Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

Analysis of Tensor Time Series: tensorTS

Rong Chen
Rutgers University

Yuefeng Han
Rutgers University

Zebang Li
Rutgers University

Han Xiao
Rutgers University

Dan Yang
The University of Hong Kong

Ruofan Yu
Rutgers University

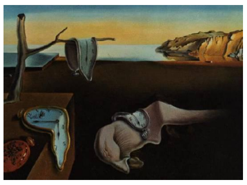
Abstract

Tensor and matrix time series data have been amassed more and more from many areas in recent years, calling for new statistical models, methods and algorithms for analyzing such data. Statistical tools for tensor time series have been developed in two major directions: autoregressive modeling and factor modeling. In both directions, a key idea is to make use of the tensor structure and adopt a multi-linear form in the model formulation. The corresponding models have advantages in terms of dimension reduction, interpretability and computing. This paper introduces a R package **tensorTS** which implements the methodologies proposed in a series of recent papers, including functions for estimation, model selection, prediction and visualization for both tensor autoregressive models and dynamic tensor factor models. The usage of the functions and the effectiveness of the proposed models are demonstrated by an example of the NYC taxi data.

Keywords: autoregressive models, factor models, model selection, number of factors, reduced rank, Tucker decomposition.

1. Introduction

Due to the advancement of computing power in the past decade, large amount of complex data are generated and collected over time in a wide range of fields such as economics, finance, biology, engineering, social science, among others. Often, at each time point, the observations routinely and naturally arise in the form of multi-dimensional arrays, or tensors, when there are multiple classifications for each observation. Such tensor observations are often observed over time, forming *tensor time series*, which is an extension of the classical mutilvariate time series (in vector form). For instance, a set of economic indicators of a group of countries



result in a matrix (order-2 tensor) time series (Wang, Liu, and Chen 2019), the import-export volume of multiple categories among several countries generate an order-3 tensor time series (Chen, Yang, and Zhang 2022b), the evolution of social network in different pathways yields an order-3 tensor time series (Goldenberg, Zheng, Fienberg, Airoldi *et al.* 2010; Hanneke, Fu, and Xing 2010; Snijders 2001; Kolaczyk and Csárdi 2014; Ji and Jin 2016; Zhao, Levina, and Zhu 2012; Phan and Airoldi 2015), and many more.

The omnipresence of tensor time series poses pressing need to develop state-of-the-art methodologies and software/package to analyze such data, explore the dependency of observations over time, understand the dynamic of the underlying process, and make accurate predictions. The new R package **tensorTS** is created timely for this purpose. It is available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=tensorTS`.

The study of tensors with independent and identically distributed (iid) assumption has been demonstrated to be powerful and versatile at the confluence of statistics, computer science, machine learning, and signal processing. Research areas range from tensor completion (Liu, Musialski, Wonka, and Ye 2012; Yuan and Zhang 2016; Zhang 2019; Xia, Yuan, and Zhang 2021; Yuan and Zhang 2017; Xia and Yuan 2019), tensor decomposition (Kolda and Bader 2009; Sun, Lu, Liu, and Cheng 2017; Anandkumar, Ge, and Janzamin 2014; Sidiropoulos, De Lathauwer, Fu, Huang, Papalexakis, and Faloutsos 2017; Liu, Shang, Fan, Cheng, and Cheng 2014; Liu, Yuan, and Zhao 2022), tensor denoise (Xia and Zhou 2019), tensor regression (Lock 2018; Zhang, Luo, Raskutti, and Yuan 2020; Raskutti, Yuan, and Chen 2019; Chen, Raskutti, and Yuan 2019; Zhou, Li, and Zhu 2013; Hoff 2015).

There are many statistical software/packages available to analyze tensor data under various models and objectives. The R packages **rTensor** (Li, Bien, and Wells 2021, 2018) and **tensor** (Rougier 2012) provide tensor operations and decomposition methods. **PTAk** (Leibovici 2021, 2010) supports CANDECOMP/PARAFAC(CP) and Tucker decompositions, and generalizes singular value decomposition (SVD) of matrices to tensors. Package **tensorr** (Zamora 2019) provides methods to manipulate and store sparse tensors. Package **tensorA** (van den Boogaart 2020) provides Einstein and Riemann summation conventions as well as some tensor algebra. Package **nnTensor** (Tsuyuzaki, Ishii, and Nikaido 2021; Cichocki, Zdunek, Phan, and Amari 2009) is for non-negative matrix factorization, non-negative CP and Tucker decompositions. Package **tensorBF** (Khan *et al.* 2016) deals with Bayesian tensor factorization. Package **tensr** (Gerard and Hoff 2018) contains a collection of functions for the estimation and testing of covariance with Kronecker structure as well as the manipulation and decomposition of tensor data. **MultiwayRegression** performs tensor-on-tensor regression (Lock 2019, 2018).

Conventional multivariate time series analysis has also been studied extensively, with a rising focus on high-dimensional time series. Recent research on high dimensional time series can be roughly divided into two categories: vector auto-regressive (VAR) models with regularization (Basu and Michailidis 2015; Davis, Zang, and Zheng 2016; Han, Lu, and Liu 2015; Kock and Callot 2015; Lin and Michailidis 2017; Loh and Wainwright 2011; Melnyk and Banerjee 2016; Nicholson, Matteson, and Bien 2017) and dynamic vector factor models (VFM) (Tiao and Tsay 1989; Engle and Kroner 1995; Forni, Hallin, Lippi, and Reichlin 2000; Stock and Watson 2016; Fan, Liao, and Mincheva 2013; Pena and Box 1987; Lam and Yao 2012). Both categories can achieve effective dimension reduction: VAR with regularization results in sparse coefficient matrix of a smaller number of parameters; VFM assumes low rank structure to extract common information embedded in a small number of factors.

To analyze multivariate time series analysis in low dimensions, there exist many statistical software/packages with classical autoregressive and moving average approaches. **MTS** (Tsay and Wood 2021) is a comprehensive package for analyzing multivariate linear time series, including functions for VAR and vector autoregressive and integrated moving average (VARIMA) models, co-integration analysis, multivariate volatility models, factor models, among many other functions; see Tsay (2013) for more detail. For VAR models, **mAr** (Barbosa 2012) provides step-wise least squares estimation. Package **sparsevar** (Vazzoler 2021; Basu and Michailidis 2015) implements the sparse VAR and sparse vector error correction models (VECM). **BigVAR** (Nicholson, Matteson, and Bien 2019; Nicholson, Wilms, Bien, and Matteson 2020) and **bigtime** (Wilms, Matteson, Bien, Basu, Nicholson, and Wegner 2021b; Wilms, Basu, Bien, and Matteson 2021a) consider VAR and vector autoregressive with exogenous variable (VARX) models with structured Lasso penalties. Package **svars** (Lange, Dalheimer, Herwartz, Maxand, and Riebl 2022; Lange, Dalheimer, Herwartz, and Maxand 2021) implements data-driven model specifications for structural VAR models. **VARshrink** (Lee, Yang, and Kim 2019) provides shrinkage methods for VAR models, and **dse** (Gilbert 2020) focuses on VARIMA models and state space models. VECM is also available through the packages **urca** (Pfaff, Zivot, and Stigler 2016; Pfaff 2008), **ecm** (Bansal 2021), **vars** (Pfaff and Stigler 2021; Pfaff 2008), and **tsDyn** (Fabio Di Narzo, Aznarte, and Stigler 2009). Network time series analysis is carried out by **GNAR** (Leeming, Nason, Nunes, and Knight 2020; Knight, Leeming, Nason, and Nunes 2020), and graphical models are implemented by **graphicalVAR** (Epskamp and Asena 2021; Epskamp, Waldorp, Mõttus, and Borsboom 2018) and **mgm** (Haslbeck 2021; Haslbeck and Waldorp 2020). Functions for Bayesian VAR models (Chan, Koop, Poirier, and Tobias 2019; Koop and Korobilis 2010; Lütkepohl 2005) are provided in **bvartools** (Mohr 2022), and **BMTAR** (Salcedo, Villanueva, and Torres 2021) implements Baysian multivariate threshold AR models with missing data. Package **mfbvar** (Ankargren, Yang, and Kastner 2021) includes tools for estimating mixed-frequency Bayesian and state space-based VAR models. **BVAR** (Kuschnig, Vashold, McCracken, and Ng 2022; Kuschnig and Vashold 2021) provides a toolkit for hierarchical Bayesian VAR models. **BGVAR** (Böck, Feldkircher, and Huber 2021) implements Bayesian Global VAR models.

To analyze multivariate time series analysis in high dimensions, many statistical software and packages have been developed via factor models, principal component analysis (PCA), or state space models. For example, **ForeCA** (Goerg 2020), **PCA4TS** (Chang, Guo, and Yao 2015) and **HDTSA** (Lin, Cheng, Chang, and Yao 2021) provide various methods for PCA on vector time series, including estimation, rank determination and several inferential tools. Package **odpc** (Peña, Smucler, and Yohai 2022) computes one-sided dynamic PCA, **freqdom** (S. and L. 2022) implements dynamic PCA in frequency domain, **tsBSS** (Matilainen, Croux, Miettinen, Nordhausen, Oja, Taskinen, and Virta 2021) provides blind source separation and supervised dimension reduction. State space models are available via the packages **KFAS** (Helske 2021, 2017), **FKF** (Luethi, Erb, Otziger, McDonald, and Smith 2021), **FKF.SP** (Aspinall, Gepp, Harris, Kelly, Southam, Vanstone, Luethi, Erb, Otziger, and Smith 2021), **dlm** (Petris and Gilks 2018; Petris 2010), **mssm** (Christoffersen and Williams 2022), **MARSS** (Holmes, Ward, Scheuerell, and Wills 2021; Holmes, Ward, and Wills 2012), and **mbsts** (Qiu and Ning 2021).

However, there have been no packages available to study autoregressive or factor models for tensor-valued time series in either low or high dimensions. The analysis of tensor time series is very different from the traditional tensor analysis with iid observations, and is also very different from vector time series analysis. There are two ad hoc ways to convert tensor time series into something that can be analyzed via existing methods. The first one is to treat time as an additional mode of the tensor, i.e. combine all observations from different time points to form a single tensor with one extra mode, and then apply standard tensor analysis tools. Such an approach does not reveal the dynamic nature of the time series, and does not in general meet the objectives of time series analysis. The second one is to vectorize the tensor observation at each time point into a long vector, and use the standard tools of vector time series analysis. But such an approach does not fully utilize the grouping information (each mode corresponds to a grouping of the individual series) provided in the tensor structure. Moreover, it is much more difficult to interpret the results from such an approach.

The recent development of analyzing tensor time series that preserves both the tensor structure and the time series structure includes two approaches: tensor auto-regressive (TenAR) models and dynamic tensor factor models (TenFM). The commonality of the proposed models in these two approaches is the adoption of the multilinear form, which can achieve substantial dimension reduction and admit interpretations adherent to each mode of the tensor. Such a multilinear form is also frequently utilized for statistical tensor analysis under iid assumptions.

Regarding TenAR models, Hoff (2015) and Chen, Xiao, and Yang (2021) pioneer the study of autoregressive model for matrix time series (MAR) by introducing the bilinear autoregression. Xiao, Han, Chen, and Liu (2021) considers the MAR model with low rank coefficient matrices. Li and Xiao (2021) extends Chen *et al.* (2021) to tensor time series, also allowing multiple lags and multiple terms for each lag in the autoregression. Wang, Zheng, and Li (2021) considers a full tensor autoregressive model which is equivalent to the VAR (after the vectorization), and simplifies the model by imposing low rank constraint on the coefficient tensor.

For TenFM model under the assumption that the noise process is white, Wang *et al.* (2019) initiates the research of matrix factor model using SVD of the outer product of the lagged matrix columns/rows. In a recent breakthrough, Chen *et al.* (2022b) considers the tensor factor models and proposes two non-iterative estimators based on the inner and outer products, named TIPUP and TOPUP estimation procedures, respectively. Han, Chen, Yang, and Zhang (2020) further improves the two estimators through iterative projections. Han, Chen, and Zhang (2022) proposes two methods for the determination of the number of factors, which is a fundamental problem in factor analysis.

There are other works on extensions and other aspects of matrix and tensor time series models, including, among others, Chen, Tsay, and Chen (2020a), Chen, Xia, Cai, and Fan (2020b) and Yu, He, Kong, and Zhang (2022), though these methods are not covered by the package **tensorTS**.

The goal of this article is to introduce the R package **tensorTS**, which implements the models/methods/algorithms proposed in the authors' series of recent works on TenAR and TenFM, including Chen *et al.* (2021); Li and Xiao (2021); Xiao *et al.* (2021); Chen *et al.* (2022b); Han *et al.* (2020), and Han *et al.* (2022). It highlights the capability of the software, and demonstrates its usage with a case study. Some major functions in package **tensorTS** are listed in Table 1. The complete list of functions and their brief descriptions can be found in Appendix A. For more details of each function, see the online manual of the package also available at https://cran.r-project.org/package=tensorTS.

The rest of this article is organized as follows. Section 2 introduces the TenAR models, the reduced rank matrix autoregressive model, their model estimation and model selection methods, and demonstrates the usage of the associated functions in the package **tensorTS** with

Function Name	Usage	Section
`tenAR.est`	Estimation of the tensor autoregressive model	2.2
`matAR.RR.est`	Estimation of the matrix reduced rank autoregressive model	2.3
`tenFM.est`	Estimation of the tensor factor model	3.4
`tenFM.rank`	Rank determination of the tensor factor model	3.4

Table 1: Major functions in package **tensorTS** for the autoregressive and factor models for tensor time series.

examples. Section 3 introduces the TenFM model, the methods for its estimation and rank determination, and the relevant functions in **tensorTS** with examples. Section 4 summarizes. Appendix A provides the complete list of functions in **tensorTS** with short descriptions. The code to download and pre-process the taxi data used in the examples is given in Appendix B. The collection of all the code in Sections 2 and 3 for the analysis of the taxi data is given in Appendix C.

Notation. Throughout this article, we make the convention that uppercase letters in boldface denote matrices, lowercase letters in boldface are vectors, and script letters symbolize tensors of order 3 or higher. We use $\otimes$ to denote the tensor product, $\odot$ the Kronecker product, and $\times_k$ the product of a tensor and a matrix along mode-k. For definitions of these products and other basic tensor operations (vectorization, unfolding etc), we refer the readers to Kolda and Bader (2009).

2. Tensor Autoregressive Models

2.1. Tensor Autoregressive Models

Chen *et al.* (2021) considers the matrix autoregressive models of order 1, denoted by MAR(1), in the following bilinear form

$$\boldsymbol{X}_t = \boldsymbol{A}_1 \boldsymbol{X}_{t-1} \boldsymbol{A}_2^\top + \boldsymbol{E}_t, \tag{1}$$

where $\boldsymbol{X}_t$ is a $d_1 \times d_2$ matrix observed at time t, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are $d_1 \times d_1$ and $d_2 \times d_2$ autoregressive coefficient matrices respectively, and $\boldsymbol{E}_t$ is a $d_1 \times d_2$ matrix white noise.

