1,k\neq i}^{n} \sum_{j=k+1,j\neq i}^{n} f_{kj}(\mathbf{Y})}_{\text{B}}.$$

Term A is relevant to $\mathbf{y}_i$, whereas term B is not. Consequently, the partial derivative of $S(\mathbf{Y})$ with respect to $\mathbf{y}_i$ applies solely to term A, yielding

$$\frac{\partial^2 S(\mathbf{Y})}{\partial \mathbf{y}_i^2} = \sum_{j=1,j\neq i}^{n} \frac{\partial^2 f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i^2}.$$

From Inequality (2), it is direct to observe that the following inequality holds:

$$\frac{\partial^2}{\partial \mathbf{y}_i^2} \sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}) \preceq 2 \underbrace{\sum_{j=1,j\neq i}^{n} w_{ij}}_{=L} \mathbf{I}. \tag{3}$$

According to Lemma 1, $S(\mathbf{Y})$ satisfies the Descent Lemma when $L = 2\sum_{j=1,j\neq i}^{n} w_{ij} > 0$. We define the update formula for gradient descent for $\mathbf{y}_i$ as follows:

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i \left.\frac{\partial S(\mathbf{Y})}{\partial \mathbf{y}_i}\right|_{\mathbf{y}_i = \mathbf{y}_i^{(t)}}, \tag{4}$$

where η_i is the step size used when updating $\mathbf{y}_i$. We subsequently derive the following crucial proposition from Lemma 2.

Proposition 2. *Consider the update rule given by Equation (4), where the step size is defined as* $\eta_i = \frac{1}{2\sum_{j=1,j\neq i}^{n} w_{ij}}$ *for updating* $\mathbf{y}_i$. *Let* t *in* $\mathbf{Y}^{(t)}$ *represent*

Algorithm 2 FastMDS

Input: Maximum number of iterations T, initial positions of embeddings $\mathbf{Y}^{(0)}$, distance matrix $\mathbf{D}$, weighting matrix $\mathbf{W}$, sampling size b, boolean variable `shuffle`.
Output: Updated embeddings $\mathbf{Y}^{(nT)}$

Initialize the indices list indices $= [1, \ldots, n]$, where n is the number of points.
for $t = 1$ to T **do**
  if `shuffle` **then**
    Shuffle indices to randomize the order of updates.
  end if
  Uniformly sample a set of data points $\mathcal{B} \subset \{1, \ldots, n\}$, with a size of $b = |\mathcal{B}|$.
  for index **in** indices **do**
    Set i to the current index.
    Compute the step size: $\eta_i = \frac{1}{\sum_{j\in\mathcal{B},j\neq i} w_{ij}}$.
    Compute the gradient:
    $\mathbf{g}_i = \sum_{j\in\mathcal{B},j\neq i} w_{ij} \left(\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(u)}\right) \left(1 - \frac{d_{ij}}{\max(\epsilon, \|\mathbf{y}_i^{(t)} - \mathbf{y}_j^{(u)}\|)}\right)$,
    where $\mathbf{y}_j^{(u)} = \begin{cases} \mathbf{y}_j^{(t+1)} & \text{if } \mathbf{y}_j \text{ has been updated} \\ \mathbf{y}_j^{(t)} & \text{otherwise.} \end{cases}$
    Update the embedding: $\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i \mathbf{g}_i$.
  end for
end for

*Differences from the StableMDS are highlighted in **color**.

Table 1: Comparison of Algorithms for WMDS. Kamada-Kawai Algorithm, Stress Majorization, and StableMDS. The variable n represents the size of the dataset, while p denotes the projection dimension.

	Kamada-Kawai Algorithm	Stress Majorization	StableMDS (**Ours**)
Computational Complexity per Iteration	$\mathcal{O}(n^2p^3)$ if $p \geq 3$ $\mathcal{O}(n^2p^2)$ if $p = 2$	$\mathcal{O}(\max(n^3, n^2p))$	$\mathcal{O}(n^2p)$
Guarantee of Non-Increasing Loss Values	✗	✓	✓
Convergence Guarantee	Requires close to a stationary point	Stationary point	Stationary point
Convergence Rate	Quadratic (Newton) or superlinear (quasi-Newton), if close to a stationary point	Unknown	Sublinear
Auxiliary Space*	$\mathcal{O}(p^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(p)$

* *Auxiliary Space* refers to the additional memory that an algorithm requires during its execution, excluding the memory required to store the input data. Evaluating auxiliary space is critical because memory access times are often comparable to or slower than CPU computation times.

the total number of updates applied to any component $\mathbf{y}_i$ *of* $\mathbf{Y}$*. Under this update scheme, the following monotonicity property holds:* $S(\mathbf{Y}^{(t+1)}) \leq S(\mathbf{Y}^{(t)})$.

This proposition demonstrates that the step size η_i guarantees that the loss value does not increase at each iteration when updating $\mathbf{y}_i$.

3.2 StableMDS

We introduce our proposed method, *StableMDS*. In StableMDS, each data point is updated using the update formula specified in Equation (4) once per iteration. Additionally, data points to be updated can be selected either in a predetermined order or randomly during each iteration. The detailed steps of *StableMDS* are outlined in Algorithm 1.

A cautionary note in Algorithm 1 concerns the calculation of the step size $\mu_i = \frac{1}{2\sum_{j=1,j\neq i}^{n} w_{ij}}$ and the partial derivative $\frac{\partial f_{ij}(\mathbf{Y})}{\partial \mathbf{y}_i} = 2w_{ij}(\mathbf{y}_i - \mathbf{y}_j)\left(1 - \frac{d_{ij}}{\|\mathbf{y}_i - \mathbf{y}_j\|}\right)$. Multiplying these two terms cancels the factor of 2 in both the numerator and denominator, simplifying the calculation by omitting the 2.

