Learning Dynamic Models from Non-sequenced Data

Tzu-Kuo Huang TZUKUOH@CS.CMU.EDU
Machine Learning Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Jeff Schneider SCHNEIDE@CS.CMU.EDU
Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Abstract

Virtually all methods of learning dynamic models from data start from the same basic assumption: that the learning algorithm will be provided with a single or multiple sequences of data generated from the dynamic model. In this work we consider the case where the data is not sequenced. The learning algorithm is presented a set of data points from the system's operation but with no temporal ordering. The data are simply drawn as individual disconnected points. While making this assumption may seem absurd at first glance, we observe that many scientific modeling tasks have exactly this property.

We begin by formalizing the task of learning fully observable, discrete-time and linear models from non-sequenced data, and briefly discuss several issues in identifiability. We then develop several learning algorithms based on optimizing approximate posterior distributions. Next we generalize these algorithms to learn a class of nonlinear dynamic models through the use of reproducing kernels. We also study a different but related problem, that of reconstructing a temporal sequence from out-of-order data points. To solve this problem, we propose a convex program that optimizes a measure of temporal smoothness, and develop efficient solution algorithms. We test these methods on several synthetic and real data sets; experimental results show both effectiveness and limitations of the proposed methods.

1. Introduction

Learning dynamic models from data is the traditional topic of system identification (Ljung, 1999) in control theory and many algorithms have been proposed. In the machine learning literature, the learning of graphical models, such as dynamic Bayesian networks (Ghahramani, 1998a; Murphy, 2002), and the learning of various types of Markov models (e.g., Rabiner (1989); Ghahramani (1998b); Beal et al. (2002); Abbeel and Ng (2005); Hsu et al. (2009); Song et al. (2010)) have been studied for the same problem, often with discrete state spaces. Virtually all of these methods start from the same basic assumption: that the learning algorithm will be provided with sequences, or trajectories, of data generated from the dynamic model. Here we consider the case where the data is not sequenced. The learning algorithm is presented a set of data points from the system's operation but with no temporal ordering. The data are simply drawn as individual disconnected points, and usually come from many separate executions of the dynamic system.

Many scientific modeling tasks have exactly this property. Consider the task of learning dynamic models of galaxy or star evolution. The dynamics of these processes are far too slow for us to collect successive data points showing any meaningful changes. However, we do have billions of single data points showing these objects at various stages of their evolution.

At more modest time scales, the same problem arises in the understanding of slow-moving human diseases such as Alzheimer's or Parkinson's, which may progress over a decade or more. It is feasible to follow patients over the full course of their disease and many studies do exactly that. However, a scientist considering new hypotheses about the effects of previously unmeasured proteins or genes would prefer to collect data from their current pool of patients and assemble a dynamic model immediately rather than wait a decade for full trajectories of the disease to be collected.

At the other end of the spectrum, cellular or molecular biological processes may be too small or too fast to permit collection of trajectories from the system. Often, the measurement techniques are destructive and thus only one data point can be collected from each sample even though a rough indication of the relative timing between samples may be known.

In all of these applications, scientists would like to construct a dynamic model using only non-sequenced, individual data points collected from the system of interest. In this work, we launch investigation of this topic by considering fully observable, continuous-state, discrete-time models. Starting with linear dynamic models, we briefly review basic methods for learning from sequenced data in Section 2.1, and formalize the problem of learning from non-sequenced data in Section 2.2. We then propose several learning methods based on optimizing approximate posteriors in Sections 3.1 to 3.3, followed by a generalization for learning nonlinear models in Section 3.4. The proposed methods essentially carry out expectation maximization, and thus require good initialization. We address this issue in Section 4, and point out the connection to a related problem: reconstructing a temporal sequence from out-of-order data points. To solve this problem, we propose in Section 5 a convex program that optimizes a measure of temporal smoothness, and develop an efficient minimization algorithm. Sections 6.1 and 6.2 report our experiments on simulated and real data, respectively, and provide some observations and comparisons. In sum, the proposed methods perform reasonably well on these data sets, but depend heavily on good initializations. Finally, we conclude this work in Section 7 and discuss future directions.

Part of this work appeared in (Huang and Schneider, 2009) and (Huang et al., 2010).

2. Learning Linear Dynamic Models

We start with first-order, discrete-time, fully observable linear dynamic models described as by the following transition function:

$$\mathbf{x}^{(t+1)} = A\mathbf{x}^{(t)} + \boldsymbol{\epsilon}^{(t)}, \tag{1}$$

where $\mathbf{x}^{(t)} \in \mathbb{R}^p$ is the state vector at time t, $A \in \mathbb{R}^{p\times p}$ is the state transition matrix, and $\boldsymbol{\epsilon}^{(t)}$ is the noise vector at time t. For simplicity, we assume hereafter that $\forall t,\ \boldsymbol{\epsilon}^{(t)} \sim \mathcal{N}(\cdot|\mathbf{0}, \sigma^2 I)$, a Gaussian distribution with zero mean and covariance $\sigma^2 I$, where I is the identity matrix. But the proposed methods in later sections all can be extended to handle general covariance matrices. The dynamical system also has a start state, which we denote as $\mathbf{x}^{(0)}$. Thus, the linear dynamic models considered in this paper are fully characterized by $\Theta = \{A, \sigma^2, \mathbf{x}^{(0)}\}$.

Algorithm 1 Sampling from multiple trajectories

```
**Input:** transition matrix $A$, $\sigma^2$, $\mathbf{x}^{(0)}$, $T_{\max}$, and $n$
for $i = 1$ to $n$ do
  Pick a random time stamp $t_i$ from $\{1, \dots, T_{\max}\}$.
  for $t = 1$ to $t_i$ do
    $\mathbf{x}^{(t)} \leftarrow A\mathbf{x}^{(t-1)} + \boldsymbol{\epsilon}^{(t)}$,  $\boldsymbol{\epsilon}^{(t)} \sim \mathcal{N}(\cdot|\mathbf{0}, \sigma^2 I)$.
  end for
  Set $\mathbf{x}_i = \mathbf{x}^{(t_i)}$.
end for
Output: A sample $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$.
```

2.1 Learning from Sequenced Data

Before discussing our approach for learning from non-sequenced data, we briefly review a basic method for learning from sequenced data. Suppose we have collected T data points, $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$, from a single execution of the model (1) where the t-th data point occurred at time t. Then we can perform a standard regression to obtain estimates of the parameters. We form an input matrix, X, with $T-1$ rows containing the observations $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T-1)}$ and a corresponding output matrix, Y, with $T-1$ rows containing the observations $\mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}$. Then A is estimated as $\widehat{A} = (X^\top X)^{-1}(X^\top Y)$, where $X^\top$ is the transpose of X, and σ^2 is estimated by $\widehat{\sigma}^2 = \|Y - X\widehat{A}\|_F^2/(T-1)/p$, where $\|\cdot\|_F$ is the matrix Frobenius norm. If more than one trajectory is observed, X and Y are assembled accordingly such that each row contains pairs of state observations at time t and $t+1$. In fact, this approach does not require long trajectories. It only needs enough *pairs of consecutive observations*. However, as mentioned in Section 1, with our target systems it is difficult to collect even two consecutive observations at desirable time intervals.

2.2 Learning from Non-sequenced Data

The problem without observed state sequences is much more difficult. We assume that n executions of the dynamic model (1) have taken place, and from each execution we have observed a single data point drawn at random from the sequence of states generated in that execution. The result is n data points, $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, each from a different trajectory and having occurred at an unknown point in time. To avoid confusion in indices, hereafter we use parenthesized super-script, e.g., $\mathbf{x}^{(t)}$, to denote the time index, but sub-script, e.g., $\mathbf{x}_i$, to denote the data index. A precise description of this generative process is given in Algorithm 1.

We focus on estimating A and σ^2, and treat the start state $\mathbf{x}^{(0)}$ as a nuisance parameter. If the time index t_i of $\mathbf{x}_i$ is known, then by the properties of the Gaussian distribution we obtain the likelihood

$$L(\mathbf{x}_i|\boldsymbol{\theta}, t_i) \;=\; \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}^{(t_i)}, \Sigma^{(t_i)}) \;=\; \int \frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i-1)}, \Sigma^{(t_i-1)}) d\mathbf{x}, \quad (2)$$

where $\|\cdot\|$ is the vector two-norm, and

$$\boldsymbol{\mu}^{(t)} \;:=\; A^t\mathbf{x}^{(0)}, \qquad \Sigma^{(t)} \;:=\; \begin{cases} \sigma^2\left(\sum_{i=0}^{t-1} A^i\right)\left(\sum_{i=0}^{t-1} A^i\right)^\top, & t \geq 1, \\ \mathbf{0}, & t = 0. \end{cases} \tag{3}$$

Since the n data points are drawn independently, we can factorize the likelihood of the sample points as

$$L(\mathbf{x}_1, \ldots, \mathbf{x}_n|\boldsymbol{\theta}, t_1, \ldots, t_n) = \prod_{i=1}^{n} L(\mathbf{x}_i|\boldsymbol{\theta}, t_i). \tag{4}$$

The maximization of (4) is a challenging task because, as suggested by (3), the transition matrix A appears in (4) as polynomials whose degrees depend on the missing time indices t_i's. In the next Section we propose several approaches that are able to learn a model while avoiding this difficulty.

2.3 Identifiability Issues in Learning from Non-sequenced Data

Before presenting our proposed methods, we discuss several possibly non-identifiable properties of the model when the true temporal information is missing. Consider a simple linear dynamic model with the following transition matrix and initial point:

$$A \;=\; \begin{bmatrix} \cos\left(\frac{2\pi}{T}\right) & -\sin\left(\frac{2\pi}{T}\right) \\ \sin\left(\frac{2\pi}{T}\right) & \cos\left(\frac{2\pi}{T}\right) \end{bmatrix}, \qquad \mathbf{x}^{(0)} \;=\; \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The ideal trajectory rolled out by this simple dynamic model lies on the unit circle in the two-dimensional Euclidean space. Suppose we observe a set of points from the ideal trajectory, but do not know their time indices. It is easy to see that all of the following dynamic models:

$$A(t) \;=\; \begin{bmatrix} \cos\left(\frac{2\pi t}{T}\right) & -\sin\left(\frac{2\pi t}{T}\right) \\ \sin\left(\frac{2\pi t}{T}\right) & \cos\left(\frac{2\pi t}{T}\right) \end{bmatrix}, \qquad t \in \{\pm 1, \pm 2, \ldots, \pm(T-1)\}$$

would explain the data equally well under any reasonable measure of goodness of fit. In the presence of process noise, some of these models may become less likely, but it would still be hard to uniquely determine the true dynamic model.

The above example suggests two possibly non-identifiable properties of the model: the overall direction in time and the speed of the underlying dynamics. In fact, it has been shown in (Peters et al., 2009) that under some linear dynamic models the true direction in time is not identifiable. The methods proposed in subsequent sections thus do not resolve these ambiguities; the learnt model may follow either of the two directions in time, but usually corresponds to the slowest dynamics.

Another perhaps more intriguing example is depicted in Table 1, which presents a non-sequenced and noiseless data set in the left column and two possible dynamic models in the right column. On the one hand, according our assumption of a single fixed start state as in Algorithm 1, Model 1 should be favored over Model 2 under any reasonable measure of goodness of fit that incorporates such an assumption. On the other hand, under a certain level of noise and/or some non-uniform sampling rate in the temporal domain, the

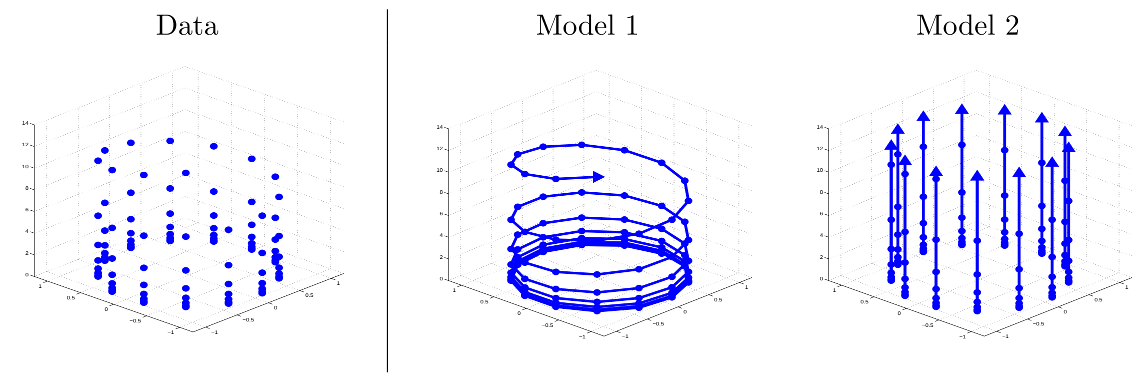


Table 1: An example of two possible dynamic models (right column) for a non-sequenced data set (left column).

data generated from Model 1 may be more similar to a cylinder than to a spiral, making Model 2 equally or even more likely to have generated the data. There are more examples of this type, such as a torus of points where rotations around the short and the long circumferences can hardly be distinguished from each other in the absence of any temporal information. Further theoretical investigation is necessary to understand these issues and find out conditions under which these ambiguities can or cannot be resolved.

3. Approximate Posterior and Expectation Maximization

We present three methods for estimating A and σ^2 in Sections 3.1 to 3.3, all of which are based on optimizing approximate posteriors. The optimization is carried out by Expectation Maximization (EM) types of algorithms. Then Section 3.4 demonstrates extensions of these three methods for learning nonlinear dynamic models, which make use of reproducing kernels.

