An Asymptotic Analysis of Random Partition Based Minibatch Momentum Methods for Linear Regression Models

Yuan Gao, Xuening Zhu, Haobo Qi, Guodong Li, Riquan Zhang, and Hansheng Wang

School of Statistics and KLATASDS-MOE, East China Normal University, Shanghai, China; School of Data Science, Fudan University, Shanghai, China; Guanghua School of Management, Peking University, Beijing, China; Department of Statistics and Actuarial Science, University of Hong Kong, Hong Kong, China; School of Statistics and Information, Shanghai University of International Business and Economics, Shanghai, China.

1. Introduction

The technological advances have revolutionized data collection in many fields. A large number of real practices such as online retailing (Akter and Wamba 2016), social media interaction (e.g., Twitter and Weibo), Internet search engines (e.g., Google and Baidu) and countless sensing devices generate huge amounts of data on a daily basis. These massive datasets present not only opportunities but also challenges for traditional statistical methods (Fan et al. 2020). Consider for example the well-known ImageNet dataset. It contains more than one million photos and takes up approximately 150GB space on a hard drive (Deng et al. 2009). Obviously, most computers available to researchers cannot easily load such an enormous dataset as a whole into its memory. Then, how to process these massive datasets efficiently with statistical guarantees becomes a problem of great interest. To address this issue, it is in urgent need to develop novel and scalable algorithms for statistical analysis.

In this regard, a lot of research efforts have been devoted in recent years to dealing with large-scale statistical optimization problems. For example, distributed computing methods, mostly based on the divide-and-conquer strategy from computer science literature, have gained great popularity in the statistics community. Many statistical learning problems have been investigated under a distributed computing framework, such as generalized linear models (Chen and Xie 2014; Battey et al. 2018; Tang, Zhou, and Song 2020), quantile regression (Chen, Liu, and Zhang 2019; Chen et al. 2020b), nonparametric regression (Chang, Lin, and Wang 2017; Chang, Lin, and Zhou 2017), support vector machine (Wang et al. 2019), principal component analysis (Fan et al. 2019; Chen et al. in press), and many others. Clearly, the distributed computing frameworks reduce the computational burden of one single machine by employing a computer cluster. However, it relies on a distributed computing system for implementation, which could be expensive for many practitioners. Then how to solve complex optimization problems with massive datasets and limited computing resource becomes a problem of great importance.

For this problem, the method of gradient descent (GD) has been popularly used. Unlike second-order methods, such as the Newton-Raphson algorithm, a first-order method like GD requires no computation of the Hessian matrix. In fact, a standard GD algorithm only requires the gradients of the objective function and a learning rate. Therefore, it is computationally more feasible for sophisticated models and massive datasets. Furthermore, the cost in computer memory for storing Hessian matrices can be significantly reduced. Unfortunately, there is no free lunch. The price paid by the GD methods for this excellent practical feasibility is slow numerical convergence. In other words, the numerical convergence rate of a GD algorithm is usually much slower than that of a Newton-Raphson algorithm, if the Newton-Raphson algorithm can indeed be implemented. This is particularly true for ill-conditioned problems (Polyak 1987). In this case, feature directions with tiny variability are likely to be ignored by a standard GD algorithm.
To fix this problem, Polyak (1964) proposed a heavy-ball method to accelerate the standard GD algorithm. Specifically, it uses the information from not only the current step but also the previous step (i.e., the momentum). As a consequence, the stability of the search direction can be much improved and the numerical convergence rate can be accelerated. This leads to the important method of gradient descent with momentum (GDM). Once proposed, the method of GDM has received considerable attention both theoretically and practically. Theoretically, the novel idea of GDM has inspired a lot of follow-up studies, including Nesterov (1983), Beck and Teboulle (2009), Kingma and Ba (2014), Gitman et al. (2019), and many others. It is also found practically useful for complex optimization problems with high dimensional parameter and heavily correlated features (Sutskever et al. 2013; Goodfellow, Bengio, and Courville 2016; Bottou, Curtis, and Nocedal 2018). In fact, the method of GDM has been well implemented by many important software libraries, including but not limited to TensorFlow (Abadi et al. 2015) and PyTorch (Paszke et al. 2017). As a consequence, the method of GDM has been extensively used in practice.

Despite its theoretical popularity, the practical implementation of the classical GDM algorithm is not immediately straightforward for massive datasets. In this case, the massive dataset cannot be easily loaded into computer memory as a whole. Consequently, it has to be processed in a minibatch-by-minibatch manner. This leads to various minibatch-based GDM (MGDM) algorithms. Here, “minibatch” refers to a subsample of the original whole dataset. Subsequently, the important quantities (e.g., gradients and momentums) can be computed based on the minibatches. This approach leads to various stochastic gradient descent (SGD) methods (Moulines and Bach 2011; Wang et al. 2013; Toulis and Airoldi 2017; Chen et al. 2020a; Zhu, Chen, and Wu in press; Li et al. 2022). Specifically, there are two typical types of stochastic minibatch methods. The first type of stochastic minibatch method concerns streaming data analysis, where the minibatches should be generated sequentially (Luo and Song 2020). The key feature of this type of method is that it should be a one-pass type of analysis. In other words, no minibatch should be repeatedly used by the algorithm. This unique feature makes streaming data analysis useful for analyzing not only independent but also time series data. For more detailed discussions in this regard, we refer to Anava et al. (2013), Xie et al. (2019), Fang, Xu, and Yang (2018), Chen et al. (2020a), and Luo and Song (2020). The second type of stochastic minibatch method constructs minibatches by independent subsampling from the whole dataset (Wang et al. 2013; Xiao and Zhang 2014; Li et al. 2014; Assran and Rabbat 2020). In this case, different minibatches are typically assumed to be conditionally independent of each other, once the whole dataset is given. Accordingly, any given observation in the whole dataset should have a positive probability to be repeatedly sampled by every two consecutive minibatches. Theoretically, the properties of the above stochastic minibatch methods have been extensively studied in the literature. Most of them are non-asymptotic type of analysis (Kidambi et al. 2018; Gitman et al. 2019; Loizou and Richtarik 2020; Liu, Gao, and Yin 2020; Assran and Rabbat 2020).

In addition to the aforementioned two types of minibatch methods, there exists a third type of minibatch method, which we call random partition in this work. This type of minibatch method has been widely used in practice (Krizhevsky, Sutskever, and Hinton 2012; Simonany and Zisserman 2014; He et al. 2016). By random partition, we mean that minibatches in the same epoch iteration are randomly partitioned in a nonoverlapping way. Consequently, every single observation is effectively used only once in every epoch iteration of the algorithm. This also makes a random partition approach different from the subsampling method of Ma, Mahoney, and Yu (2015), where only a subset of the whole sample is used for estimation. In practice, there are two different ways to construct random partition. One way is to randomly partition the whole sample only once and then fix these minibatches throughout the rest of the algorithm. The minibatches generated in this way are called fixed minibatches. The other way is to shuffle the whole sample and repartition it again in each epoch iteration. The minibatches generated in this way are called shuffled minibatches. In either case, within the same epoch, different minibatches should be mutually exclusive with each other. This means that those observations already sampled by previous minibatches cannot be resampled by subsequent minibatches within the same epoch iteration.

As one can see, the random partition methods discussed above are somewhat similar to the first type of stochastic minibatch method (i.e., the streaming data analysis), in the sense that the minibatches are processed in a one-after-the-other manner. The key difference is that a typical streaming data analysis is usually a one-pass type of analysis, whereas a random partition method is a multi-pass method. The random partition methods are also similar to the second type of stochastic minibatch method (i.e., independent subsampling). However, the key difference is that the minibatches used by a random partition method are inevitably dependent on each other, conditioned on the given whole dataset. This is because different minibatches in this case are mutually exclusive with each other. Consequently, the conditional independence property is destroyed, making the theoretical understanding of random partition methods more difficult. This also leads to an interesting phenomenon. That is the random partition methods, as one important type of minibatch methods, have been extensively used in practice and implemented by many standard software libraries (e.g., TensorFlow or PyTorch). Nevertheless, their theoretical properties are not well studied in the literature. This is an obvious discrepancy between the real practices and the existing literature. We are then inspired to develop this work to fill this theoretical gap.

We find that the well developed nonasymptotic techniques in the literature cannot be immediately used to study the theoretical properties of the algorithm in this work. The main reason is that the MGDM algorithm studied in this work assumes that the minibatches are obtained by random partition instead of streaming data or independent subsampling. This places new challenges to subsequent theoretical analysis. To solve the problem, we develop here a novel asymptotic analysis paradigm. Specifically, we study here the classical linear regression model and start with one particular type of the MGDM algorithm (i.e., the fixed minibatch-based GDM algorithm, FIXED). In this case, we are able to identify the nearly sufficient and necessary conditions for the tuning parameters (i.e., the learning rate and the momentum parameter) to ensure numerical convergence.
In addition to that, we further investigate the numerical convergence rate and provide the theoretical guidance for tuning parameters specification. Moreover, we study the asymptotic properties of the resulting MGDM estimator. We find that a diminishing learning rate is needed for the MGDM estimator to achieve the optimal statistical efficiency. Finally, we conduct extensive numerical experiments to validate our theoretical findings.

