UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

On the Reproducibility of "Learning Mixtures of Linear Dynamical Systems"

Anonymous[1]

[1]University of Michigan, Ann Arbor

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
03 February 2023

Published
–

DOI
–

Reproducibility Summary

We investigate the methodology in learning a mixture of multiple linear dynamical systems from unlabeled short trajectories proposed by Chen and Poor [1]. Our objectives are mainly to (1) validate their experimental results by reproducing their figures, (2) explore the robustness of their algorithm by experimenting on real-world datasets, and (3) analyze the behavior of the proposed algorithm under different hyperparameters. We propose to tackle these objectives by re-implementing their algorithm to test on synthetic data and the MotionSense dataset [2].

Scope of Reproducibility – This paper proposes a two-stage meta-algorithm that is guaranteed to recover the ground-truth LDS model up to an error $\mathcal{O}(\sqrt{d/T})$, where T is the total sample size. We validate these theoretical guarantees on synthetic data and test the limitations of the algorithm on a real-world dataset.

Methodology – We re-implemented all stages of the algorithm in Python, using the author's MATLAB code as a reference. We use this implementation to reproduce the figures in the paper as well generate new ones.

Results – We were able to reproduce all of the figures for the synthetic data within 0.03 error (which is small in context). Our results for the MotionSense dataset is also analagous to that of the ones shown in the paper. However, we identified a bug in their implementation, which we have informed the authors with to find a resolution. Along with our contributions, we present how this bug affects the reproduced results.

What was easy – The author's code was fairly easy to run and understand.

What was difficult – The implementation on Python was not very straightforward. There were many careful modifications we had to make (e.g. avoid using for-loops by vectorization). The reproduction of some of the figures were also difficult, as the descriptions in the paper did not always align with the authors' MATLAB code.

Communication with original authors – We contacted the corresponding author to ask for their code and at times we thought there were errors in the way they generated their figures.

Correspondence should be addressed to ()
The authors have declared that no competing interests exists.
Code is available at .

1 Introduction

Linear dynamical systems (LDSs) are a class of models that serve to capture the time evolution of a system of many real-world applications. In the classical LDS setting, we have access to a finite amount of sample trajectories that are assumed to be generated by an LDS. If we can accurately estimate the parameters of the LDS, then we can predict how the system will behave at a future time instance. That is, we wish to learn the latent matrices $\mathbf{A} \in \mathbb{R}^{d\times d}$ and $\mathbf{W} \in \mathbb{R}^{d\times d}$ given sample trajectories $\{\mathbf{x}_t\}_{0\le t\le T}$, where each trajectory is assumed to be generated by the LDS

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{w}_t. \tag{1}$$

We assume that the noise $\mathbf{w}_t$ is i.i.d. with $\mathbb{E}[\mathbf{w}_t] = 0$ and $\mathrm{Cov}[\mathbf{w}_t] = \mathbf{W} \succ 0$.
Chen and Poor [1] considers an extension of the classical LDS setting to learning multiple LDSs from a mixture of unlabeled sample trajectories. In this setting, we assume that there are K different LDSs represented by $\{\mathbf{A}^{(k)}, \mathbf{W}^{(k)}\}_{1\le k\le K}$, where $\mathbf{A}^{(k)} \in \mathbb{R}^{d\times d}$ and $\mathbf{W}^{(k)} \in \mathbb{R}^{d\times d}$. Given access to M short trajectories $\{\mathbf{X}_m\}_{1\le m\le M}$, where $\mathbf{X}_m = \{\mathbf{x}_{m,t}\}_{0\le t\le T_m}$ and

$$\mathbf{x}_{m,t+1} = \mathbf{A}^{(k)}\mathbf{x}_{m,t} + \mathbf{w}^{(k)}_{m,t}, \tag{2}$$

the objective is to estimate $\{\mathbf{A}^{(k)}, \mathbf{W}^{(k)}\}_{1\le k\le K}$. We assume that there is an associated label to each pair $(\mathbf{A}^{(k)}, \mathbf{W}^{(k)})$ and that $(\mathbf{A}^{(k)}, \mathbf{W}^{(k)}) \neq (\mathbf{A}^{(l)}, \mathbf{W}^{(l)})$ for $k \neq l$. There are three main sources of challenges in this problem: (1) the label k of each trajectory $\mathbf{X}_m$ is a latent variable, (2) the length of each trajectory is "short", making it infeasible to estimate the system from a single trajectory, and (3) there is a temporal dependence on the trajectories. To solve these issues, Chen and Poor propose a two-stage meta-algorithm that involves dimensionality reduction, clustering, classification, and model estimation. The proposed algorithm is guaranteed to recover the ground-truth system up to an error $\mathcal{O}(\sqrt{d/T})$, where T is the total sample size $\sum_{m=1}^{M} T_m$.