3.1 An Unordered Posterior

We first remove the problem of unknown time indices by marginalizing out the missing t_i's. According to the generative process in Algorithm 1, the distributions of t_i's are independent from A and σ^2, and also mutually independent. Let $P(t_i)$ denote the probability mass function of $t_i \in \{1, \ldots, T_{\max}\}$. We then write

$$\begin{aligned}
L(\mathbf{x}_1, \ldots, \mathbf{x}_n | \boldsymbol{\theta}) &:= \sum_{t_1=1}^{T_{\max}} \cdots \sum_{t_n=1}^{T_{\max}} L(\mathbf{x}_1, \ldots, \mathbf{x}_n, t_1, \ldots, t_n | \boldsymbol{\theta}) \\
&= \sum_{t_1=1}^{T_{\max}} \cdots \sum_{t_n=1}^{T_{\max}} \left(\prod_{i=1}^{n} L(\mathbf{x}_i | \boldsymbol{\theta}, t_i) P(t_i) \right) \\
&= \prod_{i=1}^{n} \sum_{t_i=1}^{T_{\max}} L(\mathbf{x}_i | \boldsymbol{\theta}, t_i) P(t_i).
\end{aligned}$$

Plugging in the conditional likelihood (2), we obtain

$$
\begin{aligned}
L(\mathbf{x}_1,\dots,\mathbf{x}_n|\boldsymbol{\theta}) &= \prod_{i=1}^{n}\sum_{t_i=1}^{T_{\max}}\left(\int \frac{\exp(-\frac{\|\mathbf{x}_i-A\mathbf{x}\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i-1)},\Sigma^{(t_i-1)})d\mathbf{x}\right)P(t_i)\\
&= \prod_{i=1}^{n}\left(\int \frac{\exp(-\frac{\|\mathbf{x}_i-A\mathbf{x}\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}}\left(\sum_{t_i=1}^{T_{\max}}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i-1)},\Sigma^{(t_i-1)})P(t_i)\right)d\mathbf{x}\right).(5)
\end{aligned}
$$

In the case of $P(t_i)=1/T_{\max}$, i.e. t_i's are uniformly distributed, we have

$$
\begin{aligned}
\sum_{t_i=1}^{T_{\max}}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i-1)},\Sigma^{(t_i-1)})P(t_i) &= \sum_{t_i=1}^{T_{\max}}\frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i-1)},\Sigma^{(t_i-1)})}{T_{\max}}\\
&\approx \sum_{t_i=1}^{T_{\max}}\frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}^{(t_i)},\Sigma^{(t_i)})}{T_{\max}}, \qquad (6)
\end{aligned}
$$

where the last term is the density that the data points $\{\mathbf{x}_1,\dots,\mathbf{x}_n\}$ are generated from. This gives another view of the generative process: a random sample point is drawn (approximately) from (6) and an observation is created by applying (1) to it. However, (6) still depends on the unknown t_i's. To remove this dependency, we replace (6) with its empirical estimate given by the sample points we have. This together with (5) lead to the following approximate likelihood:

$$
\widehat{L}(\mathbf{x}_1,\dots,\mathbf{x}_n|\boldsymbol{\theta}) := \prod_{i=1}^{n}\left(\sum_{j\neq i}\frac{\exp(-\frac{\|\mathbf{x}_i-A\mathbf{x}_j\|^2}{2\sigma^2})}{(n-1)(2\pi\sigma^2)^{\frac{p}{2}}}\right). \qquad (7)
$$

We exclude the case that $\mathbf{x}_i$ generates itself to avoid the degenerate estimate $A=I$. Moreover, we impose a zero-mean Gaussian prior on A with precision λI and an inverse Gamma prior on σ^2 with shape and scale parameters α and β, leading to the approximate log-posterior:

$$
\log\widehat{P}_{\text{UM}}(\boldsymbol{\theta}|\mathbf{x}_1,\dots,\mathbf{x}_n) := \sum_{i=1}^{n}\log\left(\sum_{j\neq i}\frac{\exp(-\frac{\|\mathbf{x}_i-A\mathbf{x}_j\|^2}{2\sigma^2})}{(n-1)(2\pi\sigma^2)^{\frac{p}{2}}}\right) - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2}. \qquad (8)
$$

This is the objective our first method aims to maximize. Since there is no notion of ordering involved in (8), we refer to it as the Unordered Posterior.

Before introducing our optimization algorithm, we point out that (8) considers the data points as if each $\mathbf{x}_i$ were generated from some other $\mathbf{x}_j$ in the sample by (1). However, according to Algorithm 1 no $\mathbf{x}_i$ was generated from any other $\mathbf{x}_j$ in the sample. Such a discrepancy is due to our replacing (6) with its empirical estimate, and an immediate consequence is that σ^2 in (8) now accounts for not only the noise $\boldsymbol{\epsilon}$ in the dynamic model (1), but also the approximation error introduced by replacing (6) with the empirical density.

To present our learning algorithm, we observe that the likelihood (7) is a product of summations of Gaussian densities. This structure is also shared by the likelihood of Gaussian Mixture Models (GMM), for which Expectation Maximization (EM) algorithms are the common choice for estimation. Although (8) is not a GMM, its similar structure allows us to derive an EM procedure with analytical update rules. We first introduce a latent variable matrix $Z \in \{0,1\}^{n\times n}$ such that

$$\begin{aligned} &Z_{ij} = \begin{cases} 1, & \mathbf{x}_i \text{ was generated from } \mathbf{x}_j \\ 0, & \text{otherwise} \end{cases}, \quad j \neq i, \\ &Z_{ii} = 0, \quad \sum_{j=1}^{n} Z_{ij} = 1. \end{aligned} \tag{9}$$

Again, here "$\mathbf{x}_i$ was generated from $\mathbf{x}_j$" is to be taken as an approximation due to our replacing (6) with the data. We then rewrite (8) using the standard variational equation (c.f. Section 9.4, Bishop (2006)):

$$\log \widehat{P}_{\text{UM}}(\boldsymbol{\theta}|\mathbf{x}_1,\ldots,\mathbf{x}_n) \;=\; \log \sum_{Z} \widehat{P}_{\text{UM}}(\boldsymbol{\theta}, Z|\mathbf{x}_1,\ldots,\mathbf{x}_n), \tag{10}$$

where

$$\widehat{P}_{\text{UM}}(\boldsymbol{\theta}, Z|\mathbf{x}_1,\ldots,\mathbf{x}_n) \;:=\; \left(\prod_{i=1}^{n}\prod_{j=1}^{n}\left(\frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2})}{(n-1)(2\pi\sigma^2)^{\frac{p}{2}}}\right)^{Z_{ij}}\right)\exp\left(-\frac{\lambda}{2}\|A\|_F^2 - \sigma^{2(\alpha+1)} - \beta/\sigma^2\right)$$

is referred to as the complete posterior. Following the standard EM derivation, in the E-step we compute the posterior probability of Z:

$$Q(Z|\boldsymbol{\theta}, \mathbf{x}_1,\ldots,\mathbf{x}_n) \;:=\; \frac{\widehat{P}_{\text{UM}}(\boldsymbol{\theta}, Z|\mathbf{x}_1,\ldots,\mathbf{x}_n)}{\widehat{P}_{\text{UM}}(\boldsymbol{\theta}|\mathbf{x}_1,\ldots,\mathbf{x}_n)},$$

which simplifies as

$$\widetilde{Z}_{ij} \;:=\; Q(Z_{ij}=1|\boldsymbol{\theta}, \mathbf{x}_1,\ldots,\mathbf{x}_n) \;=\; \begin{cases} \dfrac{\exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2}\right)}{\sum_{s\neq i}\exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_s\|^2}{2\sigma^2}\right)}, & i \neq j, \\ 0, & i = j. \end{cases} \tag{11}$$

In the M-step we maximize the expectation of the log complete posterior with respect to $Q(Z|\boldsymbol{\theta}, \mathbf{x}_1,\ldots,\mathbf{x}_n)$:

$$\begin{aligned} &\max_{\boldsymbol{\theta}'} \sum_{Z} Q(Z|\boldsymbol{\theta}, \mathbf{x}_1,\ldots,\mathbf{x}_n)\log \widehat{P}_{\text{UM}}(\boldsymbol{\theta}', Z|\mathbf{x}_1,\ldots,\mathbf{x}_n) \iff \\ &\max_{A,\sigma^2} -\sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{Z}_{ij}\left(\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} + \frac{p}{2}\log(2\pi\sigma^2)\right) - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2}, \end{aligned} \tag{12}$$

whose solution has a simple form:

$$A \;=\; \left(\sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{Z}_{ij}\mathbf{x}_i\mathbf{x}_j^\top\right)\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{Z}_{ij}\mathbf{x}_j\mathbf{x}_j^\top + \lambda\sigma^2 I\right)^{-1}, \tag{13}$$

$$\sigma^2 \;=\; \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{Z}_{ij}\|\mathbf{x}_i - A\mathbf{x}_j\|^2 + 2\beta}{pn + 2(\alpha+1)}, \tag{14}$$

Algorithm 2 Expectation Maximization for (8)

```
**Input:** Data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$
Initialize $A_{(0)}$ and $\sigma^2_{(0)}$, set $k = 0$
**repeat**
    Update $\widetilde{Z}_{(k+1)}$ by (11) with $A_{(k)}$ and $\sigma^2_{(k)}$
    Update $A_{(k+1)}$ by (13) with $\widetilde{Z}_{(k+1)}$ and $\sigma^2_{(k)}$
    Update $\sigma^2_{(k+1)}$ by (14) with $A_{(k+1)}$ and $\widetilde{Z}_{(k+1)}$
    $k \leftarrow k + 1$
**until** The approximate log posterior (8) does not increase
```

A summary of the EM procedure is given in Algorithm 2. Note that (14) can be easily generalized to handle general covariance structures.

According to (11) and (12), we can view Algorithm 2 as a version of the iteratively reweighted least square (IRLS) procedure. Although it is simple and often computationally efficient, one may worry that without enforcing any directional consistency in the Z_{ij}'s the EM algorithm may lead to degenerate dynamic models, especially when the sample size is limited. In fact, this happens in our experiments on simulated data. We thus propose a variant of (8) that incorporates additional constraints in the next section.

3.2 A Partially-ordered Posterior

As mentioned before, the replacement of the true state space density with the empirical density results in the approximate likelihood (7), where the data points are treated as if each one were actually generated from some other one. But what might be more problematic is such an approximation ignores the fact that there is a latent temporal ordering induced by the unknown time indices of the data points, even though the data points are drawn independently. A possible consequence of ignoring the latent ordering is a degenerate estimate of the dynamic model. Thus in our second approach, we take into account the ordering relation explicitly. To do so, we introduce a set of weight parameters ω_{ij}'s, collectively denoted as an n-by-n matrix $\boldsymbol{\omega}$, and modify the approximate likelihood (7) as follows:

$$\widehat{L}_2(\mathbf{x}_1, \ldots, \mathbf{x}_n | \boldsymbol{\theta}, \boldsymbol{\omega}) \ := \ \prod_{\substack{i=1,\\ i \notin S}}^{n} \sum_{j=1}^{n} \left(\frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}} \omega_{ij} \right), \tag{15}$$

in which $S \ := \ \{i : t_i \leq t_j \ \forall j\}$,

$$\begin{cases} \omega_{ij} \ \geq \ 0, & t_j < t_i, \\ \omega_{ij} \ = \ 0, & t_j \geq t_i, \\ \omega_{ii} \ = \ 0, & \forall\, i, \end{cases} \quad \text{and} \quad \sum_{j=1}^{n} \omega_{ij} = 1, \ \forall\ i \notin S. \tag{16}$$

The first set of constraints in (16) is to force the summation in (15) to be consistent with a global direction of time, while the normalization constraints are to maintain the notion of approximating the true state space density with an empirical density. The set S denotes the data points that are the earliest in time (hence cannot be generated from other data points), which can be viewed as rough estimates of the first state. If the underlying dynamic model exhibits a periodic behavior (such as rotation on a plane), the true first state may not be identifiable but A and σ^2 still may be. In that case, S is chosen arbitrarily and the relative time offsets between points may still be correct, thus leading to reasonable estimates of A and σ^2.

As mentioned before, the true time indices t_i's of the data points are missing, and even with the above approximation (15) and (16) it is still unclear how to jointly estimate them and the model parameters. We instead consider the ω_{ij}'s as unknown parameters to be estimated, which we interpret as decomposing the global ordering information into parameters of pairwise relations. Again, we make clear that as in Section 3.1, here we are also approximating the likelihood as if each point in the data were actually generated from some other point in the data. The set of constraints (16) can be restated only in terms of $\boldsymbol{\omega}$ as follows:

1. $\boldsymbol{\omega}$ is non-negative; each row of $\boldsymbol{\omega}$ sums to one or zero.

2. As a weighted adjacency matrix, $\boldsymbol{\omega}$ represents a *directed acyclic* graph.

Note that for both constraints to hold at the same time, one or more rows in $\boldsymbol{\omega}$ must sum to zero, and the corresponding data points form the set S. However, it is hard to maximize (15) with respect to ω under these constraints because they define a non-convex set. Moreover, Nicholson (1975) proved that a weighted adjacency matrix M contains no cycle if and only if permanent$(M+I) = 1$, and Valiant (1979) showed that computing the matrix permanent is #P-complete. We therefore consider a subset of the previous two constraints:

1. $\boldsymbol{\omega}$ can only take values in $\{0, 1\}$.