Tensors are multi-dimensional arrays, naturally extending and including matrices as a special case. Li and Xiao (2021) extends the MAR(1) model to multi-linear tensor autoregressive models. Consider a tensor time series $\{\mathcal{X}_t\}$, where at each time t, an order-K tensor $\mathcal{X}_t \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_K}$ is observed. The tensor autoregressive model of order p, denoted by TenAR(p) (or more precisely TenAR($p, R_1, \dots, R_p$), indicating there are R_i multilinear terms for lag i), has the form

$$\mathcal{X}_t = \sum_{i=1}^{p} \sum_{r=1}^{R_i} \mathcal{X}_{t-i} \times_1 \boldsymbol{A}_1^{(ir)} \times_2 \cdots \times_K \boldsymbol{A}_K^{(ir)} + \mathcal{E}_t. \tag{2}$$

where $\boldsymbol{A}_k^{(ir)} \in \mathbb{R}^{d_k \times d_k}$ is the coefficient matrix associated with mode k for the r-th term at lag i, $\mathcal{E}_t \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_K}$ is a tensor white noise process satisfying $\mathrm{Cov}(\mathcal{E}_t, \mathcal{E}_s) = \mathbf{0}$ whenever

$s \neq t$, and p is the autoregressive order. After vectorization by stacking all mode-1 fibers, model (2) becomes

$$\mathrm{vec}(\mathcal{X}_t) = \sum_{i=1}^{p} \mathbf{\Phi}_i \mathrm{vec}(\mathcal{X}_{t-i}) + \mathrm{vec}(\mathcal{E}_t), \tag{3}$$

where

$$\mathbf{\Phi}_i = \sum_{r=1}^{R_i} \left[\odot_{k=K}^{1} \boldsymbol{A}_k^{(ir)} \right]. \tag{4}$$

Here, $\odot$ denotes the Kronecker product of two matrices, and $\odot_{k=K}^{1} \boldsymbol{A}_k^{(ir)} := \boldsymbol{A}_K^{(ir)} \odot \boldsymbol{A}_{K-1}^{(ir)} \odot \cdots \odot \boldsymbol{A}_1^{(ir)}$. In view of (3), the TenAR model can be perceived as a VAR whose coefficient matrices bear the form of the sum of a few Kronecker products. We note that after rearrangement of the entries, (4) corresponds to a tensor CP decomposition of rearranged $\mathbf{\Phi}$, see Li and Xiao (2021) for more details. In fact, if R_i is sufficiently large, the sum in (4) can represent any $d \times d$ matrix, and the model (3) will become a VAR model without any restriction. On the other hand, when R_i are small, significant dimension reduction is achieved by model (2), compared with the unrestricted VAR model. The TenAR model allows the user to specify the R_i, and thus provides flexibility and capability of capturing the dynamics. We will refer to R_i as the *Kronecker rank* of $\mathbf{\Phi}_i$ in the sequel.

In the R package **tensorTS**, estimation for TenAR models is carried out by the function `tenAR.est`, with options to use one of three estimation methods: projection (PROJ), least squares (LSE), and maximum likelihood (MLE).

The projection method first estimates the coefficient matrices $\mathbf{\Phi}_i$'s in the vectorized AR model (3) without restrictions, and then estimates $\boldsymbol{A}_k^{(ir)}$ by projecting $\hat{\mathbf{\Phi}}_i$ onto the space of sums of Kronecker products (see (4)) under the Frobenius norm. After rearrangement of entries, this boils down to finding the best rank-R_i approximation of the rearranged $\hat{\mathbf{\Phi}}_i$. Again, see Li and Xiao (2021) for more details. PROJ is less efficient than LSE and MLE and requires a larger sample size to obtain $\hat{\mathbf{\Phi}}_i$, but it can be used as an initial value for the other two methods.

The least squares estimators are the solutions of the least squares problem

$$\left(\hat{\boldsymbol{A}}_1^{(11)}, \cdots, \hat{\boldsymbol{A}}_K^{(pR_p)} \right) = \underset{\boldsymbol{A}_1^{(11)}, \cdots, \boldsymbol{A}_K^{(pR_p)}}{\arg\min} \sum_t \left\| \mathcal{X}_t - \sum_{i=1}^{p} \sum_{r=1}^{R_i} \mathcal{X}_{t-i} \times_1 \boldsymbol{A}_1^{(ir)} \times_2 \cdots \times_K \boldsymbol{A}_K^{(ir)} \right\|_F^2 .$$

An iterative procedure is used. It updates one of the coefficient matrices while holding the others fixed in each iteration, using the PROJ estimators as the initial values.

The maximum likelihood method is based on the additional assumption that $\mathcal{E}_t$ is normal with a separable covariance structure $\mathrm{Cov}(\mathrm{vec}(\mathcal{E}_t)) = \Sigma_K \odot \Sigma_{K-1} \odot \cdots \odot \Sigma_1$. Each Σ_k is a $d_k \times d_k$ symmetric positive definite matrix, corresponding to the covariance matrix along mode k of $\mathcal{E}_t$. In this case, the error tensor process can be represented equivalently as

$$\mathcal{E}_t = \mathcal{Z}_t \times_1 \Sigma_1^{1/2} \ldots \times_K \Sigma_K^{1/2}, \tag{5}$$

where $\mathcal{Z}_t$ has iid standard normal elements. Under such an assumption, the corresponding log likelihood can be maximized through an alternating algorithm. Specifically, the algorithms updates one matrix of the set $\{\boldsymbol{A}_k^{(ir)}, \Sigma_k : 1 \leqslant i \leqslant p, 1 \leqslant r \leqslant R_i, 1 \leqslant k \leqslant K\}$ while holding others fixed, and iterates until convergence. We skip the formula of the log likelihood and the details of the algorithm which are quite complicated for tensor data. For more technical details, including theoretical properties of the estimators, see Li and Xiao (2021).

In **tensorTS**, an extended Bayesian information criterion (EBIC) is provided to facilitate model selection of the AR order p and the Kronecker ranks R_i, $i = 1, \ldots, p$. Specifically, for any given order p and $\boldsymbol{R}_p := (R_1, \ldots, R_p)$, define the information criterion as

$$\mathrm{EBIC}(p; \boldsymbol{R}_p) := \frac{1}{2} \log \left(\frac{1}{dT} \sum_t \left\| \mathcal{X}_t - \sum_{i=1}^{p} \sum_{r=1}^{R_i} \mathcal{X}_{t-i} \times_1 \hat{\boldsymbol{A}}_1^{(ir)} \times_2 \cdots \times_K \hat{\boldsymbol{A}}_K^{(ir)} \right\|_F^2 \right) + g(d,T) \sum_{i=1}^{p} R_i, \tag{6}$$

where $\hat{\boldsymbol{A}}_k^{(ir)}$ are the estimates obtained with given p and $\boldsymbol{R}_p$, and $g(d, T)$ is the penalty function. It is shown in Li and Xiao (2021) that this criterion selects the p and $\boldsymbol{R}_p$ consistently under both fixed and diverging dimension setups, as long as $g(d, T) \to 0$ and $\frac{T}{d} g(d, T) \to \infty$ as $T \to \infty$. In **tensorTS**, the penalty function $g(d, T)$ is chosen as $\log(T)/T$.

The function `tenAR.est` allows users to specify other parameters, including the maximum number of iterations (default: `niter=150`), the error tolerance in terms of the Frobenius norm (default: `tol=1e-4`), and whether to print the number of iterations (default: `print.true=FALSE`). The components of the output list of the function `tenAR.est` are summarized in Table 2.

Value	Details
`A`	a list of estimated coefficient matrices $\hat{\boldsymbol{A}}_k^{(ir)}$, such that `A[[i]][[r]][[k]]` $= \hat{\boldsymbol{A}}_k^{(ir)}$.
`SIGMA`	only if `method=MLE`, a list of estimated $\hat{\Sigma}_1, \ldots, \hat{\Sigma}_K$.
`res`	residuals.
`Sig`	sample covariance matrix of the residuals $\mathrm{vec}(\hat{\mathcal{E}}_t)$.
`cov`	grand covariance matrix of all entries of all $\hat{\boldsymbol{A}}_k^{(ir)}$.
`sd`	standard errors of $\hat{\boldsymbol{A}}_k^{(ir)}$, returned as a list aligned with `A`.
`niter`	number of iterations.
`BIC`	value of extended Bayesian information criterion.

Table 2: Components of the output list of the function `tenAR.est`.

Prediction under a TenAR model is similar to the univariate and vector AR models. Specifically, the best linear h step prediction of $\mathcal{X}_{t+h}$, based on $\mathcal{X}_1, \ldots, \mathcal{X}_t$, can be obtained recursively using

$$\hat{\mathcal{X}}_t(h) = \sum_{i=1}^{p} \sum_{r=1}^{R_i} \hat{\mathcal{X}}_t(h-i) \times_1 \hat{\boldsymbol{A}}_1^{(ir)} \times_2 \cdots \times_K \hat{\boldsymbol{A}}_K^{(ir)},$$

where $\hat{\mathcal{X}}_t(h-i) = \mathcal{X}_{t+h-i}$ if $h \leqslant i$. The prediction standard errors can also be calculated recursively based on the estimated model parameters. In the package **tensorTS**, the predictions are implemented by the function `tenAR.predict`, which also provides rolling forecasts if `rolling=TRUE` (default: `rolling=FALSE`). Users can specify the prediction horizon `n.ahead` $= h$ (default: `n.ahead=1`), and the starting point of rolling forecast `n0` if rolling forecasting is requested.

2.2. An Example for Tensor Autoregressive Model

We use New York taxi traffic data, maintained by the Taxi & Limousine Commission of New York City (available at https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page) to demonstrate the main functions of the package **tensorTS**, especially about how to understand and visualize the output. In this example, each $\mathcal{X}_t = \{\mathcal{X}_{t,ijk}\}$ is an order-three tensor for day t, with element $\mathcal{X}_{t,ijk}$ representing the number of taxi rides from pick-up region i to drop-off region j, during hour k. We select five pick-up and drop-off regions around Midtown Center, Midtown East, Midtown North, Penn Station and Times Square, and seven hours between 9am to 4pm (these are business hours between the morning and afternoon traffic peaks), on the business days from January 1, 2017 to December 31, 2019. Thus, the data is a tensor time series of length $T = 754$ and each $\mathcal{X}_t$ is a $5 \times 5 \times 7$ tensor. Due to the impact of the emergence of shared ride programs such as Uber and LYft after 2015, the original data exhibit persistent downward trend. So we estimate the trend of each individual series by exponential smoothing and then remove it from the original series. Data after prepossessing can be obtained using the code in Appendix B and is directly loaded here by the following command:

```
R> xx = load('tenAR_taxi.RData')
R> dim(xx)
[1] 754   5   5   7
```

It can be seen that the tensor data `xx` is a order-4 '`array`' object. Note that the functions in **tensorTS** assume that the first mode of the input data is the time mode. The observed tensor time series can be visualized using the function `mplot` which plots a matrix slice of the tensor-valued time series. For example, the first 100 observations of the taxi traffic data among the five locations and in the last hour (3pm - 4pm) can be plotted using

```
R> mplot(xx[1:100,,,7])
```

The resulting figure is shown in Figure 1. The (i, j)-th sub-figure shows the univariate time series of taxi traffic volume from region i to region j during 3pm–4pm for the first 100 business days. It can be seen that the de-trended time series are relatively stationary, but of different scales at different locations.
The command

```
R> set.seed(123)
R> est = tenAR.est(xx, R=2, P=1, method="MLE")
R> ## The following line fits a TenAR(2) with R1 = 2, R2 = 3.
R> ## est = tenAR.est(xx, R=c(2,3), P=2, method="MLE")
```

estimates a TenAR(1) model with two terms ($R_1 = 2$), using MLE method. The function can fit any TenAR(p) model with number of terms $\boldsymbol{R}_p = (R_1, \ldots, R_p)$. For example, a TenAR(2) model with $\boldsymbol{R}_2 = (2, 3)$ can be fitted using the last line above. The returned object `est` is a list containing many output components. In particular, the estimated coefficient matrices can be extracted by

```
R> A = est$A
```

The object `A` is a multi-layer list containing the estimated coefficients $\hat{\boldsymbol{A}}_k^{(ir)}$, the first layer for the AR lag $1 \leqslant i \leqslant P$, the second for the term $1 \leqslant r \leqslant R_i$, and the third for the tensor mode $1 \leqslant k \leqslant K$. In this example, we have $P = 1$, $R_1 = 2$ and $K = 3$. The following commands (with output) display the values of P, R_1, K:

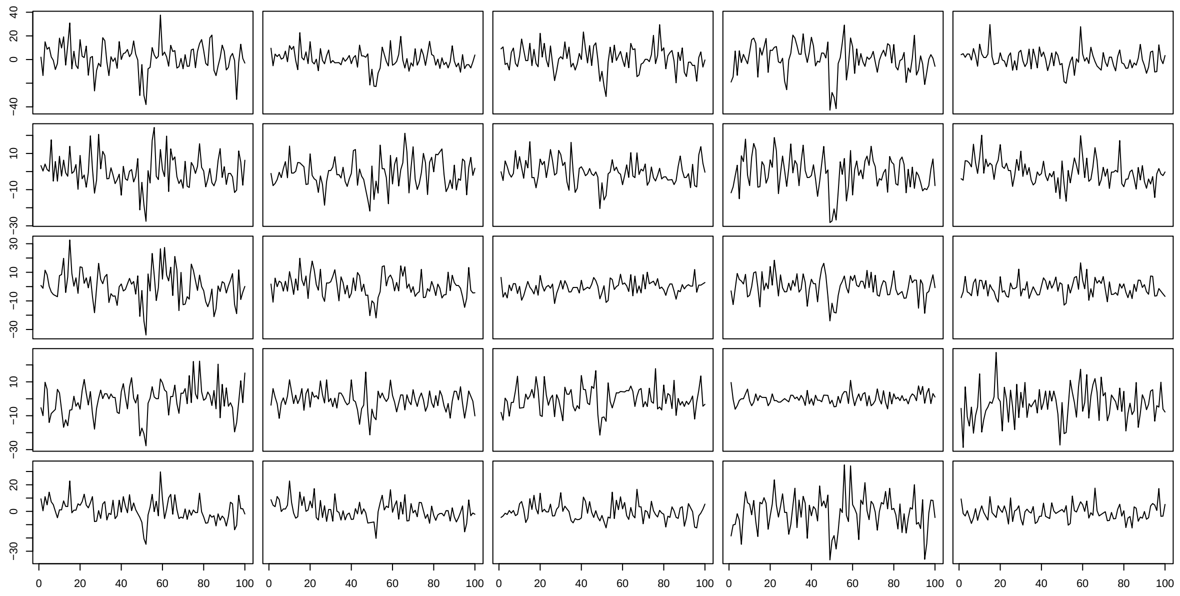


Figure 1: Time series plot by the function `mplot` for the first 100 observations of the taxi data among five pick-up and drop-off locations during 3pm - 4pm of business days from January 1, 2017 to December 31, 2019.

```
R> length(A) == 1 # order P = 1
[1] TRUE
R> length(A[[1]]) == 2 # number of terms R = 2
[1] TRUE
R> length(A[[1]][[1]]) == 3 # mode K = 3
[1] TRUE
```

The estimated coefficient matrix $\hat{\boldsymbol{A}}_k^{(ir)}$ is therefore the `[[i]][[r]][[k]]`-th element of the list. For example, $\hat{\boldsymbol{A}}_3^{(12)}$, which is a matrix of size $d_3 \times d_3$, where $d_3 = 7$, can be obtained by

```
R> A[[1]][[2]][[3]]
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
[1,] -1.005974 -0.119605  0.033134 -0.062632 -0.089681  0.006976 -0.045493
[2,] -0.143052 -0.781482 -0.080859 -0.072773 -0.127773 -0.066241 -0.249055
[3,] -0.063311 -0.157130 -0.627007 -0.115772 -0.052147 -0.100399 -0.332964
[4,] -0.225480 -0.079807 -0.096239 -0.413157 -0.104122 -0.084577 -0.343035
[5,] -0.067741 -0.076032 -0.098737 -0.088475 -0.486144 -0.122372 -0.303149
[6,] -0.082926 -0.042373 -0.099061  0.006823 -0.044823 -0.476063 -0.369031
[7,]  0.005552 -0.065352 -0.021277 -0.013186 -0.139197 -0.116763 -0.604108
```

The element-wise standard errors of the estimated coefficient matrices $\hat{\boldsymbol{A}}_k^{(ir)}$ are also included in the output of the function `tenAR.est`. For example, the standard errors of the $\hat{\boldsymbol{A}}_3^{(12)}$ shown above are,

```
R> sd = est$sd
R> sd[[1]][[2]][[3]]
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
[1,] 0.112137 0.116964 0.267138 0.175703 0.074884 0.093301 0.090958
[2,] 0.082195 0.116622 0.171140 0.150899 0.067578 0.152612 0.125235
[3,] 0.124765 0.101260 0.151318 0.169161 0.068641 0.154194 0.111107
[4,] 0.154409 0.145163 0.156418 0.182069 0.108255 0.138389 0.082096
[5,] 0.195911 0.208182 0.125951 0.155391 0.094502 0.188283 0.097203
[6,] 0.173500 0.279382 0.138692 0.123278 0.103372 0.213234 0.102576
[7,] 0.193864 0.195102 0.152109 0.173321 0.083526 0.190937 0.164743
```

When the error process $\mathcal{E}_t$ is assumed to have the separable structure (5) and the option `method="MLE"` is used in `tenAR.est`, its output object also contains estimated $\hat{\Sigma}_i$, $i = 1, \ldots, K$ in the list `SIGMA`. For example, the $\hat{\Sigma}_2$ is given by,

```
R> Sigma = est$SIGMA
R> Sigma[[2]]
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.805917 0.158538 0.128962 0.049164 0.147387
[2,] 0.158538 0.469409 0.080079 0.023978 0.084042
[3,] 0.128962 0.080079 0.403256 0.029937 0.093488
[4,] 0.049164 0.023978 0.029937 0.368991 0.031961
[5,] 0.147387 0.084042 0.093488 0.031961 0.545927
```

The residuals of the estimated model, a tensor of the same dimension as the input data, is also reported by `tenAR.est`. Residual-based diagnostics can then be performed for model selection and validation. In our example, the residuals are given by

```
R> residuals = est$res
R> dim(residuals)
[1] 753   5   5   7
```

The function `mplot.acf` provides the ACF plot of a matrix slice of the tensor-valued time series. For example, the ACF plot of the last hour of the residuals can be plotted using

```
R> mplot.acf(residuals[,,,7])
```

and is given in Figure 2. It is seen that the two-term TenAR(1) model is able to capture the serial correlations and leads to relatively clean ACF plots of the residuals.