2 Overview of the Proposed Algorithm

We dedicate this section to better illustrate the main ideas behind Chen and Poor's proposed algorithm. We only discuss the high-level ideas and refer the reader to the original paper for more details. Recall that we are given access to M short trajectories $\{\mathbf{X}_m\}_{1\le m\le M}$, where each $\mathbf{X}_m$ is generated by an LDS with some label k. Intuitively, one can think of each trajectory $\mathbf{X}_m$ as an activity – for example, $\{\mathbf{X}_m\}_{1\le m\le M}$ can be a compilation of a person running and walking, which would correspond to $K = 2$. The proposed algorithm uses these trajectories to estimate the LDS parameters with four sub-algorithms: subspace estimation, clustering, model estimation, and classification. To preserve independence throughout each sub-routine of the algorithm, the M sample trajectories are divided into three disjoint subsets $\mathcal{M}_{\text{subspace}}$, $\mathcal{M}_{\text{clustering}}$ and $\mathcal{M}_{\text{classification}}$ satisfying $M = \mathcal{M}_{\text{subspace}} \cup \mathcal{M}_{\text{clustering}} \cup \mathcal{M}_{\text{classification}}$. We assume that all of the trajectories in each subset have the same length, namely

$$T_m = \begin{cases} T_{\text{subspace}} & \text{if } m \in \mathcal{M}_{\text{subspace}} \\ T_{\text{clustering}} & \text{if } m \in \mathcal{M}_{\text{clustering}} \\ T_{\text{classification}} & \text{if } m \in \mathcal{M}_{\text{classification}}. \end{cases}$$

Let us first consider why clustering is an important step in this algorithm.
Clustering: One main challenge of this problem is that we do not know the underlying label k of each trajectory $\mathbf{X}_m$. To solve this issue, we can cluster the sample trajectories into K clusters such that the trajectories in each cluster are primarily generated by the

same LDS. Then, we can use the trajectories in each respective cluster to estimate the LDS model parameters. To perform clustering, the first important observation is that the order-0 and order-1 stationary autocovariance matrices, $\mathbf{\Gamma}^{(k)}$ and $\mathbf{Y}^{(k)}$, respectively, completely characterizes the LDS model $(\mathbf{A}^{(k)}, \mathbf{W}^{(k)})$. We estimate these autocovariance matrices as using samples $\mathbf{x}_{m,t}$ and $\mathbf{x}_{m,t+1}$ from each $\mathbf{X}_m$. Using these estimates, denoted as $\hat{\mathbf{\Gamma}}^{(k)}$ and $\hat{\mathbf{Y}}^{(k)}$, we perform clustering by computing the quantity

$$\|\hat{\mathbf{\Gamma}}^{(k)} - \hat{\mathbf{\Gamma}}^{(l)}\|_{\mathsf{F}}^2 + \|\hat{\mathbf{Y}}^{(k)} - \hat{\mathbf{Y}}^{(l)}\|_{\mathsf{F}}^2. \tag{3}$$

This quantity can be viewed as a similarity score, where the score will be smaller if two pairs of sample trajectories come from the same LDS. Then, using some appropriately chosen threshold τ, we can declare $k \neq l$ if the score exceeds τ. We store these thresholded scores as a matrix $\mathbf{S}$, which we call the similarity matrix. Notice that the similarity matrix is a square matrix with binary entries, where the $\mathbf{S}_{i,j}$ entry takes 0 if the similarity score exceeds τ and 1 otherwise, i.e. $\mathbf{X}_i$ and $\mathbf{X}_j$ come from the same LDS model. Looking at Equation (3), notice that even if two pairs of trajectories are "similar", since the score is computed in the Frobenius norm, it can blow up as d increases. This is where subspace estimation comes into play – we can reduce the dimensions of $\mathbf{\Gamma}$ and $\mathbf{Y}$ to effectively reduce the variance of Equation (3).

Subspace Estimation: For simplicity, consider the right-hand side of Equation (3). Notice that

$$\|\mathbf{Y}^{(k)} - \mathbf{Y}^{(l)}\|_{\mathsf{F}}^2 = \sum_{i=1}^{d} \|(\mathbf{Y}^{(k)})_i - (\mathbf{Y}^{(l)})_i\|_2^2 \approx \sum_{i=1}^{d} \|\mathbf{U}_i^\top((\mathbf{Y}^{(k)})_i - (\mathbf{Y}^{(l)})_i)\|_2^2,$$

if each subspace $\mathbf{U}_i$ is sufficiently close to $\mathbf{U}_i^* = \text{span}\{(\mathbf{Y}^{(j)})_i, 1 \leq j \leq K\}$, where $\mathbf{U}_i$ is i-th row of $\mathbf{U}$. Using the same argument for $\mathbf{\Gamma}$, we can effectively reduce the variance of the similarity score by projecting with $\mathbf{U}_i$ (and respectively $\mathbf{V}_i$ for $\mathbf{\Gamma}$) as long as we can faithfully estimate $\mathbf{U}_i^*$ and $\mathbf{V}_i^*$.