2. As an adjacency matrix, $\boldsymbol{\omega}$ forms a *directed spanning tree.*

The new constraints turn the problem into a combinatorial one, which at first glance seems even more difficult. As we will show below, the fact that this discrete version is computationally tractable depends entirely on our restricting $\boldsymbol{\omega}$ to be a directed spanning tree. Under the new constraints, the set S has only one data point, which is the root of the directed spanning tree. Combining these tree-based constraints with the approximate likelihood (15) and imposing the same priors on A and σ^2 as before, we propose maximizing the following approximate log posterior for estimation:

$$\max_{\substack{A,\sigma^2,\boldsymbol{\omega},\\ r\in\{1,\ldots,n\}}} \sum_{\substack{i=1,\\ i\neq r}}^{n} \log \sum_{j=1}^{n} \left(\frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}} \omega_{ij} \right) - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2} \tag{17}$$

$$\text{s.t.} \quad \omega_{ij} = \{0, 1\}, \tag{18}$$

$$\sum_{j=1}^{n} \omega_{ij} = 1, \; i \neq r, \; \sum_{j=1}^{n} \omega_{rj} = 0, \tag{19}$$

$$\boldsymbol{\omega} \text{ forms a tree with root } \mathbf{x}_r. \tag{20}$$

Algorithm 3 Alternating Maximization for (17)

```
**Input:** Data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$.
Initialize $A_{(0)}$ and $\sigma^2_{(0)}$, set $k = 0$
**repeat**
    Construct the weight matrix $W_{(k)}$ by (24) with $A_{(k)}$ and $\sigma^2_{(k)}$
    $\boldsymbol{\omega}_{(k+1)} \leftarrow \mathsf{OptimumBranch}(W_{(k)})$
    Update $A_{(k+1)}$ by (22) with $\boldsymbol{\omega}_{(k+1)}$ and $\sigma^2_{(k)}$
    Update $\sigma^2_{(k+1)}$ by (23) with $A_{(k+1)}$ and $\boldsymbol{\omega}_{(k+1)}$
    $k \leftarrow k + 1$
**until** The approximate log posterior (17) does not increase
```

We refer to (17) as the (log) Partially-ordered Posterior, and with the constraints (18) and (19) it can be simplified as follows:

$$\begin{aligned}
&\sum_{\substack{i=1,\\ i\neq r}}^{n} \log \sum_{j=1}^{n} \left(\frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}} \omega_{ij} \right) - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2} \\
= \;& \sum_{i=1}^{n} \log \prod_{j=1}^{n} \left(\frac{\exp(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2})}{(2\pi\sigma^2)^{\frac{p}{2}}} \right)^{\omega_{ij}} - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2} \\
= \;& -\sum_{i=1}^{n}\sum_{j=1}^{n} \omega_{ij} \left(\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} + \frac{p}{2}\log(2\pi\sigma^2) \right) - \frac{\lambda}{2}\|A\|_F^2 - (\alpha+1)\log\sigma^2 - \frac{\beta}{\sigma^2}. \quad (21)
\end{aligned}$$

Interestingly, this objective function is in the same form as the expected log complete posterior (12) in Section 3.1 with $\widetilde{Z}_{ij}$ being replaced by ω_{ij}. One may thus view $\boldsymbol{\omega}$ as the latent variable Z in Section 3.1 with additional constraints. However, there is a subtle difference: Z serves only as a means to derive the EM algorithm and does not appear in log Unordered Posterior (7) being maximized, whereas $\boldsymbol{\omega}$ explicitly appears in the optimization problem (17) as an unknown parameter to be estimated.

Next we discuss how to maximize (17). Since (21) has the same form as (12), the optimal A and σ^2 under a fixed $\boldsymbol{\omega}$ have the same form as (13) and (14):

$$A = \left(\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\omega}_{ij}\mathbf{x}_i\mathbf{x}_j^\top \right) \left(\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\omega}_{ij}\mathbf{x}_j\mathbf{x}_j^\top + \lambda\sigma^2 I \right)^{-1}, \quad (22)$$

$$\sigma^2 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\omega}_{ij}\|\mathbf{x}_i - A\mathbf{x}_j\|^2 + 2\beta}{p(n-1) + 2(\alpha+1)}. \quad (23)$$

When A and σ^2 are fixed, maximizing (21) with respect to $\boldsymbol{\omega}$ under (18), (19) and (20) is equivalent to finding the *maximum spanning tree on a directed weighted graph*, in which each data point $\mathbf{x}_i$ is a node, each pair of nodes is connected in both directions, and the weight on the edge (i, j) is

$$W_{ij} := -\left(\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} + \frac{p}{2}\log(2\pi\sigma^2) \right). \quad (24)$$

The problem of finding maximum spanning trees on directed graphs is a special case of the *optimum branchings* problem, which seeks a maximum or minimum forest of rooted trees (branching) on a directed graph. Chu and Liu (1965), Edmonds (1967), and Bock (1971) independently developed efficient algorithms for the optimum branchings problem. The ones by the former two are virtually identical, and are usually referred to as the Chu-Liu-Edmonds algorithm, for which Tarjan (1977) gave an efficient implementation that runs in $O(n^2)$ time, where n is the number of nodes, for densely connected graphs. Camerini et al. (1979) pointed out an error of Tarjan (1977) and provided a remedy retaining the same time complexity.

With these results, we present an alternate maximization procedure, Algorithm 3 for maximizing (17), where $\mathsf{OptimumBranch}(\cdot)$ taking an edge-weight matrix as the input argument uses the implementation of Tarjan (1977) and Camerini et al. (1979). Since Algorithm 3 always increases the objective (21), it converges to at least a local maximum.

3.3 Expectation Maximization over Directed Spanning Trees

Recently in the Natural Language Processing community, researchers (Smith and Smith, 2007; Globerson et al., 2007) have developed sum-product inference algorithms for directed spanning trees, which make use of the matrix tree theorem (Tutte, 1984). Based on their techniques, we develop an EM procedure whose E-step computes the the expectation of the latent variable Z over all directed spanning trees. Such a tree-based EM procedure can be viewed as being midway between the previous two methods, the first of which averages out entirely the latent sequential nature of the data, while the second aggressively selects the single most likely partial order.

More precisely, we consider the set of spanning trees, $T(X)$, on the complete directed graph whose nodes are the sample points. We represent a spanning tree by its adjacency matrix, whose rows correspond to heads and columns correspond to tails. We also slightly abuse the notation Z to mean both a predecessor matrix and the corresponding set of edges, so $(i,j) \in Z$ means $Z_{ij} = 1$. We then maximize the following approximate log posterior:

$$\begin{aligned} &\log \widehat{P}_{tree}(A, \sigma^2|X) \qquad (25)\\ &\propto \log\left(\sum_{Z\in T(X)} \prod_{(i,j)\in Z} \exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2}\right)\right) - \frac{\lambda}{2}\|A\|_F^2 - \left(\frac{p(n-1)}{2} + (\alpha+1)\right)\log\sigma^2 - \frac{\beta}{\sigma^2}, \end{aligned}$$

where as before we place a zero-mean Gaussian prior on A with hyper-parameter λ and an inverse Gamma prior on σ^2 with hyper-parameters α and β. A major difference between (25) and the unordered approximate log posterior (8) is that the former sums over "global" latent structures, i.e., spanning trees, whereas the latter sums over "local" latent predecessor variables as shown in (10). We thus expect (25) to be more robust against undesirable local maxima than (8).

To derive an estimation procedure based on maximizing (25), we first denote the posterior distribution over $Z \in T(X)$ by

$$Q(Z|A, \sigma^2, X) := \frac{\mathbf{1}\{Z \in T(X)\} \prod_{(i,j)\in Z} \exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2}\right)}{\sum_{Z'\in T(X)} \prod_{(i,j)\in Z'} \exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2}\right)}. \qquad (26)$$

Algorithm 4 Expectation Maximization for (25)

```
**Input:** Data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$
Initialize $A_{(0)}$ and $\sigma^2_{(0)}$, set $k = 0$
**repeat**
    Update $\widetilde{Z}_{(k+1)}$ by (33) with $A_{(k)}$ and $\sigma^2_{(k)}$
    Update $A_{(k+1)}$ by (29) with $\widetilde{Z}_{(k+1)}$ and $\sigma^2_{(k)}$
    Update $\sigma^2_{(k+1)}$ by (30) with $A_{(k+1)}$ and $\widetilde{Z}_{(k+1)}$
    $k \leftarrow k + 1$
**until** The approximate log posterior (25) does not increase
```

Then, by applying the standard variational equation we obtain the following lower bound:

$$\begin{aligned}
&\log \widehat{P}_{tree}(A, \sigma^2 | X) \\
\geq\ & \left(\sum_{Z \in T(X)} Q(Z|A, \sigma^2, X) \left(\log \prod_{(i,j) \in Z} \exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} \right) \right) \right) - \frac{\lambda}{2}\|A\|_F^2 - \frac{\beta}{\sigma^2} \\
& - \left(\frac{p(n-1)}{2} + (\alpha + 1) \right) \log \sigma^2 \\
=\ & - \left(\sum_{i,j} \widetilde{Z}_{ij} \frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} \right) - \frac{\lambda}{2}\|A\|_F^2 - \frac{\beta}{\sigma^2} - \left(\frac{p(n-1)}{2} + (\alpha + 1) \right) \log \sigma^2, \qquad (27)
\end{aligned}$$

where

$$\widetilde{Z}_{ij} \ := \ E_Q[Z] \ = \sum_{Z \in T(X)} \mathbf{1}\{(i,j) \in Z\} Q(Z|A', (\sigma')^2, X). \qquad (28)$$

The lower bound (27) holds for all choices of A' and $(\sigma')^2$ in the posterior mean (28), suggesting an EM procedure that alternates between computing $\widetilde{Z}_{ij}$ and maximizing (27) with respect to A and σ^2.

For the M-step, the lower bound (27), as a function of A and σ^2, is in the same form as the complete log posterior (12), leading to update rules similar to (13) and (14):

$$A \ = \ \left(\sum_{i=1}^n \sum_{j=1}^n \widetilde{Z}_{ij} \mathbf{x}_i \mathbf{x}_j^\top \right) \left(\sum_{i=1}^n \sum_{j=1}^n \widetilde{Z}_{ij} \mathbf{x}_j \mathbf{x}_j^\top + \lambda \sigma^2 I \right)^{-1}, \qquad (29)$$

$$\sigma^2 \ = \ \frac{\sum_{i=1}^n \sum_{j=1}^n \widetilde{Z}_{ij} \|\mathbf{x}_i - A\mathbf{x}_j\|^2 + 2\beta}{p(n-1) + 2(\alpha+1)}. \qquad (30)$$

For the E-step, we resort to the techniques in Sections 3.1 and 3.2 of (Globerson et al., 2007). Let

$$W_{ij} \ := \ \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - A\mathbf{x}_j\|^2}{2\sigma^2} \right), & i \neq j, \\ 0, & i = j. \end{cases} \qquad (31)$$

denote the weight on the edge $\mathbf{x}_j$ to $\mathbf{x}_i$. Based on the Laplacian of the corresponding weighted directed graph, we define the following matrix:

$$\widetilde{L}_{ij} \;:=\; \begin{cases} r_i, & j = 1, \\ \sum_{j'=1}^{n} W_{ij'}, & i = j,\ j > 1, \\ -W_{ij}, & i \neq j,\ j > 1, \end{cases} \tag{32}$$

which replaces the first column of the Laplacian with a non-negative root selection score vector $\mathbf{r} \in \mathcal{R}^n$. The values in $\mathbf{r}$ reflect how likely each sample point $\mathbf{x}_i$ would be the root of a spanning tree. When prior knowledge is unavailable, we simply set $r_i = 1,\ i = 1, \ldots, n$. Then, we compute $\widetilde{Z}_{ij}$ by

$$\widetilde{Z}_{ij} \;=\; (1 - \mathbf{1}\{1 = i\}) W_{ij} (\widetilde{L}^{-1})_{ii} - (1 - \mathbf{1}\{j = 1\}) W_{ij} (\widetilde{L}^{-1})_{ji}. \tag{33}$$

We determine convergence of the algorithm by checking the value of the log posterior (25), which is computed by

$$\log \widehat{P}_{tree}(A, \sigma^2 | X) \;\propto\; \log |\widetilde{L}| - \frac{\lambda}{2} \|A\|_F^2 - \left(\frac{p(n-1)}{2} + (\alpha + 1) \right) \log \sigma^2 - \frac{\beta}{\sigma^2}.$$

A summary of the EM algorithm is in Algorithm 4.

3.4 Nonlinear Extension via Kernel Regression

To learn nonlinear dynamic models, we extend the aforementioned three methods through the use of kernel regression. We consider nonlinear dynamic models of the following form:

$$\mathbf{x}^{(t+1)} = B\phi(\mathbf{x}^{(t)}) + \boldsymbol{\epsilon}^{(t)},\ \boldsymbol{\epsilon}^{(t)} \sim \mathcal{N}(\cdot | \mathbf{0}, \sigma^2 I). \tag{34}$$

where $\phi(\cdot)$ maps a point in $\mathbb{R}^p$ into a Reproducing Kernel Hilbert Space (RKHS) endowed with a kernel function $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\top \phi(\mathbf{y})$, and B is a linear mapping from the RKHS to $\mathbb{R}^p$. Replacing $A\mathbf{x}_j$ in (8), (17) and (25) by $B\phi(\mathbf{x}_j)$ then leads to nonlinear extensions of the three approximate log posteriors.