The $(d_1 \ldots d_K) \times (d_1 \ldots d_K)$ sample covariance matrix of the vectorized residual process (in this example, a 175×175 matrix, where $175 = 5 \times 5 \times 7$) is reported as `est$Sig`. The `est$cov` is the grand covariance matrix of

$$\left\{\sqrt{T} \cdot \operatorname{vec}\left[\hat{\boldsymbol{A}}_k^{(ir)} - \boldsymbol{A}_k^{(ir)}\right] : \quad 1 \leqslant i \leqslant p,\ 1 \leqslant r \leqslant R_i,\ 1 \leqslant k \leqslant K\right\},$$

where the ordering over the triplet (i, r, k) is lexicographic. For this example, `est$cov` is a 198×198 matrix, where $198 = 2 \cdot (5^2 + 5^2 + 7^2)$. They can be used to make joint inference, or to obtain prediction intervals if needed.

The value of the extended Bayesian information criterion (6) with $g(d, T) = \log T/T$ is also part of the `tenAR.est` output:

```
R> est$BIC
[1] 2.016997
```

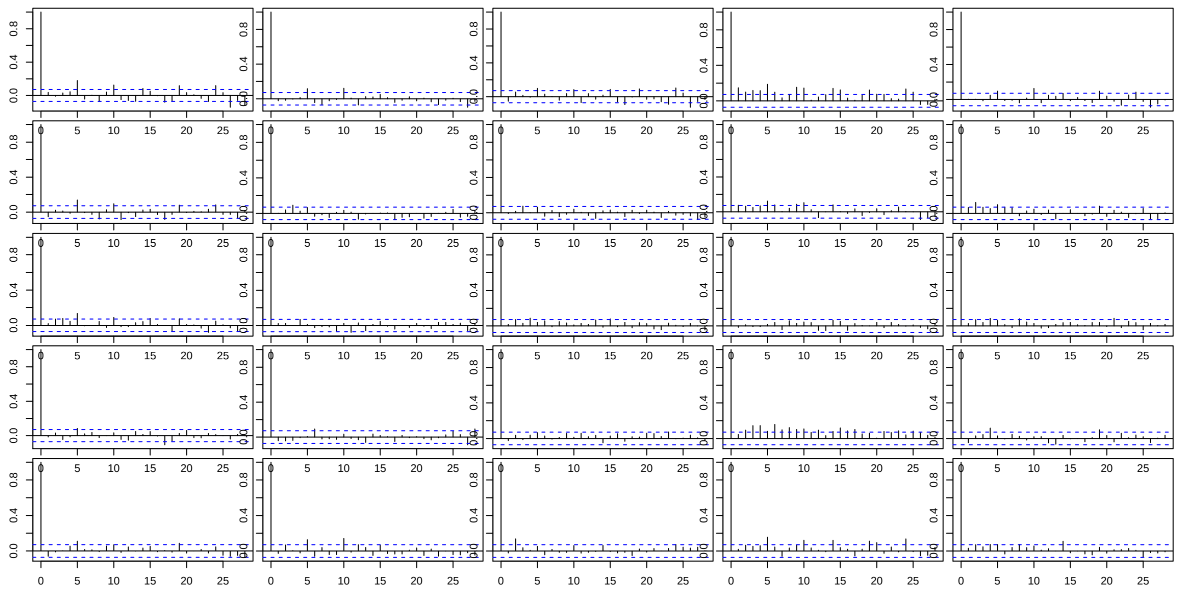


Figure 2: ACF plot by the function `mplot.acf` for the last hour (3pm-4pm) of the residuals after fitting a two-term TenAR(1) model using MLE.

After we fit the model with `tenAR.est`, predictions can be made based on the estimated model, using the function `tenAR.predict`. It takes the estimation output from `tenAR.est` as input, with the specified prediction horizon `n.ahead`. By default, it makes prediction of the next `n.ahead` $= h$ observations $\mathcal{X}_{T+1}, ..., \mathcal{X}_{T+h}$, based on $\mathcal{X}_1, \ldots, \mathcal{X}_T$. For example, the prediction of the next three days after the end of original observed data using the fitted two-term TenAR(1) model can be obtained by

```
R> pred = tenAR.predict(est, n.ahead = 3)
R> dim(pred)
[1] 3 5 5 7
```

The output of the function `tenAR.predict` is the predicted values. If `rolling=FALSE`, the output is a tensor time series of length `n.ahead=` h, containing the predicted values of $\mathcal{X}_{T+1}, ..., \mathcal{X}_{T+h}$, based on $\mathcal{X}_1, \ldots, \mathcal{X}_T$. If `rolling=TRUE`, the output is a tensor series of length `T-n0-n.ahead+1`, giving the rolling forecasts of $\mathcal{X}_{n_0+h}, \cdots, \mathcal{X}_T$. Based on the rolling forecasts, one can obtain the mean squared rolling forecast error

$$\frac{1}{d(T-h-n_0+1)} \sum_{t=n_0}^{T-h} \|\hat{\mathcal{X}}_t(h) - \mathcal{X}_{t+h}\|_F^2.$$

For example, the 1-step rolling forecasts of the last 150 days and the corresponding mean squared forecast error in our example can be obtained with

```
R> T = dim(xx)[1]
R> t0 = T - 150
R> pred.rolling = tenAR.predict(est, n.ahead = 1, rolling=TRUE, n0=t0)
R> sum((pred.rolling - xx[(t0+1):T,,,])^2)/(7*5*5*(T-t0))
[1] 45.38606
```

The mean rolling forecast errors can be used for the evaluation of prediction performance, and the comparison of different models. Table 3 reports the mean rolling forecast errors with `n0` $= 604$ and $h = 1$ of the two-term TenAR(1) model ($R_1 = 2$), one-term TenAR(2) model ($R_1 = R_2 = 1$), and the VAR model (after vectorizing the tensor observations). It is evident that for this example, the TenAR models produce better out-of-sample prediction performance than the VAR model.

	TenAR		VAR
	LSE	MLE	
$p = 1$	45.72	45.39	58.27
$p = 2$	45.21	46.87	84.41

Table 3: Mean squared rolling forecast errors of the TenAR(1) model ($R_1 = 2$), TenAR(2) model ($R_1 = R_2 = 1$), and VAR model, for the taxi data. Here, p stands for the autoregressive order of the TenAR and VAR.

2.3. Reduced Rank Matrix Autoregressive Models

Xiao *et al.* (2021) introduces the reduced rank matrix autoregressive model (RRMAR), which bears the same form of the MAR(1) model in (1), but with the additional rank constraints $\text{rank}(\boldsymbol{A}_i) = k_i \leqslant d_i$ for $i = 1, 2$. Let $\boldsymbol{A}_i = \boldsymbol{U}_i \boldsymbol{D}_i \boldsymbol{V}_i^\top$ be the SVD of $\boldsymbol{A}_i$, the RRMAR model can then be rewritten as

$$\boldsymbol{X}_t = \boldsymbol{U}_1 \boldsymbol{D}_1 \boxed{\boldsymbol{V}_1^\top \boldsymbol{X}_{t-1} \boldsymbol{V}_2} \boldsymbol{D}_2 \boldsymbol{U}_2^\top + \boldsymbol{E}_t. \tag{7}$$

Since $\boldsymbol{U}_i$ and $\boldsymbol{V}_i$ are both $d_i \times k_i$ tall matrices, the boxed part in the preceding equation $\boldsymbol{F}_t := \boldsymbol{V}_1^\top \boldsymbol{X}_{t-1} \boldsymbol{V}_2$ can be interpreted as a composite factor which summarizes the impact of $\boldsymbol{X}_{t-1}$ on $\boldsymbol{X}_t$ through a smaller $k_1 \times k_2$ matrix, and $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ play the role of loading matrices on $\boldsymbol{F}_t$.

Two estimation methods, least squares (RRLSE) and MLE (RRMLE), have been proposed and studied. RRLSE is obtained by an alternating algorithm which updates one of $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ when holding the other fixed. Specifically, the coefficient matrix $\boldsymbol{A}_1$ is updated by minimizing the sum of squares (with $\boldsymbol{A}_2$ given)

$$\min_{\boldsymbol{A}_1 \colon \text{rank}(\boldsymbol{A}_1) = k_1} \sum_{t=2}^{T} \|\boldsymbol{X}_t - \boldsymbol{A}_1 \boldsymbol{X}_{t-1} \boldsymbol{A}_2^\top\|_F^2.$$

The coefficient matrix $\boldsymbol{A}_2$ can be updated similarly under the rank constraint, given $\boldsymbol{A}_1$. The algorithm starts with initial estimates of $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ (e.g. the estimates of MAR(1) model without the rank constraints), and then iterates until convergence. Since each step reduces the residual sum of squares, the convergence is guaranteed.

RRMLE is obtained under the assumption that $\boldsymbol{E}_t$ are iid normal, and the covariance matrix Σ_e of $\text{vec}(\boldsymbol{E}_t)$ is separable of the form

$$\text{Cov}(\text{vec}(\boldsymbol{E}_t)) = \Sigma_2 \odot \Sigma_1, \quad \text{or equivalently,} \quad \boldsymbol{E}_t = \Sigma_1^{1/2} \boldsymbol{Z}_t \Sigma_2^{1/2}, \tag{8}$$

where Σ_1 and Σ_2 are $d_1 \times d_1$ and $d_2 \times d_2$ positive definite matrices respectively, corresponding to the row-wise and column-wise covariances among the entries of $\boldsymbol{E}_t$, and all the entries of

$\boldsymbol{Z}_t$ are iid standard normal. The log likelihood of the RRMAR model is (up to some additive constants)

$$-d_2(T-1)\log|\Sigma_1| - d_1(T-1)\log|\Sigma_2| - \sum_{t=2}^{T} \mathrm{tr}\Big[\Sigma_1^{-1}\left(\boldsymbol{X}_t - \boldsymbol{A}_1\boldsymbol{X}_t\boldsymbol{A}_2^\top\right)\Sigma_2^{-1}\left(\boldsymbol{X}_t - \boldsymbol{A}_1\boldsymbol{X}_t\boldsymbol{A}_2^\top\right)^\top\Big].$$

The algorithm updates one of the pairs $(\boldsymbol{A}_1, \Sigma_1)$ and $(\boldsymbol{A}_2, \Sigma_2)$ when holding the other fixed, and iterates until convergence. In the R package **tensorTS**, the initial values can be specified, and the default is the MLE of the unrestricted MAR(1). For more details of the methods and their theoretical properties, see Xiao *et al.* (2021). Once $\hat{\boldsymbol{A}}_i$ are obtained, the $\boldsymbol{U}_i$ and $\boldsymbol{V}_i$ in (7) can be estimated through the SVD of $\hat{\boldsymbol{A}}_i$.

The extended Bayesian information ceriterion (EBIC) of the estimated model is defined as

$$\begin{aligned}\mathrm{EBIC}(r_1, r_2) = &\log\left[\frac{1}{Td_1d_2}\sum_{t=2}^{T}\|\boldsymbol{X}_t - \hat{\boldsymbol{A}}_1\boldsymbol{X}_{t-1}\hat{\boldsymbol{A}}_2^\top\|_F^2\right] \\ &+ \frac{1}{Td_1d_2}\cdot\left[\log(Td_2)\cdot k_1(2d_1 - k_1) + \log(Td_1)\cdot k_2(2d_2 - k_2)\right],\end{aligned} \tag{9}$$

where we plug in the RRMLE estimates $\hat{\boldsymbol{A}}_i$.

The estimation of the RRMAR model is implemented through the function `matAR.RR.est` in **tensorTS**, with the `method="RRLSE"` or `method="RRMLE"` options. The ranks of $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are specified by the parameters `k1` and `k2`, respectively. The output of the function `matAR.RR.est` is a list containing the values summarized in Table 4. In particular, it includes the estimated $\boldsymbol{A}_i$, $\boldsymbol{U}_i$, $\boldsymbol{V}_i$ and their corresponding element-wise standard errors. Note that the current version of the function can only handle one-term RRMAR model of order 1. Extension of capability to estimate the general multi-term TenAR(p) model with reduced rank structure is still on-going.

2.4. An Example for the RRMAR Model

To demonstrate the estimation procedure and make a comparison with the MAR(1) model without rank constraints, we consider as an example the last hour (3–4pm) of the taxi data used in Section 2.2, which gives a 5×5 matrix observation for each day.

```
R> xmat = xx[,,,7]
R> dim(xmat)
[1] 754   5   5
```

We consider the estimation of the RRMAR model under the constraints that both 5×5 coefficient matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are of rank 2. The MLE (RRMLE) is obtained by

```
R> est.rr = matAR.RR.est(xmat, method="RRMLE", k1=2, k2=2)
```

The estimated coefficient matrix $\hat{\boldsymbol{A}}_1$ and the corresponding standard errors are:

```
R> est.rr$A1
          [,1]      [,2]     [,3]      [,4]      [,5]
[1,]  0.411344  0.356790 0.078945 -0.006701 -0.297310
[2,]  0.347516  0.309272 0.095708  0.057926 -0.189495
[3,]  0.140406  0.140739 0.097044  0.151341  0.047543
[4,] -0.047092 -0.016871 0.079626  0.195087  0.222534
[5,]  0.071043  0.097537 0.146460  0.289951  0.231037
```

Value	Details
`A1, A2`	estimate of $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, $d_1 \times d_1$ and $d_2 \times d_2$ matrices.
`loading`	a list of estimated $d_i \times k_i$ matrices $\boldsymbol{U}_i$, $\boldsymbol{V}_i$, where $\boldsymbol{A}_i = \boldsymbol{U}_i \boldsymbol{D}_i \boldsymbol{V}_i^\top$ is the SVD of $\boldsymbol{A}_i$, $i = 1, 2$.
`SIGMA1, SIGMA2`	estimate of Σ_1 and Σ_2, where $\mathrm{Cov}(\mathrm{vec}(\boldsymbol{E}_t)) = \Sigma_2 \odot \Sigma_1$, available only if `method=RRMLE`.
`res`	residuals $\hat{\boldsymbol{E}}_t$.
`Sig`	sample covariance matrix of the residuals $\mathrm{vec}(\hat{\boldsymbol{E}}_t)$.
`cov`	a list containing `Sigma`, asymptotic covariance matrix of $\left(\mathrm{vec}(\hat{\boldsymbol{A}}_1), \mathrm{vec}(\hat{\boldsymbol{A}}_2^\top)\right)$. `Theta1.u`, `Theta1.v`, asymptotic covariance matrices of $\mathrm{vec}(\hat{\boldsymbol{U}}_1), \mathrm{vec}(\hat{\boldsymbol{V}}_1)$. `Theta2.u`, `Theta2.v`, asymptotic covariance matrices of $\mathrm{vec}(\hat{\boldsymbol{U}}_2), \mathrm{vec}(\hat{\boldsymbol{V}}_2)$.
`sd.A1`	element-wise standard errors of $\hat{\boldsymbol{A}}_1$, aligned with `A1`.
`sd.A2`	element-wise standard errors of $\hat{\boldsymbol{A}}_2$, aligned with `A2`.
`niter`	number of iterations.
`BIC`	value of the extended Bayesian information criterion.