In summary, the key contribution we attempt to make in this work is to provide a relatively complete theoretical analysis for the random partition based MGDM algorithm. One key feature of this algorithm is that the minibatches used here are formed by random partitions. This makes the existing non-asymptotic techniques not immediately applicable. To solve the problem, a novel asymptotic analysis paradigm is developed, and fruitful theoretical findings are obtained. The rest of the article is organized as follows. Section 2 first introduces the MGDM method under the linear regression setup. Then, the numerical convergence properties and the statistical efficiency of the resulting estimator are investigated. Section 3 includes extensive numerical experiments to corroborate our theoretical findings. We conclude the article in Section 4 and discuss a number of interesting topics for future study.

2. Minibatch-Based Gradient Descent with Momentum

2.1. Classical Gradient Descent with Momentum

We first introduce some frequently used notations for this article. Let $A$ be a matrix in $\mathbb{R}^{p \times p}$. We use $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ to denote the largest and smallest eigenvalues, respectively, of $A$, if $A$ is symmetric. We define $\|A\|_F = \sqrt{\lambda_{\max}(A^T A)}$ as the operator norm and the Frobenius norm of $A$, respectively. In addition, $\rho(A) = \max\{|\lambda| : \lambda$ is the eigenvalue of $A\}$, which stands for the spectral radius of $A$. As usual, $\|v\| = \sqrt{v^T v}$ is the standard $\ell_2$-norm of an arbitrary vector $v \in \mathbb{R}^p$. We start with a classical linear regression model. Let $(Y_i, X_i)$ be the observation collected from $i$th subject with $1 \leq i \leq N$. Throughout the rest of this article, we assume that $(Y_i, X_i)$’s are independent and identically distributed. Assume

$$Y_i = X_i^T \theta + \varepsilon_i, \quad \text{with} \quad 1 \leq i \leq N, \quad (2.1)$$

where $X_i \in \mathbb{R}^p$ is the covariate, $Y_i \in \mathbb{R}$ is the response, $\varepsilon_i$ is the random error with mean 0 and variance $\sigma^2$, and $\theta \in \mathbb{R}^p$ is the unknown parameter vector with true value given by $\theta_0 = (\theta_{01}, \ldots, \theta_{0p})^T$. Suppose the whole sample is indexed by $S = \{1, \ldots, N\}$. To estimate the unknown parameter vector $\theta$, we usually minimize the ordinary least squares (OLS) loss function as

$$\mathcal{L}(\theta) = N^{-1} \sum_{i=1}^{N} \ell_i(\theta) = (2N)^{-1} \sum_{i=1}^{N} (Y_i - X_i^T \theta)^2,$$

where $\ell_i(\theta) = 2^{-1}(Y_i - X_i^T \theta)^2$ is the loss evaluated on the $i$th observation. We know that the solution to this problem is the classical ordinary least squares (OLS) estimator that has an explicit form as $\hat{\theta}_{\text{ols}} = \arg \min_{\theta} \mathcal{L}(\theta) = \Sigma_{xx}^{-1} \Sigma_{xy}$, where $\Sigma_{xx} = N^{-1} \sum_{i=1}^{N} X_i X_i^T \in \mathbb{R}^{p \times p}$ and $\hat{\Sigma}_{xy} = N^{-1} \sum_{i=1}^{N} X_i Y_i \in \mathbb{R}^p$.

Under some mild conditions, one can show that $\sqrt{N}(\hat{\theta}_{\text{ols}} - \theta_0) \rightarrow d N(0, \sigma^2 \Sigma_{xx}^{-1})$, where $\Sigma_{xx}$ is the covariance matrix of $X_i$ (Rao 1973; Shao 2003). To practically compute $\hat{\theta}_{\text{ols}}$, the sample covariance matrix $\hat{\Sigma}_{xx}$ needs to be calculated and inverted. This leads to a computational complexity of order $O(Np^2 + p^3)$ in general.

Another popular way to compute $\hat{\theta}_{\text{ols}}$ is to use the first-order optimization methods in an iterative way. Arguably, the most basic first-order optimization method is the gradient descent (GD) algorithm. For the OLS problem, a standard GD algorithm should update the estimates iteratively as $\hat{\theta}(t) = \hat{\theta}(t-1) - \alpha \nabla \mathcal{L}(\hat{\theta}(t-1))$, where $\nabla \mathcal{L}(\theta)$ denotes the gradient of $\mathcal{L}(\cdot)$ at $\theta$ and $\alpha > 0$ is the learning rate controlling the step size. As shown by Nesterov (2018), with an appropriately selected $\alpha$, the estimates $\{\hat{\theta}(t)\}$ generated by the GD algorithm should satisfy $\|\hat{\theta}(t) - \hat{\theta}_{\text{ols}}\| \leq \rho^t\|\hat{\theta}(0) - \hat{\theta}_{\text{ols}}\|$ for some convergence factor $\rho \in (0, 1)$ and an arbitrary initial value $\hat{\theta}(0)$. The optimal convergence factor can be achieved if we choose $\alpha = 2/(\lambda_{\max}(\Sigma_{xx}) + \lambda_{\min}(\Sigma_{xx}))$. In this case, the optimal convergence factor should be $\rho = (\kappa - 1)/(\kappa + 1)$, where $\kappa = \lambda_{\max}(\Sigma_{xx})/\lambda_{\min}(\Sigma_{xx}) \geq 1$ is the condition number of $\Sigma_{xx}$.

However, when the condition number $\kappa$ is large, the convergence factor $\rho$ could be very close to 1. Consequently, the convergence rate could be painfully slow. To accelerate it, Polyak (1964) proposed the heavy-ball method, which is also called the momentum method (Sutskever et al. 2013). Specifically, in this work, we investigate the gradient descent with the classical momentum, which updates the estimates as

$$\hat{\theta}(t) = \hat{\theta}(t-1) - \alpha \nabla \mathcal{L}(\hat{\theta}(t-1)), \quad (2.2)$$

$$\nu(t) = \gamma \nu(t-1) + \alpha \nabla \mathcal{L}(\hat{\theta}(t-1)), \quad (2.3)$$

where $\alpha > 0$ is the learning rate and $\gamma > 0$ is the momentum parameter. Polyak (1964) showed that, by choosing appropriate tuning parameters $\alpha$ and $\gamma$, the optimal convergence factor of the momentum method could be achieved as $\rho = (\sqrt{\kappa} - 1)/((\sqrt{\kappa} + 1)$, which is strictly larger than that of the GD with momentum (GDM) method as long as $\kappa > 1$. This suggests that the optimal convergence rate of a GDM algorithm should be faster than that of a standard GD algorithm.

2.2. Fixed Minibatch-Based Gradient Descent with Momentum

Despite its theoretical attractiveness, a standard GDM method cannot be immediately used to handle massive datasets. In many cases, the dataset could be too large to be comfortably processed as a whole by computer memory. As a consequence, it has to be placed on the hard drive and then processed in a minibatch-by-minibatch manner. In practice, this approach leads to the most popularly used minibatch-based gradient descent methods with momentum (MGDM). Specifically, to practically implement an MGDM algorithm, we first need to randomly partition the whole sample $S$ into $M$ disjoint minibatches as $S = \sqcup_{m=1}^{M} S_{(m)}$ with $S_{(m_1)} \cap S_{(m_2)} = \emptyset$ for each $m_1 \neq m_2$. Throughout the article, we assume that $M$ is fixed. Without loss of generality, we assume that $|S_{(m)}| = n$ for each $1 \leq m \leq M$. Obviously,
we should have \( N = Mn \). We start with the fixed minibatch-based GDM (FIXED) method. In other words, we assume that the random partition \( S = \cup_{m=1}^{M} S_{(m)} \), once given, should be fixed throughout the rest of the algorithm.

In the fixed minibatch setting, the OLS loss function based on \( S_{(m)} \) can be defined as \( L_{(m)}(\theta) = (2n)^{-1} \sum_{i \in S_{(m)}} (Y_i - X_i^T \theta)^2 \).

The corresponding gradient is \( \hat{L}_{(m)}(\theta) = \hat{\Sigma}_{xx} \theta - \hat{\Sigma}_{xy} \), where \( \hat{\Sigma}_{xx} = n^{-1} \sum_{i \in S_{(m)}} X_iX_i^T \in \mathbb{R}^{p \times p} \) and \( \hat{\Sigma}_{xy} = n^{-1} \sum_{i \in S_{(m)}} X_iY_i \in \mathbb{R}^p \).