Model Estimation: Once we have K clusters of sample trajectories from the previous step, we can estimate the model parameters $\{(\hat{\mathbf{A}}^{(k)}, \hat{\mathbf{W}}^{(k)})\}_{1 \leq k \leq K}$.

Classification: Upon estimating the model parameters $\{(\hat{\mathbf{A}}^{(k)}, \hat{\mathbf{W}}^{(k)})\}_{1 \leq k \leq K}$, we can infer the labels of other trajectories $\mathbf{X}_m$ by computing a quantity that is a function of $\hat{\mathbf{A}}^{(k)}$ and $\hat{\mathbf{W}}^{(k)}$. We assign the label with the smallest aforementioned quantity, and add it to the cluster with the corresponding label.

Lastly, recall that this algorithm is a two-stage algorithm – the latter two steps are repeated for refinement upon coarse estimation.

3 Scope of Reproducibility

Throughout the paper, the authors provide empirical evidence suggesting the efficiency of each sub-routine of their overall algorithm. Our first reproducibility goal is to verify the following claims of each sub-routine:

- **Claim 1:** Note that we can perform the clustering step either with or without subspace estimation. The authors report that the accuracy of the clustering algorithm [1, Figure 3 (a)] is noticeably higher when we use subspace estimation, justifying the efficacy of their algorithm.
- **Claim 2:** The authors report that the accuracy of the classification algorithm increases with the trajectory length $T_{\text{classification}}$, achieving accuracies near perfection [1, Figure 3 (b)].
- **Claim 3:** The authors report that the model estimation error is approximately $\mathcal{O}(\sqrt{Kd/T})$, where T is the total number of observations (i.e. the sum of the lengths of all trajectories) [1, Figure 2].

- **Claim 4:** Upon pre-processing the MotionSense dataset as two activities, walking and jogging, the authors report that the algorithm achieves an estimated distance matrix between trajectories that is close to the true distance matrix [1, Figure 4].

Our second reproducibility goal is to extend their results by (1) more extensively testing the algorithm on the MotionSense dataset and (2) observing the performance of the algorithm with varying hyperparameters. More specifically, we make the following contributions:

- **Contribution 1:** Upon implementing the algorithm in Python, we found that the authors' MATLAB code had a substantial bug in the subspace estimation step. The subspace estimation step involves computing the top-K eigenspace of matrices $\mathbf{H}_i$ and $\mathbf{G}_i$, which resulted in the subspaces $\mathbf{V}_i$ and $\mathbf{U}_i$, respectively. The matrices $\mathbf{H}_i$ and $\mathbf{G}_i$ are supposed to be positive semi-definite (PSD) in expectation, but not necessarily PSD in practice. The authors' code used SVD and took the singular vectors corresponding to the top-K singular values, which treated the negative eigenvalues as positive singular values! We informed the authors and have come to a resolution – we discuss these issues in more detail.

- **Contribution 2:** Recall that in the clustering step, there is a thresholding value τ that determines the similarity between two pairs of autocovariance matrices. We present an approach for choosing this value, which involves plotting a histogram of the computed similarity scores.

- **Contribution 3:** Previously, we briefly mentioned that we assumed $(\mathbf{A}^{(k)}, \mathbf{W}^{(k)}) \neq (\mathbf{A}^{(l)}, \mathbf{W}^{(l)})$ for $k \neq l$. However, what if two pairs of $(\mathbf{A}^{(k)}, \mathbf{W}^{(k)})$ were "close" to each other? If the model parameters of two different classes were close, then we hypothesize that the clustering sub-algorithm would have a harder time distinguishing the parameters of two different classes, degrading the overall performance. We test this hypothesis by varying the model separation parameter $\Delta^2_{\mathbf{A},\mathbf{W}}$, where

$$\|\mathbf{A}^{(k)} - \mathbf{A}^{(l)}\|_{\mathrm{F}}^2 + \|\mathbf{W}^{(k)} - \mathbf{W}^{(l)}\|_{\mathrm{F}}^2 \geq \Delta^2_{\mathbf{A},\mathbf{W}}, \tag{4}$$

  (and similarly $\Delta^2_{\Gamma,\mathbf{Y}}$ if we replace $\mathbf{A}$ and $\mathbf{W}$ for Γ and $\mathbf{Y}$).