Table 4: Components of the output list of the function `matAR.RR.est`.

```
R> est.rr$sd.A1
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.051055 0.058019 0.070266 0.065060 0.060357
[2,] 0.042054 0.045389 0.052925 0.049396 0.049284
[3,] 0.034772 0.036308 0.037757 0.038703 0.045868
[4,] 0.036678 0.038879 0.037760 0.040169 0.046735
[5,] 0.042952 0.048954 0.053089 0.047464 0.056643
```

The estimates $\hat{\boldsymbol{U}}_i$ and $\hat{\boldsymbol{V}}_i$ (see (7)) are reported in the list `est.rr$loading`. The list `est.rr$cov` contains the asymptotic covariance matrices of $(\mathrm{vec}(\hat{\boldsymbol{A}}_1), \mathrm{vec}(\hat{\boldsymbol{A}}_2^\top))$ (`cov$Sigma`), $\hat{\boldsymbol{U}}_i$ (`cov$Theta1.u, cov$Theta2.u`) and $\hat{\boldsymbol{V}}_i$ (`cov$Theta1.v, cov$Theta2.v`).

With the option `method="RRMLE"`, the output also contains the estimated covariance matrices $\hat{\Sigma}_1$ (`est.rr$SIGMA1`) and $\hat{\Sigma}_2$ (`est.rr$SIGMA2`), as well as the residual sample covariance matrix (`est.rr$Sig`).

As the RRMAR model is a special case of the TenAR model, its prediction procedure is the same. The output of `matAR.RR.est` can be directly plugged into the function `tenAR.predict` for prediction. For example, the prediction of the taxi traffic volume matrices of next three days after the observation period can be obtained using

```
R> pred.rr = tenAR.predict(est.rr, n.ahead = 3)
```

Similarly, rolling forecast can be made so that the mean squared rolling prediction error can be calculated.

```
R> t0 = T - 150
R> pred.rolling = tenAR.predict(est.rr, n.ahead = 1, rolling=TRUE, n0=t0)
R> sum((pred.rolling - xmat[(t0+1):T,,])^2)/(5*5*(T-t0))
[1] 44.42103
```

Table 5 summarizes the rolling forecast performance of VAR, MAR, and RRMAR models, all of order 1, on this matrix series of taxi data. It can be seen that the RRMAR model produces similar forecasts as the unrestricted MAR model, both much better than the VAR model. On the other hand, with $k_1 = k_2 = 2$, the RRMAR model involves significantly less number of parameters than other models.

MAR.LSE	MAR.MLE	RRMAR.LSE	RRMAR.MLE	VAR
44.29	43.88	44.16	44.42	44.74

Table 5: Mean squared rolling forecast errors of various models for the taxi data during 3-4pm.

3. Tensor Factor Model

3.1. Introduction to the Tensor Factor Model

Chen *et al.* (2022b) proposes a factor analysis approach for analyzing high dimensional tensor time series, and introduces the tensor factor model (TenFM) in a Tucker decomposition form

$$\mathcal{X}_t = \mathcal{M}_t + \mathcal{E}_t = \mathcal{F}_t \times_1 \boldsymbol{A}_1 \times_2 \ldots \times_K \boldsymbol{A}_K + \mathcal{E}_t, \tag{10}$$

where $\mathcal{X}_1, ..., \mathcal{X}_T \in \mathbb{R}^{d_1 \times \cdots \times d_K}$ are the observed tensor time series, $\mathcal{E}_t$ is the noise component of $\mathcal{X}_t$ which is white across time, $\boldsymbol{A}_k$ is the deterministic loading matrix of size $d_k \times r_k$ with $r_k \ll d_k$, and the core tensor $\mathcal{F}_t$ itself is a latent tensor factor process of dimension $r_1 \times \ldots \times r_K$. The core tensor $\mathcal{F}_t$ is usually much smaller than $\mathcal{X}_t$ in dimension. This structure provides an effective dimension reduction, as all the co-movements of individual time series in $\mathcal{X}_t$ are driven by $\mathcal{F}_t$. It should be noted that vector and matrix factor models can be viewed as special cases of the tensor factor model, since both vectors and matrices are tensors, with $K = 1$ and $K = 2$ respectively.

3.2. Estimation of the Tensor Factor Model

Because of the non-identifiability of Model (10), only the linear space P_k spanned by the column vectors of $\boldsymbol{A}_k$, $k = 1, \ldots, K$, can be estimated. Chen *et al.* (2022b) proposes two non-iterative estimation procedures, namely TOPUP and TIPUP, to estimate the loading spaces. Han *et al.* (2020) further proposes two iterative algorithms, iTOPUP and iTIPUP, based on the TOPUP and TIPUP respectively, and achieves sharper convergence rates. The following two subsections will cover the non-iterative and iterative methods consecutively.

The Non-iterative Estimation Methods: TIPUP and TOPUP

TOPUP and TIPUP are two methods for estimating the column space spanned by the loading matrix $\boldsymbol{A}_k$, for $k = 1, \ldots, K$. The two procedures are based on different auto-cross-product operations of the observed tensors $\mathcal{X}_t$ to accumulate information.

(i). **Time series Outer-Product Unfolding Procedure (TOPUP)**:
Let $\widehat{\Sigma}_h$ be the sample auto-cross-product of the data $\mathcal{X}_{1:T} = (\mathcal{X}_1, \dots, \mathcal{X}_T)$,

$$\widehat{\Sigma}_h = \widehat{\Sigma}_h(\mathcal{X}_{1:T}) = \sum_{t=h+1}^{T} \frac{\mathcal{X}_{t-h} \otimes \mathcal{X}_t}{T-h} \in \mathbb{R}^{d_1 \times \cdots \times d_K \times d_1 \times \cdots \times d_K}, \tag{11}$$

where $\otimes$ stands for the tensor product. Let $d = d_1 d_2 \cdots d_K$ and $d_{-k} = d/d_k$. The TOPUP method organizes all lag-h cross-outer-products of the mode-k matrix unfolding $\text{mat}_k(\mathcal{X}_t)$ to a $d_k \times (dd_{-k}h_0)$ matrix

$$\begin{aligned} \text{TOPUP}_k(\mathcal{X}_{1:T}) &= \text{mat}_1\left(\sum_{t=h+1}^{T} \frac{\text{mat}_k(\mathcal{X}_{t-h}) \otimes \text{mat}_k(\mathcal{X}_t)}{T-h},\ h = 1, ..., h_0\right) \\ &= \left(\text{mat}_k(\widehat{\Sigma}_h),\ h = 1, ..., h_0\right). \end{aligned} \tag{12}$$

Theoretically, any h_0 can be used to estimate the loading spaces. However, a relatively small h_0 is usually adopted, since the autocorrelations are often stronger at small time lags, and a larger h_0 might introduce more noises. It can be easily seen that $\mathbb{E}[\text{TOPUP}_k(\mathcal{X}_{1:T})] = \boldsymbol{A}_k[\star]$, where $\star$ is a $r_k \times (dd_{-k}h_0)$ matrix, under the assumption that the error tensor process $\mathcal{E}_t$ is white.

The TOPUP method performs SVD of $\text{TOPUP}_k(\mathcal{X}_{1:T})$ to obtain an orthonormal representation of the linear space spanned by the columns of $\boldsymbol{A}_k$

$$\widehat{U}_{k,r_k}^{TOPUP}(\mathcal{X}_{1:T}) = \text{LSVD}_{r_k}\left(\text{TOPUP}_k(\mathcal{X}_{1:T})\right),$$

where LSVD_m stands for the left singular matrix composed of the first m left singular vectors corresponding to the largest m singular values.

(ii). **Time series Inner-Product Unfolding Procedure (TIPUP)**:
Similar to (12), define a $d_k \times (d_k h_0)$ matrix as

$$\text{TIPUP}_k(\mathcal{X}_{1:T}) = \left(\sum_{t=h+1}^{T} \frac{\text{mat}_k(\mathcal{X}_{t-h})\text{mat}_k^\top(\mathcal{X}_t)}{T-h},\ h = 1, ..., h_0\right), \tag{13}$$

which replaces the tensor product in (12) with the inner product. The TIPUP method performs SVD on $\text{TIPUP}_k(\mathcal{X}_{1:T})$: to obtain an orthonormal representation of the column space of $\boldsymbol{A}_k$,

$$\widehat{U}_{k,r_k}^{TIPUP}(\mathcal{X}_{1:T}) = \text{LSVD}_{r_k}\left(\text{TIPUP}_k(\mathcal{X}_{1:T})\right), \quad k = 1, 2, \dots, K.$$

In the sequel, with an abuse of notation, we also refer to the $\hat{U}_{k,r_k}$ returned by TIPUP, TOPUP, the iTIPUP and iTOPUP to be introduced in the next subsection as $\hat{\boldsymbol{A}}_k$.

The theoretical properties of both TOPUP and TIPUP estimators are studied in detail in Chen *et al.* (2022b), along with their comparisons. Both theoretical and empirical studies show that either TOPUP or TIPUP can be better than the other in terms of convergence rates, under different circumstances. For details, see Chen *et al.* (2022b) and its accompanying discussions. In practice, when the signal is strong, TOPUP is recommended. Otherwise, TIPUP can be better.

The Iterative Estimation Methods: iTOPUP and iTIPUP

Han *et al.* (2020) proposes to use iterative projections to improve the TOPUP and TIPUP estimators for TenFM model in the Tucker form. The basic idea comes from the following observation. For illustration, assume that the loading matrices $\boldsymbol{A}_k$ are orthonormal such that $\boldsymbol{A}_k^\top \boldsymbol{A}_k = \boldsymbol{I}$. Suppose $\boldsymbol{A}_2, \ldots, \boldsymbol{A}_K$ are given, let

$$\mathcal{Z}_t = \mathcal{X}_t \times_2 \boldsymbol{A}_2^\top \times_3 \cdots \times_K \boldsymbol{A}_K^\top, \quad \text{and} \quad \mathcal{E}_t^* = \mathcal{E}_t \times_2 \boldsymbol{A}_2^\top \times_3 \cdots \times_K \boldsymbol{A}_K^\top.$$

Then (10) leads to

$$\mathcal{Z}_t = \mathcal{F}_t \times_1 \boldsymbol{A}_1 + \mathcal{E}_t^*,$$

where $\mathcal{Z}_t$ is a $d_1 \times r_2 \times \cdots \times r_K$ tensor. Since $r_k \ll d_k$, the projected tensor $\mathcal{Z}_t$ is much smaller than $\mathcal{X}_t$. Under proper conditions on the projected noise tensor $\mathcal{E}_t^*$, the estimation of the loading space of $\boldsymbol{A}_1$ based on $\mathcal{Z}_t$ can be significantly more accurate, as its convergence rate now depends on $d_1 r_2 \cdots r_K$ rather than $d_1 d_2 \cdots d_K$.

In practice, $\boldsymbol{A}_2, \ldots, \boldsymbol{A}_K$ are unknown. Similar to backfitting algorithms, starting with some initial values of the loading matrices that can be obtained using TOPUP or TIPUP estimators, one can iteratively estimate the loading space of $\boldsymbol{A}_k$ at iteration j based on

$$\mathcal{Z}_{t,k}^{(j)} = \mathcal{X}_t \times_1 \widehat{\boldsymbol{A}}_1^{(j)\top} \times_2 \ldots \times_{k-1} \widehat{\boldsymbol{A}}_{k-1}^{(j)\top} \times_{k+1} \widehat{\boldsymbol{A}}_{k+1}^{(j-1)\top} \times_{k+2} \ldots \times_K \widehat{\boldsymbol{A}}_K^{(j-1)\top}, \qquad (14)$$

using the estimates $\widehat{\boldsymbol{A}}_{k'}^{(j-1)}$, $k < k' \leqslant K$ obtained in the previous iteration and the estimates $\widehat{\boldsymbol{A}}_{k'}^{(j)}$, $1 \leqslant k' < k$, obtained in the current iteration. The estimation can be done using either TOPUP or TIPUP estimators on $\mathcal{Z}_{1:T,k}^{(j)}$ through iterations. After convergence, it results in the iTOPUP and iTIPUP estimators respectively.

Han *et al.* (2020) shows that the iterative estimators significantly improve the convergence rates over the corresponding non-iterative ones, and they are minimax optimal under certain conditions.

In the R package **tensorTS**, estimation for TenFM model is carried out by the function `tenFM.est`, with options to use one out of four estimation methods: TOPUP, TIPUP, iTOPUP and iTIPUP. Two options are used to specify the estimation method: `method` can be either `TOPUP` or `TIPUP`, and `iter=TRUE/FALSE` specifies whether to use the iterative approach or not, where `iter=TRUE` is the default. The function `tenFM.est` also requires users to specify the ranks $r_1, \ldots, r_K$.

When the iterative procedures are used (which is the default), the users can also specify other parameters, including, the maximum number of iterations (default: `niter=100`), the error tolerance in terms of the Frobenius norm (default: `tol=1e-4`), and whether to print the number of iterations (default: `print.true=FALSE`). The components of the output list of the function `tenFM.est` are summarized in Table 6.

3.3. Rank Determination of the Tensor Factor Model

A critical step in building a factor model is to correctly specify the numbers of factors, which correspond to ranks of $\boldsymbol{A}_k$ and the dimensions of the core tensor $\mathcal{F}_t$ in model (10). Estimation procedures all depend on a pre-determined number of factors to be used in the model. In Han *et al.* (2022), two rank determination procedures are proposed to estimate the dimensions r_k

Value	Details
`Ft`	estimated factor tensor time series.
`Ft.all`	the factor tensor summed over time.
`Q`	list of estimated loading matrices $\boldsymbol{A}_k$.
`x.hat`	estimated signal part $\mathcal{M}_t$.
`niter`	number of iterations before estimation stopped.
`fnorm.resid`	proportion of residual Frobensius norm over that of the observed data.

Table 6: Components of the returned list of the function `tenFM.est()`

of the core tensor process $\mathcal{F}_t$, based on the *information criterion* (IC) and *eigenvalue ratio* (ER) respectively. The IC estimators aim at truncating eigenvalues to balance goodness-of-fit and model complexity, and ER estimators are obtained by minimizing the ratio of two adjacent eigenvalues arranged in descending order. Both criteria can be applied to determine the ranks r_k by properly constructing a corresponding semi-positive definite matrix $\widehat{\boldsymbol{W}}$, based on the tensor unfolding of lagged cross-products in Section 3.2, namely TOPUP and TIPUP, and their iterative versions.

Consider a general $p \times p$ symmetric and non-negative definite matrix $\boldsymbol{W}$ with eigenvalues $\lambda_1 \geqslant \ldots \geqslant \lambda_r > \lambda_{r+1} = \ldots = \lambda_p = 0$. Note that the rank of $\boldsymbol{W}$ is r. Let $\widehat{\boldsymbol{W}}$ be a sample version of $\boldsymbol{W}$ (also symmetric and non-negative definite matrix), and assume $\mathbb{E}\widehat{\boldsymbol{W}} = \boldsymbol{W}$. Also let $\hat{\lambda}_1 \geqslant \hat{\lambda}_2 \geqslant \ldots \geqslant \hat{\lambda}_p$ be the eigenvalues of $\widehat{\boldsymbol{W}}$. Let $m^* < p$ be a predefined upper bound and functions $G(\widehat{\boldsymbol{W}})$ and $H(\widehat{\boldsymbol{W}})$ be some appropriate positive penalty functions.

The **information criterion (IC)** is defined as

$$\mathrm{IC}(\widehat{\boldsymbol{W}}) = \underset{0\leqslant m\leqslant m^*}{\arg\min} \left\{ \sum_{l=m+1}^{p} \hat{\lambda}_l + mG(\widehat{\boldsymbol{W}}) \right\}. \tag{15}$$

It is similar to an information criterion since its first term mimics the residual sum of squares of using a rank m matrix to approximate the matrix $\widehat{\boldsymbol{W}}$ while the second term $mG(\widehat{\boldsymbol{W}})$ penalizes the model complexity m.