Next we extend Algorithms 2, 3 and 4 for learning B and σ^2. For the E-steps, we only need to replace $A\mathbf{x}_j$ in (11), (24) and (31) by $B\phi(\mathbf{x}_j)$. For the M-steps, we solve the weighted least squares problems (12), (21) and (27) with $A\mathbf{x}_j$ replaced by $B\phi(\mathbf{x}_j)$. The resulting three update rules for B and σ^2 are very similar, so for brevity here we only give the one for maximizing the unordered approximate posterior:

$$\begin{aligned} B &= \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \widetilde{Z}_{ij} \mathbf{x}_i \phi(\mathbf{x}_j)^\top \right) \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \widetilde{Z}_{ij} \phi(\mathbf{x}_j) \phi(\mathbf{x}_j)^\top + \lambda \sigma^2 I \right)^{-1} \\ &= X \widetilde{Z} \phi(X)^\top \left(\phi(X) \Lambda_{\widetilde{Z}} \phi(X)^\top + \lambda \sigma^2 I \right)^{-1} && (35) \\ &= X \widetilde{Z} \left(K \Lambda_{\widetilde{Z}} + \lambda \sigma^2 I \right)^{-1} \phi(X)^\top, && (36) \\ \sigma^2 &= \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \widetilde{Z}_{ij} \|\mathbf{x}_i - B\phi(\mathbf{x}_j)\|^2 + 2\beta}{pn + 2(\alpha + 1)}, && (37) \end{aligned}$$

where $X := [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n]$ collects the data into a p-by-n matrix, $\phi(X) := [\phi(\mathbf{x}_1) \ \cdots \ \phi(\mathbf{x}_n)]$ is the RKHS mapping of the entire data set, $\Lambda_{\widetilde{Z}}$ is a diagonal matrix with $(\Lambda_{\widetilde{Z}})_{ii} := \sum_{j=1}^{n} \widetilde{Z}_{ji}$, and $K := \phi(X)^\top \phi(X)$ is the kernel matrix. We obtain (36) from (35) by using the Matrix Inversion lemma.

One issue with the above extensions is that we cannot compute B when the mapping $\phi(\cdot)$ is of infinite dimension. However, we observe that the EM procedures only make use of $B\phi(X)$, and according to (36)

$$\begin{aligned} B\phi(X) &= X\widetilde{Z}\left(K\Lambda_{\widetilde{Z}} + \lambda\sigma^2\right)^{-1} \phi(X)^\top \phi(X) \\ &= X\widetilde{Z}\left(K\Lambda_{\widetilde{Z}} + \lambda\sigma^2\right)^{-1} K. \end{aligned}$$

Therefore, instead of B we maintain and update a p-by-n matrix $M := X\widetilde{Z}\left(K\Lambda_{\widetilde{Z}} + \lambda\sigma^2\right)^{-1}$ in the EM iterations. To predict the next state for a new observation $\mathbf{x}$, we compute $M\phi(X)^\top \phi(\mathbf{x})$, which also only requires kernel evaluations. Alternatively, we may compute a finite-dimensional approximation to $\phi(X)$ by doing a low-rank factorization of the kernel matrix $K \approx \widetilde{\phi}(X)^\top \widetilde{\phi}(X)$, and replace $\phi(X)$ in the EM procedure with $\widetilde{\phi}(X) \in \mathbb{R}^{m\times n}$, $m < n$. Then we can maintain and update $B \in \mathbb{R}^{p\times m}$ explicitly. To do prediction on a set of new data points, we project them onto the basis found by factorizing the training kernel matrix, thereby computing their finite-dimensional approximation $\widetilde{\phi}$, and then apply the estimated B to the mapped points.

4. Initialization by Temporal Clustering

All of the proposed methods are solving non-convex optimization problems, and avoiding local optima is a critical issue. A common practice in applying EM methods is to run the algorithm multiple times, each with a randomly initialized model, and then choose the best local optimum as the final estimate. We follow this practice in our experiments on simulated data in Section 6.1, but observe that the number of random restarts needed to obtain a good model is usually large, meaning that a lot of random initializations lead to undesirable local optima. Moreover, our simulated data are low dimensional, but the problem caused by local optima will only become worse in a higher dimension, which is common with real data. We thus investigate an alternative way of initialization.

We begin by observing that in the case of a linear dynamic model, the samples generated by Algorithm 1 can be viewed as i.i.d. samples drawn from the following mixture of Gaussians:

$$\mathbf{x} \sim \sum_{t=1}^{T_{\max}} \pi^{(t)} \mathcal{N}(\cdot | \boldsymbol{\mu}^{(t)}, \Sigma^{(t)}), \tag{38}$$

where

$$\boldsymbol{\mu}^{(t)} := A^t \mathbf{x}^{(0)}, \qquad \Sigma^{(t)} := \sigma^2 \left(\sum_{i=1}^{t-1} A^i\right) \left(\sum_{i=1}^{t-1} A^i\right)^\top, \tag{39}$$

and $\pi^{(t)} \geq 0$ is the probability that $\mathbf{x}$ is drawn at time t. Based on this view, we devise a heuristic to initialize the model:

1. Estimate $\boldsymbol{\mu}^{(t)}$'s by fitting a GMM to the data
2. Estimate the true temporal order of $\boldsymbol{\mu}^{(t)}$'s based on their estimates from Step 1
3. Learn a dynamic model from the estimated sequence of $\boldsymbol{\mu}^{(t)}$'s by existing dynamic model learning methods

For Step 1 we can use the standard EM algorithm for learning GMMs, or simply the k-means algorithm since subsequent steps only need estimates of the means. Step 2 in its own right is a challenging problem. If we believe temporally close $\boldsymbol{\mu}^{(t)}$'s should be similar, we can compute pairwise distances between estimates of $\boldsymbol{\mu}^{(t)}$'s and solve a traveling salesman problem (TSP). Then we need to decide the direction of time on the TSP path, which is often impossible without prior or expert knowledge. In our experiments in Section 6.2 we simply try both directions and report the one that performs better. Under some linear dynamic models, as we show in Section 6.1, temporal proximity among the means is not proportional to spatial proximity, so the TSP path based on pairwise distances may be a poor estimate of the true temporal order. We thus discuss in the next section an alternative way to recover the true temporal order, which is based on the idea of temporal smoothing.

5. Reconstructing Temporal Sequences by Temporal Smoothing

Motivated by the need of reordering the estimated means as mentioned in the previous section, we study the problem of reconstructing a sequence from a set of temporally scrambled data points. The assumption about the data is different from that in the previous sections: instead of being drawn from multiple and independent executions of an underlying dynamic model, the sample now consists of out-of-order data points from a single execution of the underlying model. The goal then is to recover the true temporal ordering of the data points.

This problem has been studied by the computational biology community in the context of finding a temporal ordering of static, asynchronous microarray measurement data (Magwene et al., 2003; Gupta and Bar-Joseph, 2008). The proposed methods therein are closely related to algorithms for solving the curve reconstruction problem, which has been studied in various fields such as computational geometry (e.g., Giesen (1999)), statistics (Hastie and Stuetzle, 1989), and machine learning (Smola et al., 2001). More specifically, Magwene et al. (2003) proposed to reconstruct the temporal ordering of microarray samples through finding the minimum spanning tree on the graph formed by the sample points, while Gupta and Bar-Joseph (2008) proposed to solve an instance of the traveling salesman problem (TSP) and proved that under certain conditions on the dynamics generating the samples the optimal TSP path accurately reconstructs the true ordering. A key assumption behind these two methods is that temporally close sample points should also be spatially close. While this assumption is reasonable, we find that it is not always true even under linear dynamic models, as shown in Section 6.1. We thus propose a method that first *learns* a pairwise similarity measure between the sample points by taking into account the sequential structure of the data, and then finds the optimal TSP path based on the learnt similarity.

Before presenting our method, we point out a limitation in both the existing methods (Magwene et al., 2003; Gupta and Bar-Joseph, 2008) and ours: the inability to choose an overall direction of time. This limitation is due to the fact that the objective functions in these methods do not distinguish between the two directions of time. Nevertheless, we often expect to have some prior or domain knowledge that help to break the tie. For example, in microarray experiments scientists can usually tell samples collected near the beginning of a biological process from those collected near the end of the process, even though the ordering of samples during the process is hard to decide. We will thus leave the choice of an overall direction to the external entity using the method.

Unlike methods proposed in previous sections, the method we present in the following does not make any assumptions about the functional form of the underlying dynamic model. It only assumes the underlying dynamics to be *smooth*, i.e., the curvature of the trajectory rolled out by the dynamic model is small. More precisely, we quantify smoothness by the second order differences of temporally adjacent points generated by the dynamic model:

$$S = \sum_{t=2}^{T_{\max}-1} \|(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}) - (\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)})\|^2, \tag{40}$$

where $T_{\max}$ is the maximum time. Small values of S correspond to smooth trajectories. Such a smoothness measure has been used as the regularization term in the Hodrick-Prescott filter (Hodrick and Prescott, 1997; Lesser, 1961), a common tool in macroeconomics for obtaining a smooth and nonlinear representation of a time series.

The quantity (40) cannot be computed on our data since the true time indices of the data points are missing. Nevertheless, it can be succinctly expressed using the Laplacian L of the *temporal adjacency graph* obtained by connecting temporally adjacent pairs of data points. More specifically, we let $X := [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n]$ be the p-by-n data matrix as before, and Z be a directed *temporal* adjacency matrix such that $Z_{ij} = 1$ if $\mathbf{x}_j$ precedes $\mathbf{x}_i$ immediately in time, and 0 otherwise. Then, we define $\overline{Z} := Z + Z^\top$ to represent the undirected, symmetric temporal adjacency of the data points. If the data points were sorted according to their true temporal order, the matrix $\overline{Z}$ would consist of ones in the upper-first and lower-first off-diagonals and zeros elsewhere. The graph Laplacian based on the adjacency matrix $\overline{Z}$ is then $L = \mathsf{diag}(\overline{Z}\mathbf{e}) - \overline{Z}$, where $\mathbf{e}$ is a vector of ones and $\mathsf{diag}(\overline{Z}\mathbf{e})$ denotes the diagonal matrix with the vector $\overline{Z}\mathbf{e}$ in the main diagonal. Simple algebraic manipulation shows that the smoothness S can be expressed in terms of L (hence Z) as follows:

$$S(Z) \ = \ \|XL\|_F^2 \ = \ \mathrm{tr}((\mathsf{diag}(\overline{Z}\mathbf{e}) - \overline{Z})^\top X^\top X(\mathsf{diag}(\overline{Z}\mathbf{e}) - \overline{Z})), \tag{41}$$

which is a quadratic and convex function of $\overline{Z}$ and hence Z. Since we assume the true dynamics to be smooth, a natural way to reconstruct a temporal ordering would be to solve the following problem:

$$\begin{aligned} Z^* &= \arg\min_Z \ S(Z) \\ &\text{s.t. } Z \text{ represents a directed Hamiltonian path through the data points.} \end{aligned} \tag{42}$$

However, this problem is essentially a quadratic version of TSP, and to the best of our knowledge, no efficient solver exists for such problems. We thus consider the following two-step heuristics. In the first step, we minimize $S(Z)$ under a modified set of constraints:

$$\begin{aligned} \widehat{Z} \ &= \ \arg\min_Z \ \ S(Z) \\ \text{s.t.} \ \ Z\mathbf{e} = \mathbf{e}, &\quad Z^\top\mathbf{e} = \mathbf{e}, \quad Z_{ij} \geq 0, \quad Z_{ii} = 0. \end{aligned} \tag{43}$$

Algorithm 5 $\mathbf{w} = \ell_1$-Projection($\mathbf{v}$), $\mathbf{v} \in \mathbb{R}^n$

```
1: Sort $\mathbf{v}$ into $\boldsymbol{\mu} : \mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$.
2: Find $\rho = \max\{j \in \{1, 2, \ldots, n\} : \mu_j - \frac{1}{j}(\sum_{r=1}^{j} \mu_r - 1) > 0\}$
3: Define $\theta = \frac{1}{\rho}(\sum_{i=1}^{\rho} \mu_i - 1)$
4: Output $w$ s.t. $w_i = \max\{v_i - \theta, 0\}$
```

The constraints in (43) are not a proper relaxation of (42) because Z must have one zero row and one zero column to represent a Hamiltonian path. Nevertheless, we can interpret solving (43) as learning a pairwise similarity $\widehat{Z}$ whose (i, j)-th entry reflects how likely $\mathbf{x}_j$ is to precede $\mathbf{x}_i$ temporally. Then in the second step, we solve an instance of TSP with $1 - (\widehat{Z} + \widehat{Z}^\top)/2$ as the distance, and obtain an ordering from the optimal TSP path.

The optimization problem (43) is essentially convex quadratic programming (QP) under linear and bound constraints. However, the number of variables is *quadratic* in the number of data points, and as the data size increases, directly applying a general-purpose QP or nonlinear programming solver may become inefficient or even infeasible. We thus devise a simple and efficient *projected gradient method* that iteratively updates the rows and the columns of Z.