- **Contribution 4:** Compared to a data driven approach, we observe the performance changes as we project onto random subspaces in the clustering step. Though we hypothesize that the performance will decrease, if it does not decrease too much, it would provide another (perhaps faster) way of doing dimensionality reduction.

4 Methodology

In this section, we discuss how we generated synthetic data and pre-processed the MotionSense dataset. We also present the hyperparameters that require tuning in the algorithm, and how we went about choosing these values.

4.1 Datasets

Recall that for both our contributions and the reproduced results, we use two datasets: synthetic data and the MotionSense dataset.

Synthetic Data — We describe the procedure in generating the synthetic dataset. Each synthetic experiment generally uses slightly different models, so we defer the details to Section 4.3. The first step in generating the synthetic data is to randomly construct the

ground truth LDS model parameters $\{\mathbf{A}^{(k)}, \mathbf{W}^{(k)}\}_{1\leq k\leq K}$. These parameters were constructed by

$$\begin{cases}\mathbf{A}^{(k)} = \Delta_{\mathrm{A,W}}\mathbf{R}^{(k)} \\ \mathbf{W}^{(k)} = \mathbf{U}^{(k)}\mathbf{\Lambda}^{(k)}\mathbf{U}^{(k)\top},\end{cases}$$

where $\mathbf{R}^{(k)} \in \mathbb{R}^{d\times d}$ and $\mathbf{U}^{(k)} \in \mathbb{R}^{d\times d}$ that varied throughout each experiment. Using these models, we perform the following steps to generate trajectories $\{\mathbf{X}_m\}_{1\leq m\leq M}$:

1. Generate a list of trajectory lengths $\{T_m\}_{1\leq m\leq M}$.
2. Assign labels $1, \ldots, K$ uniformly at random to each of the trajectories
3. Generate example trajectory X_m with label $k \in \{1, \ldots, K\}$ by initializing $x_{m,0} = 0$ and iteratively computing $\mathbf{x}_{m,t+1} = \mathbf{A}^{(k)}\mathbf{x}_{m,t} + \mathbf{w}_{m,t}^{(k)}$ where $\mathbf{w}_{m,t}^{(k)}$ is drawn from a Gaussian distribution $\mathcal{N}(0, \mathbf{W}^{(k)})$

MotionSense Data — The MotionSense is a time-series dataset generated by 24 participants performing 6 activities under the same environment and conditions. Following the authors, we pre-processed this dataset from one participant to include only two of the activities, jogging and walking. There are a total of 24 examples (one for each participant) each with dimension $d = 12$. The examples were obtained by breaking the $4800 \times d$ jogging data (and walking data) into blocks of 400 consecutive rows to obtain 12 examples with trajectory length $T = 400$.

4.2 Hyperparameters

The only hyperparameter that needed tuning was the separation parameter τ used in the clustering algorithm. For the synthetic datasets, this value was often fixed by the authors and did not require further tuning. For the MotionSense dataset, we chose this value by plotting a histogram of the values in the similarity matrix in clustering (before thresholding). Then, we picked a τ which separated the first peak from the next peak in the histogram (see Section 5.2 for a detailed explanation).

4.3 Experimental Setup and Code

Here, we discuss the experimental setups for both reproducing the figures and our contributions in more detail.

Setup for Results Reproducing the Original Paper —

- **Claim 1:** To reproduce Claim 1, we fixed $d = 40$ and $K = 2$ with varying $T_{\text{clustering}} \in [0, 60]$ and $\mathcal{M}_{\text{clustering}} = 5d$. The values of $T_{\text{clustering}}$ varied as the objective of this experiment was to observe the performance of the clustering algorithm for different values of $T_{\text{clustering}}$. We generated the LDS model $\mathbf{A}^{(k)}$ with $\Delta_{\mathrm{A,W}} = (0.5 + (-1)^{k-1}\delta)$ with $\delta = 0.12$ and $\mathbf{W}^{(k)} = \mathbf{I}_d$. Using these models, we generated $M = 35d$ trajectories. We ran the clustering algorithm over 30 independent trials and averaged the clustering error. Following the authors' MATLAB code, the data was not separated into the subspace estimation and the clustering portions.
- **Claim 2:** For Claim 2, we generated the LDS model parameters in the same way described above. We used $M = 10d + 7920$ trajectories of dimension $d = 40$ and $K = 2$ labels. We partitioned the data into $10d$ trajectories for subspace estimation and clustering (which was again not further separated), and 7920 trajectories for classification. The value of $T_{\text{classification}}$ took values from $T_{\text{classification}} \in [4, 50]$, and $T_{\text{clustering}} = 30$. We obtain a coarse estimate of the model using the dataset corresponding to $\mathcal{M}_{\text{clustering}}$ and ran the classification algorithm on varying $T_{\text{classification}}$.