The **Eigen-ratio criterion (ER)** is defined as

$$\mathrm{ER}(\widehat{\boldsymbol{W}}) = \underset{1\leqslant m\leqslant m^*}{\arg\min} \frac{\hat{\lambda}_{m+1} + H(\widehat{\boldsymbol{W}})}{\hat{\lambda}_m + H(\widehat{\boldsymbol{W}})}. \tag{16}$$

It uses the ratio of two adjacent eigenvalues of $\widehat{\boldsymbol{W}}$, with a small penalty term $H(\widehat{\boldsymbol{W}})$ added to both the numerator and denominator. The use of $H(\widehat{\boldsymbol{W}})$ is to ensure that, when $m = r$ (the true rank), the ratio $(\hat{\lambda}_{m+1} + H(\widehat{\boldsymbol{W}}))/(\hat{\lambda}_m + H(\widehat{\boldsymbol{W}}))$ goes to zero, while for all other $m \neq r$, the ratios are asymptotically bounded from below. Note that all estimated eigenvalues $\hat{\lambda}_l$ $(r + 1 \leqslant l \leqslant p)$ should be relatively small, since they correspond to the zero eigenvalues of $\boldsymbol{W}$.

For either the IC (15) or the ER (16) estimator above, Han *et al.* (2022) proposes four choices of $\widehat{\boldsymbol{W}}$:

$$\begin{aligned}
\widehat{\boldsymbol{W}}_k &:= (\mathrm{TOPUP}_k(\mathcal{X}_{1:T}))(\mathrm{TOPUP}_k(\mathcal{X}_{1:T}))^\top,\\
\widehat{\boldsymbol{W}}_k^* &:= (\mathrm{TIPUP}_k(\mathcal{X}_{1:T}))(\mathrm{TIPUP}_k(\mathcal{X}_{1:T})^\top,\\
\widehat{\boldsymbol{W}}_k^{(j)} &:= (\mathrm{TOPUP}_k(\tilde{\mathcal{Z}}_{1:T,k}^{(j)}))(\mathrm{TOPUP}_k(\tilde{\mathcal{Z}}_{1:T,k}^{(j)}))^\top,\\
\widehat{\boldsymbol{W}}_k^{*(j)} &:= (\mathrm{TIPUP}_k(\tilde{\mathcal{Z}}_{1:T,k}^{(j)}))(\mathrm{TIPUP}_k(\tilde{\mathcal{Z}}_{1:T,k}^{(j)}))^\top.
\end{aligned} \tag{17}$$

This yields eight different rank determination methods, summarized in Table 7. The non-iterative rank estimates can be obtained directly using $\widehat{\boldsymbol{W}}_k$ or $\widehat{\boldsymbol{W}}_k^*$. The iterative method proceeds as follows: the j-th iteration ($j \geqslant 1$) uses the ranks $\{r_k^{(j-1)}+1,\ 1 \leqslant k \leqslant K\}$ and the previous $\hat{\boldsymbol{A}}_k^{(j-1)}$ to get the updated $\hat{\boldsymbol{A}}_k^{(j)}$, based on the $\mathcal{Z}_{1:T,k}^{(j)} = (\mathcal{Z}_{1,k}^{(j)}, \ldots, \mathcal{Z}_{T,k}^{(j)})$ defined in (14). Then the $\widehat{\boldsymbol{W}}_k^{(j)}$ or $\widehat{\boldsymbol{W}}_k^{*(j)}$ is calculated using $\tilde{\mathcal{Z}}_{1:T,k}^{(j)} = (\tilde{\mathcal{Z}}_{1,k}^{(j)}, \ldots, \tilde{\mathcal{Z}}_{T,k}^{(j)})$, where

$$\tilde{\mathcal{Z}}_{t,k}^{(j)} = \mathcal{X}_t \times_1 \widehat{\boldsymbol{A}}_1^{(j)\top} \times_2 \ldots \times_{k-1} \widehat{\boldsymbol{A}}_{k-1}^{(j)\top} \times_{k+1} \widehat{\boldsymbol{A}}_{k+1}^{(j)\top} \times_{k+2} \ldots \times_K \widehat{\boldsymbol{A}}_K^{(j)\top}.$$

The rank estimates $r_k^{(j)}$ are then updated using $\widehat{\boldsymbol{W}}_k^{(j)}$ or $\widehat{\boldsymbol{W}}_k^{*(j)}$, finishing the j-th iteration. To start the iteration, the algorithm uses $r_k^{(0)}+1$, where $r_k^{(0)}$ are rank estimates given the by the corresponding non-iterative methods.

	IC	ER
non-iterative TOPUP	$\hat{r}_k(\mathrm{IC}) = \hat{r}_k^{(0)}(\mathrm{IC}) = \mathrm{IC}(\widehat{\boldsymbol{W}}_k)$	$\hat{r}_k(\mathrm{ER}) = \hat{r}_k^{(0)}(\mathrm{ER}) = \mathrm{ER}(\widehat{\boldsymbol{W}}_k)$
non-iterative TIPUP	$\hat{r}_k^*(\mathrm{IC}) = \hat{r}_k^{*(0)}(\mathrm{IC}) = \mathrm{IC}(\widehat{\boldsymbol{W}}_k^*)$	$\hat{r}_k^*(\mathrm{ER}) = \hat{r}_k^{*(0)}(\mathrm{ER}) = \mathrm{ER}(\widehat{\boldsymbol{W}}_k^*)$
j-th iteration of iTOPUP	$\hat{r}_k^{(j)}(\mathrm{IC}) = \mathrm{IC}(\widehat{\boldsymbol{W}}_k^{(j)})$	$\hat{r}_k^{(j)}(\mathrm{ER}) = \mathrm{ER}(\widehat{\boldsymbol{W}}_k^{(j)})$
j-th iteration of iTIPUP	$\hat{r}_k^{*(j)}(\mathrm{IC}) = \mathrm{IC}(\widehat{\boldsymbol{W}}_k^{*(j)})$	$\hat{r}_k^{*(j)}(\mathrm{ER}) = \mathrm{ER}(\widehat{\boldsymbol{W}}_k^{*(j)})$

Table 7: Rank determination criteria.

The choice of the penalty functions $G(\cdot)$ and $H(\cdot)$: Both the IC and ER criteria essentially try to distinguish the smallest (true) non-zero eigenvalue from the true zero eigenvalues based on noisy estimation of the eigenvalues. Hence the penalty function is closely related to the amount of error in the eigenvalue estimation and the strength of the smallest (true) non-zero eigenvalue. Han *et al.* (2022) considers the following penalty functions $G(\cdot) = g_k(d,T)$ for IC :

$$\begin{aligned}
&g_{k,1}(d,T) = \frac{h_0 d^{2-2\nu}}{T}\log\left(\frac{dT}{d+T}\right), \quad g_{k,2}(d,T) = h_0 d^{2-2\nu}\left(\frac{1}{T}+\frac{1}{d}\right)\log\left(\frac{dT}{d+T}\right),\\
&g_{k,3}(d,T) = \frac{h_0 d^{2-2\nu}}{T}\log\left(\min\{d,T\}\right), \quad g_{k,4}(d,T) = h_0 d^{2-2\nu}\left(\frac{1}{T}+\frac{1}{d}\right)\log\left(\min\{d,T\}\right),\\
&g_{k,5}(d,T) = h_0 d^{2-2\nu}\left(\frac{1}{T}+\frac{1}{d}\right)\log\left(\min\{d_k,T\}\right),
\end{aligned} \tag{18}$$

where $d = \Pi_{k=1}^K d_k$ and ν is a tuning parameter. The rank determination function in our package uses `delta1=0` as the default $\nu = 0$.

For the ER criterion, Han *et al.* (2022) introduces the following penalty functions $H(\cdot) = h_k(d, T)$:

$$h_{k,1}(d,T) = c_0 h_0, \quad h_{k,2}(d,T) = \frac{h_0 d^2}{T^2}, \quad h_{k,3}(d,T) = \frac{h_0 d^2}{T^2 d_k^2}$$
$$h_{k,4}(d,T) = \frac{h_0 d^2}{T^2 d_k^2} + \frac{h_0 d_k^2}{T^2}, \quad h_{k,5}(d,T) = \frac{h_0 d^2}{T^2 d_k} + \frac{h_0 d d_k}{T^2}, \tag{19}$$

where c_0 is a small constant, e.g. $c_0 = 0.1$. Note that the penalty functions scale with h_0, because the strength of divergent eigenvalues increases with h_0.

In the R package **tensorTS**, the rank determination of the TenFM models is carried out by the function `tenFM.rank`, using the original tensor time series as the input. The four methods are again specified by `method` (with value `TIPUP` or `TOPUP`) and `iter` (with `TRUE` as the default), corresponding to the four $\widehat{\boldsymbol{W}}_k$ matrices in (17). The choice of rank determination criterion is given by `rank`, with two options: `IC` and `ER`. The penalty function is specified via `penalty`, with values 1 to 5, corresponding to the five penalty functions listed in (18) or (19). The function `tenFM.rank` returns a list containing two components: `factor.num` is the estimated ranks $\hat{r}_1, \ldots, \hat{r}_K$, and `path` records the intermediate results in each iteration, when the iterative method is used. The first element in `path` is thus the non-iterative result estimated by TIPUP or TOPUP.

3.4. An Example for the Tensor Factor Model

We use the the New York city taxi traffic data again to demonstrate the usage of the functions in the **tensonTS** package for the TenFM model. Here we use different part of the data from that used previously in building TenAR model and RRMAR model, for the purpose of better illustration.

The taxi data is pre-processed into a tensor time series $\mathcal{X}_t = \{\mathcal{X}_{t,ijk}\}$, where at each time a high dimensional order-3 tensor is observed. We focus on the taxi data for business days. The details of downloading and pre-processing the data can be found in Appendix A.

```
R> load('tenFM_taxi_manhattan_midtown.RData')
R> dim(y.midtown)
[1]  252  12  12  24
R> mplot(y.midtown[,,,8])
```

The first mode is the temporal mode that corresponds to 252 business days in the year of 2019, the second and third modes of the tensor correspond to the 12 pick-up and drop-off regions in midtown Manhattan highlighted in blue in Figure 3, and the last mode stands for the 24 hours in a day. One of the 12 regions is Penn Station, a heavy commuting hub serving the surrounding tri-state area with heaving traffic in the morning for business days. A slice of this data, for the morning rush hour from 7am to 8am, is shown in Figure 4.

The first step to build a TenFM model is to determine the number of factors to be used. For example, using the ER criterion (`rank='ER'`), iTIPUP estimation (`method='TIPUP'` and `iter=TRUE`), the penalty function $h_{k,1}(d, T)$ in (19) (`penalty=1`), and default lag $h_0 = 1$, the estimated ranks are 1 on every mode, i.e. $\hat{r}_1 = \hat{r}_2 = \hat{r}_3 = 1$.

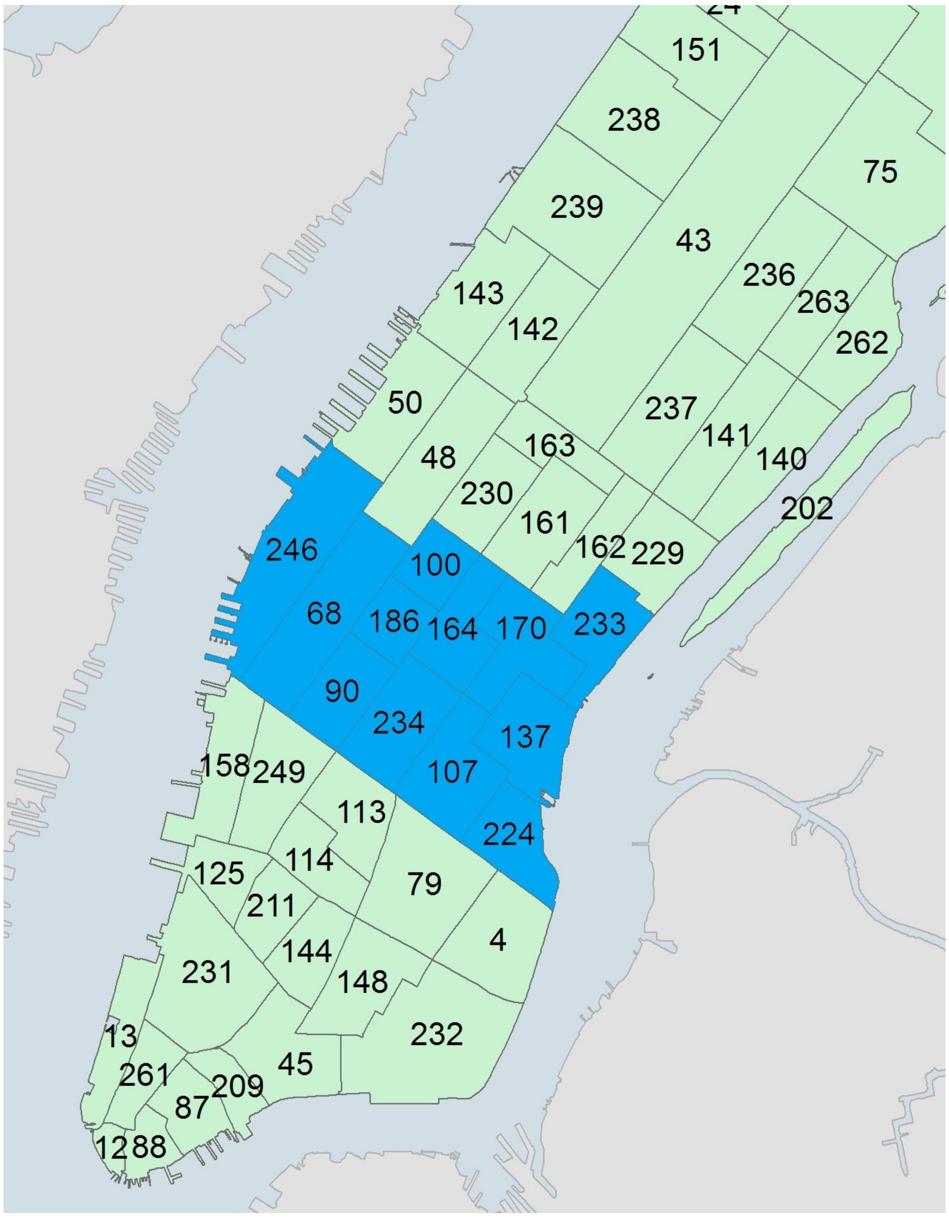


Figure 3: 12 midtown regions in Manhattan are highlighted in blue.

We use IC criterion by specifying `rank='IC'`, and display `$path`, giving the estimated ranks in each iteration. The first row (`iteration 1    4 4 6`) is the the estimated ranks by non-iterative method, the last row (`iteration 7    4 4 3`) is the estimated ranks by the iterative procedure after convergence.

```
R> rank.ans = tenFM.rank(y.midtown,h0=1,rank='IC',iter=TRUE,method='TIPUP',penalty=1)
R> rank.ans$factor.num
[1] 4 4 3

R> rank.ans$path
```

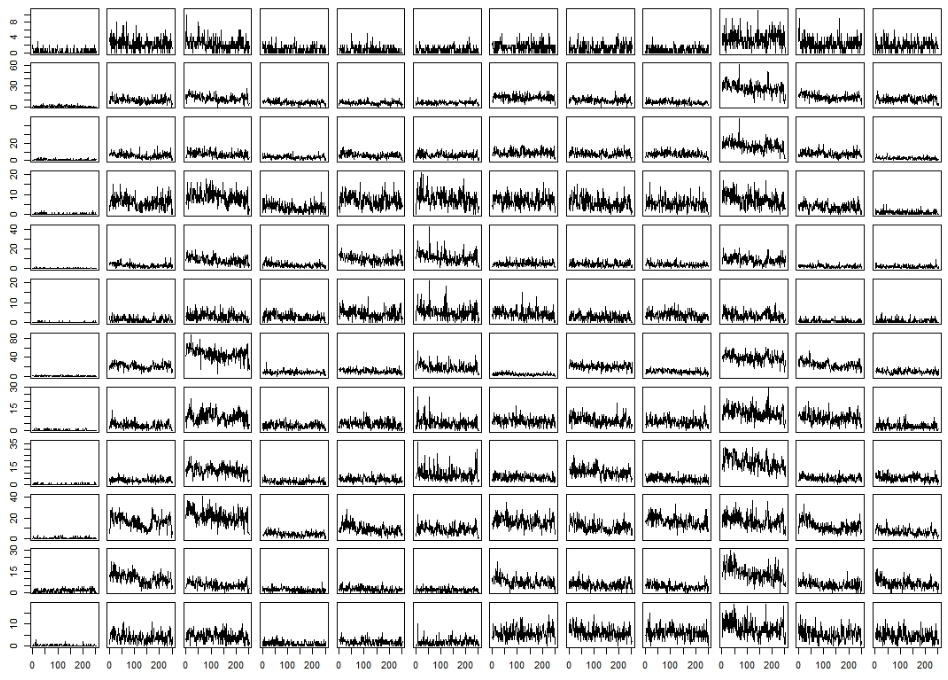


Figure 4: Time series plot of traffic volumes among 12 regions in middle town Manhattan during 7am - 8am, for business days in 2019.

```
               mode 1  mode 2  mode 3
iteration 0       4       4       6
iteration 1       1       4       4
iteration 2       4       4       3
iteration 3       4       4       4
iteration 4       4       4       3
iteration 5       4       4       3
iteration 6       4       4       3
iteration 7       4       4       3
```

Using a different penalty function may result in different answers. For example, with the penalty function $g_{k,5}(d,T)$ in (18), we get

```
R> rank.ans2 = tenFM.rank(y.midtown,h0=1,rank='IC',iter=TRUE,method='TIPUP',penalty=5)
R> rank.ans2$factor.num
[1] 4 4 4
```

In the following we estimate the TenFM model with ranks $(4,4,3)$, $h_0 = 1$ and iTIPUP method.

```
R> factor.ans = tenFM.est(y.midtown,r=rank.ans$factor.num,h0=1,iter=TRUE,method='TIPUP')
```

The estimated factor tensor `factor.ans$Ft` is a tensor time series of dimension $4 \times 4 \times 3$, and the estimated signal `factor.ans$x.hat`, where $\hat{\mathcal{X}}_t = \hat{\mathcal{F}}_t \times_{k=1}^{K} \hat{\boldsymbol{A}}_k$ has the same dimension as the input tensor data $\mathcal{X}_t$. The estimated loading matrices $\boldsymbol{A}_k$ are returned in a list `factor.ans$Q`, where the dimension of each loading matrix is $d_k \times r_k$ which matches the dimension of the input tensor and the chosen factor rank. In what follows, we provide interpretations for the estimated loading matrices $\hat{\boldsymbol{A}}_1, \hat{\boldsymbol{A}}_2, \hat{\boldsymbol{A}}_3$ and the estimated factor process $\hat{\mathcal{F}}_t$ in order.