The key idea of a projected gradient method is to move the parameter vector along the negative gradient direction, and project the updated vector back into the feasible region Ω whenever it goes out. The cost of a projected gradient procedure is mainly determined by the projection operation, so we need to compute efficiently the projection step:

$$Z^{t+1} \leftarrow \Pi_\Omega(Z^t - \eta \nabla S(Z^t)), \tag{44}$$

$$\Omega = \{Z_{i\cdot}\mathbf{e} = 1, Z_{\cdot,j}^\top \mathbf{e} = 1, Z_{ij} \geq 0, Z_{ii} = 0\}, \tag{45}$$

where $Z_{i\cdot}$ and $Z_{\cdot j}$ denote a row and a column of Z respectively, and $\Pi_\Omega(\mathbf{a}) := \arg\min_{\mathbf{b}}\{\|\mathbf{a} - \mathbf{b}\| \mid \mathbf{b} \in \Omega\}$ is the Euclidean projection of a vector a onto a region Ω. The gradient of $S(Z)$ is given by

$$\nabla S(Z) = 2(\widetilde{Q} + \widetilde{Q}^\top),$$

where

$$\begin{aligned} \widetilde{Q}_{ij} &:= Q_{jj} - Q_{ij}, \\ Q &:= X^\top X(\mathsf{diag}((Z + Z^\top)\mathbf{e}) - (Z + Z^\top)). \end{aligned}$$

Moreover, the feasible region (45) is the intersection of two closed convex sets Ω_1 and Ω_2:

$$\begin{aligned} \Omega_1 &= \{Z_{i\cdot}\mathbf{e} = 1,\ Z_{ij} \geq 0, Z_{ii} = 0, 1 \leq i, j \leq n\}, \\ \Omega_2 &= \{Z_{\cdot,j}^\top \mathbf{e} = 1,\ Z_{ij} \geq 0, Z_{ii} = 0, 1 \leq i, j \leq n\}, \end{aligned}$$

which correspond to the normalization constraints for rows and columns, respectively. Using Dykstra's cyclic projection algorithm Boyle and Dykstra (1986), we perform the projection operation (44) by alternately projecting onto Ω_1 and Ω_2. A very nice property of this procedure is that projecting onto Ω_1 or Ω_2 alone can be further decomposed as doing row-wise (or column-wise) projections, and a single-row or single-column projection can be computed

Algorithm 6 Projected Gradient Method for (43)

Input: Data matrix $X = [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n]$

Output: $\widehat{Z}$

```
1: Set $\alpha = 0.1$, $\epsilon = 10^{-6}$, $\sigma = 10^{-2}$
2: Initialize $Z_{(1)}$, set $k = 1$
3: repeat
4:    Compute the gradient $\nabla_{(k)} := \nabla S(Z_{(k)})$, $\eta \leftarrow 1.0$
5:    repeat
6:       $Z = Z_{(k)} - \eta \nabla_{(k)}$, $D^{row} = D^{col} = [0]_{n \times n}$
7:       repeat
8:          $\widetilde{Z} \leftarrow Z$
9:          $Z'_{i\cdot} \leftarrow \ell_1$-Projection$((Z - D^{row})_{i\cdot})$, $\forall i$
10:         $D^{row} \leftarrow Z' - (Z - D^{row})$
11:         $Z''_{\cdot j} \leftarrow \ell_1$-Projection$((Z' - D^{col})_{\cdot j})$, $\forall j$
12:         $D^{col} \leftarrow Z'' - (Z' - D^{col})$
13:         $Z \leftarrow Z''$
14:      until $\|Z - \widetilde{Z}\|_F \leq \epsilon$
15:      $\eta \leftarrow \alpha \eta$
16:   until $S(Z) - S(Z_{(k)}) \leq \sigma \mathrm{tr}\Big(\nabla_{(k)}(Z - Z_{(k)})\Big)$
17:   $t \leftarrow t + 1,\ Z_{(k)} \leftarrow Z$
18: until $\|Z_{(k)} - Z_{(k-1)}\|_F \leq \epsilon$
19: $\widehat{Z} \leftarrow Z_{(k)}$
```

very efficiently by the ℓ_1 projection technique proposed in Duchi et al. (2008), which we outline in Algorithm 5. The required operations are simply sorting and thresholding[1]. Algorithm 6 gives a summary of the projected gradient method for the optimization problem (43). As in all gradient-based methods, we conduct back-tracking line search for the step size η to ensure convergence.

6. Experiments

We evaluate the proposed methods on several simulated and real data sets. We first give the evaluation criteria, and then describe experiment settings and results in Sections 6.1 and 6.2 respectively for simulated and real data.

Let $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}\}$ be a test sequence of state vectors. After we learn a dynamic model from a training data set, we predict for each $\mathbf{x}^{(t)}, t = 1, \ldots, T-1$ the next state $\widehat{\mathbf{x}}^{(t+1)}$. To evaluate the predictions, we use the following two measures:

- **Cosine Score**:

$$\frac{1}{T-1} \left| \sum_{t=1}^{T-1} \frac{(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})^\top (\widehat{\mathbf{x}}^{(t+1)} - \mathbf{x}^{(t)})}{\|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\| \|\widehat{\mathbf{x}}^{(t+1)} - \mathbf{x}^{(t)}\|} \right|, \tag{46}$$

1. For the ease of presentation, in Algorithm 5 we ignore the constraint $Z_{ii} = 0$, which can be easily enforced by setting $Z_{ii} = 0$ and updating only the other $n - 1$ entries.

which measures the similarity between the predicted one-step movements and the true ones. A higher score means a better prediction. By using such a normalized score of similarity, we focus on measuring the consistency of the predicted dynamics with the truth, but do not take into account the lengths of the difference vectors and the overall direction in time.

- **Normalized Residual**:

$$\frac{1}{T-1}\sum_{t=1}^{T}\frac{\|\mathbf{x}^{(t+1)}-\widehat{\mathbf{x}}^{(t+1)}\|}{\|\mathbf{x}^{(t+1)}-\mathbf{x}^{(t)}\|}, \tag{47}$$

  which measures how close the predictions are to the true next state vectors. A smaller normalized residual means a better prediction, and a constant prediction ($\widehat{\mathbf{x}}^{(t+1)} = \mathbf{x}^{(t)}$) gives a normalized residual of one.

For the simulated data we report the cosine scores, and for the real data we report both the cosine scores and the normalized residuals.

6.1 Simulated Data

We generate data from four dynamical systems:

- **2D**, a two-dimensional divergent linear system:

$$A_{\text{2D}} = \begin{bmatrix}1.01 & 0\\ 0 & 1.05\end{bmatrix},\ \mathbf{x}^0 = \begin{bmatrix}50\\50\end{bmatrix}.$$

  A sample of 200 points is shown in Figure 1.

- **3D-lin**, a three-dimensional linear dynamical system. The transition matrix and the initial point are:

$$A = \begin{bmatrix}1.1882 & 0.3732 & 0.1660\\ -0.1971 & 0.8113 & -0.0107\\ -0.1295 & -0.1886 & 0.9628\end{bmatrix}\ \mathbf{x}^0 = \begin{bmatrix}10\\10\\10\end{bmatrix}.$$

  A sample is shown in Figure 2. From the ideal trajectory we can see that temporally close points (those along the same spiral) can be spatially further away from each other than temporally remote points (those across adjacent spirals).

- **3D-conv**, a convergent three-dimensional nonlinear system (Girard and Pappas, 2005) governed by the following differential equations:

$$\begin{aligned}dx(t)/dt &= -(1+0.1y(t)^2)x(t),\\ dy(t)/dt &= -(1-0.1x(t)^2)y(t)/2+2z(t),\\ dz(t)/dt &= -(1-0.1x(t))2y(t)-z(t)/2,\end{aligned}$$

  where $x(t), y(t)$, and $z(t)$ are the three states at time t. The initial point is set to $[5\ 1\ 5]^\top$. A sample is given in Figure 3.

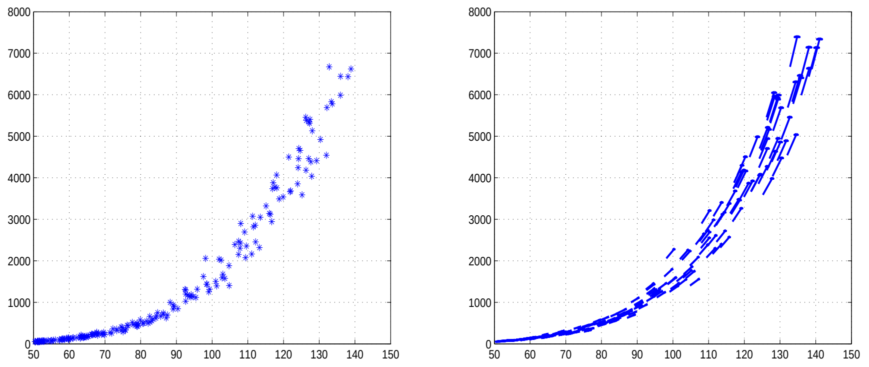


Figure 1: Left: Sample points of 2D with $\sigma = 0.2$. Right: Gradients estimated by UM.

- **Lorenz Attractor** (Lorenz, 1963):

$$\begin{aligned} dx(t)/dt &= 10(y(t) - x(t)), \\ dy(t)/dt &= x(t)(28 - z(t)) - y(t), \\ dz(t)/dt &= x(t)y(t) - 8z(t)/3. \end{aligned}$$

  The initial point is set to [0 1 1.05]. The ideal trajectory is shown in Figure 4.

We consider two different settings:

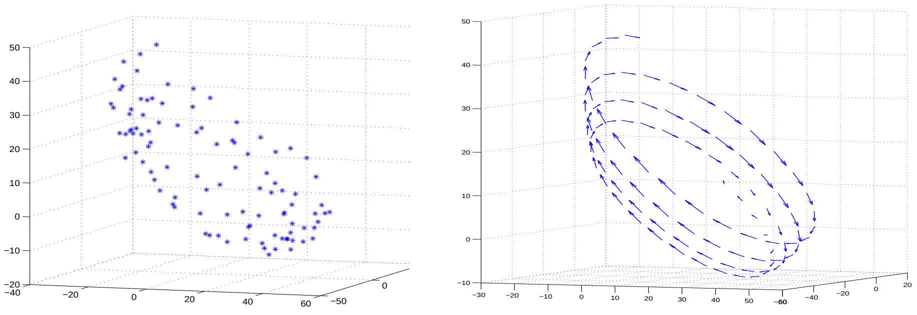
- **Single trajectory with observational noise**
  From each of 3D-lin, 3D-conv, and Lorenz attractor, we generate from the corresponding system equation an ideal trajectory of 100 points, 200 points, and 800 points, respectively, using the initial points mentioned above. Then, we add independent zero-mean Gaussian noise to the ideal trajectories, with two levels of noise standard deviation: $\sigma_{noise} = \{0.01\delta, 0.05\delta\}$, where δ is the median of all the pairwise distances of points on an ideal trajectory. For each noise level and each of the three systems[2], we generate 20 data sets. Examples of these data are in Figures 2(a), 3(a), and 4.

- **Multiple trajectories with process noise**
  We use Algorithm 1 to draw samples from multiple trajectories. For 2D and 3D-lin, we set $T_{\max} = 100$. For 3D-conv we use its discrete-time counterpart and apply a locally linear approximation to the dynamics by treating the derivatives as constant in a small duration $\Delta t = 0.1$. The process noise follows a zero-mean Gaussian, whose standard deviations are 0.2 for 2D, $\{0.01\delta, 0.05\delta\}$ for 3D-lin and $\{0.1\Delta t, 0.5\Delta t\}$ for 3D-conv. We generated 40 data sets of 200 points for 2D, and 20 data sets of 400 points for each of 3D-lin and 3D-conv under different noise levels. We did not include the Lorenz attractor here because it is a chaotic system.

2. For Lorenz attractor we only consider the last 400 points on the ideal trajectory. Nevertheless, this partial trajectory preserves the butterfly shape.

	Rand	PM	UM
Cosine Score	0.78±0.05	0.96±0.14	0.89±0.19

Table 2: Results on 2D with standard deviations, $\sigma = 0.2$

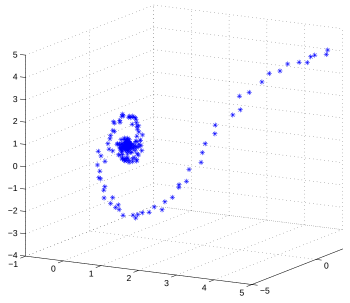


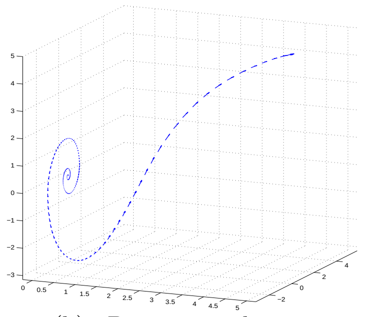
(a) 3D-lin: $\sigma_{noise} = 0.05\delta$ (b) 3D-lin: predictions

Figure 2: Examples of training data (single trajectory) and predicted dynamics on the ideal trajectory by TSM.

We compare the three of the proposed methods: the unordered approximate log posterior (UM), the partially-ordered approximate log posterior (PM), and the temporal smoothing method (TSM). We use UM and PM to refer to approximate log posteriors under linear models, and KUM and KPM to refer to the corresponding nonlinear models. We apply multiple random initializations to all of the approximate posterior based methods. We do not regularize the noise variance σ^2. For TSM, instead of following the outline in Section 4 which clusters the sample points first, here we directly apply it to the sample points. In Section 5 we suggest a two-step heuristics whose section step is solving an instance of TSP to find an ordering of the data points. However, when the sample is drawn from multiple independent trajectories, it makes little sense to find a single total ordering of the data points. Moreover, solving TSP may be computationally extensive. We thus use a different heuristics after solving the convex program (43): for single-trajectory data, we take the lower-triangular part of $\widehat{Z} + \widehat{Z}^\top$ and normalize it to be doubly-stochastic, thereby specifying the correct starting and the end points in time. For multiple-trajectory data, we treat $\hat{Z}$ as a weighted adjacency matrix, find the maximum spanning tree rooted at each data point, and choose the tree that exhibits the "smoothest" dynamics, i.e., the maximum total cosine score between the difference vectors corresponding to adjacent tree edges. After this post-processing, we fit a re-weighted regression as in (36) using the adjusted matching matrix as the weights.

In our comparison we also include a baseline approach that exploits manifold learning techniques. The baseline approach maps the data points to the real line through some manifold learning method, sorts the points according to their one-dimensional projections, and then learns a dynamic model of the form (34) from the ordered data. In our experiments, we find Maximum Variance Unfolding (MVU) by Weinberger et al. (2004) to be the best manifold learning choice.


(a) 3D-conv: $\sigma_{noise} = 0.05\delta$


(b) 3D-conv: predictions

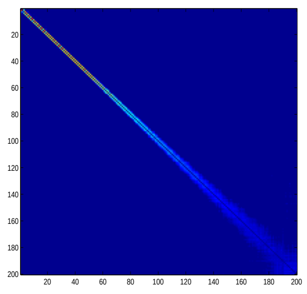

(c) Heat map of $\widehat{Z}$

Figure 3: Examples of training data (single trajectory), predicted dynamics on the ideal trajectory, and the heat map of $\widehat{Z}$ by TSM.