Subsequently, a standard GDM algorithm should be executed in a minibatch-by-minibatch manner. Specifically, let \( \hat{\theta}_{(m)} \) be the estimate obtained from the \( m \)th minibatch in the \( t \)th epoch. Then we should have

\[
\hat{\theta}_{(t,m)} = \hat{\theta}_{(t,m-1)} - \nu(t,m),
\]

\[
\nu(t,m) = \gamma\nu(t,m-1) + \alpha \hat{L}_{(m)}(\hat{\theta}_{(t,m-1)}),
\]

where \( \hat{\theta}_{(t,0)} = \hat{\theta}_{(t-1,M)}, \nu(t,0) = \nu(t-1,M) \), and \( \hat{\theta}_{(0,m)} \) (1 \( \leq m \leq M \)) is the initial estimate specified for the \( m \)th minibatch. This immediately leads to three important questions. First, if we can treat (2.4) and (2.5) as a linear dynamical system, does this dynamical system have a stable solution? Second, does the FIXED algorithm converge to this stable solution? Last, if the FIXED algorithm does converge to the stable solution under appropriate conditions, what is the statistical properties of the resulting estimator?

To support the theoretical investigation, we require a set of technical conditions, which are presented as follows.

(C1) (Covariates) The covariates \( X_1, \ldots, X_N \) are assumed to be independent and identically distributed \( p \)-dimensional random vectors with mean \( 0 \) and covariance matrix \( \Sigma_{xx} \), where \( \Sigma_{xx} \in \mathbb{R}^{p \times p} \) is a positive definite matrix with bounded eigenvalues \( \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0 \). Further assume that \( \mathbb{E}[||X||^4] < \infty \).

(C2) (Noise Terms) The random errors \( \varepsilon_1, \ldots, \varepsilon_N \) are assumed to be independent and identically distributed random variables with mean 0 and variance \( \sigma^2 \). Further assume that \( \varepsilon_i \) is independent of the covariates \( X_i \).s.

Specifically, condition (C1) assumes that the covariates have a positive definite covariance matrix. This condition is usually assumed for the linear regression model; see, for example, Zou and Li (2008) and Wang (2009). The additional finite fourth moment condition is used to elaborate the convergence rate of the sample covariance matrix of the covariates. Condition (C2) assumes that the random error has a finite variance, and is independent of the covariates. This condition is widely assumed in the literature (Shao 2003; Ma et al. 2020) to adopt the central limit theorem. We remark that the independent and identical distribution assumptions are made mainly for the sake of technical simplicity. They can be further relaxed. For example, the independence assumption about \( X_i \) and \( \varepsilon_i \) can be replaced by an uncorrelatedness assumption and some additional moment conditions. We refer to Section 5.2 of White (2001) for more detailed discussions.

### 2.3. The Linear Dynamical System and Stable Solution

We investigate the first question in this section: does there exist a stable solution to the FIXED algorithm? To address this important problem, we temporarily assume that there does exist a numerical limit \( \hat{\theta}^{(m)} \) such that \( \hat{\theta}_{(t,m)} \to \hat{\theta}^{(m)} \) as \( t \to \infty \) for each \( 1 \leq m \leq M \). This leads to an analytical solution as \( \hat{\theta}^{(m)} \) for each \( 1 \leq m \leq M \). This analytical solution further helps us to understand the conditions needed for its unique existence. Specifically, we write the stable solution vector as \( \hat{\theta}^{*} = (\hat{\theta}^{(1)}_1, \ldots, \hat{\theta}^{(M)}_M)^T \in \mathbb{R}^q \) with \( q = Mp \). Then, by (2.4) and (2.5), we know that \( \hat{\theta}^{*} \) should be the stable solution to the following linear dynamical system,

\[
\begin{align*}
\hat{\theta}^{(1)}_1 &= \Delta^{(1)} \hat{\theta}^{(t-1,1)} - \gamma \hat{\theta}^{(t-1,1)} + \alpha \hat{\Sigma}_{xy} \\
\hat{\theta}^{(2)}_2 &= \Delta^{(2)} \hat{\theta}^{(t-1,2)} - \gamma \hat{\theta}^{(t-1,2)} + \alpha \hat{\Sigma}_{xy} \\
&\vdots \\
\hat{\theta}^{(M)}_M &= \Delta^{(M)} \hat{\theta}^{(t-1,M)} - \gamma \hat{\theta}^{(t-1,M)} + \alpha \hat{\Sigma}_{xy}
\end{align*}
\]

where \( \Delta^{(m)} = (1 + \gamma)I_p - \alpha \hat{\Sigma}_{xx} \in \mathbb{R}^{p \times p} \) for \( 1 \leq m \leq M \). We further write \( \hat{\Sigma}_{xy} = (\hat{\Sigma}_{xy}^{(1)}^T, \ldots, \hat{\Sigma}_{xy}^{(M)}^T)^T \in \mathbb{R}^{q \times q} \), and

\[
\hat{\Omega} = \begin{bmatrix}
I_p & 0 & 0 & \cdots & 0 & y_{1p} & -\Delta^{(1)} \\
-\Delta^{(2)} & I_p & 0 & \cdots & 0 & 0 & y_{2p} \\
& -\Delta^{(3)} & I_p & \cdots & 0 & 0 & \vdots \\
0 & 0 & 0 & \cdots & y_{np} & -\Delta^{(M)} & I_p
\end{bmatrix}
\in \mathbb{R}^{q \times q}.
\]

Then, we should have \( \hat{\Omega} \hat{\theta}^{*} = \alpha \hat{\Sigma}_{xy}^{*} \) or equivalently \( \hat{\theta}^{*} = \alpha \hat{\Sigma}_{xy}^{*} \hat{\Omega}^{-1} \), provided \( \hat{\Omega} \) is invertible. Consequently, whether \( \hat{\Omega} \) is invertible or not determines the existence of the stable solution.

Then, in the following theorem, we elaborate the conditions to ensure its invertibility in probability.

**Theorem 1 (Stable Solution).** Assume condition (C1). Further assume that \( \gamma \neq 1, \alpha \neq 0 \), and \( \alpha \neq 2(1 + \gamma) / \lambda_j \) for each \( 1 \leq j \leq p \). Then, \( \hat{\Omega} \) is invertible with probability tending to one as \( n \to \infty \).

The proof of **Theorem 1** can be found in Appendix A.1. By **Theorem 1**, we know that the stable solution does exist for a wide range of different \( \alpha \) and \( \gamma \) values. In fact, as long as \( \gamma \neq 1, \alpha \neq 0 \), and \( \gamma \neq 2(1 + \gamma) / \lambda_j \) for every \( 1 \leq j \leq p \), we should have \( \hat{\Omega} \) invertible with high probability, and thus, the stable solution should exist. In particular, we find that the stable solution exists as \( \gamma \to \infty \). Nevertheless, we should remark that the existence of the stable solution and the FIXED estimator might be totally different. By the FIXED estimator, we mean that the FIXED algorithm should converge to a numerical limit \( \hat{\theta}^{(m)} \), and this numerical limit \( \hat{\theta}^{(m)} \) is referred to as the FIXED algorithm. Obviously, as long as such a numerical limit does exist, it must be a stable solution to (2.6). Nevertheless, this does not imply that every possible stable solution can be numerically approached by the FIXED algorithm. In fact, there do exist stable solutions (e.g., those stable solutions associated with very large \( \gamma \) values) that cannot be achieved using the FIXED algorithm. Consequently, it is not a FIXED estimator. Thus, we know that the existence of the stable solution should be a necessary condition for the existence of the FIXED estimator but might not be sufficient. Accordingly, we investigate the conditions to assure the numerical convergence of the FIXED algorithm in the next section.
2.4. Numerical Convergence Properties

We devote this section to studying the second important problem, that is, the numerical convergence properties of the FIXED algorithm. Specifically, we are interested in studying how the $\alpha$ and $\gamma$ would affect the numerical convergence of the FIXED algorithm. Recall that $\lambda_j \ (1 \leq j \leq p)$ is the $j$th largest eigenvalue of $\Sigma_{xx}$. The main results are summarized in the following theorem.

**Theorem 2 (Numerical Convergence).** Assume condition (C1).
(a) If $0 \leq \gamma < 1$ and $0 < \alpha < 2(1 + \gamma)/\lambda_1$ and assuming $n \to \infty$, then we have $\hat{\theta}^{(t,m)} \to \hat{\theta}^{(m)}$ as $t \to \infty$ holds with probability tending to one. (b) Assume $\gamma > 1$ or $\alpha > 2(1 + \gamma)/\lambda_1$. Then, for any constant $\delta > 0$, there exists an initial value $\hat{\theta}^{(0,m)}$ satisfying $0 < \|\hat{\theta}^{(0,m)} - \hat{\theta}^{(m)}\| < \delta$, such that $\|\hat{\theta}^{(t,m)} - \hat{\theta}^{(m)}\| \to \infty$ as $t \to \infty$ holds with probability tending to one as $n \to \infty$.