- **Claim 3:** For Claim 3, we ran 12 independent trials. In each trial, we generated a dataset of $M = 5040d$ trajectories of dimension $d = 80$ and $K = 4$ labels. We partitioned the M trajectories into $\mathcal{M}_{\text{subspace}} = 30d$, $\mathcal{M}_{\text{clustering}} = 20d$, $\mathcal{M}_{\text{classification}} = 5000d$, with a total T of $T_{\text{subspace}} = 20$, $T_{\text{clustering}} = 20$, $T_{\text{classification}} = 5$. The ground truth LDS model parameter $\mathbf{A}^{(k)}$ was generated with $\Delta_{\mathrm{A,W}} = 0.5$ and $\mathbf{W}^{(k)} = \mathbf{U}^{(k)}\mathbf{\Lambda}^{(k)}\mathbf{U}^{(k)\top}$, where $\mathbf{U}^{(k)} \in \mathbb{R}^{d\times d}$ are random orthogonal matrices and the diagonal entries of $\mathbf{\Lambda}^{(k)}$ are drawn independently from a uniform distribution on $[0, 1]$. We computed the model estimation error using

$$\begin{cases} \mathrm{A}: & \max_k \|\hat{\mathbf{A}}^{(k)} - \mathbf{A}^{\pi(k)}\|_2 \\ \mathrm{W}: & \max_k \|\hat{\mathbf{W}}^{(k)} - \mathbf{W}^{\pi(k)}\|_2 / \|\mathbf{W}^{\pi(k)}\|_2, \end{cases}$$

  where $\hat{\mathbf{A}}$ and $\hat{\mathbf{W}}$ are the estimated models and $\pi(k)$ is a permutation function that allows us to distinguish the classes upon clustering.

- **Claim 4:** The experimental setup for the MotionSense dataset was largely the preprocessing steps as described in Section 4.1. We ran the full algorithm on the pre-processed dataset to obtain the adjacency graph from the spectral clustering algorithm.

Setup for Results Beyond the Original Paper —

- **Contribution 1:** We used the same setup as described for Claim 1. To identify the bug, we randomly chose an iteration of the algorithm and plotted the singular values and eigenvalues of associated matrices.

- **Contribution 2:** For choosing the threshold τ, we plotted a histogram from the values given in the similarity matrix before thresholding. This is easily done using built-in packages in Python. To demonstrate the histogram plots, we used the MotionSense dataset with the same preprocessing steps as previously described.

- **Contribution 3:** We used the same setup as described in Claim 1, except we fixed with $T_{\text{clustering}} = 10, 40$ (for two different figures) and varied $\Delta_{\mathrm{A,W}} \in [0.001, 0.25]$. We averaged the results over 15 indepedent trials.

- **Contribution 4:** We followed the same setup as in Claim 1 and generated random subspaces by orthonormalizing a randomly generated matrix of appropriate dimensions. We varied $T_{\text{clustering}} \in [10, 90]$ and chose the model separation parameters as $\Delta_{\mathrm{A,W}} = 0.12, 0.30$. We averaged the results over 30 indepedent trials.

5 Results

In this section, we present our reproduced results as well as our contributions. We follow our results with an analysis of what we expected and what we obtained.

5.1 Results Reproducing Original Paper

Claims 1 and 2 — To validate Claims 1 and 2, we reproduced [1, Figure 3] that showed that the classification error reduced rapidly proportional to the trajectory length, as well as that the subspace estimation technique substantially reduced the clustering error. The reproduction and comparison of the figures is shown in Figures 1 and 2. We would like to note that the clustering plot in Figure 1 does not exactly match that of the plot in [1, Figure 3 (b)]. This is due to an improved spectral clustering code that is being used by the code that the authors provided, as well as what we have also implemented.

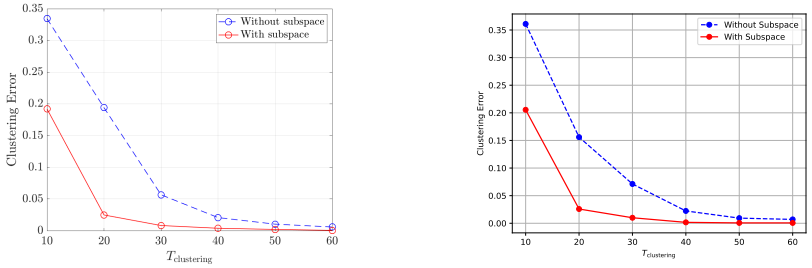

Figure 1. Clustering error for varying values of the sample size $T_{\text{clustering}}$. Left: Original figure generated from MATLAB. Right: Reproduced figure generated from Python.