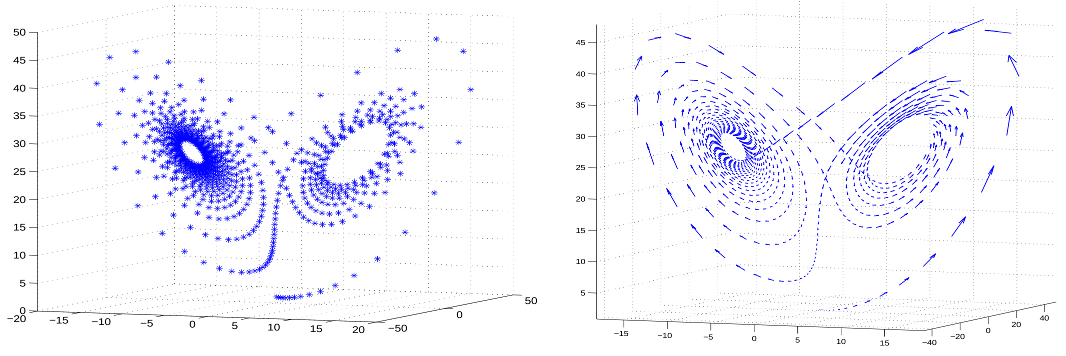


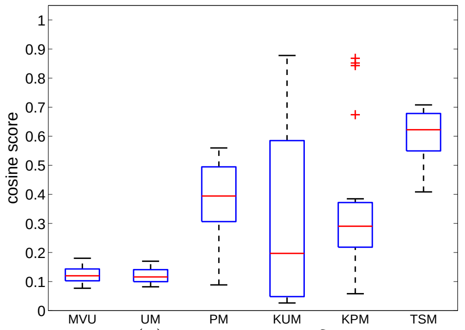
Figure 4: Left: Lorenz attractor ideal trajectory. Right: estimated dynamics by KUM.

For 3D-lin, we use a linear model for MVU and TSM in their final regression. For all the nonlinear models we use kernel regression with the Gaussian kernel $\exp(-(\|\mathbf{x}-\mathbf{y}\|^2)/(2h))$. When applying KUM and KPM, we set the kernel bandwidth h to $10\widetilde{\delta}$ and $50\widetilde{\delta}$ respectively for 3D-lin and the two nonlinear systems, where $\widetilde{\delta}$ is the median of all pairwise distances in a training data set. For TSM and MVU we set $h = \widetilde{\delta}$. Regarding the regularization parameter λ in UM and PM, on noiseless data we set it to 10^{-3}. On 3D-lin single(multiple)-trajectory data we set it to $10^{-7}(10^{-6})$ and $10^{-6}(10^{-5})$ for small and large noise levels, and on all the other data we set it to 10^{-4} and 10^{-3} for the two noise levels. When applying KUM and KPM, we use a low-rank approximation to the kernel matrix as described in Section 3.4, with the reduced rank $m = 5$. For each data set we run KUM and KPM with 50 random initializations of B and σ^2 to avoid local minima. Each entry of B is drawn independently from a zero-mean Gaussian with the standard deviation set to 100, and σ^2 is drawn uniformly random between 0 and 100 times of the median of pairwise distances.

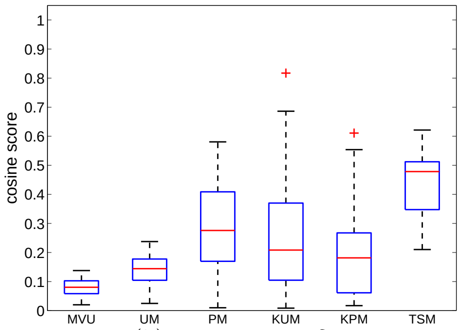
Our experimental results are in Tables 2, 3 and Figures 1 to 7. Table 2 and Figure 1 show the results on 2D, where the "Rand" column gives the performance by randomly guessing a dynamic model. In this simple two dimensional example both UM and PM perform very

	MVU	UM	PM	KUM	KPM	TSM
3D-lin	0.0152	0.9120	0.9898	**0.9972**	0.3239	0.8757
3D-conv	**0.9954**	0.9903	0.5570	0.9909	0.9225	0.9545
Lorenz	0.1383	0.5644	0.2155	**0.9884**	0.334	0.324

Table 3: Cosine scores on noiseless data



(a) $\sigma_{noise} = 0.01\delta$



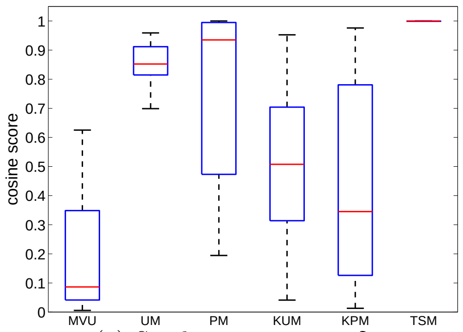
(b) $\sigma_{noise} = 0.05\delta$

Figure 5: Box plots of cosine scores for Lorenz attractor (partial trajectory).

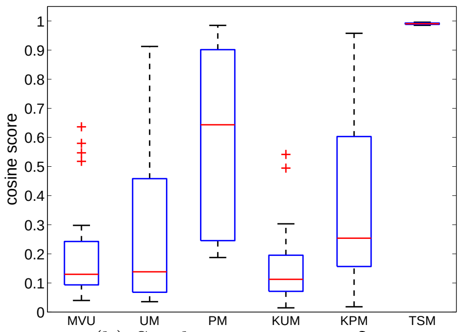
well. Table 3 reports cosine scores on single noiseless trajectories and bold-faces the best method for each dynamical system. It is interesting that KUM performs better than its linear version UM on the linear system 3D-lin, and that UM performs very well on the nonlinear system 3D-conv; the latter suggests 3D-conv may be well approximated by a linear system in terms of one-step predictions. For Lorenz attractor, however, only KUM performs well and other methods are significantly worse. Figure 4 shows the estimated dynamics by KUM, which is very close to the true dynamics.

Figure 3(c) shows the heat map of a $\widehat{Z}$ matrix obtained by solving the convex program (43) for one of the single-trajectory data set generated from 3D-conv, where the rows and the columns are sorted according to the true temporal order. The energy is concentrated around the two leading off-diagonals, showing that the temporal smoothing method nicely recovers the sequential structure in time, though does not choose one direction over the other. This suggests that solving the convex program (43) can lead to a good pairwise similarity, thereby facilitate the subsequent task of finding an ordering.

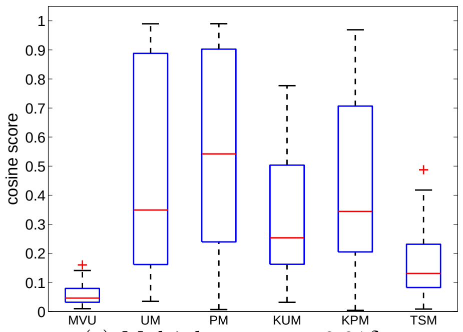
Figures 5, 6(a), 6(b), 7(a), and 7(b) report results on single-trajectory noisy data. TSM is the best for all three systems while MVU is the worst except for 3D-conv with small noise. Figure 3 shows examples of noisy training data and predicted dynamics by TSM. On Lorenz attractor, a highly nonlinear system, KUM and KPM are better than UM and PM. It is interesting that TSM performs better here than in the noiseless case; the reason may be that the partial trajectory used here does not contain the highly-dense spirals in the core of the left wing shown in Figure 4, which cause difficulties for TSM as the smoothness measure (41) may be sensitive to different spacings of the data. For 3D-lin, it is as expected that UM and PM are better than KUM and KPM, but for 3D-conv UM sometimes outperforms them by a small margin. This is aligned with our finding from noiseless data and the known fact that linear models may still be useful for nonlinear systems. Figures 6(c), 6(d),



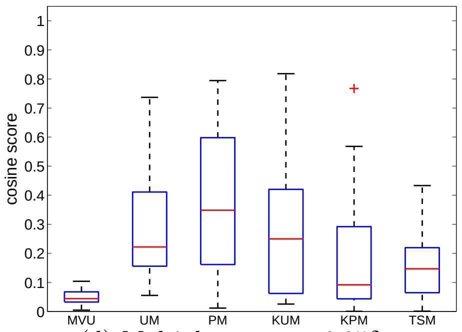
(a) Single, $\sigma_{noise} = 0.01\delta$



(b) Single, $\sigma_{noise} = 0.05\delta$



(c) Multiple, $\sigma_{noise} = 0.01\delta$



(d) Multiple, $\sigma_{noise} = 0.05\delta$

Figure 6: Box plots of cosine scores for 3D-lin.

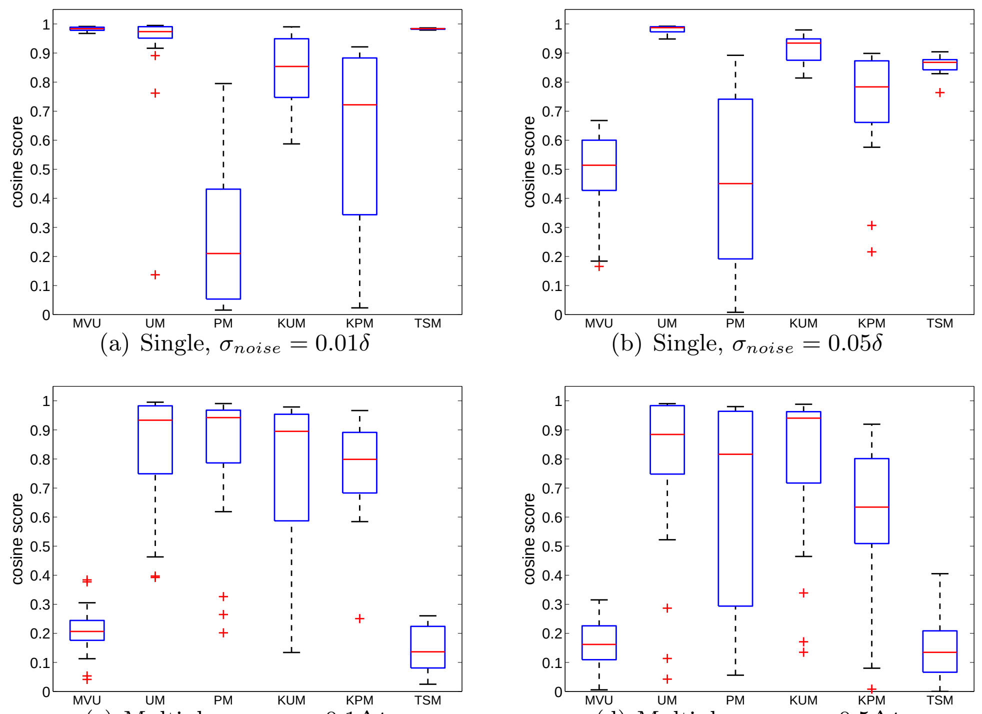
7(c), and 7(d) show results on multiple-trajectories data, which are roughly consistent with single-trajectory results except that TSM performs a lot worse. This is not surprising since TSM's assumption of a single ordering of data points is invalid here.

6.2 Real Data

We apply the proposed methods: UM, PM and their nonlinear versions, and the tree-based EM method (TEM), to two real data sets. For the purpose of evaluation, we choose two time series data sets whose temporal orderings are known: a video stream of a swinging pendulum (Section 6.2.1) and gene expression time series of the yeast metabolic cycle (Section 6.2.2). Knowing the temporal ordering allows us to obtain target dynamic models by applying existing dynamic model learning methods with the available temporal order. We then learn dynamic models by the proposed methods without the known temporal order, and compare the learnt models with the target dynamic model in terms of prediction performances.

We initialize the proposed methods by the temporal clustering heuristics in Section 4. More specifically, we use the K-means method to cluster the data points; to find an ordering of the cluster centers, we compare the following four methods:

1. MVU: Project the cluster centers to the one-dimensional space found by Maximum Variance Unfolding, and then sort the cluster centers according to the projections.
2. l1+TSP: Solve a TSP with the 1-norm pairwise distances between the cluster centers.



(a) Single, $\sigma_{noise} = 0.01\delta$ (b) Single, $\sigma_{noise} = 0.05\delta$

(c) Multiple, $\sigma_{noise} = 0.1\Delta t$ (d) Multiple, $\sigma_{noise} = 0.5\Delta t$

Figure 7: Box plots of cosine scores for 3D-conv.

3. l2+TSP: Solve a TSP with the 2-norm pairwise distances between the cluster centers.

4. TSM+TSP: The two-step heuristics outlined in Section 5.

Then, we learn a linear dynamic model (1) and a nonlinear dynamic model (34) with the Gaussian kernel from the ordered cluster centers, and initialize the proposed methods with the learnt model. As mentioned in Section 5, methods based on TSP do not decide the overall direction of time. Here we learn dynamic models using both directions, and report the one that leads to a better prediction performance. To solve a TSP, we use the state-of-the-art *Concorde TSP solver* (Applegate et al.).

For UM and TEM, we not only initialize the estimation procedures with clustering, but also consider restricting the approximate log posteriors by the ordering of the cluster centers: when summing over the latent variables in (8) and (25), we only include those consistent with the ordering of the clusters. We refer to the restricted versions of UM and TEM respectively as "UM rest" and "TEM rest."