The proof of Theorem 2 can be found in Appendix A.2. Theorem 2 provides nearly sufficient and necessary conditions for numerical convergence. Specifically, from Theorem 2(a), we know that the numerical convergence can never be achieved if $\gamma > 1$. Consequently, the FIXED estimator does not exist in this case. Furthermore, with a given momentum parameter $0 \leq \gamma < 1$, the learning rate $\alpha$ should be sufficiently small. Otherwise, the algorithm should not converge. Remarkably, the upper bound of the learning rate $\alpha$ is related to the largest eigenvalue of $\Sigma_{xx}$ (i.e., $\lambda_1$) and the momentum parameter $\gamma$. This upper bound should be larger if the momentum parameter $\gamma$ is larger. As a result, this enables a larger learning rate to be used for a FIXED algorithm compared to a standard GD algorithm ($\gamma = 0$). Theorem 2 examines the conditions for the numerical convergence for the FIXED algorithm. However, it does not provide the exact convergence rate. In particular, we want to understand how the $\alpha$ and $\gamma$ specification would affect the numerical convergence rate. We next develop the following theorem.

**Theorem 3 (Numerical Convergence Rate).** Assume condition (C1). If $0 \leq \gamma < 1$ and $0 < \alpha < 2(1 + \gamma)/\lambda_1$, then there exists a constant $\rho = \max[1 - \alpha \lambda_1, |1 - \alpha \lambda_p|]$. In particular, the minimal (i.e., optimal) $\rho$ is given by $\rho = (\sqrt{\lambda_1}/\lambda_p - 1)/(\sqrt{\lambda_1}/\lambda_p + 1)$, which represents a very slow convergence rate. Moreover, from Theorem 3(c) we find that by choosing an appropriate $\gamma$, the convergence factor of the FIXED could be strictly smaller than that of a standard GD algorithm (i.e., FIXED method with $\gamma = 0$). This finding suggests that the momentum term can indeed accelerate the numerical convergence even in the cases with small learning rates. In addition, Theorem 3(c) discovers a very interesting and novel interactive relationship between the learning rate $\alpha$ and the momentum parameter $\gamma$. Specifically, it suggests that $\gamma$ should approach 1 as $\alpha \to 0$. This result theoretically explains why practitioners often like to specify $\gamma$ to be a value very close to 1 to strike a balance between both numerical and statistical convergence (Sutskever et al. 2013).

The proof of Theorem 3 is shown in Appendix A.3. From Theorem 3, we know that the FIXED algorithm should enjoy a linear convergence rate by choosing appropriate $\alpha$ and $\gamma$, as long as both $n$ and $t$ are large enough. By Theorem 3(a), we know that the optimal convergence factor $\rho = (\sqrt{\lambda_1}/\lambda_p - 1)/(\sqrt{\lambda_1}/\lambda_p + 1)$ can be achieved by carefully specified $\alpha$ and $\gamma$. This specification is closely related to the eigenvalue structure of the covariance matrix $\Sigma_{xx}$. Our findings are in line with theoretical findings in the classical GD method without minibatch (Polyak 1964). It is known that this convergence factor also attains the achievable lower bound of various first-order methods (Nesterov 2018). Moreover, Theorem 3(b) claims that, for a standard GD algorithm with $\gamma = 0$, the best convergence factor should be $\rho = (\lambda_1/\lambda_p - 1)/(\lambda_1/\lambda_p + 1)$. This represents a convergence rate that is strictly slower than that of the FIXED algorithm as long as $\lambda_1 > \lambda_p$.

Although the optimal numerical convergence can be guaranteed by Theorem 3(a) and (b) under different circumstances, the stable solution is not necessarily optimal in statistical efficiency. For most SGD-related methods, the decaying learning rate $\alpha$ is typically needed to improve statistical efficiency (Robbins and Monro 1951; Polyak and Juditsky 1992). In this case, from Theorem 3(c) we know that the convergence factor in this case tends to 1 as $\alpha \to 0$, which represents a very slow convergence rate. Moreover, from Theorem 3(c) we find that by choosing an appropriate $\gamma$, the convergence factor of the FIXED could be strictly smaller than that of a standard GD algorithm (i.e., FIXED method with $\gamma = 0$). This finding suggests that the momentum term can indeed accelerate the numerical convergence even in the cases with small learning rates. In addition, Theorem 3(c) discovers a very interesting and novel interactive relationship between the learning rate $\alpha$ and the momentum parameter $\gamma$. Specifically, it suggests that $\gamma$ should approach 1 as $\alpha \to 0$. This result theoretically explains why practitioners often like to specify $\gamma$ to be a value very close to 1 to strike a balance between both numerical and statistical convergence (Sutskever et al. 2013).

2.5. Statistical Properties of the Stable Solution

In this section, we investigate how the tuning parameters $\alpha$ and $\gamma$ would affect the statistical properties of the stable solution. Note that the OLS estimator is the global optimal solution to our problem. Ideally, we should have the resulting FIXED estimator stay with the OLS estimator as closely as possible. Consequently, we should examine the relationship between the stable solution and the OLS estimator. To this end, we need to analyze $\hat{\Omega}$ and its inverse (assume it exists) carefully. Accordingly, we decompose it into $\hat{\Omega} = A + \alpha \hat{B}$, where

$$
\begin{align*}
A = \begin{bmatrix}
I_p & 0 & \cdots & 0 & \gamma I_p & -(1 + \gamma) I_p \\
-(1 + \gamma) I_p & I_p & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & \gamma I_p & -(1 + \gamma) I_p & I_p \\
\end{bmatrix} 
\end{align*}
$$

(2.8)
and
\[
\hat{B} = \begin{bmatrix}
0 & 0 & \cdots & 0 & \hat{\Sigma}_{1x}^{(1)} \\
\hat{\Sigma}_{2x}^{(2)} & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & \hat{\Sigma}_{Mx}^{(M)} \\
0 & 0 & \cdots & 0 & 0
\end{bmatrix} \in \mathbb{R}^{d \times q}.
\]

Note that \(A \) and \( \hat{B} \) do not involve the learning rate \( \alpha \). Furthermore, it can be easily verified that \( A^\top P^* = A \hat{F}^* = 0 \), where \( P^* = I_M \otimes I_p \). Here, \( I_M \) denotes the \( M \)-dimensional vector filled with ones, \( I_p \in \mathbb{R}^{p \times p} \) is the identity matrix, and \( \otimes \) denotes the Kronecker product. Thus, the columns of \( P^* \) are the eigenvectors of \( A \) and \( \hat{F} \) corresponding to the eigenvalue 0. Next, let \( P_1 = P^*/\sqrt{M} \in \mathbb{R}^{p \times p} \). Then, we can construct \( P_2 \in \mathbb{R}^{d \times (p-1)} \) that has orthonormal columns and is orthogonal to \( P_1 \). We then have \( P = [P_1, P_2] \). The following equation is verified:

\[
P^\top \hat{\Omega} P = P^\top AP + \alpha P^\top \hat{B} P \in \begin{bmatrix}
\alpha P^\top \hat{B} P & \alpha P^\top \hat{B} P \\
\alpha P^\top \hat{B} P & \alpha P^\top \hat{B} P
\end{bmatrix}.
\]

Denote \( \hat{B}'_1 \), \( \hat{B}'_2 \), and \( \hat{B}'_3 \) as \( P^\top \hat{B} P \), \( P^\top \hat{B} P \), and \( P^\top \hat{B} P \), respectively. The following equation is simplified:

\[
P^\top \hat{\Omega} P = [\alpha P^\top \hat{B} P, \alpha P^\top \hat{B} P, \alpha P^\top \hat{B} P].
\]

Theorem 4 (Relationship with the OLS Estimator). Assume conditions (C1) and (C2). Further denote \( \hat{\theta}_0 \in I \otimes \theta_0 \). Then we have

(a) If \( \gamma \geq 0 \) is fixed and \( \alpha \to 0 \), then \( \hat{\theta}^* = \hat{\theta}_0^* + \alpha \hat{E}(1 + O_p(\alpha)) \), where \( \hat{E} = (P^1 \hat{\Sigma}^{(1)} P^1 \hat{B} - I_q) \{P_1(A^{(1)}_2)^{-1} P^\top_2 \} \hat{\Sigma}_0 - \hat{\Sigma}_0^* \).

(b) If \( \alpha \to 0 \) is fixed and \( \gamma \to \infty \), then \( \hat{\theta}^* = \hat{\theta}_0^* \alpha \hat{E} + \alpha \hat{E}(1 + O_p(\gamma)) \), where \( \hat{E} \) is the same as in (a) and \( \| (A^{(2)}_2)^{-1} \| \to 0 \) as \( \gamma \to \infty \).