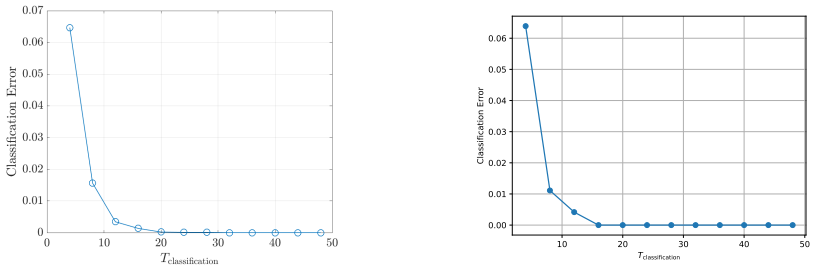

Figure 2. Classification error for varying values of the sample size $T_{\text{classification}}$. Left: Original figure generated from MATLAB. Right: Reproduced figure generated from Python.

Claim 3 — To validate Claim 3, we reproduced Figure 2 of the paper that stated that the model estimation error with respect to the spectral norm reduced at a rate $\mathcal{O}(\sqrt{Kd/T})$. Upon reproducing this figure, we indeed verify that this trend is true – see Figure 3.

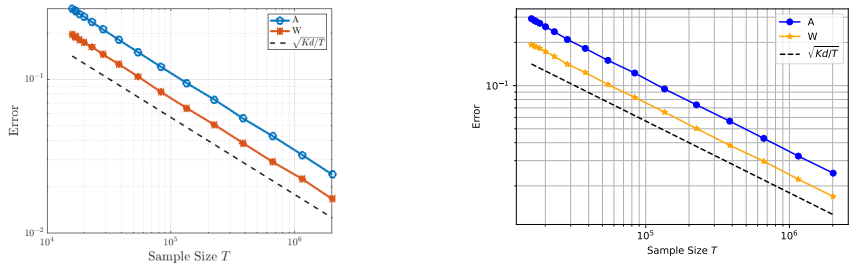

Figure 3. Model estimation error for varying values of the sample size T. Left: Original figure generated from MATLAB. Right: Reproduced figure generated from Python.

Claim 4 — Regarding the MotionSense dataset, we reproduced the distance matrix estimate obtained by the clustering algorithm as shown in Figure 4 of the paper. We can observe the reproduced figure in Figure 4. The block structure of the matrix tells us that the algorithm will successfully cluster the MotionSense data if a good threshold is chosen, thereby validating Claim 4.

5.2 Results Beyond Original Paper

Contribution 1 — Our first major contribution was that we identified a bug in the MATLAB code provided by the authors. Upon reaching out to the authors with this issue, Chen verified that it was indeed a bug, and that the top-K eigenvectors should be used along

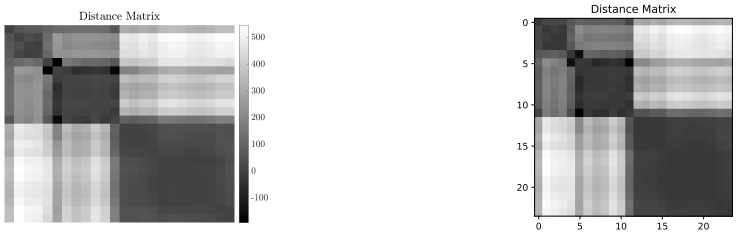

Figure 4. Distance matrix obtained from the clustering algorithm. Left: Original figure generated from MATLAB. Right: Reproduced figure generated from Python.

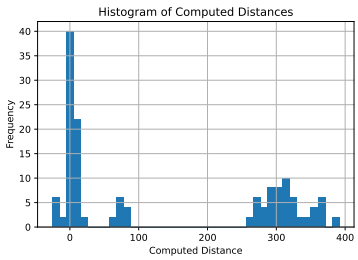

Figure 5. Histogram of distance estimates from clustering

with sample splitting. Due to limited space, we defer the details of identifying the bug and comparison of using eigenvectors and singular vectors to the appendix.
Even though we identified a bug, for the sake of reproducibility, we followed the authors' MATLAB code exactly to generate the figures in Chen and Poor's paper [1]. Throughout the rest of this report, we adopt to using eigenvectors instead of singular vectors.

Contribution 2 – Our second contribution is presenting how one can choose a threshold for making the graph used for clustering. Our simple yet effective method is to plot a histogram of the estimated distances before thresholding. The histogram is expected to be multimodal, with a big peak at zero, corresponding to distances estimated between trajectories coming from the same model. The threshold should be chosen at a value separating this big peak and the next big peak (which would correspond to the smallest distance between two different models). We plot a histogram of computed distances for the MotionSense data, which is shown in Figure 5. From Figure 5, it is clear that a threshold of $100 \leq \tau \leq 240$ should return ideal results for clustering.