We generalize the proposed methods to allow each state variable to have a different noise variance. The update rules can be easily derived as in previous sections and are quite similar to those under a common noise variance. We choose the regularization parameter λ (and the kernel bandwidth in the Gaussian kernel) by leave-one-out cross validation on the ordered cluster centers, but set α and β by manual selection. Our choice of α and β is mainly to avoid numerical issues caused by small values of the estimated noise variances during the EM iterations. We find the follow choice to be effective: $\alpha = 1$ and $\beta \approx n$, which

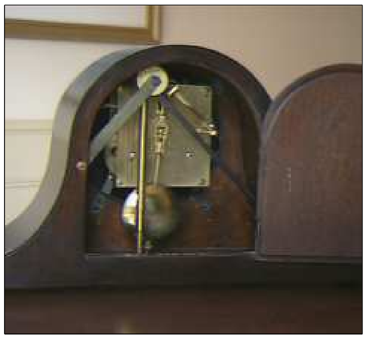


Figure 8: A frame of the swinging pendulum video stream.

correspond to a prior of noise variance whose mean is around n, and in our experiments leads to a posterior mean around 2.

6.2.1 Video of a swinging pendulum

Out first real data is a video analyzed by Siddiqi et al. (2010). The video consists of 500 frames of 240-by-240 colored images of a swinging pendulum. An image is shown in Figure 8. The underlying dynamics is highly periodic and stable as the pendulum completes about 22 full swings[3]. We center the pixel values to be zero across the 500 frames, and then apply Singular Value Decomposition (SVD) to reduce the dimension from $240 \times 240 \times 3 = 172800$ to 20 by projecting the data onto the subspace corresponding to the 20 largest singular values. Such a subspace preserves about 72 percent of the total energy. We further normalize each of the 20 temporal sequences to be zero-mean and unit-variance. Then we use the first 400 points as training data and the last 100 points as testing data.

In the initialization step we combine the K-means method with the AIC criterion to determine the number of clusters. For each possible number of clusters in our search range, we run the K-means method with 30 random restarts and choose the best clustering to initialize the dynamic model. We repeatedly train 30 linear and nonlinear dynamic models with Gaussian kernels, all of which are initialized by K-means combined with AIC. In most of the 30 runs the number of clusters determined by K-means and AIC is 31, which is about the number of time steps one full swing takes. We then evaluate these learnt models by their prediction performances on the test data.

We present the box plots of the testing cosine scores and normalized residuals in Figures 10 to 13 in Appendix A. The left most column in each plot, the "kmeans" column, gives the performance of the initial model found by K-means and some ordering method. In each box-plot we also indicate the performance of the target model learnt with the known temporal ordering. There are two main observations:

- Comparing the four ordering methods, we find that MVU is worse than l1+TSP and l2+TSP, which are in turn worse than TSM+TSP. Moreover, TSM+TSP does almost

3. A full swing means the pendulum ended where it started.

as well as the model learnt with the known temporal ordering. This suggests that orderings solely based on pairwise distances, such as those by MVU, l1+TSP, and l2+TSP, may be more sensitive to distances between cluster centers, which are not always equally separated in space. On the contrary, TSM+TSP is more robust against irregular distances, suggesting that the pairwise similarity learnt through solving the convex program (43) better captures the dynamic nature of the data.

- The initial models learnt from ordered cluster centers already perform quite well, and the proposed methods result in only marginal improvements. Moreover, without the restriction imposed by cluster orderings UM even performs worse than the initial model. This suggests that our approximation to the log posteriors may introduce too many undesirable local maxima.

6.2.2 Gene expression time series of yeast metabolic cycle

To study gene expression dynamics of yeasts during the metabolic cycle, Tu et al. (2005) collected expression profiles of about 6,000 yeast genes along three consecutive metabolic cycles, each containing 12 samples. Due to the destructive nature of the measurement technique, gene expression profiles were measured on different yeast cells, and therefore synchronization of yeast cells in the metabolic cycle is necessary for obtaining reliable gene expression time series data. To address this issue, Tu et al. (2005) developed a continuous culture system that provides a stable environment for yeast cells to grow, and chose a particular strain of yeasts that exhibit "unusually robust periodic behavior," i.e., cells of that strain of yeasts are in a sense self-synchronizing. However, Tu et al. (2005) noted that the periodic gene expression observed in their experiment is more robust than those in certain other species (c.f. **Discussion** in (Tu et al., 2005)), suggesting that in general it may be quite difficult to obtain reliable time series gene expression measurements. In those cases, our proposed methods of learning dynamic models from non-sequenced data may be very useful.

We focus on a subset of 3,552 genes found by Tu et al. (2005) to exhibit strong periodical behaviors during the metabolic cycle. We normalize each gene expression time series to be zero-mean and unit-variance, and use the first two cycles (24 points) as training data and the last cycle (12 points) as testing data. Here the number of state variables (genes) is much higher than the sample size, and thus learning dynamic models is much more difficult than in the previous experiment.

Since the number of sample points is disproportionally smaller than the dimension, in the initialization step we specify the number of clusters in the K-means method to be 12, the number of time steps in one cycle. This means each cluster will contain only few data points. We repeatedly train 20 linear models, and in each of the 20 runs we randomly restart the K-means method 30 times and choose the best clustering to initialize the model. We do not consider nonlinear models here due to the high dimension.

To evaluate the proposed methods, we first qualitatively examine our initial step of temporal clustering and ordering. Among the 3,552 genes, MRPL10, POX1 and RPL17B were found to be strongly periodical and yet exhibit different dynamics. Treating these three genes as fixed seeds in clustering analysis, Tu et al. (2005) identified three major clusters of genes. From each cluster we pick the 24 most representative genes and plot their average

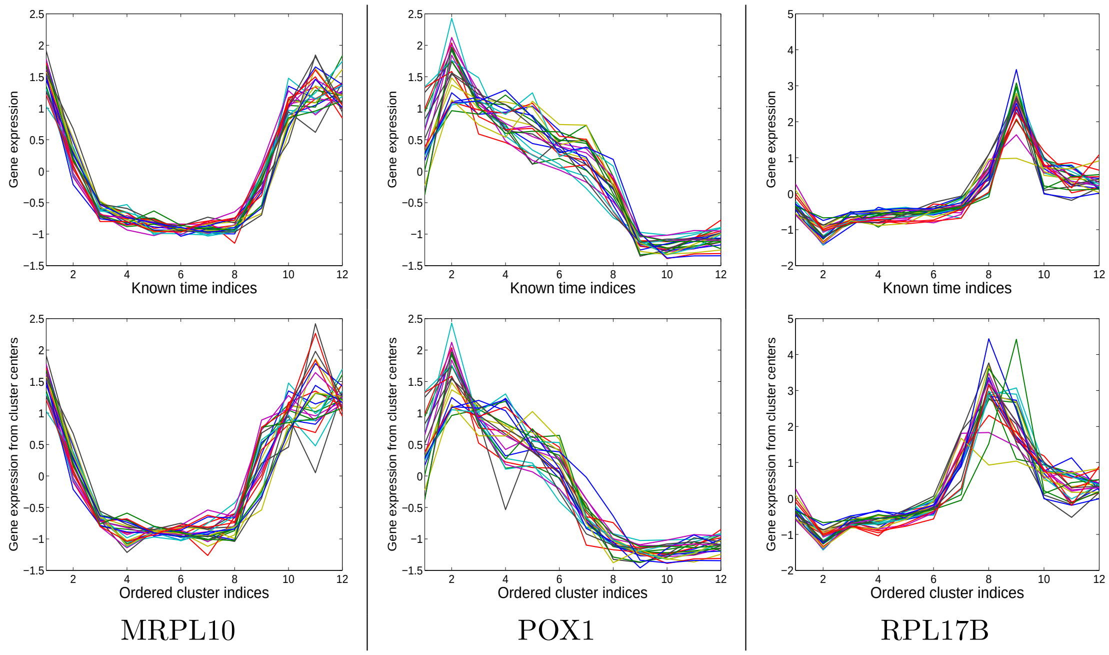


Figure 9: Gene expression profiles in three major gene groups: MRPL10, POX1, and RPL17B. Top row: original gene expression. Bottom row: gene expression from estimated cluster centers ordered by TSM+TSP.

expression profiles over the first two cycles in the top row of Figure 9. In the bottom row of the same figure we plot the expression profiles of the same genes from the estimated cluster centers in the order found by TSM+TSP. Comparing the two rows shows our initial step of temporal clustering and ordering effectively recovers the major trends of gene dynamics.

We then evaluate the proposed methods quantitatively. Figures 14 and 15 in Appendix A present box-plots of cosine scores and normalized residuals. Our first observation, as expected, is that the target model learnt with the known temporal ordering does not perform as well as in the previous experiment, confirming the difficulty of this learning task. However, the improvements due to the proposed EM-based methods over the initial model are more significant here than in the previous experiment, though all of them become further away from the target performance than before. Most of the performance measures here are rather stable across different runs, which is to be expected since the number of sample points here is very small. The only exception is TEM, which occasionally results in extremely poor performance. This is due to numerical difficulties encountered in its E-step; the main computation there is inverting a matrix of exponentiated negative distances, which are numerically unstable for high dimensional data points. Regarding the different ordering methods, unlike in the previous experiment TSM+TSP does not outperform the other ordering methods; all four methods perform equally well. Again, this may be attributed to the training points being too few for different ordering methods to behave differently.

7. Conclusions and Future Directions

We propose and study the problem of learning dynamic models from non-sequenced data. This problem appears in several disciplines, ranging from Astronomy to Biology and Medicine. We formalize a generative process for this problem under the assumption that the true dynamic model is fully observable, linear and first-order Markovian, and point out several identifiability issues in the model learning task. We propose several learning methods based on maximizing approximate log posteriors, and devise simple EM-type optimization algorithms, all of which are generalized for learning nonlinear models. In investigating ways to initialize these EM-type algorithms, we study a relevant but different problem: reconstructing a temporal sequence from out-of-order time series data points, for which we propose a two-step method combining a novel convex program that optimizes temporal smoothness and the traveling sales problem. We evaluate our proposed methods on several synthetic and real-world data sets. Experimental results demonstrate the effectiveness of the proposed methods, but also reveal their limitations.

In the future, we plan to study the theoretical properties of the problem, aiming at identifying rigorous conditions under which the learning task can or cannot be solved, and developing more theoretically sound learning procedures. We also want to investigate the problem in the partially observable setting, which is closer to real-world situations. Lastly, we hope to apply the proposed methods to other real-world data, with the final goal of making scientific discoveries from non-sequenced data.

References

Pieter Abbeel and Andrew Y. Ng. Learning first order Markov models for control. In *Advances in Neural Information Processing Systems 17*, 2005.

David Applegate, Ribert Bixby, Vašek Chvátal, and William Cook. Concorde TSP solver. URL `http://www.tsp.gatech.edu/concorde/index.html`.

Matthew J. Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. The infinite hidden Markov model. In *Advances in Neural Information Processing Systems 14*, pages 577–584, 2002.

Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, New York, 2006.

F. Bock. An algorithm to construct a minimum directed spanning tree in a directed network. In *Developments in Operations Research*, pages 29–44. 1971.

James P. Boyle and Richard L. Dykstra. A method for finding projections onto the intersection of convex sets in Hilbert spaces. *Lecture Notes in Statistics*, 37:28–47, 1986.

P. M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309–312, 1979.

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 272–279, 2008.

J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

Zoubin Ghahramani. Learning dynamic bayesian networks. *Lecture Notes in Computer Science*, 1387:168–197, 1998a.

Zoubin Ghahramani. Variational learning for switching state-space models. *Neural Computation*, 12:963–996, 1998b.

Joachim Giesen. Curve reconstruction in arbitrary dimension and the traveling salesman problem. *Proceedings of the 8th International Conference on Discrete Geometry for Computational Imagery, Lecture Notes in Computer Science*, 1568:164–176, 1999.

Antoine Girard and George J. Pappas. Approximate bisimulations for nonlinear dynamical systems. In *44th IEEE Conference on Decision and Control and European Control Conference*, pages 684–689, 2005.

Amir Globerson, Terry Koo, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Poceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 141–150, 2007.

Anupam Gupta and Ziv Bar-Joseph. Extracting dynamics from static cancer expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5:172–182, 2008.

Trevor Hastie and Werner Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84:502–516, 1989.

Robert Hodrick and Edward C. Prescott. Postwar U.S. business cycles: An empircal investigation. *Journal of Money, Credit, and Banking*, 29:1–16, 1997.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden Markov models. In *Proceedings of the Twenty-Second Annual Conference on Learning Theory*, 2009.

Tzu-Kuo Huang and Jeff Schneider. Learning linear dynamical systems without sequence information. In *Proceedings of the 26th International Conference on Machine Learning*, pages 425–432, 2009.

Tzu-Kuo Huang, Le Song, and Jeff Schneider. Learning nonlinear dynamic models from non-sequenced data. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

C. E. V. Lesser. A simple method of trend construction. *Journal of the Royal Statistical Society, Series B*, 23:91–107, 1961.

Lennart Ljung. *System Identification: Theory for the User.* Prentice Hall, 1999.

Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

Paul M. Magwene, Paul Lizardi, and Junhyong Kim. Reconstructing the temporal ordering of biological samples using microarray data. *Bioinformatics*, 19:842–850, 2003.

Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning.* PhD thesis, University of California at Berkeley, 2002.

Victor A. Nicholson. Matrices with permanent equal to one. *Linear Algebra and its Applications*, 12(2):185–188, 1975.

Jonas Peters, Dominik Janzing, Arthur Gretton, and Bernhard Schölkopf. Detecting the direction of causal time series. In *Proceedings of the 26th International Conference on Machine Learning*, pages 801–808, 2009.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

Sajid M. Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden Markov models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

David A. Smith and Noah A. Smith. Probabilistic models of nonprojective dependency trees. In *Poceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 132–140, 2007.

Alex J. Smola, Sebastian Mika, Bernhard Schölkopf, and Robert C. Williamson. Regularized principal manifolds. *Journal of Machine Learning Research*, 1:179–209, 2001.