The proof of Theorem 4 can be found in Appendix A.4. With a fixed \( \gamma \), it can be verified that \( \gamma \hat{\Sigma}^* = O_p(1) \) as \( n \to \infty \). Then by Theorem 4, we should have \( \alpha \sqrt{n} \hat{E} \to \hat{E} \) if \( \alpha \to 0 \). Hence, it can be ignored since we have \( \sqrt{n} \theta \) as \( \theta \to \infty \). With a fixed \( \alpha \), it can be verified that \( (A^{(2)}_2)^{-1} \) as \( \gamma \to \infty \). This further implies that \( \gamma \hat{\Sigma}^* \to 0 \) as \( \gamma \to \infty \). Consequently, we should have \( \hat{\theta}^* \to \hat{\theta}_0^* \) for \( 1 \leq m \leq M \) as long as \( \alpha \to 0 \) and \( \gamma \to \infty \). This immediately suggests two different ways to force the stable solution to stay close to the global OLS estimator. They are, respectively, \( \alpha \to 0 \) and \( \gamma \to \infty \). However, from Theorem 2(b), we know that the FIXED algorithm would not converge if \( \gamma > 1 \). This suggests that \( \alpha \) should be taken as small as possible for the best statistical efficiency. In the meanwhile, Theorem 3(b) and (c) suggest that too small \( \alpha \) leads to painfully slow convergence rate, and thus, is not practically acceptable either. Consequently, \( \alpha \) should be taken as small as possible, as long as the numerical convergence rate can be practically tolerated. Furthermore, for a given \( \alpha \), Theorem 3 suggests that the parameter momentum \( \gamma \) can be used to improve the numerical convergence. In particular, if \( \alpha \) is sufficiently small, Theorem 3(c) suggests that the optimal \( \gamma \) should be close to 1. In the past literature, \( \gamma = 0.9 \) has been popularly used (Simonyan and Zisserman 2014; He et al. 2016). We confirm it as a practically useful specification by extensive numerical experiments in Section 3.

To further investigate the stability of the parameter solution, we establish its asymptotic normality for fixed \( \gamma \) in the following theorem.

Theorem 5 (Asymptotic Normality). Assume conditions (C1) and (C2). Further assume that \( \gamma \geq 0 \) is fixed and \( \alpha \to 0 \). This yields

\[
\sqrt{N} \{ (\gamma^2 \hat{V}_m(\alpha) )^{-1/2} \{ \hat{\theta}(m) - \theta_0 + \hat{\theta}_m(\alpha) \} \} \to_d N_p(0, I_p),
\]

as \( N \to \infty \), where \( \hat{\theta}_m(\alpha) = O_p(\alpha^{2n/3-1}) \), \( V_m(\alpha) = \Sigma_{x}^{-1} + \alpha^2 M_{x}^2 \Sigma_{x} + O(\alpha^4) \), and \( d \) is the \( \ell_2 \) norm of the first row of the matrix \( P_2(A^{(2)}_2)^{-1} P_2^\top \).

The proof of Theorem 5 is shown in Appendix A.5 and the explicit formulas for calculating \( d \) can be found in Appendix A.6. Theorem 5 gives the asymptotic behavior of the stable solution. Specifically, for fixed \( \gamma \) and \( M \), the stable solution is still \( \sqrt{N} \)-consistent as long as \( \sqrt{N} \hat{\theta}_m(\alpha) = O_p(1) \). In addition, from the bias and variance term, we can see that as \( \alpha \to 0 \), the stable solution should be asymptotically equivalent to the OLS estimator.

2.6. Shuffled Minibatch-Based Gradient Descent with Momentum

The shuffled minibatch-based GDM (SHUFFLED) method is also a popularly used MGDM algorithm in practice, but little is understood regarding its statistical properties. However, Theorem 4 developed in the previous section provides us an opportunity to obtain some theoretical insights. Specifically, let \( S^{(k,m)} \) be the \( m \)th minibatch used in the \( k \)th shuffled partition \( (1 \leq k \leq K) \). We should have \( S = \bigcup_{m=1}^{M} S^{(k,m)} \) for every \( k \) and \( S^{(k,m_1)} \cap S^{(k,m_2)} = \emptyset \) for any \( m_1 \neq m_2 \) and every \( k \). Different shuffled partitions subsequently determine different dynamical systems and thus different stable solutions. More specifically, consider one particular shuffled partition \( \{ S^{(k,m)} : 1 \leq m \leq M \} \) for one particular \( 1 \leq k \leq K \). Then, the corresponding dynamical system becomes

\[
\left\{ \begin{array}{l}
\hat{\theta}(t+1) = \hat{\Delta}(k,1) \hat{\theta}(t-1,M) - \gamma \hat{\theta}(t-1,1,M-1) + \alpha \hat{\Sigma}_{kx}^{(1,1)} \\
\hat{\theta}(t+2) = \hat{\Delta}(k,2) \hat{\theta}(t-1,1,M) + \alpha \hat{\Sigma}_{kx}^{(1,2)} \\
\vdots \\
\hat{\theta}(t+M) = \hat{\Delta}(k,M) \hat{\theta}(t-1,1,M-1) - \gamma \theta(t-1,M-2) + \alpha \hat{\Sigma}_{kx}^{(1,1)} 
\end{array} \right.
\]

where \( \hat{\Delta}(k,m) = (1 + \gamma I_p - \alpha \hat{\Sigma}^{(k,m)}_{x}) \in \mathbb{R}^{p \times p}, \hat{\Sigma}^{(k,m)}_{xx} = n^{-1} \sum_{t \in S^{(k,m)}} X_t X_t^\top, \) and \( \hat{\Sigma}^{(k,m)}_{x} = n^{-1} \sum_{t \in S^{(k,m)}} X_t Y_t \) for each \( 1 \leq m \leq M \). We denote the corresponding stable solution by \( \hat{\theta}(k,m) \). Then, from Theorem 4, we know that \( \hat{\theta}(k,m) - \hat{\theta}_0 = \alpha \hat{E}(k,1) \{1 + O_p(\alpha)\} \), where \( \hat{E}(k) = (P_1 \hat{\Sigma}_0^{-1} P_1^\top \hat{B}(k) - I_q) \{P_2(A^{(2)}_2)^{-1} P_2^\top \} (\hat{\theta}(k,m) - \hat{\theta}_0) \). It has the same form as (2.9) but with \( \hat{\Sigma}_{xx}^{(m)} \) replaced by \( \hat{\Sigma}_{xx}^{(k,m)} \) for each \( 1 \leq m \leq M \),
and $\hat{\Sigma}_{xx}^{(k)\ast} = (\hat{\Sigma}_{xy}^{(k)\ast})^T$. Consequently, as long as we can show that $\alpha \max_{1 \leq k \leq K} \| \hat{E}^{(k)} \| = o_p(1/\sqrt{N})$, the difference between the shuffled estimator $\hat{\theta}_{ols}$ and the global OLS estimator is ignorable uniformly over all shuffled partitions.

Specifically, we first note that

$$\max_{1 \leq k \leq K} \| \hat{E}^{(k)} \| \leq \left( \max_{1 \leq k \leq K} \| P_1 \Sigma_{xx}^{-1} P_1 \hat{B}^{(k)} \| + 1 \right) \| P_2 (A_{xx}^{(2)} - P_2) \| \max_{1 \leq k \leq K} \| \hat{B}^{(k)} \| \hat{\theta}_{ols} - \hat{\Sigma}_{xy}^{(k)\ast},$$

where

$$P_1 \Sigma_{xx}^{-1} P_1 \hat{B}^{(k)} = M^{-1} \begin{bmatrix} \Sigma_{xx}^{-1} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} & \cdots & \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \\ \Sigma_{xx}^{-1} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} & \cdots & \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \\ \vdots & \ddots & \vdots \\ \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} & \cdots & \Sigma_{xx}^{-1} \hat{\Sigma}_{xx} \hat{\Sigma}_{xx} \end{bmatrix}.$$
the statistical properties of the stable solutions corresponding to the different tuning parameters specifications. Third, we compare the performances of the different types of MGDM methods. Forth, we apply the MGDM methods to the $\ell_2$-penalized least squares problems. Last, we examine the performance of the MGDM methods on a real-word dataset.

### 3.1. Numerical Convergence

In this section, we demonstrate how the tuning parameters $\alpha$ and $\gamma$ affect the numerical convergence rate of the FIXED method. Specifically, we consider a standard linear regression model (2.1) with $N = 5000$ and $p = 50$. The random noise $\varepsilon_i$'s are simulated from the standard normal distribution with mean 0 and variance $\sigma^2 = 1$. The true regression coefficient $\theta_0$ is given by $\theta_0 = (\theta_{01}, \ldots, \theta_{0p})$ with $\theta_{0j} = 10 \exp(-0.5j)$ for $1 \leq j \leq p$. Furthermore, the covariates $X_i$'s are simulated from a multivariate normal distribution $N_p(0, \Sigma_{xx})$, where $\Sigma_{xx} = \Sigma_1 + \lambda_1 \Sigma_0$. One can easily verify that $\lambda_1 = \lambda_{\max}(\Sigma_{xx}) = \kappa p + 1$ and $\lambda_p = \lambda_{\min}(\Sigma_{xx}) = 1$. Hence, the condition number of $\Sigma_{xx}$ is given by $\kappa p + 1$. To demonstrate the acceleration effect of the momentum term, we consider two different simulation experiments. Specifically, we consider two different momentum parameter $\alpha$ values, $0, 0.3, 0.6, 0.9$. For the second case, we fix $\alpha = 0.9$ and let $\gamma$ vary from 0.005 to 0.0005. For the second case, we fix $\gamma = 0.9$ and let $\alpha = 0.005, 0.002, 0.001, 0.0005$. Subsequently, we generate the simulated dataset as above example with $p = 1$. All other settings are the same as above example. This leads to a total of 100 $\delta(t)$ values for each epoch $1 \leq t \leq T = 30$. The median of these values are then plotted in Figure 2. From Figure 2, we obtain the following findings. First, by the left panel of Figure 2, we find that larger $\gamma$ value leads to faster numerical convergence. Specifically, the FIXED method with $\gamma = 0$ demonstrates the slowest convergence rate, while the FIXED method with $\gamma = 0.9$ demonstrates the fastest convergence rate. This verifies the conclusion in Theorem 3(c), that is a momentum parameter close to 1 should be beneficial for faster numerical convergence. Second, by the right panel of Figure 2, we find that the numerical convergence rate drops as the learning rate $\alpha$ decreases. This is in line with the conclusion of Theorem 3(c.2). That is smaller $\alpha$ value leads to slower numerical convergence.