We also establish the convergence speed of StableMDS. This ensures that StableMDS will always converge to a stationary point at a sublinear rate.

Proposition 3. *StableMDS guarantees that for each* $\mathbf{y}_i$*, an* ϵ*-approximate stationary point is achieved within* $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ *iterations. An* ϵ*-approximate stationary point is one that satisfies* $\left\| \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}) \right\| \leq \epsilon$.

We compare the performance of three different WMDS solvers, as summarized in Table 1. StableMDS, with a computational complexity of $\mathcal{O}(n^2p)$ and convergence guarantees, emerges as a versatile choice depending on the dataset size and specific downstream tasks.

3.3 FastMDS

Despite StableMDS significantly reducing the computational complexity while retaining the stability of Stress Majorization, it still encounters a bottleneck due to the requirement of $n-1$ gradient computations per data point, resulting in a significant computational burden. To address this issue, we propose a technique inspired by stochastic gradient descent. In this method, a random subset $\mathcal{B} \subset \{1, 2, \ldots, n\}$ of size $b = |\mathcal{B}|$ is selected, and only the corresponding rows of $\mathbf{Y}$ are used. This subset $\mathcal{B}$ approximates the original $n-1$ gradient computations. Specifically, the approximation is given by:

$$\frac{\partial S(\mathbf{Y})}{\partial \mathbf{y}_i} = \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}) \approx \frac{\partial}{\partial \mathbf{y}_i} \sum_{j\in\mathcal{B},j\neq i} f_{ij}(\mathbf{Y}).$$

The update formula for $\mathbf{y}_i$ becomes:

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i \left. \frac{\partial}{\partial \mathbf{y}_i} \sum_{j\in\mathcal{B},j\neq i} f_{ij}(\mathbf{Y}) \right|_{\mathbf{y}_i = \mathbf{y}_i^{(t)}},$$

where the step size η_i is approximated as $\eta_i \approx \frac{1}{2\sum_{j\in\mathcal{B},j\neq i} w_{ij}}$. This approach reduces the number of gradient computations per variable update to b, decreasing the computational complexity per iteration from $\mathcal{O}(n^2p)$ to $\mathcal{O}(nbp)$, where $b \ll n$. We refer to this method as *FastMDS*. Further details of the algorithm are provided in Algorithm 2.

The limitation of this method is the stochastic nature of the system, which complicates theoretical convergence proofs. Although theoretical convergence cannot be formally established at this stage, empirical convergence is demonstrated in the experimental section.

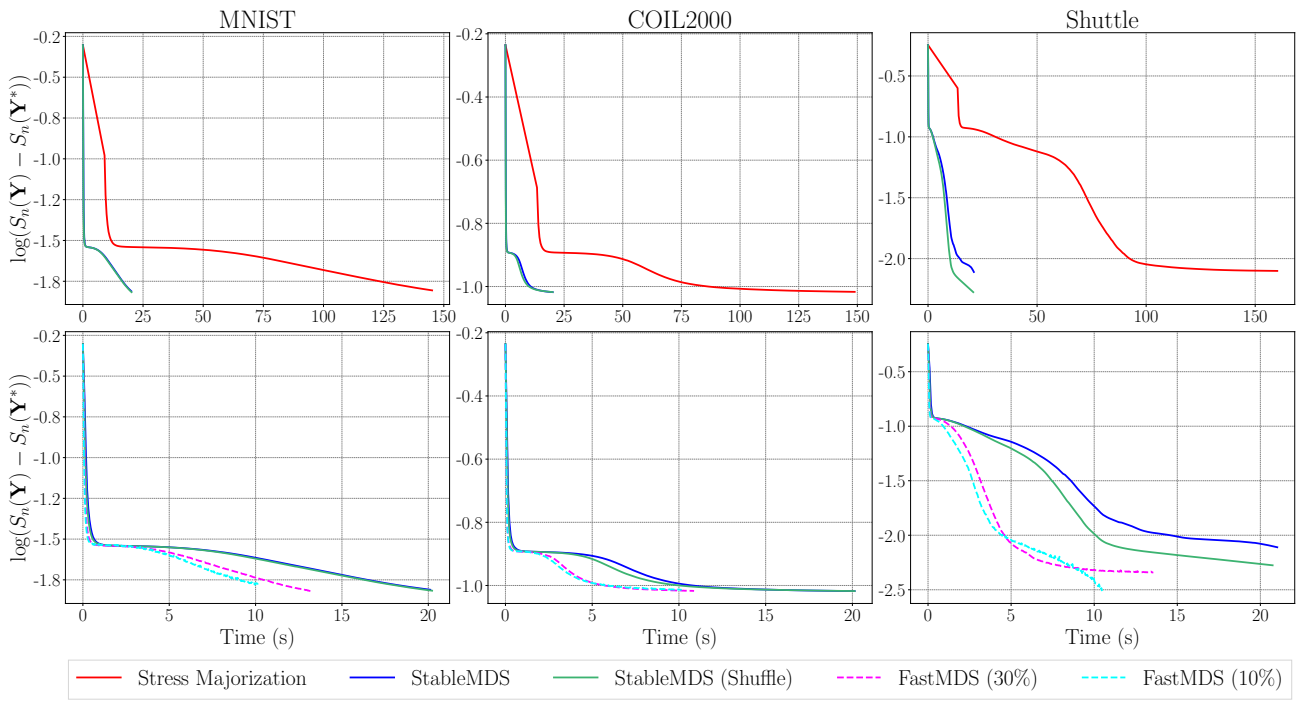

Figure 1: Optimization loss curves for Stress Majorization and the proposed methods (StableMDS, FastMDS). The horizontal axis represents time (s), and the vertical axis shows the loss $\log\left(S_n(\mathbf{Y}) - S_n(\mathbf{Y}^*)\right)$, where $S_n(\mathbf{Y}^*)$ is the minimum stress across all methods. This loss highlights convergence differences. **Top:** Stress Majorization vs. StableMDS. **Bottom:** StableMDS vs. FastMDS. In FastMDS ($q\%$), $q = \frac{b}{n} \times 100$.