Le Song, Byron Boots, Sajid Siddiqi, Geoffrey Gordon, and Alex Smola. Hilbert space embeddings of hidden Markov models. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning*, 2010.

Robert E. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.

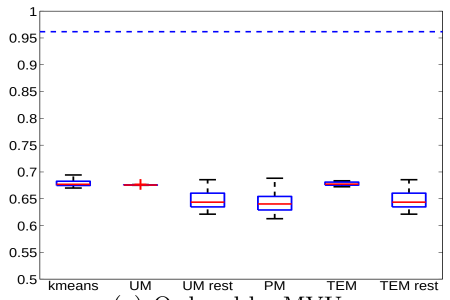
Benjamin P. Tu, Andrzej Kudlicki, Maga Rowicka, and Steven L. McKnight. Logic of the yeast metabolic cycle: Temporal compartmentalization of cellular processes. *Science*, 310 (5751):1152–1158, 2005.

William Thomas Tutte. *Graph Theory.* Addison-Wesley, 1984.

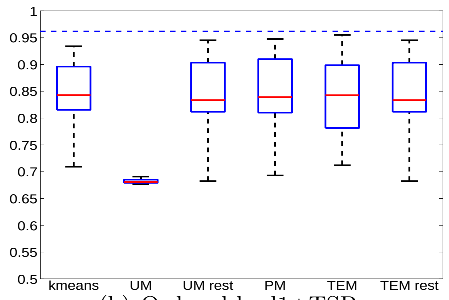
Leslie G. Valiant. The complexity of computing the permanet. *Theoretical Computer Science*, 8(2):189–201, 1979.

Kilian Q. Weinberger, Fei Sha, and Lawrence K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the 21st International Conference on Machine Learning*, pages 839–846, 2004.

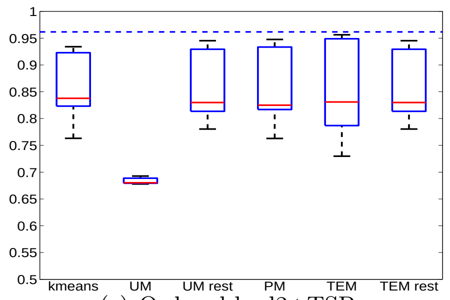
Appendix A. Experimental Results on Real Data



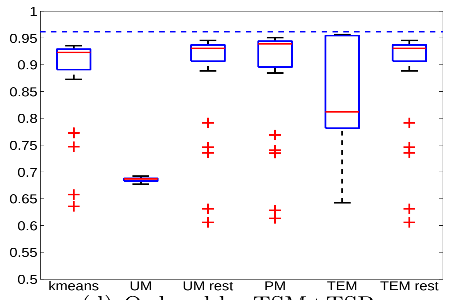
(a) Ordered by MVU



(b) Ordered by l1+TSP

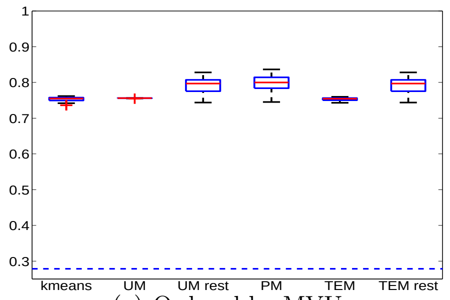


(c) Ordered by l2+TSP

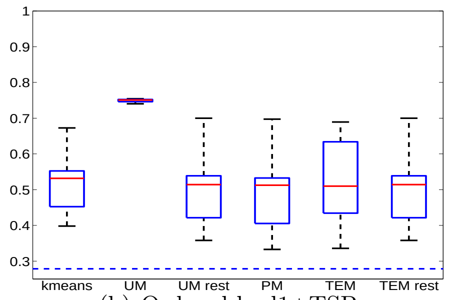


(d) Ordered by TSM+TSP

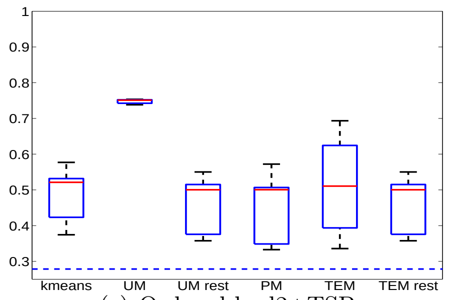
Figure 10: Cosine scores on the pendulum data by the linear model. Larger is better. The blue dashed line is by a dynamic model learnt with the known temporal order.



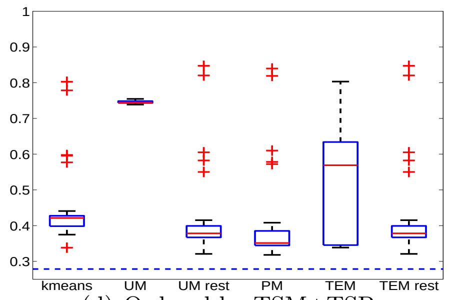
(a) Ordered by MVU



(b) Ordered by l1+TSP

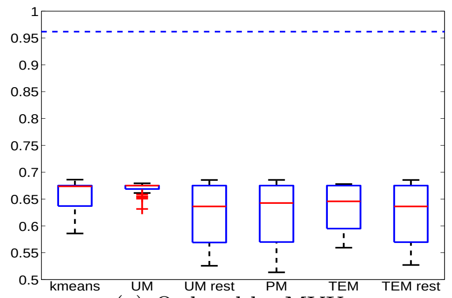


(c) Ordered by l2+TSP

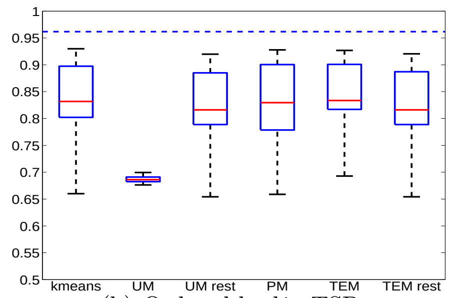


(d) Ordered by TSM+TSP

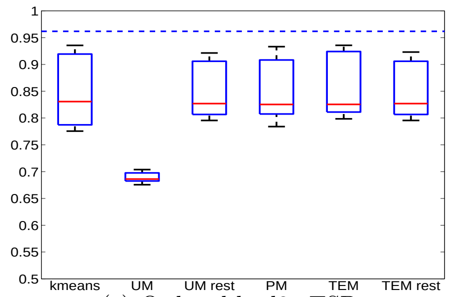
Figure 11: Normalized residuals on the pendulum data by the linear model. Smaller is better. The blue dashed line is by a model learnt with the known order.



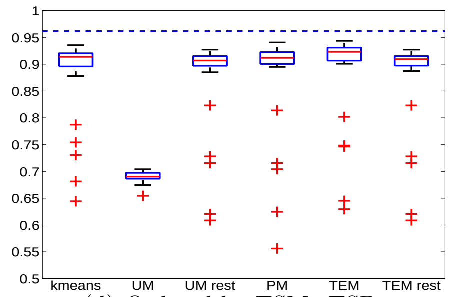
(a) Ordered by MVU



(b) Ordered by l1+TSP

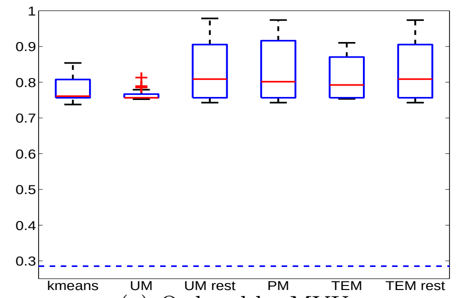


(c) Ordered by l2+TSP

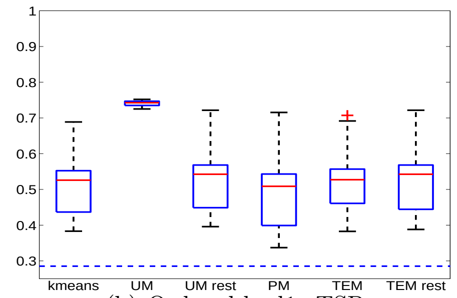


(d) Ordered by TSM+TSP

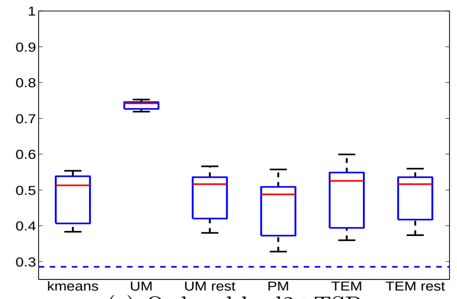
Figure 12: Cosine scores on the pendulum data using the nonlinear model with the Gaussian kernel. Larger is better. The blue dashed line is by a dynamic model learnt using the known temporal order.



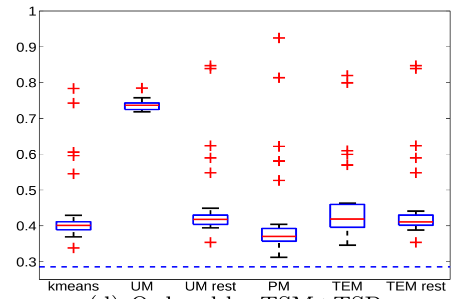
(a) Ordered by MVU



(b) Ordered by l1+TSP

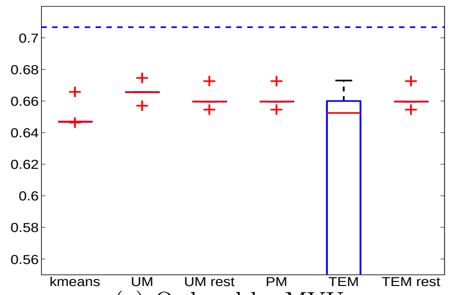


(c) Ordered by l2+TSP

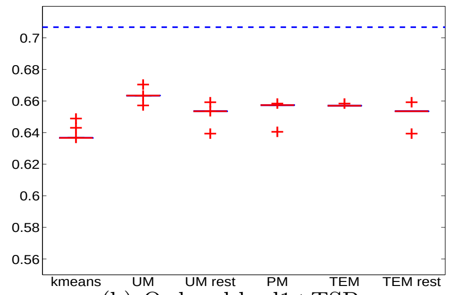


(d) Ordered by TSM+TSP

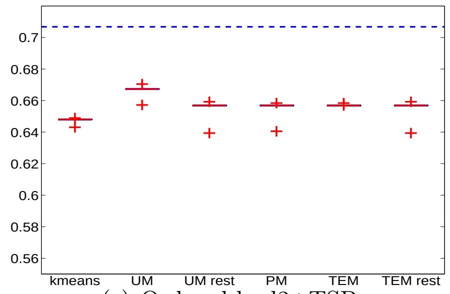
Figure 13: Normalized residuals on the pendulum data using the nonlinear model with the Gaussian kernel. Smaller is better.



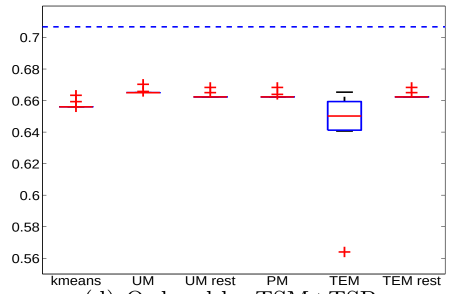
(a) Ordered by MVU



(b) Ordered by l1+TSP

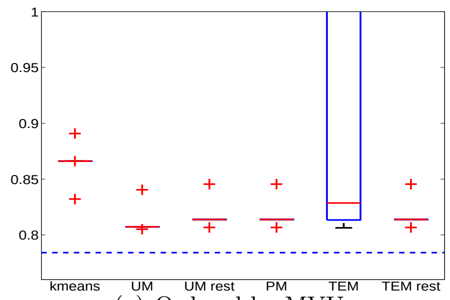


(c) Ordered by l2+TSP

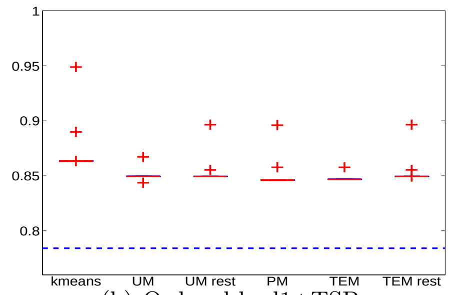


(d) Ordered by TSM+TSP

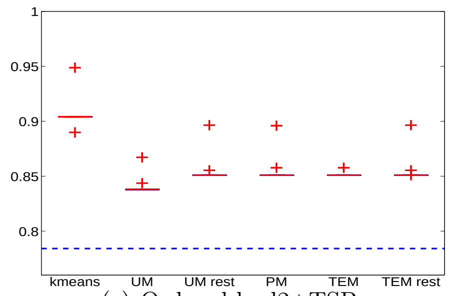
Figure 14: Cosine scores on the yeast time series. Larger is better. The blue dashed line is by a dynamic model learnt using the known temporal order.



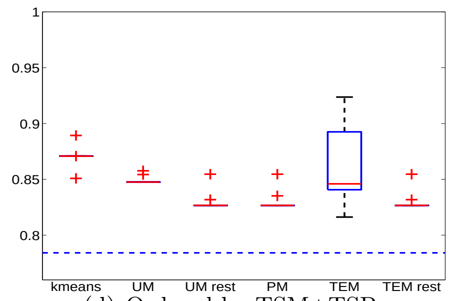
(a) Ordered by MVU



(b) Ordered by l1+TSP



(c) Ordered by l2+TSP



(d) Ordered by TSM+TSP

Figure 15: Normalized residuals on the yeast time series. Smaller is better. The blue dashed line is by a dynamic model learnt using the known temporal order.