### 3.2. Statistical Efficiency of the Stable Solution

This section is devoted to verifying the theoretical claims made in Section 2.5, that is, how the two tuning parameters (i.e., $\alpha$ and $\gamma$) affect the statistical efficiency of the stable solution. Specifically, we consider two different simulation experiments. One experiment studies the effects of the learning rate $\alpha$ with a fixed momentum parameter $\gamma$. The other experiment examines...
Figure 2. The numerical convergence performances of FIXED method with different \((\alpha, \gamma)\)-specifications. The vertical axis corresponds to the \(\delta^{(t)}\) value, while the horizontal axis corresponds to the epoch number. In the left panel, we fix the learning rate as \(\alpha = 0.005\) and let the momentum parameter \(\gamma\) vary from 0 to 0.9. In the right panel, we fix the momentum parameter as \(\gamma = 0.9\) and let the learning rate \(\alpha\) vary from 0.005 to 0.0005.

Figure 3. Boxplots of log(EE) values of the FIXED method under different \((\alpha, \gamma)\)-specifications. In the left panel, we fix the momentum parameter as \(\gamma = 0.9\) and let learning rate \(\alpha\) vary from 0.04 to 0.001. In the right panel, we fix the learning rate as \(\alpha = 0.02\) and let the momentum parameter \(\gamma\) vary from 0 to 10. The dotted reference line is the median of the log(EE) of the OLS estimator.

the effect of the momentum parameter \(\gamma\) while holding the learning rate \(\alpha\) fixed. In the first case, we fix \(\gamma = 0.9\) and let \(\alpha = 0.04, 0.02, 0.01, 0.005, 0.001\), while in the second case, we fix \(\alpha = 0.02\) and let \(\gamma = 0, 0.5, 0.9, 1, 2, 5, 10\). Subsequently, we generate the simulated dataset as in Section 3.1 with \(\kappa = 1\). For each given \((\alpha, \gamma)\)-specification, the stable solution is computed for the last minibatch as \(\hat{\theta}(M)\) by solving the linear dynamical system (2.6) directly. Then we evaluate the efficiency of different estimators by computing the estimation error (EE) as 

\[
\|\hat{\theta}(M) - \theta_0\|
\]

where \(\theta_0\) is the true regression coefficient. For comparison, we also compute the EE values of the OLS estimator. This experiment is randomly repeated a total of 100 times, leading to a total of 100 EE values for each estimator, which are then log-transformed and shown as boxplots in Figure 3.

From Figure 3, we can draw the following conclusions. First, from the left panel, we can see that, when the learning rate is large (e.g., 0.04, 0.02), the log(EE) values of corresponding stable solutions are much larger than that of the global OLS estimator. As the learning rate \(\alpha\) decays, the log(EE) values steadily drop and then converge to that of the OLS estimator. These findings verify the results in Theorem 4(a) and Theorem 5. That is, by letting \(\alpha \to 0\), the stable solution should have the same statistical efficiency as the global OLS estimator. Second, from the right panel of Figure 3 we can see that when \(\gamma\) is small (e.g., \(\gamma \leq 1\)), the log(EE) values of the corresponding stable solutions are slightly larger than that of the OLS estimator. As \(\gamma\) increases, the log(EE) values finally converge to that of the OLS estimator. This finding is in agreement with the claim in Theorem 4(b).

Finally, we find that the log(EE) values of the stable solution seem to increase from \(\gamma = 0\) to \(\gamma = 0.9\). To investigate this phenomenon, recall that by Theorem 5, the extra variance term \(\alpha^2 Md^2_{\gamma} \Sigma_{ex}\) should be the dominant term of the log(EE) value. Here, \(d_{\gamma}\) is the only factor involving \(\gamma\). Therefore, we calculate values of \(d_{\gamma}\) for \(\gamma\) ranging from 0 to 10 by the explicit formulas given in Appendix A.6. These values are then plotted, as shown in Figure 4. It can be easily seen that the curve of \(d_{\gamma}\) has a similar
pattern as that of the log(EE) values in the right panel of Figure 3. That is, the value of $d_γ$ first increases with increasing $γ$ from 0 to some value slightly smaller than 1, and then, $d_γ$ decreases as $γ$ increases. This result further validates the asymptotic variance formula given in Theorem 5.

### 3.3. Different Types of MGDM Methods

Next, we compare the performance of the different types of MGDM methods. Specifically, we consider three different types of minibatch methods, that is, fixed, shuffled and stochastic minibatches. The corresponding MGDM methods are referred to as FIXED, SHUFFLED and STOCHASTIC methods, respectively. For the STOCHASTIC method, we use simple random sampling without replacement to generate each minibatch. In this experiment, we consider three different learning rates as $α = 0.02, 0.01, 0.005$ and fix the momentum parameter as $γ = 0.9$ by convention. Subsequently, we simulate the dataset as in Section 3.1 with $κ = 1$. Once the data are generated, we run each MGDM algorithm for a total of $T = 100$ epochs and record the resulting estimator $\hat{θ}(T,M)$ from the $M$th minibatch. For a fair comparison, passing over every $M = N/n = 10$ minibatches is considered as one epoch for each MGDM method. Then we compute the corresponding estimation error (EE) as $\|\hat{θ}(T,M) - θ_0\|$ for each method under each $(α, γ)$-specification. For comparison, we also compute the EE values of the OLS estimator. We randomly replicate the experiment a total of 100 times, leading to a total of 100 EE values for each estimator. These EE values are then log-transformed and shown as boxplots in Figure 5.

From Figure 5, we immediately observe that the FIXED estimator has the smallest log(EE) values among the three MGDM methods. For the SHUFFLED method, we can find that the corresponding log(EE) values converge to that of the OLS estimator as the learning rate $α$ decreases. This finding corroborates our justifications in Section 2.6 very well. That is, the SHUFFLED estimator should enjoy the same statistical efficiency as the OLS estimator with a learning rate $α \to 0$.

The log(EE) values of the STOCHASTIC estimator also decrease with decreasing learning rate. However, there is still a clear gap between the STOCHASTIC estimator and the OLS estimator in terms of log(EE) values, even though the learning rate is sufficiently small (i.e., $\alpha = 0.005$). This result may be due to the extra random errors created by simple random sampling. These results demonstrate that the FIXED and SHUFFLED methods are more preferable in terms of statistical efficiency compared to the STOCHASTIC method.

### 3.4. More Complex Loss and Parallel Computation

The objective of this section is 3-fold. First, we study here a more complex loss function so that the power of the FIXED method can be better demonstrated. Specifically, the loss function considered here is an $ℓ_2$-penalized loss function as $L(θ) = (2N)^{-1} \sum_{i=1}^{N} (Y_i - X_i^T \hat{θ})^2 + (λ/2)\|θ\|^2$, where $λ ≥ 0$ is a regularized parameter. Following the advice of Zhang (2005),
we set \( \lambda = p/N \) in the subsequent experiments. Second, we consider the cases with a diverging feature dimension. Lastly, we study the problem of parallel computation with the help of a multi-CPU system. For all the experiments presented in this section, the covariates \( X_i \)'s are independently generated from a multivariate normal distribution \( N_p(0, \Sigma_{xx}) \), where \( \Sigma_{xx} = (\sigma_{ij})_{1 \leq i \leq J} \) and \( \sigma_{ij} = 0.95^{|i-j|} \). The random noise \( e_i \) and the true regression coefficient \( \theta_0 \) are generated in the same way as in Section 3.1. For the FIXED method, we set \( \alpha = 0.005 \) and \( \gamma = 0.9 \) unless otherwise specified. The number of minibatches is fixed to be \( M = 50 \).