4 Experiments

This section provides a detailed description of the experimental setup designed to evaluate the performance and effectiveness of the proposed methodologies, StableMDS and FastMDS, as efficient alternatives to existing WMDS solvers.

To benchmark their performance, we compare them against two widely used solvers: the Kamada-Kawai algorithm and Stress Majorization. Given their distinct applications, Stress Majorization is applied across all experimental settings, whereas the Kamada-Kawai algorithm is used exclusively for graph drawing experiments to ensure a fair comparison within its intended domain.

The baseline implementations are provided in well-established Python packages, whereas our proposed methods are implemented in Python with *Numba*-based acceleration. The details of the implementation are as follows: **(i) Kamada-Kawai Algorithm:** Implemented in the *NetworkX* package (Hagberg et al., 2008), using SciPy's `scipy.optimize.minimize` function. This function contains the L-BFGS-B quasi-Newton method (Byrd et al., 1995; Zhu et al., 1997), with its optimization routines written in highly optimized FORTRAN, enabling efficient computation. **(ii) Stress Majorization:** Implemented in the *scikit-learn* package (Pedregosa et al., 2011). In contrast to the Kamada-Kawai implementation in *NetworkX*, which leverages compiled code, the Stress Majorization implementation is written in pure Python. To enhance its performance, we accelerate the performance-critical components using *Numba* (Lam et al., 2015), achieving execution speeds comparable to those of C++. **(iii) StableMDS and FastMDS:** Both methods are implemented in Python and similarly accelerated using *Numba*, following an approach analogous to that used for Stress Majorization. The implementation is available on GitHub.

While the Kamada-Kawai algorithm is implemented in highly optimized FORTRAN, our proposed methods are implemented in Python with *Numba*-based acceleration to ensure a fair comparison. All experiments are conducted on an Apple M2 Pro (10-core CPU, 16-core GPU, 32GB RAM) using float64 precision.

4.1 Convergence Behaviors

In Section 3, we demonstrated that StableMDS theoretically does not increase the loss value per iteration, similar to Stress Majorization. To substantiate this claim, we conduct an experiment compar-

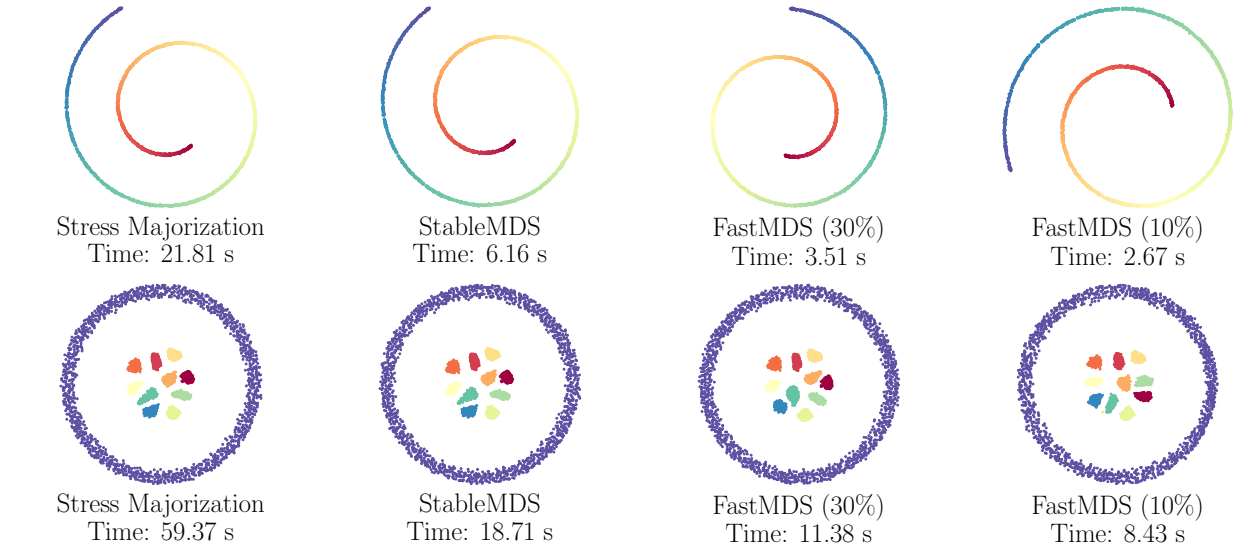

Figure 2: Visualization of embeddings generated by the Stress Majorization and the proposed methods (StableMDS, FastMDS). Time indicates the execution time until convergence. **Top:** SwissRoll. **Bottom:** Spheres.

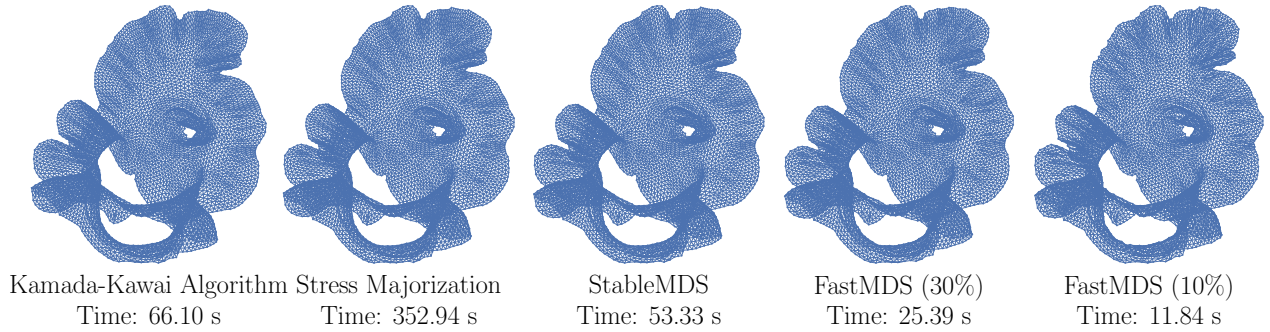

Figure 3: Graph layouts generated by different algorithms: Kamada-Kawai, Stress Majorization, and the proposed methods (StableMDS, FastMDS). Time indicates the execution time until convergence. Dataset: 3elt.