As mentioned earlier, each estimate $\hat{\boldsymbol{A}}_k$ obtained by the function `tenFM.est` is one specific representation of the loading space spanned by the column vectors of $\boldsymbol{A}_k$. Any orthogonal (or non-orthogonal but full rank) rotation of the estimate is also a valid estimate. To assist the interpretation of the typically large $(d_k \times r_k)$ loading matrices, a varimax rotation (Kaiser 1958) can be used. We use the `varimax` function in the **stats** package to implement the rotation and show the varimax-rotated $\hat{\boldsymbol{A}}_1$ and $\hat{\boldsymbol{A}}_2$ here, which correspond to the pick-up and drop-off locations respectively.

```
R> A1.varimax=varimax(factor.ans$Q[[1]])$loadings
R> A1.varimax=A1.varimax %*% diag(apply(A1.varimax,2,function(x){sign(sum(x))}))
R> round(t(A1.varimax),2)

region index  224   107   234    90    68   246   186  164   100   170   137   233
factor 1     -0.02  0.52  0.77 -0.01  0.01  0.06 -0.07 0.21  0.16 -0.17  0.15  0.10
factor 2     -0.02 -0.30  0.19  0.49  0.60  0.46  0.00 0.09  0.10  0.17 -0.16 -0.07
factor 3      0.04  0.20 -0.13  0.04  0.13 -0.08  0.92 0.06  0.25 -0.05  0.04 -0.04
factor 4      0.05  0.15 -0.03  0.04 -0.09 -0.04  0.02 0.22 -0.05  0.89  0.24  0.27

R> A2.varimax=varimax(factor.ans$Q[[2]])$loadings
R> A2.varimax=A2.varimax %*% diag(apply(A2.varimax,2,function(x){sign(sum(x))}))
R> round(t(A2.varimax),2)

region index 224   107   234    90    68   246   186   164   100   170   137   233
factor 1     0.07  0.13  0.09 0.03 -0.15 -0.03 -0.09  0.13 -0.07  0.87  0.33  0.24
factor 2     0.01 -0.15  0.00 0.32  0.64  0.57  0.22  0.08  0.03  0.23 -0.10 -0.13
factor 3     0.19  0.30 -0.50 0.06  0.16 -0.41  0.65 -0.02  0.08  0.06  0.04  0.04
factor 4     -0.02 0.44  0.68 0.01  0.04  0.03  0.29  0.25  0.35 -0.18  0.10  0.17
```

To visualize the estimated (and varimax rotated) loading matrices $\hat{\boldsymbol{A}}_1$ and $\hat{\boldsymbol{A}}_2$, we align each column of $\hat{\boldsymbol{A}}_k$ with the regions on the spatial map and use heat-maps for the entries of the column, as depicted in Figure 5. The four heat-maps in the top panel of Figure 5 correspond to the four columns of the estimated loading matrix (after varimax rotation) of the pick-up locations $\hat{\boldsymbol{A}}_1$. It can be seen that for the pick-up loading matrix, each factor is heavily loaded on only one or few regions. The third and fourth factors are almost dominantly loaded on one region, which are Penn Station and Murray Hills respectively, and both of them are regions with busy traffic. The phenomenon is similar for the drop-off locations in the bottom panel of the figure, and the heavily-loaded regions are almost the same for pick-up and drop-off locations. This indicates that these regions are both busy departures and destinations on a business day.

The third loading matrix $\hat{\boldsymbol{A}}_3$ of size 24×3 corresponds to the hour of the day and contains three columns. We again perform the varimax rotation on this loading matrix and show the resulting matrix in Table 8. Each entry has been amplified by 100 for easier reading, and the gray cells highlight the large values. It is seen that the first factor captures the morning rush hours from 5am to 9am, the second summarizes the subsequent business hours in the daytime, and the last factor represents the evening peak till midnight.

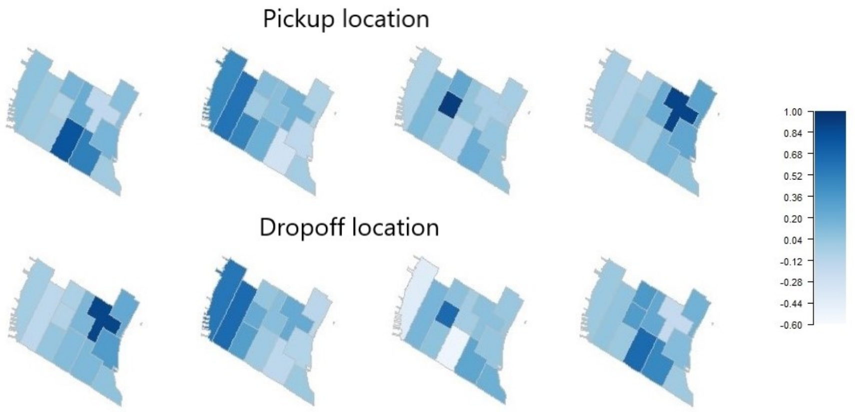


Figure 5: Visualization of loading matrices of taxi pick-up and drop-off locations for business days.

	0am		2		4		6		8		10		12pm		2		4		6		8		10	12am
1	2	-1	-2	-3	1	12	57	66	25	9	8	-4	-11	-13	-13	-10	-5	-9	-15	0	4	15	16	9
2	-2	0	1	1	1	0	-3	5	44	53	35	31	30	25	19	15	9	12	19	6	-5	-12	-11	-7
3	11	6	3	2	1	2	3	5	-12	-16	-4	3	7	12	19	23	26	31	29	34	39	39	32	24

Table 8: Loading matrix that corresponds to the hour-of-the-day mode for business days.

The estimated factor process requires rotations as well if the loading matrices are rotated. It can be obtained with the following commands.

```
A.varimax=list(A1.varimax,A2.varimax,A3.varimax)
Ft.varimax=rTensor::ttl(rTensor::as.tensor(y.midtown),lapply(A.varimax,t),c(2,3,4))@data
mplot(Ft.varimax[,,,1])   # For Figure 6
```

In Figure 6 we plot one slice of the rotated factor tensor process corresponding to the first factor of the third mode (the time-of-the-day mode) – the morning rush hour slice. The slice is a $252 \times 4 \times 4$ matrix time series. The factor processes have different impacts on the co-movement of the observed tensor time series. For example, it can been seen that the factor process in the $(3, 1)$ position of Figure 6 has large volume during the morning rush hour period. It corresponds to the pick-up areas that are heavily loaded on the third pick-up factor (the third column of the top row in Figure 5), which is mainly the area around Penn Station, and the drop-off areas that are heavily loaded on the first drop-off factor (the first column of the bottom row in Figure 5), which is mainly the area around Murray Hill.

Another useful way to jointly visualize the loading matrices and factor process in the TenFM model is to produce the transport network plot, which is used in Chen and Chen (2019), as Figure 7. Chen and Chen (2019) interprets the factors as 'virtual transportation hubs' that gather 'incoming' and 'out-going' traffic from individual areas. In our example, we can imagine that the passengers from different regions are all first transported to one of the four 'pick-up' hubs (the pick-up factors), next transported to one of the four 'drop-off' hubs (the drop-off factors), and eventually transported to their final destinations/regions.

We introduce a tensor $\mathcal{G}$ to summarize the information of the factor process by summation over time, $\mathcal{G}_{ijk} = \sum_t \hat{\mathcal{F}}_{t,ijk}$. The tensor $\mathcal{G}$ is returned by the function `tenFM.est` in

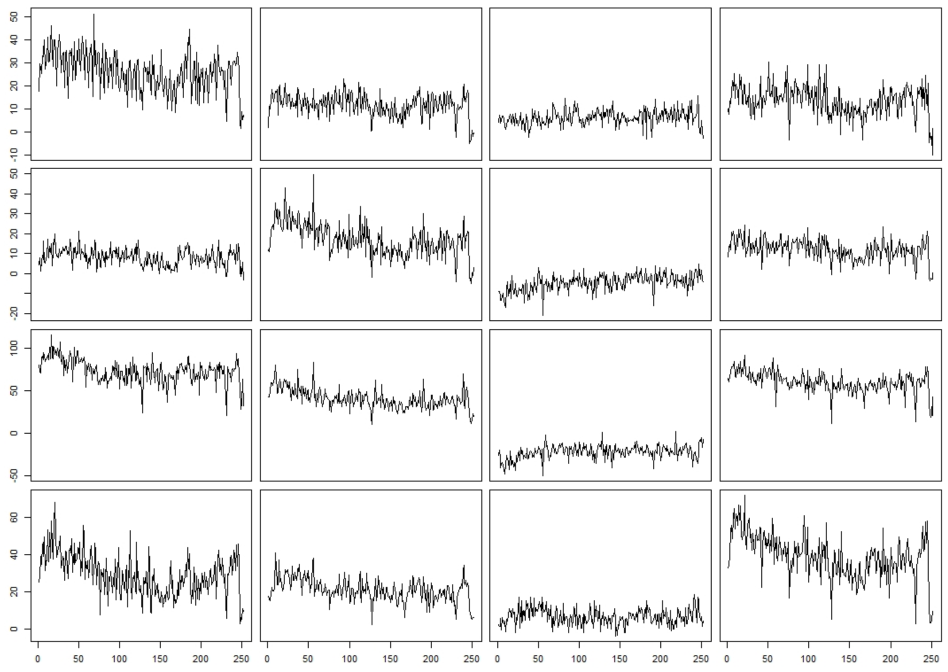


Figure 6: Time series plots of a slice of the varimax-rotated factor process, corresponding to the first factor on the mode of the hour-of-the-day, i.e. the morning rush hour. Four rows and four columns correspond to the four factors (recall $r_1 = r_2 = 4$) along the pick-up and drop-off modes respectively. In other words, the time plot of the i-th row and j-th column is for the univarite time series $\hat{\mathcal{F}}_{t,ij1},\ i,j = 1,\ldots,4$.

`factor.ans$Ft.all`. The transport network plot can only illustrate the effect of two tensor modes, so we again focus on the first factor along the third mode (the morning rush hour factor, $\mathcal{G}_{ij1}$). The two middle columns of the nodes in Figure 7 represent the pick-up and drop-off hubs respectively, and their sizes reflect the total numbers of passengers departing from ($\sum_j \mathcal{G}_{ij1}$) or arriving at ($\sum_i \mathcal{G}_{ij1}$) the hubs respectively. The thickness of the line connecting the hubs reflects the total volume ($\mathcal{G}_{ij1}$) of passengers transported from the i-th pick-up hub to the j-th drop-off hub. It can be seen that pick-up Hub 3 has the most pick-up traffic and its traffic mainly goes to drop-off Hubs 1 and 4. Drop-off Hub 3 has minimum traffic in the morning rush hour period.

On the two sides of Figure 7, we visualize the two varimax-rotated loading matrices $\hat{\boldsymbol{A}}_1$ and $\hat{\boldsymbol{A}}_2$. The numbers near these nodes on the sides are the indices of the 12 midtown areas in Figure 3. The thickness of the lines linking the i-th area node on the left-hand side and the j-th pick-up hub in the middle reflects how much the j-th pick-up factor is loaded on the i-th area, i.e, $\hat{\boldsymbol{A}}_{1,ij}$. The size of the i-th area node reflects its total loading among all hubs, i.e. $\sum_{j=1}^{4} \hat{\boldsymbol{A}}_{1,ij}$. For example, Regions 170 and 186 are the main contributors to pick-up Hubs 4 and 3, respectively. Similarly, the right-hand side of the figure shows the drop-off loading matrix $\hat{\boldsymbol{A}}_2$. For example, Regions 107 and 234 are the main areas for the drop-off volume from drop-off Hub 4.

The function `tenFM.est` also provides the ratio of the squared residual Frobenius norm over that of the original tensor data, in `fnorm.resid`. Hence 1-`fnorm.resid` measures how well

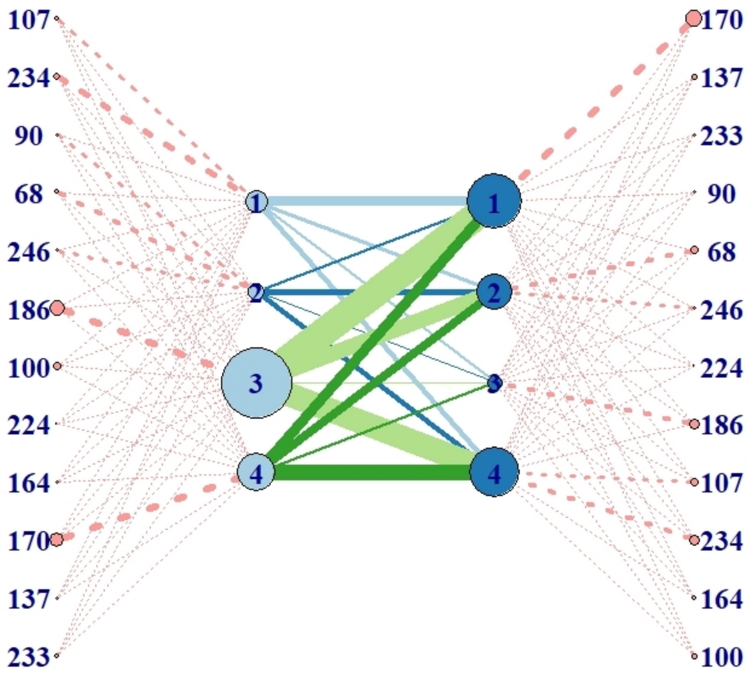


Figure 7: Transport network for the morning rush hour, on business days.

the factor model fits the data, similar to in-sample R^2 in regression analysis. In this taxi data example, $1 - 11.5\% = 88.5\%$ of the variation in the observed tensor time series is explained by the TenFM model with ranks $(4, 4, 3)$.

```
R> factor.ans$fnorm.resid
[1] 0.1153
```

The number of iterations executed by the iterated algorithm is reported by `niter`, which is important for monitoring the convergence and stability of the estimation process. This particular example converges after 4 iterations.

```
R> factor.ans$niter
[1] 4
```

4. Summary and Conclusion

The fast development of technology has led to the explosion of tensor time series data, which has its unique features that differ intrinsically and dramatically from iid tensor data and vector time series data. The existing packages/software to analyze iid tensor data or vector time series data fail to be applicable for tensor time series, because they ignore either the specialty of the temporal mode of the tensor or the information embedded in different modes of the tensor. Hence, they will produce inferior and less interpretable results. This accelerates the urgent need for a package that targets at tensor time series by preserving the tensor structure and treating the temporal mode with care.