**Example 1.** We conduct this experiment to numerically examine the theoretical claims made in Theorem 6. That is the numerical convergence of the FIXED method in the \( \ell_2 \)-penalized least squares problem. Specifically, we set \( N = 1 \times 10^4 \) and \( p = 100 \). We then implement the FIXED method with \( \gamma = 0, 0.5, 0.9 \) for a total of \( T = 50 \) epochs. For comparison purpose, we also apply the FIXED algorithm to the ordinary least squares problem (i.e., \( \lambda = 0 \)) with the same \( (\alpha, \gamma) \)-specification. Recall that \( \hat{\theta}^{(t,M)} \) is the estimate obtained from the \( M \)th minibatch in the \( t \)th epoch, and \( \hat{\theta}^{(t,M)} \) is the stable solution on the \( M \)th minibatch obtained by solving the corresponding linear dynamic system directly. We then record the difference \( \delta^{(i)} = \| \hat{\theta}^{(t,M)} - \hat{\theta}(M) \| \) for each epoch \( 1 \leq t \leq T \) so that the numerical convergence behavior of \( \hat{\theta}(M) \) can be gauged. We randomly replicate the experiment a total of 50 times, leading to a total of 50 \( \delta^{(i)} \) values for each epoch. The medians of these values are then plotted in Figure 6. From Figure 6, we can obtain the following findings. First, across all settings, we find that the curve corresponding to the \( \ell_2 \)-penalized least squares estimation converges toward 0 at a speed much faster than that of the ordinary squares problem. This corroborates our theoretical findings in Theorem 6 very well. That is, the \( \ell_2 \)-regularization term is helpful to improve the numerical convergence rate. Second, by comparing the three panels we find that the FIXED method with \( \gamma = 0.9 \) demonstrates the fastest convergence rate for both the two least squares problems. This further validates the common choice of the momentum parameter with \( \gamma = 0.9 \).

**Example 2.** In the second experiment, we investigate the finite sample performances of FIXED estimators with the diverging feature dimension. Specifically, we consider different sample sizes as \( N = 1 \times 10^4, 2 \times 10^4, 5 \times 10^4, 1 \times 10^5, 2.5 \times 10^5 \). With a given sample size \( N \), we set \( p = \lfloor \sqrt{N} \rfloor \) accordingly, where \( \lfloor x \rfloor \) denotes the integer part of \( x \). We then implement the FIXED method for a total of \( T = 100 \) epochs. We then compute the EE of the resulting FIXED estimator as \( \| \hat{\theta}(T,M) - \theta_0 \| \). We randomly replicate the experiment a total of 50 times, leading to a total of 50 EE values, which are then log-transformed and boxplotted in Figure 7. From Figure 7, we find that the log(EE) values of the FIXED estimator steadily drop as \( N \) and \( p \) increase. This verifies the usefulness of the FIXED method for the \( \ell_2 \)-penalized least squares problems with a diverging feature dimension.

**Example 3.** In the last experiment, we run the FIXED method in a parallel way with the help of a multi-CPU system. Specifically, we divide one given minibatch into several nonoverlapping mini-minibatches according to the number of CPUs. These mini-minibatches are then assigned to different CPUs to compute the gradients in a parallel way. In this example, we consider the cases with a diverging feature dimension. Lastly, we study the problem of parallel computation with the help of a multi-CPU system. For all the experiments presented in this section, the covariates \( X_i \)'s are independently generated from a multivariate normal distribution \( N_p(0, \Sigma_{xx}) \), where \( \Sigma_{xx} = (\sigma_{ij})_{1 \leq i \leq J} \) and \( \sigma_{ij} = 0.95^{|i-j|} \). The random noise \( e_i \) and the true regression coefficient \( \theta_0 \) are generated in the same way as in Section 3.1. For the FIXED method, we set \( \alpha = 0.005 \) and \( \gamma = 0.9 \) unless otherwise specified. The number of minibatches is fixed to be \( M = 50 \).

**Example 4.** In the second experiment, we investigate the finite sample performances of FIXED estimators with the diverging feature dimension. Specifically, we consider different sample sizes as \( N = 1 \times 10^4, 2 \times 10^4, 5 \times 10^4, 1 \times 10^5, 2.5 \times 10^5 \). With a given sample size \( N \), we set \( p = \lfloor \sqrt{N} \rfloor \) accordingly, where \( \lfloor x \rfloor \) denotes the integer part of \( x \). We then implement the FIXED method for a total of \( T = 100 \) epochs. We then compute the EE of the resulting FIXED estimator as \( \| \hat{\theta}(T,M) - \theta_0 \| \). We randomly replicate the experiment a total of 50 times, leading to a total of 50 EE values, which are then log-transformed and boxplotted in Figure 7. From Figure 7, we find that the log(EE) values of the FIXED estimator steadily drop as \( N \) and \( p \) increase. This verifies the usefulness of the FIXED method for the \( \ell_2 \)-penalized least squares problems with a diverging feature dimension.
of $T = 50$ epochs, and record the time consumed by each epoch. We randomly replicate the experiment a total of 50 times, leading to a total of 50 computation times for every given epoch. They are then boxplotted in Figure 8. From Figure 8, we obtain the following interesting findings. First, we find that the computation time steadily decreases as the number of CPUs increases from 1 to 8. Specifically, the 2-CPU system reduces the computation time by nearly half compared to the single-CPU system. However, the computation time of the 8-CPU system is only slightly less than that of the 4-CPU system. This is partly because communication costs become higher as the number of CPUs increases. Second, we find that the computation time increases when the number of CPUs increases from 8 to 16. This indicates that the communication time becomes the dominant part of the total computation time. This is because the minibatch size allocated to each CPU is merely $n/16 \approx 312$, making the pure computation time on each CPU much less than the communication time. These results suggest that the number of CPUs should be appropriately selected according to the minibatch size so that the least computation time can be achieved.

### 3.5. Application to Airline Data

To demonstrate the performance of the MGDM methods on a real-world dataset, we consider here the U.S. Airline Dataset. The whole dataset is available for download at http://stat-computing.org/dataexpo/2009. It contains the flight arrival and departure details for all commercial flights within the USA from 1987 to 2008. We use the data from 2008, and aim to predict the delay time in the arrival of a flight given other flight information. Each record of the data contains the arrival delay, information of the carrier, origin and destination. The detailed variable information is described in Table 1. The six continuous variables are standardized to be mean 0 and variance 1, and the five categorical variables are converted to dummy variables with appropriate dimensions. Finally, a total of 126 predictors and one response variable are used in the linear regression model, and the total sample size is over 2.5 million observations.

In this experiment, we set the minibatch size as $n = 1000$, momentum parameter as $\gamma = 0.9$ and consider four different learning rates $\alpha = 0.4, 0.2, 0.1, 0.05$. For comparison purpose, we first randomly choose a subsample of size $N = 1 \times 10^5$ from the whole sample, and then run each MGDM algorithm with $M = N/n = 200$ for a total of $T = 200$ epochs. Next, we compute the estimation error (EE) as $\|\hat{\theta}(T,M) - \theta_{\text{obs}}\|$ for each method under each $(\alpha, \gamma)$-specification, where $\theta_{\text{obs}}$ is the OLS estimator of the regression coefficient based on the whole sample. We replicate the experiment through 50 times random subsampling, leading to a total of 50 EE values for each estimator. These EE values are then log-transformed and shown as boxplots in Figure 9. From Figure 9, we can see a similar pattern as shown in Figure 5. That is, as the learning rate $\alpha$ decreases, the log(EE) values of all three methods decrease. In addition, the FIXED and SHUFFLED methods show a competitive performance. The STOCHASTIC method has the worst performance under all three $\alpha$ specifications.

### 4. Concluding Remarks

In this article, we study the theoretical properties of MGDM algorithms for linear regression models. We focus mainly on two random partition based MGDM methods, that is, fixed minibatch-based GDM (FIXED) and shuffled minibatch-based GDM (SHUFFLED) methods. We first investigate the FIXED method by introducing a linear dynamical system. Then, we provide nearly sufficient and necessary conditions for numerical convergence. Furthermore, the numerical convergence rate of FIXED algorithm is investigated. This analysis leads to the theoretically optimal specification of the tuning parameters.

![Figure 8. Boxplots of the computation times consumed by each epoch by using different numbers of CPUs. We run the FIXED algorithm for a total of 50 epochs, and vary the number of CPUs from 1 to 16.](image)

### Table 1. Variable description for the U.S. airline data from 2008.

| Variable       | Description                                      | Variable used in the model          |
|---------------|--------------------------------------------------|-------------------------------------|
| ArrDelay      | How long was the delay in the arrival of the flight | Used as the numerical response variable |
| Month         | Which month of the year                         | Converted to 11 dummies             |
| DayofMonth    | Which day of the month                          | Used as numerical variable          |
| DayOfWeek     | Which day of the week                           | Converted to 6 dummies              |
| DepTime       | Actual departure time                           | Used as numerical variable          |
| CRSDepTime    | Scheduled departure time                         | Used as numerical variable          |
| CRSArrTime    | Scheduled arrival time                           | Used as numerical variable          |
| Distance      | Distance between the origin and destination in miles | Used as numerical variable          |
| UniqueCarrier | Flight carrier code for 29 carriers              | Top 7 carriers converted to 6 dummies |
| Origin        | Departing origin                                | Top 50 origin cities converted to 49 dummies |
| Dest          | Destination of the flight                        | Top 50 origin cities converted to 49 dummies |

*NOTE: Numerical variables are standardized to have mean 0 and variance 1.*
Theoretical findings reveal that the extra momentum term could greatly speed up numerical convergence rate compared to a standard gradient descent method without momentum. This is particularly true for problems with large condition numbers. Subsequently, the statistical properties of the resulting FIXED estimator are carefully investigated and the asymptotic normality is further established. We then find that with a sufficiently small learning rate, the MGDM estimators (both fixed and shuffled) can share the same statistical efficiency as the global estimator. Finally, these theoretical results are fully validated through extensive numerical studies.