ing StableMDS and Stress Majorization. We also evaluate the optimization stability and computational speed of FastMDS, thereby including FastMDS in our evaluation. We select the MNIST (Deng, 2012), COIL2000 (Van Der Putten and Van Someren, 2004), and Shuttle (Ranka and Singh, 1998) datasets. Moreover, we sample 3,000 points from each dataset, as the original dataset sizes render matrix computations for Stress Majorization computationally difficult.

As illustrated in the top three subfigures of Figure 1, StableMDS produces non-increasing loss curves, similar to those observed with Stress Majorization, thereby ensuring stable optimization dynamics. The loss values of StableMDS at convergence closely match those obtained through Stress Majorization, with the shuffled variant of StableMDS achieving even lower loss values. Furthermore, StableMDS converges significantly faster than Stress Majorization, reducing computational overhead while maintaining high-quality solutions. These results not only confirm the theoretical guarantee of non-increasing loss values for StableMDS but also highlight its practical advantages in terms of efficiency and reliability in large-scale WMDS applications. Additionally, the bottom three subfigures reveal that FastMDS is considerably faster than StableMDS, further accelerating the optimization process. Despite its increased speed, FastMDS maintains stable learning behavior and superior convergence performance, striking a balance between efficiency and accuracy. This suggests that FastMDS is well-suited for scenarios where computational cost is a critical factor, such as real-time or large-scale embedding tasks. These observations collectively demonstrate the advantages of StableMDS and FastMDS over Stress Majorization, reinforcing their potential for scalable and efficient WMDS solvers.

4.2 Dimensionality Reduction

This experiment compares the proposed method with Stress Majorization for dimensionality reduction tasks. The objective is to assess whether the proposed method yields results comparable to those of Stress Majorization while requiring less execution time. Two topological datasets are selected for evaluation: Swiss-Roll (Marsland, 2011) and Spheres (Moor et al., 2020). The SwissRoll is a three-dimensional dataset consisting of 3,000 data points arranged on a two-dimensional manifold that spirals in three-dimensional space, resembling a rolled-up sheet. The Spheres is a 101-dimensional dataset, which is more complex than the SwissRoll. It contains a large empty sphere that includes 10 different smaller spheres. Convergence for each method is considered achieved when the following condition is met: $r_t \leq v_{\text{ftol}}$, where $r_t := \frac{|S_n(\mathbf{Y}^{(t)}) - S_n(\mathbf{Y}^{(t-1)})|}{\max(|S_n(\mathbf{Y}^{(t-1)})|, |S_n(\mathbf{Y})|, 1)}$ and $v_{\text{ftol}} := v_{\text{factr}} \times \epsilon_{\text{machine}}$. Here, v_{ftol} denotes the function tolerance, v_{factr} adjusts the convergence criteria stringency, and $\epsilon_{\text{machine}}$ is the machine epsilon, defined as the smallest positive number such that $1.0 + \epsilon$ is distinguishable from 1.0 in floating-point arithmetic. This method of convergence assessment is widely utilized in optimization libraries such as *SciPy* (Virtanen et al., 2020). The weighting matrix $\mathbf{W}$ is configured with off-diagonal elements set to 1 and diagonal elements set to 0, while v_{ftol} is set to 1×10^{10} for all methods.

The results presented in Figure 2 indicate that the proposed StableMDS and FastMDS methods produce outcomes comparable to those of Stress Majorization while significantly reducing runtime. For the Swiss-Roll dataset, both StableMDS and FastMDS achieve substantial runtime reductions relative to Stress Majorization. Similarly, for the Spheres, the proposed methods demonstrate notable improvements in computational efficiency. Moreover, the embeddings generated by StableMDS and FastMDS preserve essential topological structures and remain consistent with those obtained from Stress Majorization.

4.3 Graph Drawing

In the task of graph drawing, we compare our proposed methods with the Kamada-Kawai algorithm and Stress Majorization. Similar to the dimensionality reduction experiment, the purpose of this comparison is to ascertain whether our methods yield results comparable to those of the Kamada-Kawai algorithm and Stress Majorization, but within a shorter runtime. For evaluation, we utilize the large graph dataset 3elt (Rossi and Ahmed, 2015), which comprises 4,720 nodes and 13,722 edges. We apply the same convergence criteria introduced in the dimensionality reduction experiment, with one modification: the employed loss function is defined as $E_n(\mathbf{Y}) := \sqrt{\sum_{i<j} \left(\frac{\|\mathbf{y}_i - \mathbf{y}_j\|}{d_{ij}} - 1 \right)^2}$ instead of $S_n(\mathbf{Y})$, where $E_n^2(\mathbf{Y})$ is used in the Kamada-Kawai algorithm implemented in the *NetworkX* package.