Contribution 3 – To demonstrate the importance of model separation in the algorithm, we observed the change in clustering error when varying $\Delta_{\mathrm{A,W}}$ while fixing $T_{\mathrm{clustering}}$. The results are shown in Figure 6. In Figure 6, we can see that there is a tradeoff between $T_{\mathrm{clustering}}$ and $\Delta_{\mathrm{A,W}}$. For example, when $\Delta_{\mathrm{A,W}} = 0.10$ and $T_{\mathrm{clustering}} = 40$, the clustering error is near zero, but substantially higher when $T_{\mathrm{clustering}} = 10$. This corroborates the paper's theory in that these values are inversely proportional.

Contribution 4 – Our last contribution is observing the performance of the algorithm when projecting onto random subspaces. We performed this experiment on two different values of $\Delta_{\mathrm{A,W}}$ to see if perhaps if the model was well-separated, then we could get away with using random subspaces (or vice versa). The results are shown in Figure 7. In Figure 7, we can see that the error is high regardless of $T_{\mathrm{clustering}}$ and $\Delta_{\mathrm{A,W}}$, which indicates that this may not be a good idea.

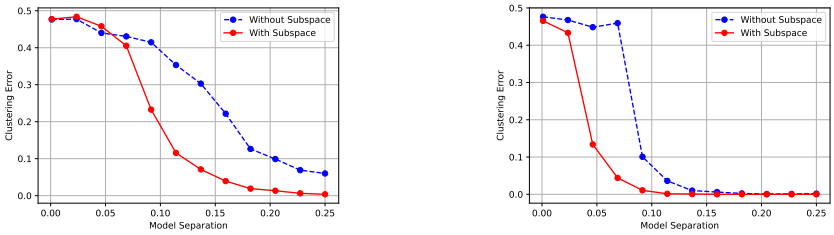

Figure 6. Performance of spectral clustering for varying model separation parameters $\Delta_{\mathrm{A,W}}$. Left: Clustering error for $T_{\text{clustering}} = 10$. Right: Clustering error for $T_{\text{clustering}} = 40$.

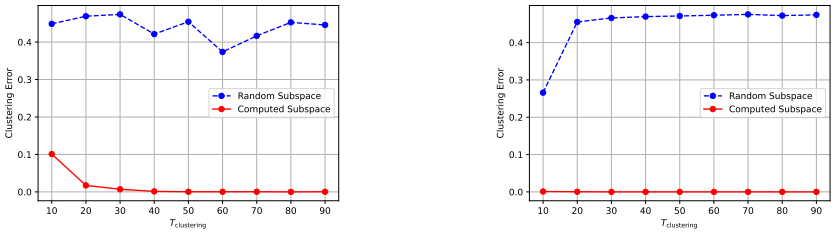

Figure 7. Performance of spectral clustering using random subspaces vs. proposed subspace estimation method. Left: Clustering error for $\Delta_{\mathrm{A,W}} = 0.12$. Right: Clustering error for $\Delta_{\mathrm{A,W}} = 0.30$.

6 Discussion

Through this reproducibility project, we were able to (mostly) corroborate Chen and Poor's empirical results. We reproduced the results by re-implementing the entire algorithm in Python using the authors' MATLAB code as a reference. We were able to further contribute to the authors' results by (1) identifying an error in their code, (2) presenting a method in choosing the hyperparameter τ, (3) validating their theory on the dependence on the model separation parameter, and (4) demonstrating the performance of projecting onto random subspaces. We think another fruitful direction for contribution might be to derive update rules for the EM algorithm and observing its performance. Once the blind review process is completed, we plan on publicizing our code.

6.1 What was Easy

The description of the algorithms in the paper were well-written and easy to understand. The authors' code was also complete, making it easy to run and understand.

6.2 What was Difficult

Although the MATLAB code was easy to understand, the implementation in Python was rather difficult. The MATLAB code used a lot of for-loops, which took significantly longer to run on Python (see Section A.1). To circumvent this, we needed to take time to modify the code to use vectorization tricks. We also identified a lot of discrepancies between the MATLAB code and the paper, which were difficult at times to resolve.

6.3 Communication with Original Authors

We requested the authors for their code and they sent it promptly. Chen was very responsive and answered questions very clearly.

A Appendix

A.1 Computational Requirements

For all experiments, we used a Python environment on a Macbook Pro with 2.2 GHz Intel Core i7 and 16 GB RAM. The approximate runtimes for generating each figure is displayed in Table 1. The runtime for the model estimation error is significantly greater, as the dimensions of the each trajectory is doubled with larger sample sizes.

Figures	Python	MATLAB
Figure 1 (Clustering Error)	1264.0795	266.3131
Figure 2 (Classification Error)	14.81847	22.4947
Figure 3 (Model Estimation Error)	22058.2458	6600.5657
Figure 4 (MotionSense)	0.7906	2.7165

Table 1. Runtimes in seconds for reproducing each figure in both Python and MATLAB

We observe that the runtimes for MATLAB are generally much lower than that of the Python counterparts. This was even after we had modified the Python code to use vectorization tricks to avoid the use of for-loops.

A.2 Differences Between Using Eigenvectors and Singular Vectors

We identified the bug in their code by plotting the top-5 eigenvalues and singular values of a randomly chosen iteration of the algorithm. We observed that although the first eigenvalue and singular value were always equivalent, for the next $2-5$, the negative eigenvalues was always dominant, as shown in Figure 8. In Figure 8, we see that the singular value is always greater, which corresponds to the negative eigenvalue.

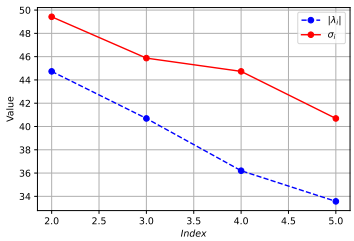

Figure 8. Plot of the differences in singular values and eigenvalues for indices $2-5$.

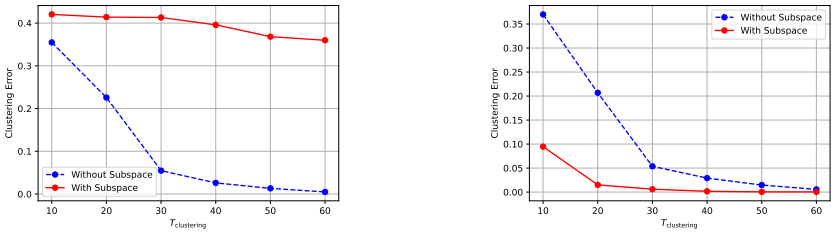

Figure 9. Performance of spectral clustering using top-K eigenvalues. Left: Clustering error for various $T_{\text{clustering}}$ without sample splitting. Right: Clustering error for various $T_{\text{clustering}}$ with sample splitting.

Now that we have identified the bug, we modified the code to take the top-K eigenvectors. Upon modification, we ran the clustering algorithm to see the change in its performance.

This result is shown on the left of Figure 9. However, it turned out using eigenvectors returned very poor results. Upon reaching out to the authors, they replied saying that while the use of eigenvectors is indeed correct, if we wanted to use eigenvectors, then we needed to use sample splitting amongst the sub-routines. We thought this was purely for theoretical purposes (which was to guarantee independence), but it turns out that this was needed in practice as well. The results for clustering with sample splitting is shown in Figure 9, and it indeed returns optimal results.

A.3 Reproduced Results Using Eigenvectors and Sample Splitting

In this section, we briefly discuss how the use of eigenvectors affects the reproduced results. Upon sample splitting as suggested by Chen, the general trend remains the same – the errors decrease as a function of T. However, the distance matrix shown in Figure 4 changes using eigenvectors, as the histogram of the computed distances change. The differences are shown in Figure 10. In Figure 10, we can see that with eigenvectors and sample splitting, the performance of the clustering algorithm signficantly increases, as we obtain better separation.

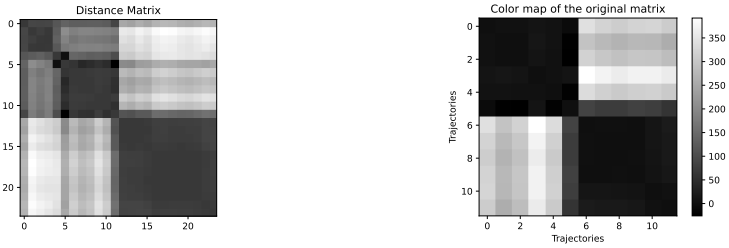

Figure 10. Distance matrix obtained from the clustering algorithm. Left: Figure without sample splitting. Right: Figure with sample splitting.

References

1. Y. Chen and H. V. Poor. "Learning Mixtures of Linear Dynamical Systems." In: **Proceedings of the 39th International Conference on Machine Learning**. Ed. by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 3507–3557.
2. M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. "Mobile Sensor Data Anonymization." In: **Proceedings of the International Conference on Internet of Things Design and Implementation**. IoTDI '19. Montreal, Quebec, Canada: ACM, 2019, pp. 49–58.