The R package **tensorTS** arrives timely for this purpose. There are two main lines of research on tensor time series: TenAR and TenFM. Both of them have many model formulations, various potential extensions, multiple estimation procedures, and several strategies for model selection. The package **tensorTS** implements all of these state-of-the-art methodologies from a sequence of recent developments. This article reviews the key materials from those works, and offers a comprehensive and stand-alone instruction on the usage and understanding of the main functions and outputs by illustration with a concrete taxi data example. Other models and further extensions are being studied and will be incorporated into the package **tensorTS** as new methodologies are developed.

Computational details

The results in this paper were obtained using R 4.0.5 with the **tensorTS** 1.0.0 package (Chen, Han, Li, Xiao, Yang, and Yu 2022a), **tensor** 1.5 package (Rougier 2012), **rTensor** 1.4.8 package (Li *et al.* 2021), **expm** 0.999-6 package (Goulet, Dutang, Maechler, Firth, Shapira, and Stadelmann 2021) **MASS** 7.3-57 package (Ripley, Venables, Bates, Hornik, Gebhardt, and Firth 2022), **abind** 1.4-5 package, **Matrix** 1.4-1 package, **pracma** 2.3.8 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

Acknowledgments

Chen's research is supported in part by National Science Foundation grants DMS-1503409, DMS-1737857, IIS-1741390, CCF-1934924, DMS-2027855, and DMS-2052949. Xiao's research is supported in part by National Science Foundation grants DMS-1454817, DMS-2027855, DMS-2052949, and a research grant from NEC Labs America. Yang's research is supported in part by National Science Foundation grant IIS-1741390, Hong Kong GRF 17301620 and CRF C7162-20GF.

References

Anandkumar A, Ge R, Janzamin M (2014). "Guaranteed Non-Orthogonal Tensor Decomposition via Alternating Rank-1 Updates." *arXiv preprint arXiv:1402.5180*.

Ankargren S, Yang Y, Kastner G (2021). ***mfbvar**: Mixed-Frequency Bayesian VAR Models.* R package version 0.5.6, URL `https://CRAN.R-project.org/package=mfbvar`.

Aspinall T, Gepp A, Harris G, Kelly S, Southam C, Vanstone B, Luethi D, Erb P, Otziger S, Smith P (2021). ***FKF.SP**: Fast Kalman Filtering Through Sequential Processing.* R package version 0.1.3, URL `https://CRAN.R-project.org/package=FKF.SP`.

Bansal G (2021). ***ecm**: Build Error Correction Models.* R package version 6.3.0, URL `https://CRAN.R-project.org/package=ecm`.

Barbosa SM (2012). ***mAr**: Multivariate AutoRegressive Analysis.* R package version 1.1-2, URL `https://CRAN.R-project.org/package=mAr`.

Basu S, Michailidis G (2015). "Regularized estimation in sparse high-dimensional time series models." *The Annals of Statistics*, **43**(4), 1535–1567.

Böck M, Feldkircher M, Huber F (2021). ***BGVAR**: Bayesian Global Vector Autoregressions.* R package version 2.4.3, URL `https://CRAN.R-project.org/package=BGVAR`.

Chan J, Koop G, Poirier DJ, Tobias JL (2019). *Bayesian Econometrics Methods*. Econometric Exercises, 2 edition. Cambridge University Press.

Chang J, Guo B, Yao Q (2015). ***PCA4TS**: Segmenting Multiple Time Series by Contemporaneous Linear Transformation.* R package version 0.1, URL `https://CRAN.R-project.org/package=PCA4TS`.

Chen EY, Chen R (2019). "Modeling dynamic transport network with matrix factor models: with an application to international trade flow." *arXiv preprint arXiv:1901.00769*.

Chen EY, Tsay RS, Chen R (2020a). "Constrained Factor Models for High-Dimensional Matrix-Variate Time Series." *Journal of the American Statistical Association*, **115**(530), 775–793.

Chen EY, Xia D, Cai C, Fan J (2020b). "Semiparametric tensor factor analysis by iteratively projected SVD." *arXiv preprint arXiv:2007.02404*.

Chen H, Raskutti G, Yuan M (2019). "Non-convex projected gradient descent for generalized low-rank tensor regression." *Journal of Machine Learning Research*, **20**(1), 172–208.

Chen R, Han Y, Li Z, Xiao H, Yang D, Yu R (2022a). ***tensorTS**: Factor and Autoregressive Models for Tensor Time Series.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=tensorTS`.

Chen R, Xiao H, Yang D (2021). "Autoregressive models for matrix-valued time series." *Journal of Econometrics*, **222**(1), 539–560.

Chen R, Yang D, Zhang CH (2022b). "Factor models for high-dimensional tensor time series." *Journal of the American Statistical Association*, **117**(537), 94–116.

Christoffersen B, Williams A (2022). ***mssm**: Multivariate State Space Models.* R package version 0.1.6, URL `https://CRAN.R-project.org/package=mssm`.

Cichocki A, Zdunek R, Phan AH, Amari Si (2009). *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley & Sons.

Davis RA, Zang P, Zheng T (2016). "Sparse vector autoregressive modeling." *Journal of Computational and Graphical Statistics*, **25**(4), 1077–1096.

Engle RF, Kroner KF (1995). "Multivariate simultaneous generalized ARCH." *Econometric Theory*, **11**(1), 122–150.

Epskamp S, Asena E (2021). ***graphicalVAR**: Graphical VAR for Experience Sampling Data.* R package version 0.3, URL `https://CRAN.R-project.org/package=graphicalVAR`.

Epskamp S, Waldorp LJ, Mõttus R, Borsboom D (2018). "The Gaussian graphical model in cross-sectional and time-series data." *Multivariate behavioral research*, **53**(4), 453–480.

Fabio Di Narzo A, Aznarte JL, Stigler M (2009). ***tsDyn**: Time series analysis based on dynamical systems theory.* R package version 0.7, URL https://CRAN.R-project.org/package=tsDyn.

Fan J, Liao Y, Mincheva M (2013). "Large covariance estimation by thresholding principal orthogonal complements." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(4), 603–680.

Forni M, Hallin M, Lippi M, Reichlin L (2000). "The generalized dynamic-factor model: Identification and estimation." *Review of Economics and statistics*, **82**(4), 540–554.

Gerard D, Hoff P (2018). ***tensr**: Covariance Inference and Decompositions for Tensor Datasets.* R package version 1.0.1, URL https://CRAN.R-project.org/package=tensr.

Gilbert P (2020). ***dse**: Dynamic Systems Estimation (Time Series Package).* R package version 2020.2-1, URL https://CRAN.R-project.org/package=dse.

Goerg GM (2020). ***ForeCA**: Forecastable Component Analysis.* R package version 0.2.7, URL https://CRAN.R-project.org/package=ForeCA.

Goldenberg A, Zheng AX, Fienberg SE, Airoldi EM, *et al.* (2010). "A survey of statistical network models." *Foundations and Trends in Machine Learning*, **2**(2), 129–233.

Goulet V, Dutang C, Maechler M, Firth D, Shapira M, Stadelmann M (2021). ***expm**: Matrix Exponential, Log, etc.* R package version 0.999-6, URL https://CRAN.R-project.org/package=expm.

Han F, Lu H, Liu H (2015). "A Direct Estimation of High Dimensional Stationary Vector Autoregressions." *Journal of Machine Learning Research*, **16**(97), 3115–3150.

Han Y, Chen R, Yang D, Zhang CH (2020). "Tensor factor model estimation by iterative projection." *arXiv preprint arXiv:2006.02611*.

Han Y, Chen R, Zhang CH (2022). "Rank determination in tensor factor model." *Electronic Journal of Statistics*, **16**(1), 1726–1803.

Hanneke S, Fu W, Xing EP (2010). "Discrete temporal models of social networks." *Electronic Journal of Statistics*, **4**, 585–605.

Haslbeck J (2021). ***mgm**: Estimating Time-Varying k-Order Mixed Graphical Models.* R package version 1.2-12, URL https://CRAN.R-project.org/package=mgm.

Haslbeck JMB, Waldorp LJ (2020). "**mgm**: Estimating Time-Varying Mixed Graphical Models in High-Dimensional Data." *Journal of Statistical Software*, **93**(8), 1–46.

Helske J (2017). "**KFAS**: Exponential Family State Space Models in R." *Journal of Statistical Software*, **78**(10), 1–39.

Helske J (2021). ***KFAS**: Kalman Filter and Smoother for Exponential Family State Space Models.* R package version 1.4.6, URL https://CRAN.R-project.org/package=KFAS.

Hoff PD (2015). “Multilinear tensor regression for longitudinal relational data.” *The Annals of Applied Statistics*, **9**(3), 1169.

Holmes EE, Ward EJ, Scheuerell MD, Wills K (2021). ***MARSS**: Multivariate Autoregressive State-Space Modeling.* R package version 3.11.4, URL https://CRAN.R-project.org/package=MARSS.

Holmes EE, Ward EJ, Wills K (2012). “**MARSS**: multivariate autoregressive state-space models for analyzing time-series data.” *R journal*, **4**(1).

Ji P, Jin J (2016). “Coauthorship and citation networks for statisticians.” *The Annals of Applied Statistics*, **10**(4), 1779–1812.

Kaiser HF (1958). “The varimax criterion for analytic rotation in factor analysis.” *Psychometrika*, **23**(3), 187–200.

Khan SA, *et al.* (2016). “**tensorBF**: an R package for Bayesian tensor factorization.” *bioRxiv*, p. 097048.

Knight M, Leeming K, Nason G, Nunes M (2020). “Generalized Network Autoregressive Processes and the **GNAR** Package.” *Journal of Statistical Software*, **96**(5), 1–36.

Kock AB, Callot L (2015). “Oracle inequalities for high dimensional vector autoregressions.” *Journal of Econometrics*, **186**(2), 325–344.

Kolaczyk ED, Csárdi G (2014). *Statistical analysis of network data with R*, volume 65. Springer.

Kolda TG, Bader BW (2009). “Tensor decompositions and applications.” *SIAM review*, **51**(3), 455–500.

Koop G, Korobilis D (2010). *Bayesian multivariate time series methods for empirical macroeconomics.* Now Publishers Inc.

Kuschnig N, Vashold L (2021). “**BVAR**: Bayesian vector autoregressions with hierarchical prior selection in R.” *Journal of Statistical Software*, **100**(14), 1–27.

Kuschnig N, Vashold L, McCracken M, Ng S (2022). ***BVAR**: Hierarchical Bayesian Vector Autoregression.* R package version 1.0.3, URL https://CRAN.R-project.org/package=BVAR.

Lam C, Yao Q (2012). “Factor modeling for high-dimensional time series: inference for the number of factors.” *The Annals of Statistics*, **40**(2), 694–726.

Lange A, Dalheimer B, Herwartz H, Maxand S (2021). “svars: An R package for data-driven identification in multivariate time series analysis.” *Journal of Statistical Software*, **97**(5), 1–34.

Lange A, Dalheimer B, Herwartz H, Maxand S, Riebl H (2022). ***svars**: Data-Driven Identification of SVAR Models.* R package version 1.3.9, URL https://CRAN.R-project.org/package=svars.

Lee N, Yang HY, Kim SH (2019). ***VARshrink**: Shrinkage Estimation Methods for Vector Autoregressive Models.* R package version 0.3.1, URL `https://CRAN.R-project.org/package=VARshrink`.

Leeming K, Nason G, Nunes M, Knight M (2020). ***GNAR**: Methods for Fitting Network Time Series Models.* R package version 1.1.1, URL `https://CRAN.R-project.org/package=GNAR`.

Leibovici DG (2010). "Spatio-temporal multiway data decomposition using principal tensor analysis on k-modes: The R package **PTAk**." *Journal of Statistical Software*, **34**(10), 1–34.

Leibovici DG (2021). ***PTAk**: Principal Tensor Analysis on k modes.* R package version 1.4-0, URL `https://CRAN.R-project.org/package=PTAk`.

Li J, Bien J, Wells M (2021). ***rTensor**: Tools for Tensor Analysis and Decomposition.* R package version 1.4.8, URL `https://CRAN.R-project.org/package=rTensor`.

Li J, Bien J, Wells MT (2018). "rTensor: An R package for multidimensional array (tensor) unfolding, multiplication, and decomposition." *Journal of Statistical Software*, **87**(1), 1–31.

Li Z, Xiao H (2021). "Multi-linear Tensor Autoregressive Models." *arXiv preprint arXiv:2110.00928*.

Lin C, Cheng G, Chang J, Yao Q (2021). ***HDTSA**: High Dimensional Time Series Analysis Tools.* R package version 1.0.1, URL `https://CRAN.R-project.org/package=HDTSA`.

Lin J, Michailidis G (2017). "Regularized estimation and testing for high-dimensional multi-block vector-autoregressive models." *Journal of Machine Learning Research*, **18**(1), 4188–4236.

Liu J, Musialski P, Wonka P, Ye J (2012). "Tensor completion for estimating missing values in visual data." *IEEE transactions on pattern analysis and machine intelligence*, **35**(1), 208–220.

Liu T, Yuan M, Zhao H (2022). "Characterizing Spatiotemporal Transcriptome of the Human Brain Via Low-Rank Tensor Decomposition." *Statistics in Biosciences*, pp. 1–29.

Liu Y, Shang F, Fan W, Cheng J, Cheng H (2014). "Generalized higher-order orthogonal iteration for tensor decomposition and completion." *Advances in Neural Information Processing Systems*, **27**.

Lock EF (2018). "Tensor-on-tensor regression." *Journal of Computational and Graphical Statistics*, **27**(3), 638–647.

Lock EF (2019). ***MultiwayRegression**: Perform Tensor-on-Tensor Regression.* R package version 1.2, URL `https://CRAN.R-project.org/package=MultiwayRegression`.

Loh PL, Wainwright MJ (2011). "High-dimensional regression with noisy and missing data: Provable guarantees with non-convexity." *Advances in Neural Information Processing Systems*, **24**.

Luethi D, Erb P, Otziger S, McDonald D, Smith P (2021). ***FKF**: Fast Kalman Filter.* R package version 0.2.3, URL `https://CRAN.R-project.org/package=FKF`.

Lütkepohl H (2005). *New introduction to multiple time series analysis.* Springer Science & Business Media.

Matilainen M, Croux C, Miettinen J, Nordhausen K, Oja H, Taskinen S, Virta J (2021). ***tsBSS**: Blind Source Separation and Supervised Dimension Reduction for Time Series.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=tsBSS`.

Melnyk I, Banerjee A (2016). "Estimating structured vector autoregressive models." In *International Conference on Machine Learning*, pp. 830–839.

Mohr FX (2022). ***bvartools**: Functions for Bayesian Inference of Vector Autoregressive Models.* R package version 0.2.1, URL `https://CRAN.R-project.org/package=bvartools`.

Nicholson W, Matteson D, Bien J (2019). ***BigVAR**: Dimension Reduction Methods for Multivariate Time Series.* R package version 1.0.6, URL `https://CRAN.R-project.org/package=BigVAR`.

Nicholson WB, Matteson DS, Bien J (2017). "VARX-L: Structured regularization for large vector autoregressions with exogenous variables." *International Journal of Forecasting*, **3**(33), 627–651.

Nicholson WB, Wilms I, Bien J, Matteson DS (2020). "High Dimensional Forecasting via Interpretable Vector Autoregression." *Journal of Machine Learning Research*, **21**(166), 1–52.

Pena D, Box GE (1987). "Identifying a simplifying structure in time series." *Journal of the American statistical Association*, **82**(399), 836–843.

Petris G (2010). "An R package for dynamic linear models." *Journal of Statistical Software*, **36**(12), 1–16.

Petris G, Gilks W (2018). ***dlm**: Bayesian and Likelihood Analysis of Dynamic Linear Models.* R package version 1.1-5, URL `https://CRAN.R-project.org/package=dlm`.

Peña D, Smucler E, Yohai V (2022). ***odpc**: One-Sided Dynamic Principal Components.* R package version 2.0.5, URL `https://CRAN.R-project.org/package=odpc`.

Pfaff B (2008). *Analysis of integrated and cointegrated time series with R.* Springer Science & Business Media.

Pfaff B, Stigler M (2021). ***vars**: VAR Modelling.* R package version 1.5-6, URL `https://CRAN.R-project.org/package=vars`.