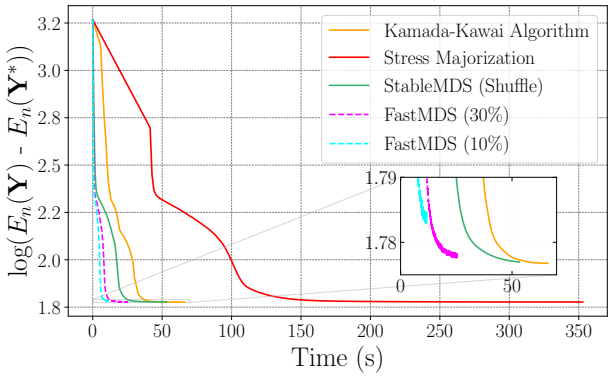

Figure 4: Optimization loss curves for the Kamada-Kawai algorithm, Stress Majorization, and the proposed methods (StableMDS, FastMDS). The horizontal axis represents time in seconds, and the vertical axis displays the loss value, defined as $\log\left(E_n(\mathbf{Y}) - E_n(\mathbf{Y}^*)\right)$.

As illustrated in Figure 3, StableMDS and FastMDS (30%) produce results comparable to those of the Kamada-Kawai algorithm and Stress Majorization while achieving a shorter runtime. Figure 4 shows that although FastMDS (10%) does not reduce the loss as much as the other methods, it remains sufficient for obtaining a comprehensive overview of the layout.

5 Conclusion

In this paper, we theoretically proved that the proposed method, StableMDS, ensures non-increasing loss values and convergence to stationary points while reducing computational complexity compared to the Kamada-Kawai algorithm and Stress Majorization. We evaluated its practical performance through three experiments, demonstrating its effectiveness. The results also indicate that our other proposed method, FastMDS, achieves comparable performance at a significantly higher speed, albeit without theoretical convergence guarantees. This makes FastMDS particularly suitable for applications requiring fast solvers for large-scale processing while maintaining reasonable accuracy, such as dimensionality reduction and graph drawing. Consequently, both methods serve as effective and efficient alternatives to existing solvers.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 22K12175.

References

Agarwal, S., Wills, J., Cayton, L., Lanckriet, G., Kriegman, D., and Belongie, S. (2007). Generalized non-metric multidimensional scaling. In Artificial Intelligence and Statistics (AISTATS), pages 11–18. PMLR.

Balanzá-Martínez, V., Kapczinski, F., de Azevedo Cardoso, T., Atienza-Carbonell, B., Rosa, A. R., Mota, J. C., and De Boni, R. B. (2021). The assessment of lifestyle changes during the covid-19 pandemic using a multidimensional scale. Revista de Psiquiatria y Salud Mental, 14(1):16–26.

Bertsekas, D. (2009). Convex Optimization Theory, volume 1. Athena Scientific.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. SIAM Journal on scientific computing, 16(5):1190–1208.

De Leeuw, J. (1988). Convergence of the majorization method for multidimensional scaling. Journal of Classification, 5:163–180.

Demaine, E., Hesterberg, A., Koehler, F., Lynch, J., and Urschel, J. (2021). Multidimensional scaling: Approximation and complexity. In International Conference on Machine Learning (ICML), volume 139, pages 2568–2578.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6):141–142.

Gansner, E., Koren, Y., and North, S. (2004). Graph drawing by stress majorization. Lect Notes Comput Sc, 3383.

Ghojogh, B., Crowley, M., Karray, F., and Ghodsi, A. (2023). Multidimensional Scaling, Sammon Mapping, and Isomap. Springer International Publishing, Cham.

Golub, G. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2(2):205–224.

Hagberg, A., Swart, P. J., and Schult, D. A. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs. Information Processing Letters, 31(1):7–15.

Kruskal, J. B. and Wish, M. (1978). Multidimensional Scaling. Number 11. Sage.

Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: A llvm-based python jit compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, pages 1–6.

Lerner, B., Guterman, H., Aladjem, M., Romem, Y., et al. (1998). On pattern classification with sammon's nonlinear mapping an experimental study. Pattern Recognition, 31(4):371–381.

Mardia, K. (1978). Some properties of clasical multidimesional scaling. Communications in Statistics - Theory and Methods, 7(13):1233–1241.

Marsland, S. (2011). Machine Learning: An Algorithmic Perspective. Chapman and Hall/CRC.

Moor, M., Horn, M., Rieck, B., and Borgwardt, K. (2020). Topological autoencoders. volume 119, pages 7045–7054.

Nesterov, Y. (2018). Lectures on Convex Optimization, volume 137. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. Journal of Machine Learning Research, 12:2825–2830.

Ranka, S. and Singh, V. (1998). Clouds: A decision tree classifier for large datasets. In Knowledge Discovery and Data Mining Conference (KDD), pages 2–8.

Rossi, R. and Ahmed, N. (2015). The network data repository with interactive graph analytics and visualization£. In AAAI conference on artificial intelligence (AAAI), volume 29.

Saeed, N., Nam, H., Al-Naffouri, T. Y., and Alouini, M.-S. (2019). A state-of-the-art survey on multidimensional scaling-based localization techniques. IEEE Communications Surveys & Tutorials, 21(4):3565–3583.

Samet, H. (2006). Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann.

Sonthalia, R., Van Buskirk, G., Raichel, B., and Gilbert, A. (2021). How can classical multidimensional scaling go wrong? Neural Information Processing Systems (NeurIPS), 34:12304–12315.

Sun, Y., Babu, P., and Palomar, D. P. (2016). Majorization-minimization algorithms in signal processing, communications, and machine learning. IEEE Transactions on Signal Processing, 65(3):794–816.

Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. Science, 290(5500):2319–2323.

Van Der Putten, P. and Van Someren, M. (2004). A bias-variance analysis of a real world learning problem: The coil challenge 2000. Machine Learning, 57:177–195.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272.

Wang, M., Xu, X., Yue, Q., and Wang, Y. (2021). A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proceedings of the VLDB Endowment, 14(11):1964–1978.

Zheng, J. X., Pawar, S., and Goodman, D. F. M. (2017). Graph drawing by stochastic gradient descent. IEEE Transactions on Visualization and Computer Graphics, 25(9):2738–2748.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on mathematical software (TOMS), 23(4):550–560.

Checklist

The checklist follows the references. For each question, choose your answer from the three possible options: Yes, No, Not Applicable. You are encouraged to include a justification to your answer, either by referencing the appropriate section of your paper or providing a brief inline description (1-2 sentences). Please do not modify the questions. Note that the Checklist section does not count towards the page limit. Not including the checklist in the first submission won't result in desk rejection, although in such case we will ask you to upload it during the author response period and include it in camera ready (if accepted).

In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Materials

A Proof of Lemma 1

Lemma 3 (Mean Value Theorem, Convex Optimization Theory (Bertsekas, 2009), Prop. A.3.1, p. 238)**.** *: Let $g : \mathbb{R}^n \to \mathbb{R}$ be a function that is differentiable over an open set (or ball) S, let $\mathbf{x}$ be a vector in S, and let $\mathbf{d}$ be such that $\mathbf{x} + \mathbf{d} \in S$. Then there exists an $\alpha \in [0, 1]$ such that*

$$g(\mathbf{x} + \mathbf{d}) = g(\mathbf{x}) + \nabla g(\mathbf{x} + \alpha\mathbf{d})^\top \mathbf{d}.$$

Moreover, if g is additionally twice continuously differentiable over S, there exists an $\alpha \in [0, 1]$ such that

$$g(\mathbf{x} + \mathbf{d}) = g(\mathbf{x}) + \nabla g(\mathbf{x})^\top \mathbf{d} + \frac{1}{2}\mathbf{d}^\top \nabla^2 g(\mathbf{x} + \alpha\mathbf{d})\mathbf{d}. \tag{5}$$

Proof. If g satisfies $\nabla^2 g \preceq L\mathbf{I}$, then g also satisfies $\mathbf{v}^\top \nabla^2 g(\mathbf{x})\mathbf{v} \leq L\|\mathbf{v}\|^2$ for any $\mathbf{v}$. From Equation (5), we derive the inequality:

$$g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) + \nabla g(\mathbf{x})^\top \mathbf{d} + \frac{L}{2}\|\mathbf{d}\|^2,$$

which illustrates the Descent Lemma. □

B Proof of Lemma 2

Proof. We have the inequality $g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) + \nabla g(\mathbf{x})^\top \mathbf{d} + \frac{L}{2}\|\mathbf{d}\|^2$ from Lemma 1, and the update rule $\mathbf{d} = -\eta\nabla g(\mathbf{x})$ from the gradient descent method. Substituting $\mathbf{d} = -\eta\nabla g(\mathbf{x})$ into this inequality, we obtain:

$$\begin{aligned} & g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) - \eta\|\nabla g(\mathbf{x})\|^2 + \frac{L\eta^2}{2}\|\nabla g(\mathbf{x})\|^2, \\ \implies \quad & g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) + \eta\left(\frac{L\eta}{2} - 1\right)\|\nabla g(\mathbf{x})\|^2. \end{aligned}$$

Since $\|\nabla g(\mathbf{x})\|^2 \geq 0$ and $L > 0$, we deduce that for $\eta \in (0, \frac{2}{L})$, the term $\eta\left(\frac{L\eta}{2} - 1\right)\|\nabla g(\mathbf{x})\|^2$ is non-positive. Specifically, the function value decreases most rapidly when $\eta = \frac{1}{L}$, and the inequality simplifies to:

$$g(\mathbf{x} + \mathbf{d}) \leq g(\mathbf{x}) - \frac{1}{2L}\|\nabla g(\mathbf{x})\|^2.$$

□

C Lemmas for Proposition 1

To substantiate Proposition 1, we introduce two lemmas: Lemma 4 and Lemma 5.

Lemma 4. *Let $\mathbf{u} = [u_1, \ldots, u_n]^\top$ be a non-zero vector in $\mathbb{R}^n$. Consider the matrix $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$, which represents the outer product of $\mathbf{u}$ with itself. Then, $\mathbf{A}$ has rank 1.*

Lemma 5. *Suppose $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$ is a rank-1 matrix, where $\mathbf{u}$ is a non-zero vector in $\mathbb{R}^n$. The matrix $\mathbf{A}$ possesses exactly one non-zero eigenvalue. This eigenvalue, denoted as λ, is the square of the norm of $\mathbf{u}$, i.e., $\lambda = \|\mathbf{u}\|^2$. All other eigenvalues are zero.*

D Proof of Lemma 4

Proof. To demonstrate that the matrix $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$ has rank 1, we proceed with the following rigorous proof, assuming that $\mathbf{u}$ is a non-zero vector. This assumption implies the existence of at least one component $u_i \neq 0$.

Given a vector $\mathbf{u} \in \mathbb{R}^n$, the matrix $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$ is defined such that its elements a_{ij} are given by the outer product of $\mathbf{u}$ with itself, i.e., $a_{ij} = u_i u_j$, where $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, n\}$.

Consider an arbitrary vector $\mathbf{v} = [v_1, \ldots, v_n]^\top \in \mathbb{R}^n$. The product $\mathbf{A}\mathbf{v}$ results in a new vector where the i-th component is given by:

$$(\mathbf{A}\mathbf{v})_i = \sum_{j=1}^{n} a_{ij} v_j = \sum_{j=1}^{n} u_i u_j v_j = u_i \left(\sum_{j=1}^{n} u_j v_j \right).$$

Let $k = \sum_{j=1}^{n} u_j v_j$, which is a scalar. Thus, the product $\mathbf{A}\mathbf{v}$ can be expressed as

$$\mathbf{A}\mathbf{v} = k\mathbf{u}.$$

This indicates that the result of multiplying $\mathbf{A}$ by any vector $\mathbf{v}$ is a scalar multiple of $\mathbf{u}$.

The rank of a matrix is defined as the maximum number of linearly independent column vectors in the matrix. Since any vector resulting from the transformation $\mathbf{A}\mathbf{v}$ is a scalar multiple of $\mathbf{u}$, all columns of $\mathbf{A}$ are linearly dependent on the single vector $\mathbf{u}$. Therefore, the maximum number of linearly independent columns in $\mathbf{A}$ is 1, indicating that the rank of $\mathbf{A}$ is 1.

This proof holds under the assumption that $\mathbf{u}$ is non-zero. If $\mathbf{u}$ were the zero vector, then $\mathbf{A}$ would be the zero matrix, and its rank would be 0. □

E Proof of Lemma 5

Proof. Consider the matrix $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$, where $\mathbf{u}$ is a non-zero vector in $\mathbb{R}^n$. Let $\mathbf{v}$ be an arbitrary vector in $\mathbb{R}^n$ and examine the equation $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ for some scalar λ.

Case 1: $\mathbf{v}$ is proportional to $\mathbf{u}$. Assume $\mathbf{v} = c\mathbf{u}$ for some scalar c. Then,

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \mathbf{A}(c\mathbf{u}) \\ &= (\mathbf{u}\mathbf{u}^\top)(c\mathbf{u}) \\ &= c(\mathbf{u}\mathbf{u}^\top\mathbf{u}) \\ &= c\|\mathbf{u}\|^2\mathbf{u}. \end{aligned}$$

Hence,

$$\begin{aligned} \mathbf{A}\mathbf{v} &= c\|\mathbf{u}\|^2\mathbf{u} \\ &= \|\mathbf{u}\|^2\mathbf{v}, \end{aligned}$$

indicating that $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ with eigenvalue $\lambda = \|\mathbf{u}\|^2$.

Case 2: $\mathbf{v}$ is orthogonal to $\mathbf{u}$. In this scenario, $\mathbf{u}^\top\mathbf{v} = 0$ and

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \mathbf{u}\mathbf{u}^\top\mathbf{v} \\ &= \mathbf{u}(0) \\ &= \mathbf{0}. \end{aligned}$$

Thus, $\mathbf{A}\mathbf{v} = 0\mathbf{v}$, indicating that $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ with eigenvalue $\lambda = 0$.

Since $\mathbf{A}$ is a rank-1 matrix, there can only be one linearly independent vector that results in a non-zero product when transformed by $\mathbf{A}$, which is any scalar multiple of $\mathbf{u}$. Consequently, $\mathbf{A}$ possesses exactly one non-zero eigenvalue, $\|\mathbf{u}\|^2$, associated with the eigenvector $\mathbf{u}$, and all other eigenvalues must be zero.

This completes the proof that $\mathbf{A} = \mathbf{u}\mathbf{u}^\top$ has one non-zero eigenvalue $\lambda = \|\mathbf{u}\|^2$ and that all other eigenvalues are zero. □

F Proof of Proposition 1

Proof. We demonstrate that α is an eigenvalue of $\mathbf{B} = \alpha\mathbf{I} + \beta\mathbf{u}\mathbf{u}^\top$ with multiplicity $n-1$ and that $\alpha + \beta\|\mathbf{u}\|^2$ is an eigenvalue with multiplicity 1.

Case 1: Eigenvalue α with multiplicity $n-1$. Let V denote the subspace of $\mathbb{R}^n$ consisting of all vectors orthogonal to $\mathbf{u}$. Since $\mathbf{u}$ is non-zero, the dimension of V is $n-1$. For any $\mathbf{v} \in V$, we have $\mathbf{u}^\top\mathbf{v} = 0$, implying that $\mathbf{u}\mathbf{u}^\top\mathbf{v} = 0$. Consequently, for any $\mathbf{v} \in V$, we obtain

$$\begin{aligned}\mathbf{B}\mathbf{v} &= (\alpha\mathbf{I} + \beta\mathbf{u}\mathbf{u}^\top)\mathbf{v} \\ &= \alpha\mathbf{I}\mathbf{v} + \beta\mathbf{u}\mathbf{u}^\top\mathbf{v} \\ &= \alpha\mathbf{v}.\end{aligned}$$

Therefore, each $\mathbf{v} \in V$ is an eigenvector of $\mathbf{B}$ corresponding to the eigenvalue α, establishing that the multiplicity of α is at least $n-1$.

Case 2: Existence of a distinct eigenvalue. Now, consider the vector $\mathbf{u}$ and calculate $\mathbf{B}\mathbf{u}$:

$$\begin{aligned}\mathbf{B}\mathbf{u} &= (\alpha\mathbf{I} + \beta\mathbf{u}\mathbf{u}^\top)\mathbf{u} \\ &= \alpha\mathbf{I}\mathbf{u} + \beta\mathbf{u}\mathbf{u}^\top\mathbf{u} \\ &= \alpha\mathbf{u} + \beta\|\mathbf{u}\|^2\mathbf{u}.\end{aligned}$$

Since $\|\mathbf{u}\|^2 > 0$, it follows that $\mathbf{u}$ is an eigenvector of $\mathbf{B}$ corresponding to the eigenvalue $\alpha + \beta\|\mathbf{u}\|^2$. This eigenvalue is distinct from α, as $\beta \neq 0$ and $\mathbf{u} \neq \mathbf{0}$.

Case 3: Multiplicities of the eigenvalues. From Case 1, α is an eigenvalue with at least $n-1$ independent eigenvectors. Case 2 establishes the existence of a distinct eigenvalue $\alpha + \beta\|\mathbf{u}\|^2$. Since the sum of the multiplicities of the eigenvalues must equal n, the dimension of our matrix, and since the multiplicities are equal for diagonalizable matrices (as is the case for $\mathbf{B}$), the multiplicity of α is exactly $n-1$, and the multiplicity of $\alpha + \beta\|\mathbf{u}\|^2$ is exactly 1.

Thus, we have proven that the matrix $\mathbf{B}$ has two distinct eigenvalues: α with multiplicity $n-1$ and $\alpha + \beta\|\mathbf{u}\|^2$ with multiplicity 1. □

G Proof of Proposition 3

Proof. We begin by considering the update formula as given in Equation (4), which is iteratively applied to each row of the matrix $\mathbf{Y}$. By leveraging Lemma 2, we first establish the following inequality:

$$\begin{aligned} & S(\mathbf{Y}^{(t+1)}) \leq S(\mathbf{Y}^{(t)}) - \frac{1}{2L}\left\|\frac{\partial}{\partial \mathbf{y}_i} S(\mathbf{Y}^{(t)})\right\|^2 \\ \Longrightarrow \quad & S(\mathbf{Y}^{(t+1)}) \leq S(\mathbf{Y}^{(t)}) - \frac{1}{2L}\left\|\frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1, j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2, \end{aligned} \tag{6}$$

where L is a non-negative constant.

Step 1: Summing Over Iterations. By summing Inequality (6) over the first N iterations, we obtain:

$$\begin{aligned}
&\sum_{t=0}^{N-1}\left[S(\mathbf{Y}^{(t+1)})-S(\mathbf{Y}^{(t)})\right] \leq -\frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \\
\implies \quad & S(\mathbf{Y}^{(N)})-S(\mathbf{Y}^{(0)}) \leq -\frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \\
\implies \quad & S(\mathbf{Y}^{(N)}) \leq S(\mathbf{Y}^{(0)})-\frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 . \qquad (7)
\end{aligned}$$

Step 2: Bounding the Gradient Norms. Let $\mathbf{Y}^*$ denote the global optimum of the stress function $S(\mathbf{Y})$. Since the stress function is non-negative, we have:

$$S(\mathbf{Y}^{(N)}) \geq S(\mathbf{Y}^*). \qquad (8)$$

Substituting this into Inequality (7) yields:

$$\begin{aligned}
& S(\mathbf{Y}^{(N)}) \leq S(\mathbf{Y}^{(0)})-\frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \\
\implies \quad & S(\mathbf{Y}^{*}) \leq S(\mathbf{Y}^{(0)})-\frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \\
\implies \quad & S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*}) \geq \frac{1}{2L}\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \\
\implies \quad & \sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \leq 2L\left(S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*})\right). \qquad (9)
\end{aligned}$$

Step 3: Contradiction Argument. To establish a contradiction, assume that for all time steps t, the following inequality holds:

$$\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 > \frac{2L}{N}\left(S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*})\right). \qquad (10)$$

Summing this assumption over all time steps t yields:

$$\begin{aligned}
&\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 > \sum_{t=0}^{N-1}\frac{2L}{N}\left(S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*})\right) \\
\implies \quad &\sum_{t=0}^{N-1}\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 > 2L\left(S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*})\right).
\end{aligned}$$

However, this yields a contradiction with Inequality (9) derived in Step 2. Therefore, the assumed Inequality (10) must be false, implying that there must exist at least one iteration t such that:

$$\left\|\frac{\partial}{\partial \mathbf{y}_i}\sum_{j=1,j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)})\right\|^2 \leq \frac{2L}{N}\left(S(\mathbf{Y}^{(0)})-S(\mathbf{Y}^{*})\right).$$

Step 4: Conclusion. We seek to demonstrate that there exists at least one iteration t where the gradient norm satisfies:

$$\left\| \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1, j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)}) \right\|^2 \leq \epsilon^2.$$

By considering Inequality (9), we observe:

$$\sum_{t=0}^{N-1} \left\| \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1, j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)}) \right\|^2 \leq 2L \left(S(\mathbf{Y}^{(0)}) - S(\mathbf{Y}^{*}) \right),$$

we consider the average gradient norm over N iterations. If the sum of the squared gradient norms is bounded, then on average, the gradient norm squared will also be bounded:

$$\frac{1}{N} \sum_{t=0}^{N-1} \left\| \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1, j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)}) \right\|^2 \leq \frac{2L}{N} \left(S(\mathbf{Y}^{(0)}) - S(\mathbf{Y}^{*}) \right). \tag{11}$$

To ensure that the gradient norm at some iteration t is less than or equal to ϵ^2, we set the right-hand side of Inequality (11) to be less than or equal to ϵ^2:

$$\frac{2L}{N} \left(S(\mathbf{Y}^{(0)}) - S(\mathbf{Y}^{*}) \right) \leq \epsilon^2.$$

Solving for N gives:

$$N \geq \frac{2L}{\epsilon^2} \left(S(\mathbf{Y}^{(0)}) - S(\mathbf{Y}^{*}) \right).$$

By considering the average, we use the fact that if the average value of a set of non-negative numbers (in this case, the squared gradient norms) is small, then at least one of those numbers must be small. Therefore, there must exist at least one iteration t such that:

$$\left\| \frac{\partial}{\partial \mathbf{y}_i} \sum_{j=1, j\neq i}^{n} f_{ij}(\mathbf{Y}^{(t)}) \right\|^2 \leq \epsilon^2.$$

This demonstrates that the number of iterations N required for the gradient norm to reach the desired threshold ϵ is:

$$N = \mathcal{O}\left(\frac{2L}{\epsilon^2} \left(S(\mathbf{Y}^{(0)}) - S(\mathbf{Y}^{*}) \right) \right).$$

Thus, the complexity of the gradient descent method is

$$\mathcal{O}\left(\frac{1}{\epsilon^2} \right).$$

□