To conclude this article, we discuss here several interesting topics for future study. First, we examine the FIXED algorithm for the quadratic objective function, due to its analytical simplicity. Of note, many learning problems have non-quadratic objection functions, such as logistic regression and quantile regression models. However, it seems to us that the theoretical properties of either the fixed or shuffled minibatch-based GDM estimators for these problems remain unknown. This is a research topic of great interest. Second, modern statistical inference often involves ultrahigh-dimensional problems. It is valuable to extend our results on the MGDM methods to the models with $p > N$. Third, online estimation for streaming data is also an important research topic (Xie et al. 2019). In fact, our MGDM methods can be readily applied to this scenario if we give up the multiple-pass idea and keep one pass only. However, whether the resulting estimator remains to be consistent or statistically efficient requires future in-depth study. Last, there are many momentum-related methods (e.g., Nesterov’s momentum method, RMSprop and ADAM) that are widely used in practice. Therefore, it is also very important to investigate their theoretical properties under a fixed or shuffled minibatch setup.

Supplementary Materials
Proofs of theorems and extended numerical experiments. (pdf file)

Acknowledgments
The authors thank the editor, associate editor, and two referees for their insightful comments that have led to significant improvement of this article.

Disclosure Statement
The authors report there are no competing interests to declare.

Funding
Xuening Zhu’s research is supported by the National Natural Science Foundation of China (nos. 72222009, 71991472, 11901105, U1811461). Guodong Li’s research is supported by the Hong Kong RGC (grant no. 17305319). Riquan Zhang’s research is supported by the National Natural Science Foundation of China (nos. 11971171, 11831008, 12171310), and the Basic Research Project of Shanghai Science and Technology Commission (22JC1400800). Hansheng Wang’s research is partially supported by National Natural Science Foundation of China (nos. 12271012, 11831008) and also partially supported by the Open Research Fund of Key Laboratory of Advanced Theory and Application in Statistics and Data Science (KLATASDS-MOE-ECNU-KLATASDS2101).

ORCID
Xuening Zhu https://orcid.org/0000-0001-5824-5279
Guodong Li https://orcid.org/0000-0003-3137-8471
Hansheng Wang https://orcid.org/0000-0003-2386-0209

References
Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016), “Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” arXiv preprint arXiv:1603.04467. [1084]
Akter, S., and Wamba, S. F. (2016), “Big Data Analytics in e-commerce: A Systematic Review and Agenda for Future Research,” Electronic Markets, 26, 173–194. [1083]
Anava, O., Hazan, E., Mannor, S., and Shamir, O. (2013), “Online Learning for Time Series Prediction,” in Proceedings of the 26th Annual Conference on Learning Theory, volume 30 of Proceedings of Machine Learning Research, eds. S. Shalev-Shwartz and I. Steinwart, pp. 172–184, Princeton, NJ, USA. PMLR. [1084]
Assran, M., and Rabbat, M. (2020), “On the Convergence of Nesterov’s Accelerated Gradient Method in Stochastic Settings,” arXiv preprint arXiv:2002.12414. [1084]
Battey, H., Fan, J., Liu, H., Lu, J., and Zhu, Z. (2018), “Distributed Testing and Estimation Under Sparse High Dimensional Models,” The Annals of Statistics, 46, 1352–1382. [1083]
Beck, A., and Teboulle, M. (2009), “A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems,” SIAM Journal on Imaging Sciences, 2, 183–202. [1084]
Bickel, P. J., and Levina, E. (2008), “Regularized Estimation of Large Covariance Matrices,” The Annals of Statistics, 36, 199–227. [1089]
Bottou, L., Curtis, F. E., and Nocedal, J. (2018), “Optimization Methods for Large-Scale Machine Learning,” Siam Review, 60, 223–311. [1084]
Chang, X., Lin, S.-B., and Wang, Y. (2017), “Divide and Conquer Local Average Regression,” Electronic Journal of Statistics, 11, 1326–1350. [1083]
Chang, X., Lin, S.-B., and Zhou, D.-X. (2017), “Distributed Semi-Supervised Learning with Kernel Ridge Regression,” Journal of Machine Learning Research, 18, 1493–1514. [1083]
Chen, X., Lee, J. D., Li, H., and Yang, Y. (in press), “Distributed Estimation for Principal Component Analysis: An Enlarged Eigenspace Analysis,” Journal of the American Statistical Association. [1083]
Chen, X., Lee, J. D., Tong, X. T., and Zhang, Y. (2020a), “Statistical Inference for Model Parameters in Stochastic Gradient Descent,” The Annals of Statistics, 48, 251–273. [1084]
Chen, X., Liu, W., Mao, X., and Yang, Z. (2020b), “Distributed High-Dimensional Regression under a Quantile Loss Function,” Journal of Machine Learning Research, 21, 1–43. [1083]
Chen, X., Liu, W., and Zhang, Y. (2019), “Quantile Regression under Memory Constraint,” *The Annals of Statistics*, 47, 3244–3273. [1083]

Chen, X., and Xie, M.-g. (2014), “A Split-and-Conquer Approach for Analysis of Extraordinarily Large Data,” *Statistica Sinica*, 24, 1655–1684. [1083]

Deng, J., Dong, W., Socher, R., Li, J.-l., Li, K., and Fei-Fei, L. (2009), “Imagenet: A Large-Scale Hierarchical Image Database,” in 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. [1083]

Fan, J., Li, R., Zhang, C.-H., and Zou, H. (2020), Statistical Foundations of Data Science, Boca Raton, FL: Chapman and Hall/CRC. [1083]

Fan, J., Wang, D., Wang, K., and Zhu, Z. (2019), “Distributed Estimation of Principal Eigenspaces,” *The Annals of Statistics*, 47, 3009–3031. [1083]

Feng, Y., Xu, J., and Yang, L. (2018), “Online Bootstrap Confidence Intervals for the Stochastic Gradient Descent Estimator,” *The Journal of Machine Learning Research*, 19, 3053–3073. [1084]

Girman, I., Lang, H., Zhang, P., and Xiao, L. (2019), “Understanding the Role of Momentum in Stochastic Gradient Methods,” in *Advances in Neural Information Processing Systems* (Vol. 32). [1084]

Goodfellow, I., Bengio, Y., and Courville, A. (2016), *Deep Learning*, Cambridge, MA: MIT Press. [1084]

He, K., Zhang, X., Ren, S., and Sun, J. (2016), “Deep Residual Learning for Image Recognition,” in proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [1084,1088]

Kidambi, R., Netrapalli, P., Jain, P., and Kakade, S. (2018), “On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization,” in *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9. [1083]

Kingma, D. P., and Ba, J. (2014), “Adam: A Method for Stochastic Optimization,” arXiv preprint arXiv:1412.6980. [1084]

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), “Imagenet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems* (Vol. 25), pp. 1097–1105. [1084]

Li, M., Zhang, T., Chen, Y., and Smola, A. J. (2014), “Efficient Mini-Batch Training for Stochastic Optimization,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 661–670, New York, NY, USA, Association for Computing Machinery. [1084]

Li, X., Liang, J., Chang, X., and Zhang, Z. (2022), “Statistical Estimation and Online Inference via Local sgd,” in *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pp. 1613–1661. PMLR. [1084]

Liu, Y., Gao, Y., and Yin, W. (2020), “An Improved Analysis of Stochastic Gradient Descent with Momentum,” arXiv preprint arXiv:2007.07989. [1084]

Loizou, N., and Richtárik, P. (2020), “Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods,” *Computational Optimization and Applications*, 77, 653–710. [1084]

Luo, L., and Song, P. X.-K. (2020), “Renewable Estimation and Incremental Inference in Generalized Linear Models with Streaming Datasets,” *Journal of the Royal Statistical Society, Series B*, 82, 69–97. [1084]

Ma, P., Mahoney, M. W., and Yu, B. (2015), “A Statistical Perspective on Algorithmic Leveraging,” *Journal of Machine Learning Research*, 16, 861–911. [1084]

Ma, P., Zhang, X., Xing, X., Ma, J., and Mahoney, M. (2020), “Asymptotic Analysis of Sampling Estimators for Randomized Numerical Linear Algebra Algorithms,” in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, eds. S. Chiappa and R. Calandra, pp. 1026–1035. PMLR. [1086]

Moulines, E., and Bach, F. (2011), “Non-asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning,” in *Advances in Neural Information Processing Systems* (Vol. 24), eds. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Curran Associates, Inc. [1084]

Nesterov, Y. (2018), *Lectures on Convex Optimization* (Vol. 137), Springer. [1085,1087]

Nesterov, Y. E. (1983), “A Method for Solving the Convex Programming Problem with Convergence Rate Ο(1/k²),” *Doklady Akademii Nauk SSSR*, 269, 543–547. [1084]