Pfaff B, Zivot E, Stigler M (2016). ***urca**: Unit Root and Cointegration Tests for Time Series Data.* R package version 1.3-0, URL `https://CRAN.R-project.org/package=urca`.

Phan TQ, Airoldi EM (2015). "A natural experiment of social network formation and dynamics." *Proceedings of the National Academy of Sciences*, **112**(21), 6595–6600.

Qiu J, Ning N (2021). ***mbsts**: Multivariate Bayesian Structural Time Series.* R package version 2.2, URL `https://CRAN.R-project.org/package=mbsts`.

Raskutti G, Yuan M, Chen H (2019). "Convex regularization for high-dimensional multiresponse tensor regression." *The Annals of Statistics*, **47**(3), 1554–1584.

Ripley B, Venables B, Bates DM, Hornik K, Gebhardt A, Firth D (2022). ***MASS**: Matrix Exponential, Log, etc.* R package version 7.3-57, URL `https://CRAN.R-project.org/package=MASS`.

Rougier J (2012). ***tensor**: Tensor product of arrays.* R package version 1.5, URL `https://CRAN.R-project.org/package=tensor`.

S H, L K (2022). ***freqdom**: Frequency Domain Based Analysis: Dynamic PCA.* R package version 2.0.2, URL `https://CRAN.R-project.org/package=freqdom`.

Salcedo VB, Villanueva SAC, Torres ADR (2021). ***BMTAR**: Bayesian Approach for MTAR Models with Missing Data.* R package version 0.1.1, URL `https://CRAN.R-project.org/package=BMTAR`.

Sidiropoulos ND, De Lathauwer L, Fu X, Huang K, Papalexakis EE, Faloutsos C (2017). "Tensor decomposition for signal processing and machine learning." *IEEE Transactions on Signal Processing*, **65**(13), 3551–3582.

Snijders TA (2001). "The statistical evaluation of social network dynamics." *Sociological methodology*, **31**(1), 361–395.

Stock JH, Watson MW (2016). "Dynamic factor models, factor-augmented vector autoregressions, and structural vector autoregressions in macroeconomics." In *Handbook of Macroeconomics*, volume 2, pp. 415–525. Elsevier.

Sun WW, Lu J, Liu H, Cheng G (2017). "Provable sparse tensor decomposition." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **79**(3), 899–916.

Tiao GC, Tsay RS (1989). "Model specification in multivariate time series." *Journal of the Royal Statistical Society: Series B (Methodological)*, **51**(2), 157–195.

Tsay RS (2013). *Multivariate time series analysis: with R and financial applications.* John Wiley & Sons.

Tsay RS, Wood D (2021). ***MTS**: All-Purpose Toolkit for Analyzing Multivariate Time Series (MTS) and Estimating Multivariate Volatility Models.* R package version 1.0.3, URL `https://CRAN.R-project.org/package=MTS`.

Tsuyuzaki K, Ishii M, Nikaido I (2021). ***nnTensor**: Non-Negative Tensor Decomposition.* R package version 1.1.5, URL `https://CRAN.R-project.org/package=nnTensor`.

van den Boogaart KG (2020). ***tensorA**: Advanced Tensor Arithmetic with Named Indices.* R package version 0.36.2, URL `https://CRAN.R-project.org/package=tensorA`.

Vazzoler S (2021). ***sparsevar**: Sparse VAR/VECM Models Estimation.* R package version 0.1.0, URL `https://CRAN.R-project.org/package=sparsevar`.

Wang D, Liu X, Chen R (2019). "Factor models for matrix-valued high-dimensional time series." *Journal of Econometrics*, **208**(1), 231–248.

Wang D, Zheng Y, Li G (2021). “High-Dimensional Low-Rank Tensor Autoregressive Time Series Modeling.” *arXiv preprint arXiv:2101.04276*.

Wilms I, Basu S, Bien J, Matteson DS (2021a). “Sparse identification and estimation of large-scale vector autoregressive moving averages.” *Journal of the American Statistical Association*, **0**(0), 1–12.

Wilms I, Matteson DS, Bien J, Basu S, Nicholson W, Wegner E (2021b). ***bigtime**: Sparse Estimation of Large Time Series Models*. R package version 0.2.1, URL https://CRAN.R-project.org/package=bigtime.

Xia D, Yuan M (2019). “On polynomial time methods for exact low-rank tensor completion.” *Foundations of Computational Mathematics*, **19**(6), 1265–1313.

Xia D, Yuan M, Zhang CH (2021). “Statistically optimal and computationally efficient low rank tensor completion from noisy entries.” *The Annals of Statistics*, **49**(1), 76–99.

Xia D, Zhou F (2019). “The Sup-norm Perturbation of HOSVD and Low Rank Tensor Denoising.” *Journal of Machine Learning Research*, **20**(61), 1–42.

Xiao H, Han Y, Chen R, Liu C (2021). “Reduced Rank Autoregressive Models for Matrix Time Series.” *Technical report*, Rutgers University.

Yu L, He Y, Kong X, Zhang X (2022). “Projected estimation for large-dimensional matrix factor models.” *Journal of Econometrics*, **229**(1), 201–217.

Yuan M, Zhang CH (2016). “On tensor completion via nuclear norm minimization.” *Foundations of Computational Mathematics*, **16**(4), 1031–1068.

Yuan M, Zhang CH (2017). “Incoherent tensor norms and their applications in higher order tensor completion.” *IEEE Transactions on Information Theory*, **63**(10), 6753–6766.

Zamora R (2019). ***tensorr**: Sparse Tensors in R*. R package version 0.1.1, URL https://CRAN.R-project.org/package=tensorr.

Zhang A (2019). “Cross: Efficient low-rank tensor completion.” *The Annals of Statistics*, **47**(2), 936–964.

Zhang AR, Luo Y, Raskutti G, Yuan M (2020). “ISLET: Fast and Optimal Low-Rank Tensor Regression via Importance Sketching.” *SIAM Journal on Mathematics of Data Science*, **2**(2), 444–479.

Zhao Y, Levina E, Zhu J (2012). “Consistency of community detection in networks under degree-corrected stochastic block models.” *The Annals of Statistics*, **40**(4), 2266–2292.

Zhou H, Li L, Zhu H (2013). “Tensor regression with applications in neuroimaging data analysis.” *Journal of the American Statistical Association*, **108**(502), 540–552.

A. List of Functions in R Package tensorTS

Functions	Details
`tenAR.est`	Estimation function for the tensor autoregressive model. Methods include: projection (`PROJ`), least squares (`LSE`) and maximum likelihood (`MLE`), as determined by the value of `method`.
`tenAR.predict`	Prediction based on the tensor autoregressive model or reduced rank MAR model. If `rolling=TRUE`, returns the rolling forecasts.
`tenAR.sim`	Generate a TenAR(p) tensor time series.
`matAR.RR.est`	Estimation of the reduced rank MAR(1) model, using least squares (`RRLSE`) or MLE (`RRMLE`), as determined by the value of `method`.
`mplot`	Plot a matrix-valued time series, or a slice of a tensor-valued time series.
`mplot.acf`	Plot ACF of a matrix-valued time series, or a slice of a tensor-valued time series.
`tenFM.est`	Estimation function for the tensor factor model. Methods include TIPUP, TOPUP, iterative TIPUP and iterative TOPUP.
`tenFM.rank`	Rank determination function for the tensor factor model. `rank` specifies the criterion: Eigen Ratio (`ER`) or Information Criterion (`IC`). Methods include TIPUP, TOPUP, iterative TIPUP and iterative TOPUP.
`tenFM.sim`	Generate a TenFM tensor time series with a given tensor factor process.

Table 9: Functions in the R package **tensorTS**.

B. Taxi Data Downloading and Pre-processing

This part is for data preprocessing only, it would first download the taxi data in years 2017 to 2019, then convert the files into a $365 \times 69 \times 69 \times 24$ tensor. We also divide these days into business days and holidays and model the dynamics separately.

```
# Downloading 2017-2019 taxi data , 36 files in total
for(i in 1:36){
  year = (i-1)%/%12+2017
  month = (i-1)%%12+1
  filename = paste('yellow_tripdata_',year,'-',sprintf("%02d",month),'.csv',sep='')
  download_url = paste('https://s3.amazonaws.com/nyc-tlc/trip+data/',filename,sep='')
  download.file(download_url,destfile = filename, method="curl")
}

# taxi pre-processing, only use the 69 regions on manhattan
download.file('https://s3.amazonaws.com/nyc-tlc/misc/taxi+_zone_lookup.csv',
              destfile = 'taxi+_zone_lookup.csv', method="curl")
taxi_dict = read.csv('taxi+_zone_lookup.csv',header=T)
taxi_dict_manhattan = taxi_dict[taxi_dict$Borough=="Manhattan",]
locationID_manhattan = taxi_dict_manhattan$LocationID

y.manhattan = array(0,c(69,69,24,365*3))
# number of days in each month, for years 2017-2019
day_index = c(31,28,31,30,31,30,31,31,30,31,30,31)
day_start_index=0 # the index for day-fiber
for(i in 1:36){
  file.year = (i-1)%/%12+2017
  file.month = (i-1)%%12+1
  filename = paste('yellow_tripdata_',file.year,'-',sprintf("%02d",file.month),'.csv',sep='')
  data = read.csv(filename,header=T)
  data = data[,c(2,8,9)]  # pick-up time, pick-up location, drop-off location
  N = length(data[,1])

  tt = data$tpep_pickup_datetime
  year = as.numeric(substring(tt,1,4))
  month = as.numeric(substring(tt,6,7))
  day = as.numeric(substring(tt,9,10))
  hour = as.numeric(substring(tt,12,13))+1
  PL = match(as.numeric(data$PULocationID),locationID_manhattan)
  DL = match(as.numeric(data$DOLocationID),locationID_manhattan)

  # Only consider trips within manhattan and current date
  # Trips out of manhattan and trips spanning across two days won't be counted
  mask = year==file.year & month==file.month & !is.na(PL) & !is.na(DL)
  for(j in 1:N){
    if(mask[j]==TRUE){
      y.manhattan[PL[j],DL[j],hour[j],day[j]+day_start_index] =
        y.manhattan[PL[j],DL[j],hour[j],day[j]+day_start_index] + 1
    }
  }
  day_start_index = 365*(file.year-2017) + sum(day_index[1:file.month])
}
```

```
# Separate the business days and holidays in year 2019
library(bizdays)
library(RQuantLib)
load_quantlib_calendars('UnitedStates/NYSE','2019-01-01','2019-12-31')
day1 = as.Date("2019-01-01")
bizday_idx = c()
bizday_date = c()
for(i in 1:365){
  if(is.bizday(day1+i-1,'QuantLib/UnitedStates/NYSE')){
    bizday_idx = c(bizday_idx,i)
    bizday_date = c(bizday_date,as.character(day1+i-1))
  }
}
bizday_idx = bizday_idx+365*2 # skip year 2017 and 2018

# The 12 manhattan midtown regions chosen to demonstrate tenFM
locationID_manhattan_midtown = c(224,107,234,90,68,246,186,164,100,170,137,233)
region_idx_midtown = match(locationID_manhattan_midtown,locationID_manhattan)
y.midtown = aperm(y.manhattan[region_idx_midtown,region_idx_midtown,,bizday_idx],c(4,1,2,3))
save(y.midtown,file='tenFM_taxi_manhattan_midtown.RData')

# year 2017-2019 business days data, example for tensor autoregressive model
load_quantlib_calendars('UnitedStates/NYSE','2017-01-01','2019-12-31')
day1 = as.Date("2017-01-01")
bizday_idx = c()
bizday_date = c()
for(i in 1:1095){
  if(is.bizday(day1+i-1,'QuantLib/UnitedStates/NYSE')){
    bizday_idx = c(bizday_idx,i)
    bizday_date = c(bizday_date,as.character(day1+i-1))
  }
}
area = c(41,42,43,47,54)
h = aperm(y.manhattan[area,area,c(9:15),bizday_idx],c(4,1,2,3))   # 754*5*5*7

exponential.smooth = function(x, lambda){
  if(length(lambda) > 1)
    stop("lambda must be a single number")
  if(lambda > 1 || lambda <= 0)
    stop("lambda must be between zero and one")
  xlam = x * lambda
  xlam[1] = x[1]
  filter(xlam, filter = 1 - lambda, method = "rec")
}

dim = dim(h)[-1]
T = dim(h)[1]
trend = array(0,c(T,dim))
for(i in 1:dim[1]){
  for(j in 1:dim[2]){
    for (k in 1:dim[3]){
      trend[,i,j,k] = exponential.smooth(h[,i,j,k], 2/(63+1))
    }
  }
}
xx = h - trend
save(xx,file='tenAR_taxi.RData')
```

C. R Code for the Examples

```
# tenAR part
xx = load('tenAR_taxi.RData')
dim(xx)
mplot(xx[1:100,,,7])
set.seed(123)
est = tenAR.est(xx, R=2, P=1, method="MLE")
A = est$A
length(A) == 1 # order P = 1
length(A[[1]]) == 2 # number of terms R = 2
length(A[[1]][[1]]) == 3 # mode K = 3
A[[1]][[2]][[3]]
sd = est$sd
sd[[1]][[2]][[3]]
Sigma = est$SIGMA
Sigma[[2]]
residuals = est$res
dim(residuals)
mplot.acf(residuals[,,,7])
est$BIC
pred = tenAR.predict(est, n.ahead = 3)
dim(pred)
T = dim(xx)[1]
t0 = T - 150
pred.rolling = tenAR.predict(est, n.ahead = 1, rolling=TRUE, n0=t0)
sum((pred.rolling - xx[(t0+1):T,,,])^2)/(7*5*5*(T-t0))

# matAR.RR part
xmat = xx[,,,7]
dim(xmat)
est.rr = matAR.RR.est(xmat, method="RRMLE", k1=1, k2=1)
est.rr$A1
est.rr$sd.A1
pred.rr = tenAR.predict(est.rr, n.ahead = 3)
t0 = T - 150
pred.rolling = tenAR.predict(est.rr, n.ahead = 1, rolling=TRUE, n0=t0)
sum((pred.rolling - xmat[(t0+1):T,,])^2)/(5*5*(T-t0))

# tenFM part
# load manhattan midtown, year 2019 data
y.midtown=load('tenFM_taxi_manhattan_midtown.RData')
dim(y.midtown)
mplot(y.midtown[,,,8])

# rank selection and factor estimation
rank.ans = tenFM.rank(y.midtown,h0=1,rank='IC',iter=TRUE,method='TIPUP',penalty=1)
rank.ans$factor.num
rank.ans2 = tenFM.rank(y.midtown,h0=1,rank='IC',iter=TRUE,method='TIPUP',penalty=5)
rank.ans2$factor.num
factor.ans = tenFM.est(y.midtown,c(4,4,3),h0=1,iter=TRUE,method='TIPUP')
factor.ans$fnorm.resid
factor.ans$niter

# varimax rotation of the loading and factor
A1.varimax=varimax(factor.ans$Q[[1]])$loadings
A1.varimax=A1.varimax %*% diag(apply(A1.varimax,2,function(x){sign(sum(x))}))
round(t(A1.varimax),2)
A2.varimax=varimax(factor.ans$Q[[2]])$loadings
A2.varimax=A2.varimax %*% diag(apply(A2.varimax,2,function(x){sign(sum(x))}))
round(t(A2.varimax),2)
A3.varimax=varimax(factor.ans$Q[[3]])$loadings
A3.varimax=A3.varimax %*% diag(apply(A3.varimax,2,function(x){sign(sum(x))}))
round(t(A3.varimax),2)*100
A.varimax=list(A1.varimax,A2.varimax,A3.varimax)
Ft.varimax=rTensor::ttl(rTensor::as.tensor(y.midtown),lapply(A.varimax,t),c(2,3,4))@data
mplot(Ft.varimax[,,,1])
```

Affiliation:

Rong Chen, Yuefeng Han, Zebang Li, Han Xiao, Ruofan Yu
Department of Statistics
Rutgers University
110 Frelinghuysen Rd
Piscataway, NJ 08904, USA
E-mail: rongchen@stat.rutgers.edu
yuefeng.han@rutgers.edu
zebang.li@rutgers.edu
hxiao@stat.rutgers.edu
ry166@stat.rutgers.edu

Dan Yang
Faculty of Business and Economics
The University of Hong Kong
Pokfulam Road
Hong Kong
E-mail: dyanghku@hku.hk

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd