6-2017

Development of Traditional and Rank-Based Algorithms for Linear Models with Autoregressive Errors and Multivariate Logistic Regression with Spatial Random Effects

Shaofeng Zhang
Western Michigan University, 0029nezsf@gmail.com

DEVELOPMENT OF TRADITIONAL AND RANK-BASED ALGORITHMS FOR LINEAR MODELS WITH AUTOREGRESSIVE ERRORS AND MULTIVARIATE LOGISTIC REGRESSION WITH SPATIAL RANDOM EFFECTS

by

Shaofeng Zhang

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy
Statistics
Western Michigan University
June 2017

Doctoral Committee:

Joseph W. McKean, Ph.D., Chair
Rajib Paul, Ph.D.
Jeffrey Terpstra, Ph.D.
Bradley Huitema, Ph.D.

DEVELOPMENT OF TRADITIONAL AND RANK-BASED ALGORITHMS FOR LINEAR MODELS WITH AUTOREGRESSIVE ERRORS AND MULTIVARIATE LOGISTIC REGRESSION WITH SPATIAL RANDOM EFFECTS

Shaofeng Zhang, Ph.D.

Western Michigan University, 2017

Linear models are the most commonly used statistical methods in many disciplines. One of the model assumptions is that the error terms (residuals) are independent and identically distributed. This assumption is often violated and autoregressive error terms are often encountered by researchers. The most popular technique to deal with linear models with autoregressive errors is perhaps the autoregressive integrated moving average (ARIMA). Another common approach is generalized least squares, such as Cochrane–Orcutt estimation and Prais–Winsten estimation. However, these usually have poor behaviors when fitting small samples. To address this problem, a double bootstrap method was proposed by McKnight et al. (2000). One purpose of this study is to transfer their algorithm from Fortran to the R computing environment and, ultimately develop an R software package, which, as R, is freeware and runs on all platforms. Furthermore, this study fixes some flaws of the original method and develops a rank-based alternative, which is robust in terms of resistance to outliers. An R package is created and the usage is demonstrated via examples. Monte Carlo studies for different sample sizes (20, 30, 50, and 100) show that both the original and robust algorithm have the expected properties, even for small sample sizes.

In addition to the original algorithm, we also develop a robust rank-based alternative

algorithm. By adopting the rank-based estimator, this new algorithm is resistant to outliers. This is the most important feature of the rank-based estimator. In the same time, this estimator does not loss much efficiency compared to the ordinary least square (OLS) estimator, when the random errors are normally distributed. Comparison of this new algorithm and the original one is made by simulation studies under different settings.

This research also includes an application of the variational approximation in fitting multivariate logistic regression with spatial effects in the Bayesian framework. Variational approximation is much faster than Markov Chain Monte Carlo (MCMC), without losing accuracy. Hence this technique becomes an important alternative to MCMC. Spatial models, such as Conditional Autoregressive (CAR) Models, are extremely popular in characterizing spatial dependencies when datasets are collected over aggregated spatial regions, such as, counties, census tracts, zip codes, etc. Modeling spatially correlated multiple health outcomes requires specification of cross-correlations. Statisticians developed several forms of multivariate conditional autoregressive models (MCAR) for joint modeling of multiple diseases. More specifically, this research investigates the generalized multivariate logistic regression with the spatial random effects modeled via MCAR. For the Bayesian inference of the parameters, both variational approximation and MCMC are developed. They are then compared in terms of the parameter point estimation, confidence interval (CI) and deviance information criterion (DIC). The simulation results exhibit the speedup and accuracy of the estimation and inference of the parameters.

Acknowledgements

I would like to express my appreciation to everyone that helped me throughout the completion of my doctoral education and degree. My deepest appreciation goes to my committee chair Professor Joseph McKean, who is a real genius, has great passion in teaching and teaches me a lot, more than just statistics. I would not have completed the dissertation without his guidance and great help. In addition, I would like to thank all my other committee members, Professor Rajib Paul, Professor Jeffrey Terpstra and Professor Bradley Huitema, for their great suggestions and insights. Lastly, I would like to give my special thanks to my family, who have been strongly supporting me.

Shaofeng Zhang

Table of Contents

Table of Contents—Continued

List of Tables

List of Tables—Continued

List of Figures

Chapter 1

Introduction

Linear models are perhaps the most commonly used statistical methods in many disciplines. One of the model assumptions is that the error terms (residuals) are independent and identically distributed. This assumption is often violated and autoregressive error terms are often encountered by researchers. Therefore, many techniques, including the autoregressive integrated moving average (ARIMA), are developed to deal with linear models with autoregressive errors. The application of ARIMA modeling to time series data was popularized by Box and Jenkins (1970) and Box & Tiao (1975). It is also referred to as Box-Jenkins models. However, they usually have poor behaviors when fitting small samples ($N < 50$) (see, e.g., Box, Jenkins, & Reinsel, 1994, p. 17). McKnight, McKean, & Huitema (2000) showed that, using ARIMA, the estimate of the autoregressive parameter ρ is negatively biased for the sample size $N = 30$.

Another popular approach for analyzing time series data is generalized least squares (GLS). Cochrane–Orcutt estimation (Cochrane & Orcutt, 1949) partitions the fit procedure into two steps. In the first step, ordinary least square (OLS) is used to obtain the residuals $\hat{\epsilon}$. Then regressing $\hat{\epsilon}_t$ on $\hat{\epsilon}_{t-1}$ yields the estimate of the autoregressive parameter ρ. In the second step, it transforms the model using $\hat{\rho}$ and refits the trans-

formed model using OLS. Prais & Winsten (1954) (Prais–Winsten estimation) modified the above procedure by adding the first observation and utilizing another form of transformation. However, those procedures still need large samples (Johnston & Dinardo, 1984). It is also showed by McKnight et al. (2000) that Prais–Winsten estimation underestimates ρ.

To address this problem, a double bootstrap method is proposed by McKnight et al. (2000). On the basis of Durbin two-stage estimation (Durbin, 1960), they introduce two bootstrap procedures. The first one is to correct the bias of Durbin two-stage estimates, and the second one results in a more accurate inference of the parameters. This algorithm was first implemented using Fortran and has been adopted by many researchers (see, e.g. Johnston & Johnston, 2013). For a time, the routine was accessible at the website: http://www.stat.wmich.edu/slab/Software/Timeseries.html. However, the site is no longer supported.

One purpose of this study is to develop an **R** algorithm and code for the double bootstrap procedure. Furthermore, we fix some flaws of the original method, which occasionally overestimates the autoregressive parameter and outputs an estimate to be 0.99. We propose to use the Fisher confidence interval to avoid this overestimation. An **R** package is developed and its usage is demonstrated using examples. Monte Carlo studies for different sample sizes (20, 30, 50, and 100) show that this **R** package works as well as the Fortran version.

In addition to the original algorithm, we also develop a robust rank-based alternative algorithm. By adopting the rank-based estimator, this new algorithm is resistant to outliers. This is the most important feature of the rank-based estimator. At the same time, this estimator does not lose much efficiency compared to the ordinary least squares (OLS) estimator, when the random errors are normally distributed. Comparison of this new algorithm and the original one is made by simulation studies under different

settings.

This research also includes an application of the variational approximation in fitting the multivariate logistic model with spatial effects in the Bayesian framework. Variational approximation is much faster than Markov Chain Monte Carlo (MCMC), without losing accuracy. Hence this technique becomes an important alternative to MCMC. Spatial models, such as Conditional Autoregressive (CAR) Models, are extremely popular in characterizing spatial dependencies when datasets are collected over aggregated regions, such as, counties, census tracts, zip codes, etc. Modeling spatially correlated multiple health outcomes requires specification of cross-correlations. Statisticians developed several forms of multivariate conditional autoregressive models (MCAR) for joint modeling of multiple diseases. More specifically, this research investigates the generalized multivariate logistic regression with the spatial random effects modeled by MCAR. For the Bayesian inference of the parameters, both variational approximation and MCMC are developed. They are then compared in terms of the parameter point estimation, confidence interval (CI) and deviance information criterion (DIC). The simulation results exhibit the speedup and accuracy of the estimation and inference of the parameters.

Chapter 2

Development of the R Package

DBfit

In this chapter, we first briefly review the double bootstrap method. This review is taken from McKnight et al. (2000). Based on this procedure and previous Fortran codes, we build the `DBfit` package in **R**. The instructions of the package are discussed through a real data analysis and simulations. All functions and the reference manual are listed in Appendix A and B. Monte Carlo simulations are also conducted to validate this package. During the simulation studies, we found a weak point of the original algorithm and proposed a corresponding solution.

Consider the general autoregressive (AR) model,

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + u_t \ , \quad t = 1, \ldots, N \ , \tag{2.1}$$

where y_t is the dependent variable, $\mathbf{x}_t$ is a $(p+1) \times 1$ vector of independent variables, the error term u_t follows a stationary autoregressive series with order k,

$$u_t = \rho_1 u_{t-1} + \cdots + \rho_k u_{t-k} + e_t \ , \tag{2.2}$$

and the errors e_t are independently and identically distributed with mean zero and finite variance. The stationarity condition is that the modulus of all roots of the equation $m^k - \rho_1 m^{k-1} - \cdots - \rho_k = 0$ must be less than one. Note that for $k = 1$, this stationarity condition simplifies to that $|\rho| < 1$. Also note that $\mathbf{x}_t$ includes the intercept. So there are p explanatory variables (input series). This is our basic model for Chapter 2 and 3.

2.1 Durbin Two-Stage Procedure

The Durbin two-stage procedure was proposed by Durbin (1960). It leads to efficient estimates for large samples. Hence, they are used as the initial estimates of the autoregressive parameters $\boldsymbol{\rho}$ and regression coefficients $\boldsymbol{\beta}$.

Stage 1

The goal of stage 1 is to get an initial estimate of $\boldsymbol{\rho}$. We can rewrite model (2.1) as

$$y_t = \rho_1 y_{t-1} + \cdots + \rho_k y_{t-k} + \mathbf{x}_t'\boldsymbol{\beta} - \mathbf{x}_{t-1}'\rho_1\boldsymbol{\beta} - \cdots - \mathbf{x}_{t-k}'\rho_k\boldsymbol{\beta} + e_t \ . \tag{2.3}$$

Equation (2.3) is fitted by ordinary least squares (OLS), which leads to the estimates of both $\boldsymbol{\rho}$ and $\boldsymbol{\beta}$. We obtain the estimates of $\boldsymbol{\rho}$ in the lagged y part, and use them as the initial estimates. We denote them by $\hat{\boldsymbol{\rho}}_1$.

Stage 2

In stage 2, we use $\hat{\boldsymbol{\rho}}_1$ to get initial estimates of $\boldsymbol{\beta}$. Consider the following transformation

$$\begin{array}{rcl} v_t(\boldsymbol{\rho}) & = & y_t - \rho_1 y_{t-1} - \cdots - \rho_k y_{t-k} \\ \mathbf{w}_t(\boldsymbol{\rho}) & = & \mathbf{x}_t - \rho_1 \mathbf{x}_{t-1} - \cdots - \rho_k \mathbf{x}_{t-k} \end{array} , \tag{2.4}$$

Suppose $\boldsymbol{\rho}$ is known for now and errors are normally distributed, regressing $v_t(\boldsymbol{\rho})$ on $\mathbf{w}_t(\boldsymbol{\rho})$ results in the BLUE estimates of $\boldsymbol{\beta}$. In fact, $\boldsymbol{\rho}$ is unknown. So we replace $\hat{\boldsymbol{\rho}}$ with $\hat{\boldsymbol{\rho}}_1$ from stage 1. Then the obtained regression coefficients are $\widehat{\gamma}_i, i = 0, 1, ..., p$. Note that $\mathbf{w}_t(\boldsymbol{\rho})$ is a $(p+1) \times 1$ vector, of which the first element is not 1 but $1 - \rho_1 - \cdots - \rho_k$. This fact leads to the following transformation:

$$\widehat{\beta}_{1j} = \begin{cases} \frac{\widehat{\gamma}_0}{1-\widehat{\rho}_1-\cdots-\widehat{\rho}_k} & \text{if } j = 0 \\ \widehat{\gamma}_j & \text{if } j = 1, \ldots, p \end{cases} . \tag{2.5}$$

We denote $\widehat{\boldsymbol{\beta}}_1$ as the final estimates of $\boldsymbol{\beta}$ in the Durbin two-stage procedure. Durbin (1960) shows that the estimator $\widehat{\boldsymbol{\beta}}_1$ is a consistent estimate of $\boldsymbol{\beta}$. Furthermore, the asymptotic distribution of $\sqrt{n}(\widehat{\boldsymbol{\beta}}_1 - \boldsymbol{\beta})$ is the same as that of the LS estimate of $\boldsymbol{\beta}$ in the case that the autoregressive parameters $\rho_1, \ldots, \rho_k$ are known, and vice versa.

2.2 Bootstrap Procedures

The initial estimates of $\boldsymbol{\rho}$ and $\boldsymbol{\beta}$ are not satisfactory. Through simulation studies (McKnight et al., 2000, p. 94), the stage 1 fit usually underestimates $\boldsymbol{\rho}$, i.e. the estimate is negatively biased. Also, the estimator $\hat{\boldsymbol{\beta}}_1$ is not reliable either because it depends on $\hat{\boldsymbol{\rho}}_1$. Furthermore, the use of these estimates usually results in a liberal inference which can be severe at times (McKnight et al., 2000). Therefore, the bootstrap is utilized to generate replicated series of the original time series. Based on the bootstrap samples, we obtain an estimate of the bias in estimating $\boldsymbol{\rho}$. This is the first bootstrap procedure. The purpose of second bootstrap procedure delivers inference on $\boldsymbol{\rho}$ and $\boldsymbol{\beta}$. That is why the method is named the double bootstrap. Although the method holds for any order of $AR(k)$, for ease of discussion, we only consider order 1.

2.2.1 First Bootstrap

As mentioned earlier, the first bootstrap is used to generate replicate series of the original series, from this we can collect information of bias of $\hat{\rho}_1$ and obtain an estimate of the bias. Let $\hat{b}_{bias}$ be the estimate of the bias, then the final estimate of ρ is $\hat{\rho}_1 - \hat{b}_{bias}$. However, one iteration of bootstrap is not enough for estimating the bias. This leads to a loop which is explained later in this section.

To generate the replicate series, we first obtain residuals from the Durbin two-stage procedure. Denote the residuals $\widehat{e}_t$ by

$$\widehat{e}_t = y_t - \widehat{\rho}_1 y_{t-1} - (\mathbf{x}_t - \widehat{\rho}_1 \mathbf{x}_{t-1})'\widehat{\boldsymbol{\beta}}_1 \ , \quad t = 2, \ldots, \mathrm{N} \ . \tag{2.6}$$

Due to possible deflation caused by the fitting (see Stine (1987)), the residuals cannot be used directly. We need to center the residuals by subtracting the average, then rescale them by multiplying the inflation factor $\sqrt{(N-k-p)/(N-2(k+p))}$. Denote the centered and rescaled residuals by $\widehat{e}_{s2}, \ldots, \widehat{e}_{sN}$. These are the residuals we use in the first bootstrap.

In the first bootstrap, we draw N_B bootstrap samples of the residuals. For each sample, say $\widehat{e}^*_{s1}, \ldots, \widehat{e}^*_{sN}$, form the response series as

$$y^*_t = \widehat{\rho}_1 y^*_{t-1} + (\mathbf{x}_t - \widehat{\rho}_1 \mathbf{x}_{t-1})'\widehat{\boldsymbol{\beta}}_1 + \widehat{e}^*_{st} \ , \quad t = 2, \ldots, N \ , \tag{2.7}$$

where y_1 is used to start the series. Once we have this new series, we use the Durbin two-stage procedure to fit them and get $\widehat{\rho}^*_i$ and $\widehat{\boldsymbol{\beta}}^*_i, i = 1, \ldots, N_B$. Note that the "true" autocorrelation for the series y^*_t is $\hat{\rho}_1$. Hence, for the i^{th} bootstrap, $\hat{\rho}^*_i - \hat{\rho}_1$ estimates the bias. So as our estimate of bias, we take the average over all N_B bootstraps.

$$\widehat{b}^{(1)}_{bias} = \sum_{i=1}^{N_B} \frac{\widehat{\rho}^*_i - \widehat{\rho}_1}{N_B} = \frac{1}{N_B}\sum_{i=1}^{N_B} \widehat{\rho}^*_i - \widehat{\rho}_1 \ . \tag{2.8}$$

To estimate the bias more accurately, a loop is proposed in McKnight et al. (2000). We describe the loop as follows:

Initialize: $\boldsymbol{\beta} \leftarrow \widehat{\boldsymbol{\beta}}_1$ and $\rho \leftarrow \widehat{\rho}_1$

Cycle:

use the aforementioned boostrap procedure to obtain $\widehat{\rho}_i^*$ and $\widehat{\boldsymbol{\beta}}_i^*, i = 1, \ldots, N_B$

$\widehat{b}_{bias} \leftarrow \sum_{i=1}^{N_B} \frac{\widehat{\rho}_i^* - \rho}{N_B}$

$\rho \leftarrow \widehat{\rho}_1 - \widehat{b}_{bias}$

use ρ to calculate the Durbin estimate of $\boldsymbol{\beta}$

repeat up to T times or until the increase in ρ is negligible.

In our **R** algorithm, we let this loop stop if the increase in ρ is less than 0.01 or $T = 8$. After this loop, we obtain the final estimate of ρ, denoted by $\widehat{\rho}_F$ and the corresponding $\widehat{\boldsymbol{\beta}}_F$. The subscript F stands for final. Note that we have the stationary assumption, so we force the estimates of ρ in each iteration to be between -.99 and .99. As our earlier Monte Carlo studies showed, this is not a good solution, see section 2.6 for our fix.

For model diagnostics, we obtain the residuals as

$$\widehat{e}_{Ft} = y_t - \widehat{\rho}_F y_{t-1} - (\mathbf{x}_t - \widehat{\rho}_F \mathbf{x}_{t-1})\widehat{\boldsymbol{\beta}}_F \ , \quad t = 2, \ldots, \mathrm{N} \ . \tag{2.9}$$

2.2.2 Second Bootstrap

Using the first bootstrap, we obtain final estimates of ρ and $\boldsymbol{\beta}$. It is straightforward to use the bootstrap to obtain inference on ρ and $\boldsymbol{\beta}$ as well. For inference of $\boldsymbol{\beta}$, we bootstrap the residuals obtained from (2.9), plug $\widehat{\rho}_F$ and $\widehat{\boldsymbol{\beta}}_F$ into equation (2.7) and create N_B replicate series. One change from the previous bootstrap that should be

noted here is a different starting value of the response variable is used. To alleviate the underestimation of the variance of the intercept, McKnight et al. (2000) randomly choose the starting value instead of y_1. This idea is also used in Stine (1987).

For each replicate series, we calculate the Durbin two-stage estimates $\widehat{\boldsymbol{\beta}}_i^*$ for $i = 1, \ldots, N_B$. Then the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}_F$ is

$$\widehat{\mathrm{Var}}(\widehat{\beta}_F) = \frac{1}{N_B} \sum_{i=1}^{N_B} (\widehat{\boldsymbol{\beta}}_i^* - \widehat{\boldsymbol{\beta}}_F)(\widehat{\boldsymbol{\beta}}_i^* - \widehat{\boldsymbol{\beta}}_F)' \ . \tag{2.10}$$

But in their simulation study, McKnight et al. (2000) found a better method of constructing the confidence interval using percentile t method of Hall (1988). For the ith replicated model, $i = 1, \ldots, \mathrm{N}_B$, let $\widehat{\mathbf{e}}_{si}^*$ denote the vector of residuals. Denote the mean square error of these residuals by

$$\mathrm{MSE}(\widehat{\mathbf{e}}_{si}^*) = \frac{1}{N-1} \sum_{t=1}^{N-1} (\widehat{e}_{sit}^* - \overline{\widehat{e}_{st}^*})^2 \ , \quad i = 1, \ldots, N_B \ . \tag{2.11}$$

Let $\mathrm{MSE}(\widehat{\mathbf{e}}_F)$ denote the mean square error based on the residuals (2.9). Then the modified estimate of the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}_F$ is $\widehat{\mathbf{V}}_M$ which is defined by

$$\widehat{\mathbf{V}}_M = \frac{\mathrm{MSE}(\widehat{\mathbf{e}}_F)}{N_B} \sum_{i=1}^{N_B} \frac{(\widehat{\boldsymbol{\beta}}_i^* - \widehat{\boldsymbol{\beta}}_F)(\widehat{\boldsymbol{\beta}}_i^* - \widehat{\boldsymbol{\beta}}_F)'}{\mathrm{MSE}(\widehat{\mathbf{e}}_{si}^*)} \ . \tag{2.12}$$

Since we already obtained the estimate of the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}_F$, we can perform the general linear test of the form

$$H_0 : \ \mathbf{M}\boldsymbol{\beta} = \mathbf{0} \text{ versus } H_A : \ \mathbf{M}\boldsymbol{\beta} \neq \mathbf{0} \tag{2.13}$$

where $\boldsymbol{M}$ is a full row rank $q \times (p+1)$ matrix. The test statistic is

$$F = (\boldsymbol{M}\widehat{\boldsymbol{\beta}})^T [\boldsymbol{M}\hat{V}_M \boldsymbol{M}^T]^{-1} (\boldsymbol{M}\widehat{\boldsymbol{\beta}}) \tag{2.14}$$

The null hypothesis is rejected if $F > F_{1-\alpha,q,n-p-1}$. As discussed in McKnight et al. (2000), the double bootstrap estimates are asymptotically equivalent to the Durbin two-stage estimator with known $\boldsymbol{\rho}$. Treating the test statistic as having an F-distribution, instead of the asymptotic χ^2 is a degree of freedom correction.

For a test of $H_0: \beta_j = 0$, we can perform the t-test using diagonal elements of the variance-covariance matrix estimator.

$$t_j = \frac{\widehat{\beta}_{Fj}}{\sqrt{\widehat{\mathbf{V}}_{Mjj}}} . \tag{2.15}$$

We reject the null hypothesis if $|t_j| > t_{\alpha/2,n-p-1}$. Again, using the t critical values is a degree of freedom correction.

As for the inference of ρ, we are still validating and investigating the inference of ρ proposed in McKnight et al. (2000). And we also propose several new structures for the confidence interval of ρ in Section 2.9.

2.3 History of the Fortran Version and Current R version

The above algorithm was first programmed using Fortran and put on the WMU website `http://www.stat.wmich.edu/slab/Software/Timeseries.html`. It has been widely used by both domestic and international researchers. As noted in a editorial by Derek W. Johnston: "Methods of analysing smaller data sets are being developed, perhaps the most hopeful being the double bootstrap approach of McKnight, McKean, and Huitema (2000), for regression analyses of small sets of time series data." (Johnston & Johnston, 2013). However due to technical issues, the web application is no longer in service. Although the Fortran version is still working, it is not portable. This motivated

us to develop an **R** algorithm for the procedure. Needless to say, **R** programming (R Core Team, 2016) has become a standard programing tool in statistics. **R** is free, open source and user friendly. More importantly, there are many packages (methods) developed by worldwide statisticians on CRAN. We have developed an **R** package `DBfit` for the double bootstrap algorithm.

This package consists of 23 functions. All functions are listed in Appendix A. Here we briefly explain several key functions. The details of all functions are listed in Appendix B.

- durbin1fit carries out the Durbin stage 1 fit as stated in model (2.3)
- durbin1xy constructs the new response variable and design matrix in model (2.3)
- durbin2fit carries out the Durbin stage 2 fit as stated in model (2.4)
- nurho provides the response variable $v_t(\boldsymbol{\rho})$ as in model (2.4)
- wrho provides the design matrix $\mathbf{w}_t(\boldsymbol{\rho})$ as in model (2.4)
- boot1 runs the first bootstrap procedure, completes the loop and outputs the final estimates of $\boldsymbol{\rho}$ and $\boldsymbol{\beta}$
- boot2 runs the second bootstrap procedure to obtain the inference of $\boldsymbol{\beta}$
- dbfit.default is the main function for users to implement the entire method

We plan to publish this package to CRAN (the comprehensive **R** archive network) https://cran.r-project.org/. So users can download and install this package, and run the double bootstrap procedure in the **R** environment.

2.4 Illustration of the DBfit Package

Our **R** software DBfit obtains the double bootstrap fit and associated inference for model (2.1). One of the primary reasons for developing the double bootstrap procedure is to fit and analyze multiphase designs. These phases often occur due to interventions. For examples illustrating the use of DBfit, we first discuss designs for multiphase studies.

We use the design matrices discussed in Huitema, McKean, & McKnight (1999). In two-phase design: β_0 = intercept, β_1 = slope for Phase 1, β_2 = level change from Phase 1 to Phase 2, and β_3 slope change from Phase 1 to Phase 2. The first four coefficients in three-phase design have the same meanings as in two-phase design, plus β_4 = level change from Phase 2 to Phase 3 and β_5 = slope change from Phase 2 to Phase 3.

In the DBfit package, the function hmmat (see Appendix A) is used to construct the design matrix for a specified numbers of phases, say k. The first argument of the function is a vector $v = (n_1, ..., n_k)$, where n_i is the number of observations in phase i. The second argument is the number of phases. In the following **R** output, we show how to use this function and two examples for the two-phase design and three-phase design respectively. Object mat1 is a design matrix for two-phase design with 10 observations in each phase. Object mat2 is a design matrix for three-phase design with 10 observations in each phase as well.

```
> library(DBfit)
> mat1<-hmmat(c(10,10),2)
> mat1

      [,1] [,2] [,3] [,4]
 [1,]    1    1    0    0
 [2,]    1    2    0    0
 [3,]    1    3    0    0
 [4,]    1    4    0    0
 [5,]    1    5    0    0
 [6,]    1    6    0    0
```

```
 [7,]    1    7    0    0
 [8,]    1    8    0    0
 [9,]    1    9    0    0
[10,]    1   10    0    0
[11,]    1   11    1    0
[12,]    1   12    1    1
[13,]    1   13    1    2
[14,]    1   14    1    3
[15,]    1   15    1    4
[16,]    1   16    1    5
[17,]    1   17    1    6
[18,]    1   18    1    7
[19,]    1   19    1    8
[20,]    1   20    1    9
```

```
> mat2<-hmmat(c(10,10,10),3)
> mat2
```

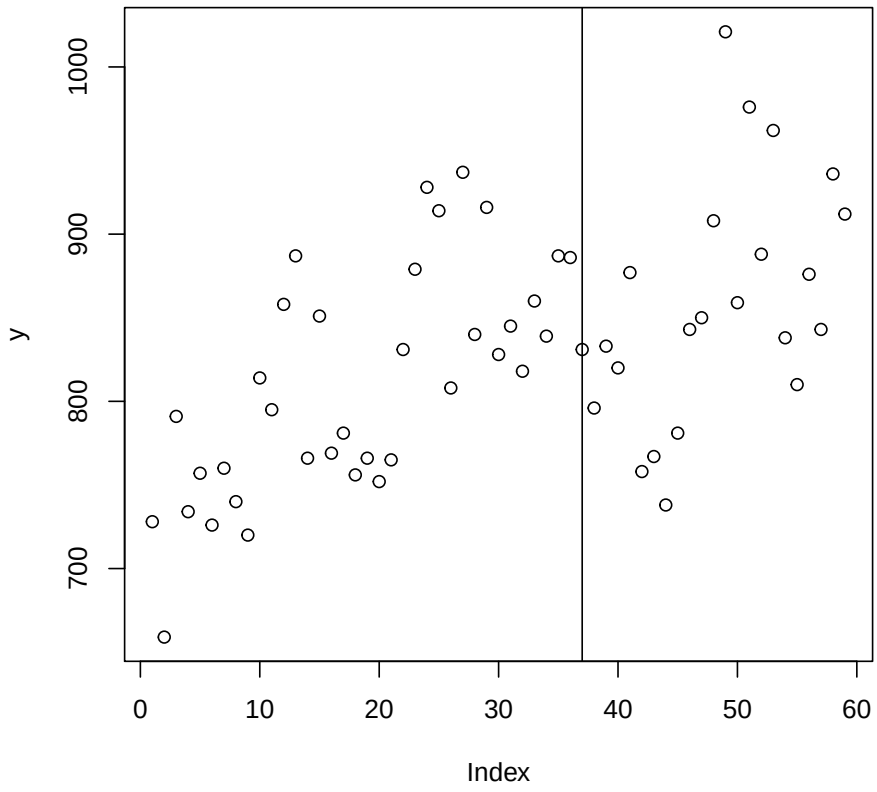
```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    1    0    0    0    0
 [2,]    1    2    0    0    0    0
 [3,]    1    3    0    0    0    0
 [4,]    1    4    0    0    0    0
 [5,]    1    5    0    0    0    0
 [6,]    1    6    0    0    0    0
 [7,]    1    7    0    0    0    0
 [8,]    1    8    0    0    0    0
 [9,]    1    9    0    0    0    0
[10,]    1   10    0    0    0    0
[11,]    1   11    1    0    0    0
[12,]    1   12    1    1    0    0
[13,]    1   13    1    2    0    0
[14,]    1   14    1    3    0    0
[15,]    1   15    1    4    0    0
[16,]    1   16    1    5    0    0
[17,]    1   17    1    6    0    0
[18,]    1   18    1    7    0    0
[19,]    1   19    1    8    0    0
[20,]    1   20    1    9    0    0
[21,]    1   21    1   10    1    0
[22,]    1   22    1   11    1    1
[23,]    1   23    1   12    1    2
[24,]    1   24    1   13    1    3
[25,]    1   25    1   14    1    4
```

```
[26,]    1   26    1   15    1    5
[27,]    1   27    1   16    1    6
[28,]    1   28    1   17    1    7
[29,]    1   29    1   18    1    8
[30,]    1   30    1   19    1    9
```

As the first example illustrating the usage of the package, we use the smoking ban example discussed by Bernal, Cummins, & Gasparrini (2016). The outcome is the monthly rates of the Acute Coronary Events (ACE) admissions in Sicily, Italy. The goal is to evaluate the impact of the smoking ban, in terms of whether it reduces the ACEs. This can be considered as a two-phase interrupted time series design.

First of all, we need to input the data into an **R** session. In the ACE data, phase 1 (before intervention, i.e. the smoking ban) has 36 observations and phase 2 (after intervention) has 23 observations. So in the first argument of function `hmmat`, we specify a vector of [36 23] and write 2 in the second argument. The outcome variable, ACEs, is input from a CSV file. Before fitting the data with the double bootstrap method, we first plot the data and fit the data using OLS.

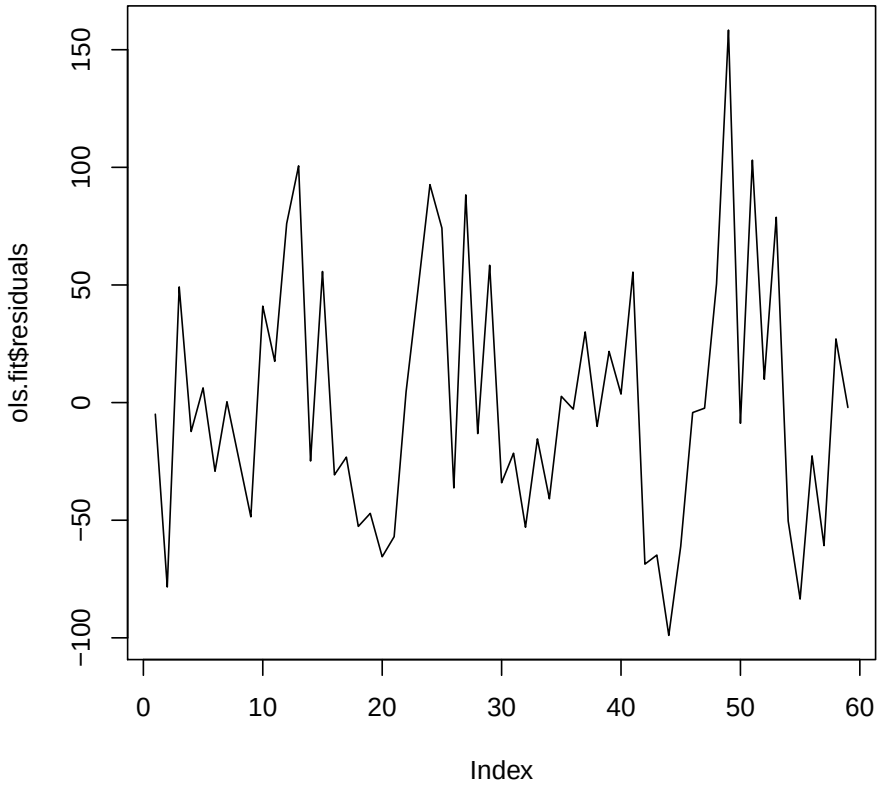
```
> library(DBfit)
> library(car)

> #set the path of the data
> data<-read.table("sicily.csv",sep=",",header = T)
> y<-data$aces

> plot(y)
> abline(v=37)
```




```
> x<-hmmat(c(36,23),2)
> ols.fit<-lm(y~x[,2:4])
> summary(ols.fit)

Call:
lm(formula = y ~ x[, 2:4])

Residuals:
    Min      1Q  Median      3Q     Max
-98.957 -38.575  -4.926  35.513 158.337

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 728.4730    18.9129  38.517  < 2e-16 ***
x[, 2:4]1     4.4534     0.8914   4.996 6.28e-06 ***
x[, 2:4]2   -92.2818    29.3429  -3.145  0.00268 **
x[, 2:4]3     0.6879     1.9609   0.351  0.72707
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 55.56 on 55 degrees of freedom
Multiple R-squared:  0.4418,        Adjusted R-squared:  0.4113
F-statistic: 14.51 on 3 and 55 DF,  p-value: 4.441e-07
```

```
> plot(ols.fit$residuals,type='l')
```




```
> DW1<-durbinWatsonTest(ols.fit)
> DW1

 lag Autocorrelation D-W Statistic p-value
   1       0.2184243      1.562983   0.034
 Alternative hypothesis: rho != 0
```

By looking at the plot of ACEs, we see a clear level change before and after the intervention. However, the slope change is doubtable. This is found out in the summary table of the OLS fit, in which all regression coefficients are significant except for the

slope change. When we plot the residuals of the OLS fit along with the time index, we see a clear pattern of the autocorrelation: adjacent residuals tend to have the same signs. And this is also verified by the Durbin-Watson test. In the test, the DW test statistic is 1.563, which favors the alternative hypothesis that the true autocorrelation is greater than 0. In such a case, we implement the double bootstrap method on this data.

The main function to implement the double bootstrap method is `dbfit`. As presented in the reference manual in Appendix B, similar to `lm`, the data can be passed to the function via a formula or by separate arguments, i.e. $dbfit(y \sim x - 1, arp = 1)$ is equivalent to $dbfit(x, y, arp = 1)$. Note that we require that the design matrix includes the column of ones. So for the S3 formula method, one must use $y \sim x - 1$. Otherwise, the algorithm will double count the intercept and output unexpected results. Argument `arp` is the order of autoregressive errors. There is no default and in this analysis we assume the order is 1. All the other arguments of `dbfit` have default values and presented with details in Appendix B. After calling the fit, one can also use the `summary` function to get the a detailed summarized result of the fit. As we can see below, the result includes the initial estimate of ρ (Durbin two-stage fit), final estimate of ρ (after the bootstrap loop), a flag indicating whether the stationarity assumption holds (see section 2.6) and a summary table of estimates, standard errors, t-ratios and p-values for the regression coefficients.

```
> DB.fit<-dbfit(x,y,arp=1)
> summary(DB.fit)

Call:
dbfit.default(x = x, y = y, arp = 1)

Initial rho:
[1] 0.2189036

Final rho:
[1] 0.3296316
```

```
Nonstationarity flag:
[1] 0
               beta        SE t-ratio   p-value
Intercept 730.50140  29.12813 25.0789 < 2.2e-16 ***
beta_ 1     4.32028   1.30234  3.3173  0.001631 **
beta_ 2   -86.12776  39.12226 -2.2015  0.031993 *
beta_ 3     0.58679   2.86918  0.2045  0.838719
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> DW2<-durbinWatsonTest(as.vector(DB.fit$residuals))#$
> DW2
```

```
[1] 2.395776
```

From the above summary table, the final estimate of the autocorrelation parameter is 0.34. The estimates of the baseline level, phase 1 slope and level change are similar to those in the OLS fit. When we perform the Durbin-Watson test on the residuals again, the test statistic exceeds the upper bound, and hence, favors the null hypothesis that ρ=0.

For the second example to demonstrate our package, we use a simulated data. In `DBfit`, the function `simpgen1hm2` (see Appendix A) creates a time series data with the two-phase design matrix under the normal distribution. The following segment of code generates a data set from AR(1) model with the autoregressive parameter $\rho = 0.6$. For this example, the dataset has two phases of respective sizes 25 and 25. The true regression coefficients are 0. The sizes of both bootstrap procedures are by default 500. Analogous to the `lm` function, one can use the `summary` function to get detailed information of the fit, such as the estimates and standard errors of regression coefficients, and p-values. Also the residuals and fitted values can be obtained from the values of the function `dbfit`.

```
> library(DBfit)
>   n1 <- 25
>   n2 <- 25
```

```
>   rho <- 0.6
>   beta <- c(0,0,0,0)
>   dat <- simpgen1hm2(n1, n2, rho, beta)
>   dat

      c1 c2 c3 c4
 [1,]  1  1  0  0 -0.87597432
 [2,]  1  2  0  0 -0.51883139
 [3,]  1  3  0  0 -0.27036331
 [4,]  1  4  0  0 -0.56511210
 [5,]  1  5  0  0 -0.83302704
 [6,]  1  6  0  0  1.57770620
 [7,]  1  7  0  0  1.65304246
 [8,]  1  8  0  0  1.05869103
 [9,]  1  9  0  0  1.25089163
[10,]  1 10  0  0 -0.35677359
[11,]  1 11  0  0 -0.59675815
[12,]  1 12  0  0 -1.78592873
[13,]  1 13  0  0 -1.78078567
[14,]  1 14  0  0 -1.44224762
[15,]  1 15  0  0 -0.43713681
[16,]  1 16  0  0  0.87528545
[17,]  1 17  0  0 -0.89020209
[18,]  1 18  0  0 -1.49568494
[19,]  1 19  0  0  0.62013654
[20,]  1 20  0  0 -0.20479734
[21,]  1 21  0  0 -0.28220413
[22,]  1 22  0  0 -0.81752737
[23,]  1 23  0  0 -0.38735805
[24,]  1 24  0  0  1.34658310
[25,]  1 25  0  0  0.07921761
[26,]  1 26  1  0 -1.17221424
[27,]  1 27  1  1 -1.35827570
[28,]  1 28  1  2  0.35698717
[29,]  1 29  1  3 -1.87478493
[30,]  1 30  1  4  0.58235707
[31,]  1 31  1  5 -0.14157943
[32,]  1 32  1  6 -0.51616441
[33,]  1 33  1  7  0.04599690
[34,]  1 34  1  8 -1.18802219
[35,]  1 35  1  9 -0.92740282
[36,]  1 36  1 10  0.61562289
[37,]  1 37  1 11  1.35739941
[38,]  1 38  1 12  0.88799678
[39,]  1 39  1 13  0.90589054
[40,]  1 40  1 14 -0.95589703
```

```
[41,]  1 41  1 15 -1.53352804
[42,]  1 42  1 16 -0.71784299
[43,]  1 43  1 17 -0.66844027
[44,]  1 44  1 18  1.16693531
[45,]  1 45  1 19  2.01704741
[46,]  1 46  1 20  1.22246492
[47,]  1 47  1 21  0.21521068
[48,]  1 48  1 22 -0.89977207
[49,]  1 49  1 23 -1.62084243
[50,]  1 50  1 24 -0.13485245
```

```
>   x <- dat[, 1:4] # design matrix
>   y <- dat[, 5] # response
>   arp <- 1 # AR(1) random errors
>   nbs <- 500
>   nbscov <- 500
>   fit <- dbfit(x, y, arp, nbs, nbscov)
>   see <- summary(fit)
>   see
```

```
Call:
dbfit.default(x = x, y = y, arp = arp, nbs = nbs, nbscov = nbscov)

Initial rho:
[1] 0.3969362

Final rho:
[1] 0.5689654

Nonstationarity flag:
[1] 0
                beta         SE t-ratio p-value
Intercept -0.0747251  1.0431499 -0.0716  0.9432
beta_ 1   -0.0010225  0.0622679 -0.0164  0.9870
beta_ 2   -0.6883761  0.9059463 -0.7598  0.4513
beta_ 3    0.0427817  0.0927601  0.4612  0.6469
```

```
> c(fit$residuals)
```

```
 [1]  0.01324005  0.05894692 -0.37673084 -0.47650320  2.08710512  0.79125843
 [7]  0.15448403  0.68529075 -1.03128921 -0.35612720 -1.40831413 -0.72613346
[13] -0.39008088  0.42285426  1.16384404 -1.34792556 -0.94846640  1.51229458
[19] -0.51602766 -0.12363491 -0.61447561  0.12071483  1.61034529 -0.64313189
[25] -0.48465984 -0.39270308  1.41042264 -1.81527428  1.89366906 -0.24629577
[31] -0.22698563  0.53030189 -1.04156719 -0.09683336  1.27990929  1.12575796
[37]  0.21631052  0.48327845 -1.40668971 -0.94302776  0.18330966 -0.24938382
[43]  1.53988365  1.32773097  0.03146447 -0.54169952 -1.10158918 -1.20627264
[49]  0.67198173
```

```
> c(fit$fitted.values)

 [1] -0.53207143 -0.32931023 -0.18838125 -0.35652385 -0.50939892  0.86178403
 [7]  0.90420700  0.56560088  0.67451561 -0.24063095 -0.37761460 -1.05465222
[13] -1.05216674 -0.85999107 -0.28855859  0.45772347 -0.54721853 -0.89215805
[19]  0.31123032 -0.15856922 -0.20305175 -0.50807288 -0.26376219  0.72234950
[25] -0.68755439 -0.96557261 -1.05343547 -0.05951065 -1.31131200  0.10471634
[31] -0.28917878 -0.48430499 -0.14645501 -0.83056946 -0.66428639  0.23164145
[37]  0.67168625  0.42261208  0.45079268 -0.59050028 -0.90115265 -0.41905645
[43] -0.37294834  0.68931644  1.19100045  0.75691020  0.20181710 -0.41456979
[49] -0.80683418

>  mat<-matrix(c(0,0,1,0,0,0,0,1),nrow=2,ncol=4,byrow=T)
>  mat

     [,1] [,2] [,3] [,4]
[1,]    0    0    1    0
[2,]    0    0    0    1

>  hypothmat(sfit=fit,mmat=mat,n=30,p=4)

[1] 0.8202454 0.6677868
```

As shown above, the estimate of ρ is 0.57 which is close to the true value 0.6. But the initial estimate is only 0.40, indicating that the Durbin two-stage procedure underestimates ρ under this small size sample. For the inference of regression coefficients, all p-values are greater than 0.05, indicating that all the regression coefficients may be insignificant. For the hypothesis test: $H_0 : \beta_2 = 0$ and $\beta_3 = 0$, the design matrix is shown in the above output. The F test statistic is 0.82 and p-value is 0.668. At the 0.05 α-level, we cannot reject the null hypothesis. Under the null hypothesis, the models for the two phases are the same.

2.5 Simulation Study 1

The double bootstrap Fortran software has been tested by practitioners and simulation studies for nearly 20 years. One method of validation for the **R** software `DBfit`

is to compare its results with the results of the Fortran version for a large simulation study.

The first simulation aims to compare the results of **R** codes to Fortran codes. We use the three-phase design in this simulation, where the settings are: the underlying distribution is normal; sample size is 50; all β's are set to 0; autoregressive order is 1 and ρ 0.65; the size of both bootstrap procedures is 200; the number of simulations is 5000. As shown in Table 2.1, the results of both versions are extremely similar, which indicates the **R** package works as well as the Fortran version. Note that both versions not only have very close estimates, but they have quite similar standard errors. This implies that both versions will have similar inference based on their respective fits. For example, their respective confidence intervals for the regression parameters will be quite similar. We will examine other settings later.

Table 2.1: Comparison of Fortran and first version of **R**

	Fortran	DBfit	Diff
rho	0.6574	0.6588	-0.0014
SE rho	0.0025	0.0025	0.0002
rho bias	0.0074	0.0088	-0.0014
SE bias	0.0025	0.0025	0.0002
Beta1	0.0153	0.0161	-0.0009
SE B1	0.0232	0.0235	0.0031
Beta2	-0.0008	-0.0009	0.0000
SE B2	0.0017	0.0017	0.0002
Beta3	-0.0146	-0.0140	-0.0006
SE B3	0.0155	0.0155	0.0004
Beta4	0.0023	0.0023	0.0000
SE B4	0.0023	0.0023	0.0002
Beta5	0.0015	0.0014	0.0002
SE B5	0.0151	0.0151	0.0004
Beta6	-0.0021	-0.0021	0.0000
SE B6	0.0019	0.0019	0.0001

2.6 The .99 issue and Simulation 2

While running the validation study, we noticed a serious problem with the basic algorithm used by both the **R** and Fortran versions. For each iteration of the bootstrap bias correction, the algorithm increments the bias. Occasionally the addition of these increments results in an estimate of rho which exceeds 1.0. In these cases, the algorithm resets the estimate of ρ to 0.99. In these simulation studies, this occurred about 5% of the time. Since the stationarity condition is one of the key assumptions for the model, $\hat{\rho}$ should not exceed 1. For these cases, we have an indication that the error time series is not stationary which violates an assumption on the model and hence on the bootstrap.

We have investigated several ways of handling this problem. One solution which seems to be empirically successful is the following: If 0.99 problem is detected, then construct the Fisher confidence intervals for ρ for both the initial estimate (in Durbin stage 1) and the first bias-corrected estimate (first bootstrap); if the midpoint of latter is smaller than 0.95, then this midpoint is the final estimate for ρ; otherwise the midpoint of the former confidence interval is the final estimate.

The Fisher confidence interval for the correlation coefficient uses the log-transformation (Rao, 1952, p. 231):

$$\frac{\sqrt{n-3}}{2} \ln \frac{(1+r)(1-\rho)}{(1-r)(1+\rho)} \xrightarrow{D} N(0,1) \; . \tag{2.16}$$

Then through transformation the CI is:

$$\left[h\left(Z_{\alpha/2} \frac{2}{\sqrt{n-3}} \right), h\left(-Z_{\alpha/2} \frac{2}{\sqrt{n-3}} \right) \right] \tag{2.17}$$

where

$$h(u) = \frac{1 - \frac{1-r}{1+r}e^u}{1 + \frac{1-r}{1+r}e^u} \ , \tag{2.18}$$

and r is the estimate of correlation coefficient ρ. Note that the asymptotic property in (2.16) does not hold in our model, since the confidence interval and ρ are based on the assumption that the observations in each sample are independent. We only consider this as an approximation.

A flag indicating this problem is also added into the **R** package. By default, when the original algorithm outputs an estimate of ρ to be .99, our correction gets involved. Then a corrected estimate of ρ is output and the non-stationarity flag is 1. From the perspective of researchers, when the .99 case materializes, it indicates the error terms are not stationary and they may utilize some common techniques to alleviate the non-stationary problem, such as first order differencing on the response variable and outlier detection via diagnostics. That is why an option to use the above solution or not is added. We suggest use of this solution with caution.

Simulation Study 2

In simulation 2, we use the same settings as simulation 1 and compare the original **R** package to the revised package, which uses the default 0.99 correction. After the introduction of previous solution, the new package produces almost the same good results (Table 2.2)as before. As shown in Table (2.2), the mean estimate of ρ is a little smaller than before. This is because the revised package corrected the 0.99 cases.

Table 2.2: Comparison of original **R** package and Fisher CI correction version

	orig	corrected	diff
rho	0.655619	0.647732	0.007886
SE rho	0.002540	0.002378	0.000161
Beta1	0.004541	0.008968	-0.004427
SE1	0.022788	0.023799	-0.001010
Beta2	0.001432	0.001165	0.000267
SE2	0.001632	0.001692	-0.000059
Beta3	-0.010871	-0.010304	-0.000567
SE3	0.015315	0.015296	0.000019
Beta4	-0.002455	-0.002117	-0.000338
SE4	0.002233	0.002268	-0.000035
Beta5	0.000610	-0.000344	0.000954
SE5	0.015323	0.015265	0.000058
Beta6	0.000690	0.000456	0.000234
SE6	0.001923	0.001895	0.000028

2.7 Simulation Study 3

The purpose of this large simulation is to further validate the revised package with the Fortran version under different combinations of ρ and sample sizes. In this simulation study, we still use the two-phase interrupted time series model. ρ varies from -0.9 to 0.9 and we choose different sample sizes as 20, 30, 50, and 100. Table 2.3 to 2.6 serve as direct comparisons to Table 3 and 4 in McKnight et al. (2000). In each table, $\hat{\rho}_1$ and Var_1 are the empirical means and variances of Durbin two-stage estimator of rho. $\hat{\rho}_F$ is the empirical mean of the final estimate of ρ, and Var_F is its variance. For sample sizes of 20 and 30, our package tend to yield smaller estimates than the Fortran version, especially when the autocorrelation is strong. Again, this is because the package corrected those 0.99 cases. For relative large sample sizes 50 and 100, the difference between both versions is very little. These simulation studies have shown that our R version is quite similar to the Fortran version. Hence, we have validated the `DBfit` package.

Table 2.3: N=20 **R** version

	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.245	0.072	0.543	0.071
0.8	0.216	0.066	0.530	0.074
0.7	0.179	0.064	0.504	0.080
0.6	0.136	0.065	0.468	0.094
0.5	0.082	0.062	0.414	0.101
0.4	0.026	0.062	0.351	0.113
0.3	-0.029	0.058	0.277	0.115
0.2	-0.094	0.053	0.192	0.115
0.1	-0.158	0.053	0.100	0.116
0	-0.224	0.050	0.004	0.112
-0.1	-0.290	0.047	-0.094	0.103
-0.2	-0.363	0.043	-0.202	0.091
-0.3	-0.424	0.041	-0.289	0.085
-0.4	-0.494	0.039	-0.388	0.079
-0.5	-0.566	0.036	-0.490	0.069
-0.6	-0.640	0.030	-0.589	0.052
-0.7	-0.712	0.025	-0.682	0.041
-0.8	-0.788	0.022	-0.772	0.032
-0.9	-0.862	0.015	-0.856	0.019

Table 2.4: N=30 **R** version

	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.451	0.039	0.691	0.036
0.8	0.417	0.037	0.675	0.040
0.7	0.359	0.037	0.629	0.048
0.6	0.298	0.038	0.570	0.059
0.5	0.226	0.037	0.488	0.063
0.4	0.161	0.036	0.403	0.065
0.3	0.084	0.036	0.303	0.065
0.2	0.009	0.035	0.203	0.062
0.1	-0.063	0.034	0.108	0.058
0	-0.142	0.033	0.006	0.055
-0.1	-0.217	0.032	-0.090	0.053
-0.2	-0.299	0.029	-0.194	0.046
-0.3	-0.383	0.028	-0.300	0.044
-0.4	-0.456	0.024	-0.392	0.038
-0.5	-0.544	0.023	-0.499	0.035
-0.6	-0.616	0.021	-0.587	0.031
-0.7	-0.703	0.018	-0.688	0.024
-0.8	-0.786	0.014	-0.780	0.017
-0.9	-0.867	0.010	-0.865	0.011

Table 2.5: N=50 **R** version

	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.639	0.018	0.797	0.015
0.8	0.577	0.018	0.761	0.020
0.7	0.503	0.019	0.691	0.027
0.6	0.428	0.018	0.605	0.028
0.5	0.341	0.020	0.499	0.029
0.4	0.264	0.020	0.406	0.029
0.3	0.179	0.020	0.307	0.028
0.2	0.091	0.020	0.204	0.028
0.1	0.002	0.021	0.101	0.028
0	-0.084	0.020	0.002	0.026
-0.1	-0.175	0.019	-0.102	0.025
-0.2	-0.259	0.018	-0.198	0.024
-0.3	-0.344	0.017	-0.294	0.022
-0.4	-0.433	0.016	-0.395	0.021
-0.5	-0.522	0.014	-0.496	0.019
-0.6	-0.610	0.012	-0.596	0.015
-0.7	-0.700	0.011	-0.694	0.012
-0.8	-0.789	0.008	-0.787	0.009
-0.9	-0.879	0.005	-0.879	0.005

Table 2.6: N=100 **R** version

	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.778	0.006	0.870	0.005
0.8	0.697	0.006	0.799	0.008
0.7	0.610	0.008	0.703	0.009
0.6	0.518	0.008	0.601	0.010
0.5	0.429	0.009	0.504	0.010
0.4	0.335	0.009	0.404	0.011
0.3	0.240	0.010	0.303	0.011
0.2	0.147	0.010	0.203	0.012
0.1	0.054	0.010	0.104	0.011
0	-0.041	0.010	0.002	0.011
-0.1	-0.134	0.010	-0.096	0.011
-0.2	-0.230	0.009	-0.199	0.010
-0.3	-0.322	0.009	-0.297	0.010
-0.4	-0.416	0.008	-0.398	0.010
-0.5	-0.510	0.007	-0.499	0.008
-0.6	-0.606	0.006	-0.600	0.007
-0.7	-0.699	0.005	-0.696	0.005
-0.8	-0.793	0.004	-0.792	0.004
-0.9	-0.888	0.002	-0.888	0.002

2.8 Recommendation of the Bootstrap Size

McKnight et al. (2000) set the default bootstrap size at 200. But that was 20 years ago and at that time computing resources were limited compared to now. Computation is much faster. Hence, we decided to investigate the bootstrap size. We next describe our study.

Table 2.7 informs users of how the bootstrap sizes will effect the speed of the algorithm. Since our method mainly focuses on analyzing short series, we simulate 5000 data sets with sample size 30, true $\rho = 0.6$ and $\boldsymbol{\beta} = [0\ 0\ 0\ 0]^T$. This set of simulations is done on a personal computer with specifications as following: Intel(R) Core(TM) i5-4590 CPU @ 3.30 GHz, 3301 MHz, 4 Core(s), 4 Logical Processor(s); Microsoft Windows 7; Physical Memory (RAM) 8.00 GB.

Table 2.7: Calculation times for different bootstrap sizes

Bootstrap.Size	AvgTime (sec.)	$\hat{\rho}_F$
500.00	5.44	0.53
1000.00	9.08	0.53
1500.00	12.81	0.53
2000.00	16.65	0.53
2500.00	20.61	0.53
3000.00	24.28	0.53

In Table (2.7), running one fit with the bootstrap size set to 5000 takes almost 4-times more time than that with the bootstrap size set to 500. We conclude that with the increase of bootstrap size, the computing time increases significantly. However, the final estimation of ρ has not improved. So we believe 500 bootstrap samples will suffice to yield valid estimates of both autoregressive parameters and regression coefficients. Hence the default bootstrap size in both bootstrap procedures is 500. Note that the bootstrap size is an argument of the `dbfit` function; hence, the user has the option to set the bootstrap size.

2.9 New Proposal of the Confidence Interval for ρ

In this section we consider the confidence interval for ρ. In the algorithm of McKnight et al. (2000), replicated series are still generated as in Section 2.2.2. Then the four-step loop is utilized for each of the series and an estimate of ρ is obtained. Similar to equation (2.12), the variance-covariance matrix of $\boldsymbol{\rho}$ is

$$V_{\widehat{\rho_F}} = \frac{\mathrm{MSE}(\widehat{\mathbf{e}}_F)}{N_B} \sum_{i=1}^{N_B} \frac{(\widehat{\boldsymbol{\rho}}_i^* - \widehat{\boldsymbol{\rho}}_F)(\widehat{\boldsymbol{\rho}}_i^* - \widehat{\boldsymbol{\rho}}_F)'}{\mathrm{MSE}(\widehat{\mathbf{e}}_{si}^*)} \ . \tag{2.19}$$

However, because this is an nested bootstrap, with the bootstrap size being 500, it will be extremely computing intensive. During the development and test of this algorithm, it takes about half an hour to get the result for one dataset. Part of the reasons is due to the low computing efficiency of **R** compared to Fortran. Moreover, based on the Monte Carlo studies of McKnight et al. (2000), this structure of the confidence interval of $\boldsymbol{\rho}$ was not satisfactory.

Hence, as shown in the function `simula` in Appendix A, we propose three new structures of the confidence interval of ρ. All of them are based on the second bootstrap procedure in Section 2.2.2. In that bootstrap procedure, for the i^{th} replicate series, a consistent estimate of $\boldsymbol{\beta}$ is obtained. Hence, we can subtract the $\boldsymbol{X\beta}$ part from the right-hand side of model (2.3). Then regressing y on the rest of the right-hand side yields the estimate $\hat{\rho}_i^{**}$ of ρ. Note that this still provides a biased estimate of ρ. When constructing the confidence interval based on these N_B biased estimates $\hat{\rho}_i^{**}$, we need the following correction.

Let K be a multiplier such that:

$$K \times \widehat{\boldsymbol{\rho}}_F = \widehat{\boldsymbol{\rho}}_F + \mathrm{bias} \tag{2.20}$$

where the bias is the bias of ρ found in the first bootstrap procedure, i.e. bias=$\hat{\rho_F} - \hat{\rho}_1$.

Here $\hat{\rho}_1$ and $\hat{\rho_F}$ are the Durbin two-stage estimator and final estimator, respectively.

Then we construct the three proposals of the confidence interval based on the $\hat{\rho}_i^{***} = \hat{\rho}_i^{**} \times K$. The first one is the same as equation (2.19) except for using $\hat{\rho}_i^{***}$ instead. The second one is to strip out the MSE correction in equation (2.19). And the third one uses the quantiles of $\hat{\rho}_i^{***}$ to build the confidence interval directly.

In a simulation study of 5000 datasets with N=100, ρ=0.6 and normally distributed errors, the empirical coverage probabilities of the three types of confidence intervals are 0.8444, 0.8418 and 0.4194, respectively, at the 0.95 confidence level. In another simulation study of 5000 datasets with N=30, ρ=0.1 and normally distributed errors, the empirical coverages at the 0.95 confidence level are summarized in Table 2.8. The default proposal of the confidence interval for ρ is the first proposal.

Table 2.8: Comparison of four types of CI of ρ

Original	Proposal1	Proposal2	Proposal3
0.966	0.798	0.782	0.45

The results of our proposed confidence intervals are not ideal. But the calculation requires no extra time, since constructing these confidence intervals are based on the second bootstrap procedure, while the original proposal in McKnight et al. (2000) requires another nested bootstrap.

Chapter 3

The Robust Version

We have developed a robust version based on the rank-based estimator for the parameters in model (2.1). One of the most important features of the rank-based estimator is that it is robust, i.e. resistant to outliers, without losing much efficiency compared to lease squares (LS) estimators (under the assumption that the random errors are normally distributed). This feature has a substantial effect on analyzing short time series. In this section, we briefly discuss theories of the rank-based fit, its implementation in **R** code, and simulation results comparing the robust and LS fit. For the details of the theories, the interested reader is referred to Hettmansperger & McKean (2011) and Kloke & McKean (2014).

3.1 Rank-Based Methods

The rank-based analysis can trace its history back to the work of Wilcoxon (1945) and Mann & Whitney (1947). Their work includes proposing the Wilcoxon signed rank test and rank sum test and the relationship between them. In the early days of these rank analyses, they were criticized for low efficiency and power. However, this viewpoint changed with the publication of Hodges Jr & Lehmann (1956), in which the

relative efficiency of several rank-based tests compared to traditional lease squares tests was shown to be quite high. These rank analyses were generalized to the field of linear regression through the work of Jaeckel (1972) and McKean (1975). These methods are generally referred to as rank-based procedures. Since then, the rank-based fit for linear regression has become an important alternative to least squares fit. A brief history of rank-based analysis can be found in the article by McKean & Hettmansperger (2016).

In particular, Kloke and McKean have implemented many aspects of this analysis in the **R** computing environment. Their package **Rfit** (see Kloke & McKean, 2012) can be downloaded at **CRAN** (http://cran.us.r-project.org/). This package is used as one of the workhorses in the robust version of our Double Bootstrap analysis. The **R** function `rfit` plays the same role as `lm` in the fitting of a linear model. This important feature ensures the smooth transition from the traditional Double Bootstrap method to the rank-based version.

We first give a brief discussion of the rank-based analysis. Consider the general linear model:

$$\boldsymbol{Y} = \mathbf{1}\alpha + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{3.1}$$

where $\boldsymbol{Y}$ is a $N \times 1$ column vector of the response variable, $\mathbf{1}$ is a $N \times 1$ column vector of ones, α is the intercept, $\boldsymbol{X}$ is a $N \times p$ matrix of the p explanatory variables, $\boldsymbol{\beta}$ is a $p \times 1$ column vector of regression coefficients, and $\boldsymbol{\epsilon}$ is a $N \times 1$ column vector of residuals.

Traditional LS is based on the Euclidean norm: for a vector $\boldsymbol{x} = (x_1, ..., x_n)$ defined on $\mathbb{R}$, $\|\boldsymbol{x}\| = \sqrt{x_1^2 + ... + x_n^2}$. The fundamental difference between rank-based fit and LS fit is that a rank-based norm is used instead of Euclidean norm. Before defining this new norm, we need to introduce score generating functions and the generated scores.

A score generating function $\varphi(u)$, $u \in (0, 1)$ is a nondecreasing square-integrable function that satisfies the standardizing conditions

$$\int_0^1 \varphi(u)du = 0 \text{ and } \int_0^1 [\varphi(u)]^2 du = 1. \tag{3.2}$$

A score $a(i)$, i is an integer, is based on $\varphi(u)$ and defined as $a(i) = \varphi[i/(n+1)]$. With these scores, we define the rank-based norm as

$$\|\boldsymbol{x}\| = \sum_{i=1}^{n} a[R(x_i)]x_i, \tag{3.3}$$

where $R(x_i)$ is the rank of x_i among all entries of vector $\boldsymbol{x}$. As McKean & Hettmansperger (2016) discuss, the above norm is a pseudo-norm, because it is invariant to constant shifts, i.e. $\|\boldsymbol{x} + a\mathbf{1}\|_\varphi = \|\boldsymbol{x}\|_\varphi$. This is the reason why we have to separate α from $\boldsymbol{\beta}$ in equation (3.1). We cannot estimate both intercept and other coefficients at the same time. In fact, for the rank-based fit of linear model, we first estimate $\boldsymbol{\beta}$ then estimate α by $\hat{\alpha} = \underset{i}{\text{med}}\left\{y_i - \boldsymbol{x}_i^T\hat{\boldsymbol{\beta}}\right\}$. In the LS fit, we have the analogous situation, i.e., if we obtain the OLS estimate of β, then the OLS estimate of the intercept is $\hat{\alpha} = \underset{i}{\text{avg}}\left\{y_i - \boldsymbol{x}_i^T\hat{\boldsymbol{\beta}}\right\}$.

Different selection of score generating functions correspond to different norms. For example, letting $\varphi(u) = \text{sign}[u - \frac{1}{2}]$, where sign is the sign function, yields the L_1 norm. Other common examples include $\varphi(u) = \sqrt{12}[u - \frac{1}{2}]$ (Wilcoxon norm), and $\varphi(u) = \Phi^{-1}(u)$ (normal scores).

Returning to the linear model, it is well known that the least squares estimator $\hat{\boldsymbol{\beta}}_{LS}$ minimizes $\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_{L2}$, where L_2 stands for Euclidean norm. In the same way, the rank-based estimator $\hat{\boldsymbol{\beta}}_\varphi$ is obtained by minimizing dispersion function (Jaeckel, 1972)

$$D(\boldsymbol{\beta}) = \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_\varphi. \tag{3.4}$$

It follows that

$$\hat{\boldsymbol{\beta}}_{\varphi} = \text{Argmin}\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_{\varphi} \tag{3.5}$$

If the random errors int the model are independently and identically distributed, then Jaeckel (1972) shows that $\hat{\boldsymbol{\beta}}_{\varphi}$ is a consistent estimate of $\boldsymbol{\beta}$ with asymptotic distribution

$$\hat{\boldsymbol{\beta}}_{\varphi} \sim N(\boldsymbol{\beta},\ \tau_{\varphi}^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}), \tag{3.6}$$

where τ_{φ} is the scale parameter defined as

$$\tau_{\varphi}^{-1} = \int_0^1 \varphi(u)\varphi_f(u)du \tag{3.7}$$

and

$$\varphi_f(u) = -\frac{f'(F^{-1}(u))}{f(F^{-1}(u))},\ f \text{ and } F \text{ are pdf and cdf of residuals respectively.} \tag{3.8}$$

3.2 Implementation in R Code

In the aforementioned Durbin two-stage procedure, we use OLS for regression. Here, we substitute the OLS fit with the rank-based fit. More specifically, we replace **R** function `lm` , which is in the **R** built-in package `stat`, by function `rfit` in the package `Rfit`. We add an argument "method" in the main function `dbfit` for users to control the method to be used. If method="OLS", then it is our previous algorithm; if method="RANK" then it is the rank-based version.

We also add an argument for the score selection in the rank-based fit. In the case that the user knows the form of the population distribution, they can choose scores which result in more efficient estimators. Similar to the `score` argument in the function `rfit`,

`score=bentscores4` is optimal for a symmetric distribution; `score=bentscores2` is optimal for distributions with lighter tails than the normal distribution; `score=bentscores3` and `score=bentscores4` are optimal for left-tailed and right-tailed distributions, respectively. For the detailed discuss of the score selection, see Section 3.5 and 3.6 in Kloke & McKean (2014). By default, the rank-based version uses the Wilcoxon scores that are generated from the linear function $\varphi(u) = \sqrt{12}[u - \frac{1}{2}]$.

Next we use the `testdata` in package `dbfit` as an illustration. In `testdata`, the data is from a two-phase design with 20 observations in each phase. By specifying `method = "RANK"`, the package implements the rank-based version algorithm. Note that by default, the package uses the OLS version.

```
> library(DBfit)
> data(testdata)
> y<-testdata[,5]
> x<-testdata[,1:4]
> fit1<-dbfit(x,y,1,method="RANK")
> summary(fit1)

Call:
dbfit.default(x = x, y = y, arp = 1, method = "RANK")

Initial rho:
[1] 0.2199898

Final rho:
[1] 0.3783017

Nonstationarity flag:
[1] 0
               beta        SE t-ratio p-value
Intercept -0.127056  0.889413 -0.1429  0.8872
beta_ 1    0.066947  0.070143  0.9544  0.3464
beta_ 2   -0.518683  0.943608 -0.5497  0.5860
beta_ 3   -0.115581  0.095301 -1.2128  0.2333
```

3.3 Comparison of the OLS and Rank-Based Versions Via a Simulated Data Set

In this section, we use a simulation data set to exhibit the robustness of the rank-based version. There are three analyses: in the first one, we analyze the original data with both versions and compare the results; in the second one, we intentionally change one data point to 5, then fit the data; in the third one, we change that data point to 50 and analyze the data.

First, we generate a short series data (N=30) with the two-phase design with normally distributed errors. Without lost of generality, we set all coefficients to be zero. The true ρ is 0.6. The plot of the data is in the left panel of Figure 3.1. The fits of both versions are summarized in Table 3.1 and 3.2. $\hat{\rho}_1$ is the Durbin two-stage estimate and $\hat{\rho}_F$ is the final estimate. The results are quite similar.

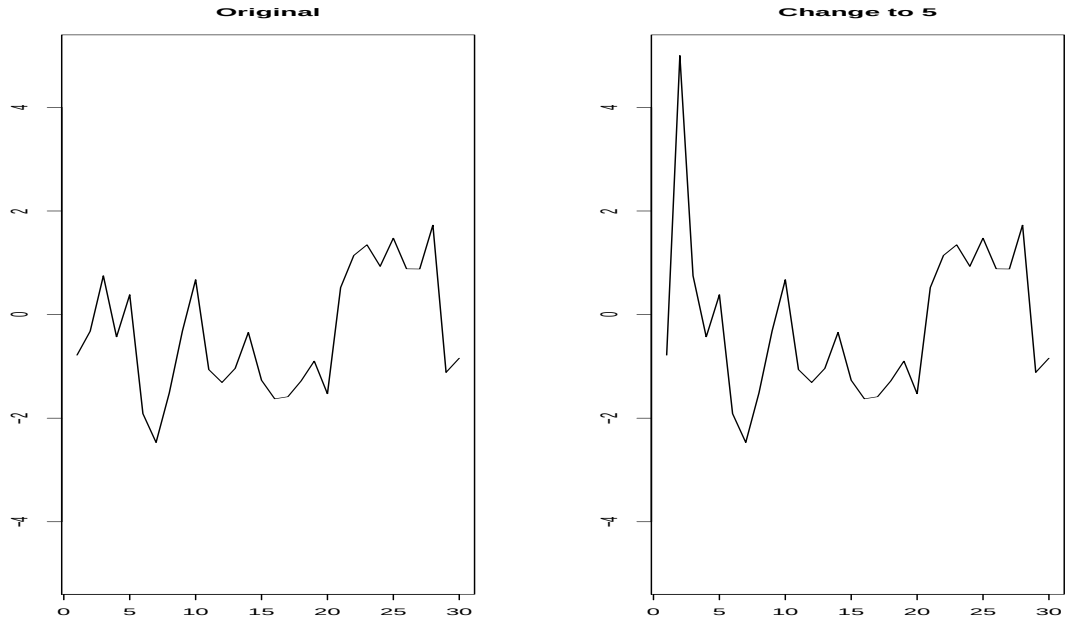


Figure 3.1: Plots of the example

Table 3.1: Comparison of the fits of original data (a)

version	$\hat{\rho}_1$	$\hat{\rho}_F$
rank-based	0.495	0.918
OLS	0.452	0.850

Table 3.2: Comparison of the fits of original data (b)

	OLS				Rank-based			
	Estimate	SE	t-ratio	p-value	Estimate	SE	t-ratio	p-value
Intercept	0.59	2.82	0.21	0.84	-3.00	4.06	-0.74	0.47
beta_ 1	-0.09	0.27	-0.34	0.74	0.23	0.36	0.63	0.54
beta_ 2	-0.20	1.23	-0.16	0.87	-0.75	1.28	-0.59	0.56
beta_ 3	0.16	0.39	0.41	0.69	-0.15	0.40	-0.37	0.71

Next, we change the response value of the second observation to 5, as shown in the right panel in Figure 3.1. Then we fit the model using both the OLS and rank-based fit. As shown in Table 3.3, the rank-based method improves the estimation of autoregressive parameters, because both $\hat{\rho}_1$ and $\hat{\rho}_F$ are closer to the true value 0.6. Table 3.4 shows the estimates of regression coefficients. Although both methods fail to reject the null hypothesis that all beta's are zero, the rank-based version yields smaller values of t-ratio.

Table 3.3: Illustration of robustness of the rank-based version (a)

version	$\hat{\rho}_1$	$\hat{\rho}_F$
rank-based	0.276	0.478
OLS	0.181	0.419

Table 3.4: Illustration of robustness of the rank-based version (b)

	OLS				Rank-based			
	Estimate	SE	t-ratio	p-value	Estimate	SE	t-ratio	p-value
Intercept	1.58	1.59	0.99	0.33	-0.40	1.63	-0.25	0.81
beta_ 1	-0.22	0.16	-1.32	0.20	-0.02	0.16	-0.14	0.89
beta_ 2	0.70	1.52	0.46	0.65	-0.13	1.45	-0.09	0.93
beta_ 3	0.35	0.22	1.57	0.13	0.18	0.22	0.80	0.43

In the third analysis, we change the second data point to 50. In Table 3.5, the OLS fit even yields a negative autocorrelation, while the rank-based estimate is still positive. More importantly, in the summary table of the regression coefficients in Table 3.6, the OLS fit outputs three significant estimates and the rank-based fit shows that none of them are significant. For this example, the rank-based estimates of the regression coefficients exhibit robustness. On the other hand, while the rank-based estimate of ρ was more robust to the changes than the OLS estimate, it was affected in the third situation.

Table 3.5: Illustration of robustness of the rank-based version (c)

version	$\hat{\rho}_1$	$\hat{\rho}_F$
rank-based	0.0179	0.074
OLS	-0.257	-0.146

Table 3.6: Illustration of robustness of the rank-based version (d)

	OLS				Rank-based			
	Estimate	SE	t-ratio	p-value	Estimate	SE	t-ratio	p-value
Intercept	14.61	4.80	3.04	0.01	1.70	11.32	0.15	0.88
beta_ 1	-1.45	0.53	-2.75	0.01	-0.09	0.44	-0.21	0.83
beta_ 2	7.41	6.15	1.21	0.24	0.13	4.85	0.03	0.98
beta_ 3	1.58	0.70	2.26	0.03	0.29	0.62	0.46	0.65

3.4 Simulation Study 4

To further validate the rank-based version of the double bootstrap method, we ran simulation studies with different values of ρ and N using the two-phase interrupted time series model. For each combination, we analyze 5000 simulated data sets, in which $\boldsymbol{\beta} = \mathbf{0}$ and the errors are generated from the normal distribution. Because the computation of the rank-based version usually takes more time than the OLS version, we ran the simulation studies in conjunction with parallel computing, such as the **R** package `snowfall`. In Table 3.7, 3.8, 3.9 and 3.10, we list the actual values of ρ, the empirical means and variances of the estimators of both versions.

Table 3.7: N=20 Robust version

	OLS				Rank-based			
Actual ρ	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.245	0.072	0.543	0.071	0.250	0.079	0.523	0.080
0.7	0.179	0.064	0.504	0.080	0.184	0.072	0.478	0.090
0.5	0.082	0.062	0.414	0.101	0.088	0.068	0.387	0.108
0.3	-0.029	0.058	0.277	0.115	-0.024	0.065	0.249	0.121
0.1	-0.158	0.053	0.100	0.116	-0.151	0.059	0.079	0.122

Table 3.8: N=30 Robust version

	OLS				Rank-based			
Actual ρ	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.451	0.039	0.691	0.036	0.454	0.042	0.677	0.039
0.7	0.359	0.037	0.629	0.048	0.360	0.040	0.610	0.052
0.5	0.226	0.037	0.488	0.063	0.230	0.039	0.468	0.065
0.3	0.084	0.036	0.303	0.065	0.087	0.039	0.286	0.068
0.1	-0.063	0.034	0.108	0.058	-0.061	0.037	0.093	0.061

Table 3.9: N=50 Robust version

	OLS				Rank-based			
Actual ρ	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.639	0.018	0.797	0.015	0.639	0.019	0.790	0.016
0.7	0.503	0.019	0.691	0.027	0.504	0.020	0.680	0.028
0.5	0.341	0.020	0.499	0.029	0.343	0.021	0.490	0.030
0.3	0.179	0.020	0.307	0.028	0.181	0.022	0.299	0.029
0.1	0.002	0.021	0.101	0.028	0.004	0.022	0.094	0.029

Table 3.10: N=100 Robust version

	OLS				Rank-based			
Actual ρ	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F	$\hat{\rho}_1$	Var_1	$\hat{\rho}_F$	Var_F
0.9	0.778	0.006	0.870	0.005	0.779	0.006	0.868	0.005
0.7	0.610	0.008	0.703	0.009	0.611	0.008	0.699	0.010
0.5	0.429	0.009	0.504	0.010	0.429	0.009	0.501	0.011
0.3	0.240	0.010	0.303	0.011	0.241	0.010	0.299	0.011
0.1	0.054	0.010	0.104	0.011	0.054	0.011	0.100	0.012

From the above tables, we can see that when the errors are normally distributed, the final estimates of the rank-based version are slightly smaller than those of the OLS version, and also have a little bit larger variance. However, in the presence of a heavier tailed distribution, we believe the rank-based version, which is based on the Wilcoxon estimator, will perform better than the OLS version.

Next, based on the above simulation data and results, we compare the empirical confidence interval coverages of the four coefficients $\boldsymbol{\beta}$ for both versions, when the nominal confidence is 0.95. We also calculate the asymptotic relative efficiencies (ARE's) of $\boldsymbol{\beta}$ for the two versions. The ARE is defined as: suppose we have two estimators $\hat{\Delta}_1$ and $\hat{\Delta}_2$ for a parameter Δ and they satisfy the condition that $\sqrt{n}(\hat{\Delta}_i - \Delta) \xrightarrow{D} Z \sim N(0, \delta_i^2)$, then the ARE of these two estimators is the reciprocal of the ratio of their variances:

$$\text{ARE}(\hat{\Delta}_1, \hat{\Delta}_2) = \frac{\delta_2^2}{\delta_1^2}. \tag{3.9}$$

If the ARE is greater than 1, then $\hat{\Delta}_1$ is more efficient than $\hat{\Delta}_2$. In our case, the ARE's of the two estimators of $\boldsymbol{\beta}$ are the variances of $\hat{\boldsymbol{\beta}}_{OLS}$ divided by the variances of $\hat{\boldsymbol{\beta}}_{Rank}$.

Table 3.11, 3.12, 3.13 and 3.14 list the results of the empirical coverages at different

sample sizes. As the sample size increases, the empirical coverages are closer to the nominal confidence 0.95. The coverages of the rank-based version are slightly lower than those of the OLS version in any sample size and for any actual ρ.

Table 3.15, 3.16, 3.17 and 3.18 show the results of ARE's. With the facts that all ARE's are smaller than 1, we conclude that the OLS estimators are more efficient than the rank-based estimators when the errors are normally distributed. Recall that the efficiency of the Wilcoxon tests relative to the t-tests at the normal model is 0.955. At N=20, the ARE's are smaller than 0.955. But as the sample size increases, the ARE's are close to 0.955.

Table 3.11: N=20 Empirical Confidence Interval Coverages

	OLS				Rank-based			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.90	0.758	0.821	0.908	0.793	0.759	0.806	0.903	0.774
0.70	0.882	0.890	0.914	0.856	0.868	0.867	0.904	0.837
0.50	0.915	0.914	0.914	0.902	0.905	0.895	0.902	0.884
0.30	0.928	0.931	0.922	0.922	0.919	0.915	0.910	0.907
0.10	0.945	0.937	0.931	0.931	0.931	0.926	0.921	0.922

Table 3.12: N=30 Empirical Confidence Interval Coverages

	OLS				Rank-based			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.90	0.790	0.825	0.921	0.800	0.789	0.803	0.915	0.784
0.70	0.906	0.907	0.922	0.893	0.888	0.884	0.912	0.868
0.50	0.926	0.928	0.924	0.920	0.916	0.915	0.910	0.905
0.30	0.932	0.933	0.931	0.931	0.921	0.916	0.917	0.913
0.10	0.943	0.940	0.941	0.943	0.933	0.933	0.930	0.929

Table 3.13: N=50 Empirical Confidence Interval Coverages

	OLS				Rank-based			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.90	0.844	0.847	0.927	0.832	0.829	0.829	0.924	0.803
0.70	0.922	0.919	0.928	0.917	0.910	0.908	0.923	0.902
0.50	0.936	0.937	0.932	0.931	0.925	0.923	0.921	0.918
0.30	0.946	0.946	0.937	0.945	0.936	0.936	0.925	0.931
0.10	0.943	0.946	0.946	0.947	0.932	0.933	0.933	0.932

Table 3.14: N=100 Empirical Confidence Interval Coverages

	OLS				Rank-based			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.90	0.882	0.888	0.936	0.886	0.874	0.878	0.938	0.872
0.70	0.944	0.941	0.931	0.940	0.929	0.926	0.926	0.925
0.50	0.945	0.945	0.939	0.942	0.933	0.936	0.935	0.932
0.30	0.948	0.947	0.947	0.939	0.944	0.936	0.937	0.935
0.10	0.942	0.940	0.951	0.944	0.935	0.933	0.942	0.937

Table 3.15: Asymptotic Relative Efficiencies of $\boldsymbol{\beta}$ for N=20

ρ	β_0	β_1	β_2	β_3
0.9	0.90	0.88	0.92	0.94
0.7	0.76	0.79	0.93	0.90
0.5	0.70	0.75	0.92	0.86
0.3	0.72	0.78	0.92	0.84
0.1	0.74	0.81	0.92	0.86

Table 3.16: Asymptotic Relative Efficiencies of $\boldsymbol{\beta}$ for N=30

ρ	β_0	β_1	β_2	β_3
0.9	0.86	0.85	0.95	0.93
0.7	0.71	0.75	0.93	0.86
0.5	0.67	0.73	0.92	0.83
0.3	0.77	0.81	0.93	0.86
0.1	0.91	0.92	0.93	0.91

Table 3.17: Asymptotic Relative Efficiencies of $\boldsymbol{\beta}$ for N=50

ρ	β_0	β_1	β_2	β_3
0.9	0.80	0.81	0.96	0.90
0.7	0.66	0.73	0.96	0.85
0.5	0.83	0.86	0.93	0.91
0.3	0.92	0.92	0.93	0.93
0.1	0.94	0.93	0.93	0.94

Table 3.18: Asymptotic Relative Efficiencies of $\boldsymbol{\beta}$ for N=100

ρ	β_0	β_1	β_2	β_3
0.9	0.78	0.81	0.98	0.90
0.7	0.86	0.87	0.95	0.90
0.5	0.94	0.94	0.95	0.94
0.3	0.94	0.93	0.93	0.94
0.1	0.95	0.96	0.94	0.96

Similar to the ARE's of $\boldsymbol{\beta}$, we present the ARE of ρ, which is the ratio of respective mean square errors (MSE) of the OLS and rank-based estimates. Table 3.19 list the

ARE's of ρ at different sample sizes. Except for N=20, the other ARE's are close to 0.955. In the next subsection, we present the results when the model is not normal. And we can see that the rank-based algorithm is superior to the OLS version in that situation.

Table 3.19: Asymptotic Relative Efficiencies of ρ for N=20, 30, 50 and 100

	N=20	N=30	N=50	N=100
0.9	0.89	0.90	0.91	0.94
0.7	0.85	0.88	0.94	0.96
0.5	0.89	0.96	0.96	0.95
0.3	0.93	0.95	0.96	0.96
0.1	0.95	0.96	0.95	0.96

3.5 Simulation Study 5

In all the previous simulation studies, we consider situations that the random errors are normally distributed. However, in this subsection, we consider contaminated normal distributions. The contaminated normal distribution was originally studied by Tukey (1960). It is a simple but useful distribution to simulate outliers. A contaminated normal distribution has the cdf as

$$F(x) = (1-\epsilon)\Phi(x;\mu,\sigma) + \epsilon\Phi(x;\mu,\lambda\sigma) \tag{3.10}$$

where $\Phi(x;\mu,\sigma)$ is the cdf of a normal distribution with mean μ and standard deviation σ; $\lambda > 1$ is the parameter that makes the other normal distribution have a larger standard deviation (heavier tailed); ϵ is the mix probability. So we are sampling from the main distribution with a probability $1-\epsilon$, and from a same mean, but heavier tailed normal distribution with the probability ϵ.

The Wilcoxon estimator has been shown to be more efficient than the LS estimator

in the case of contaminated normal distributions, even with 1% contamination rate (see Kloke & McKean, 2014, p. 72.). So we expect the rank-based version of the double bootstrap method should be superior to the OLS version in the same situations.

We first specify that the uncontaminated component is the standard normal distribution, the contaminated component is a normal distribution with mean 0, standard deviation 100 and the contamination rate is 0.2. Based on this contaminated normal distribution, we simulate the random errors and generate two-phase intervention data with sample size 50. Then we use both versions of the double bootstrap methods to fit the simulation data. Note that for the rank-based fit, we use the default Wilcoxon scores.

Table 3.20 and Table 3.21 list the summaries of both versions. Both the initial and final estimates of ρ in the rank-based fit are better than the OLS version in that the estimates are closer to the true value and have smaller variances. Considering the higher efficiency of the Wilcoxon estimators to the OLS estimates in the case of contaminated normal distributions, the above results are what we expect. In addition, the summary of bias shows that the rank-based estimates have less bias than the OLS. Another thing should be noted is that the initial estimates of the rank-based fit are also close to the true values of ρ. Table 3.22 lists the ARE's of the rank-based estimates and OLS estimates. The ARE's are much greater than 1. So in this contaminated normal model, the Wilcoxon estimates have much higher efficiency than the OLS estimates. All of the results show the advantage of the rank-based version of the double bootstrap method in the case of contaminated normal distributions.

Table 3.20: Summary of the rank-based estimates when contaminated normal errors at N=50

	$\hat{\rho}_1$	Var_1	$bias_{max}$	$bias_{min}$	$bias_{range}$	$\hat{\rho}_F$	Var_F	$bias_{max}$	$bias_{min}$	$bias_{range}$
0.9	0.883	0.003	0.090	-0.881	0.971	0.919	0.003	0.088	-0.873	0.961
0.7	0.691	0.001	0.051	-0.494	0.545	0.710	0.001	0.134	-0.468	0.602
0.5	0.494	0.000	0.194	-0.325	0.519	0.505	0.000	0.280	-0.299	0.579
0.3	0.294	0.001	0.122	-0.395	0.517	0.302	0.001	0.130	-0.395	0.525
0.1	0.094	0.001	0.324	-0.568	0.892	0.099	0.001	0.371	-0.567	0.938

Table 3.21: Summary of the OLS estimates when contaminated normal errors at N=50

	$\hat{\rho}_1$	Var_1	$bias_{max}$	$bias_{min}$	$bias_{range}$	$\hat{\rho}_F$	Var_F	$bias_{max}$	$bias_{min}$	$bias_{range}$
0.9	0.639	0.020	0.090	-1.063	1.153	0.796	0.018	0.090	-0.996	1.086
0.7	0.512	0.017	0.232	-0.690	0.922	0.697	0.025	0.289	-0.664	0.953
0.5	0.362	0.015	0.185	-0.741	0.926	0.518	0.023	0.460	-0.688	1.148
0.3	0.178	0.016	0.318	-0.598	0.916	0.300	0.022	0.541	-0.547	1.087
0.1	-0.007	0.015	0.335	-0.667	1.002	0.088	0.019	0.509	-0.648	1.156

Table 3.22: ARE's of the rank-based estimates (Wilcoxon scores) and OLS estimates of ρ when contaminated normal errors at N=50

Actual ρ	ARE
0.9	8.702
0.7	23.189
0.5	50.361
0.3	39.625
0.1	16.873

Next, we consider the skewed contaminated normal case, in which the contaminated

component has a different mean than the uncontaminated component. We specify the mean of the heavier tailed normal distribution to be 1, while keeping all other setting the same as above. After generating the simulation data, we fit the data with the OLS fit and rank-based fit. However, for the rank-based fit, we compare the results of the Wilcoxon scores and `bentscore1` type scores (see figures in Kloke & McKean, 2014, p. 74.).

Table 3.23, Table 3.24 and Table 3.25 show the results of the Wilcoxon scores, the `bentscore1` type scores and the OLS estimates, respectively. Since the error distribution is right skewed, we expect that the `bentscore1` type scores perform better. In fact, as we can see in the tables, both the initial and final estimates with the `bentscore1` type scores have smaller variances than the Wilcoxon scores. Although the mean estimates are quite similar for both types of scores, the biases for the `bentscore1` type scores generally have smaller maximum values and ranges. Table 3.26 lists the ARE's of the rank-based estimates with the `bentscore1` type scores and the OLS estimates. Again, the ARE's are greater than 1, indicating that the rank-based estimates has higher efficiencies. So, as expected, both rank-based fits are superior to the OLS fit for the skewed contaminated normal distribution.

Table 3.23: Summary of the rank-based estimates with the Wilcoxon scores when skewed contaminated normal errors at N=50

	$\hat{\rho}_1$	Var_1	$bias_{max}$	$bias_{min}$	$bias_{range}$	$\hat{\rho}_F$	Var_F	$bias_{max}$	$bias_{min}$	$bias_{range}$
0.9	0.831	0.004	0.090	-0.597	0.687	0.905	0.004	0.090	-0.558	0.648
0.7	0.654	0.003	0.200	-0.470	0.671	0.711	0.003	0.251	-0.442	0.694
0.5	0.465	0.003	0.190	-0.419	0.609	0.505	0.003	0.264	-0.374	0.638
0.3	0.275	0.003	0.300	-0.325	0.624	0.304	0.003	0.411	-0.303	0.715
0.1	0.081	0.003	0.224	-0.460	0.684	0.102	0.003	0.260	-0.446	0.706

Table 3.24: Summary of the rank-based estimates with the `bentscore1` type scores when skewed contaminated normal errors at N=50

	$\hat{\rho}_1$	Var_1	$bias_{max}$	$bias_{min}$	$bias_{range}$	$\hat{\rho}_F$	Var_F	$bias_{max}$	$bias_{min}$	$bias_{range}$
0.9	0.851	0.002	0.054	-0.454	0.509	0.916	0.002	0.090	-0.418	0.508
0.7	0.666	0.002	0.103	-0.498	0.600	0.708	0.002	0.174	-0.474	0.647
0.5	0.474	0.002	0.174	-0.411	0.585	0.504	0.002	0.234	-0.370	0.604
0.3	0.281	0.002	0.189	-0.317	0.506	0.303	0.002	0.233	-0.303	0.536
0.1	0.086	0.002	0.224	-0.516	0.740	0.102	0.002	0.253	-0.508	0.762

Table 3.25: Summary of the OLS estimates when skewed contaminated normal errors at N=50

	$\hat{\rho}_1$	Var_1	$bias_{max}$	$bias_{min}$	$bias_{range}$	$\hat{\rho}_F$	Var_F	$bias_{max}$	$bias_{min}$	$bias_{range}$
0.9	0.647	0.016	0.090	-0.724	0.814	0.802	0.013	0.090	-0.622	0.712
0.7	0.507	0.015	0.258	-0.593	0.850	0.689	0.022	0.288	-0.499	0.787
0.5	0.349	0.017	0.257	-0.567	0.824	0.501	0.025	0.485	-0.485	0.970
0.3	0.179	0.017	0.374	-0.529	0.904	0.302	0.024	0.670	-0.473	1.143
0.1	0.003	0.016	0.369	-0.542	0.912	0.098	0.022	0.546	-0.507	1.053

Table 3.26: ARE's of the rank-based estimates (`bentscore1` type scores) and OLS estimates of ρ when skewed contaminated normal errors at N=50

Actual ρ	ARE
0.9	8.884
0.7	8.837
0.5	11.224
0.3	10.485
0.1	9.677

Finally we present the validity check for the regression coefficients. The random errors still follow the contaminated normal distribution as discussed above. With the same settings, we generate the data with sample sizes $N = 30$ and $N = 50$. The

rank-based fit with the Wilcoxon scores are still used to analyze both data and the comparisons between the initial and final estimates of $\boldsymbol{\beta}$ are provided. The nominal confidence is 0.95.

Table 3.27 and Table 3.28 list the summaries for $N = 30$ and $N = 50$, respectively. Generally speaking, the final estimates are closer to the nominal confidence 0.95, which the confidence intervals of the initial estimates are liberal.

Table 3.27: The comparison of validity checks between the initial and final estimates of β at N = 30 with contaminated normal errors

	Final				Initial			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.900	0.895	0.907	0.891	0.892	0.877	0.756	0.793	0.730
0.700	0.970	0.958	0.909	0.940	0.951	0.879	0.786	0.861
0.500	0.985	0.981	0.961	0.975	0.965	0.950	0.890	0.937
0.300	0.987	0.982	0.981	0.986	0.979	0.967	0.963	0.973
0.100	0.989	0.986	0.990	0.989	0.977	0.974	0.983	0.977

Table 3.28: The comparison of validity checks between the initial and final estimates of β at N = 50 with contaminated normal errors

	Final				Initial			
ρ	β_0	β_1	β_2	β_3	β_0	β_1	β_2	β_3
0.900	0.965	0.944	0.924	0.902	0.984	0.908	0.826	0.877
0.700	0.991	0.990	0.976	0.988	0.995	0.980	0.885	0.982
0.500	0.993	0.993	0.992	0.990	0.992	0.986	0.979	0.986
0.300	0.991	0.997	0.993	0.996	0.996	0.993	0.996	0.992
0.100	0.990	0.993	0.994	0.992	0.996	0.995	0.996	0.989

All the results in this subsection show that the rank-based algorithm of the double

bootstrap method has better performance than the OLS version in the presence of outliers, which are generated from the contaminated normal distributions.

Chapter 4

The Bivariate Logistic Spatial Model

The application of variational approximations can be first found in the machine learning and computer science field (e.g., Jordan, Ghahramani, Jaakkola, & Saul, 1999, Titterington, 2004 and Winn & Bishop, 2005). Later, it was proved to be effective in various statistical models (e.g., Wang, Titterington et al., 2006, McGrory & Titterington, 2007 and Wand, Ormerod, Padoan, Fuhrwirth et al., 2011). As a Bayesian inference method, the goal is to approximate the true posterior distributions of parameters.

In the field of Bayesian inference, Markov chain Monte Carlo (MCMC) is the most common method. However when dealing with generalized linear models (GLM), especially with generalized linear mixed models (GLMM), there exists the intractability problem. And usually it requires intensive computing to implement MCMC in analyzing such models. In these cases, variational approximations provide a practical alternative to MCMC. In particular, we utilize this method to analyze bivariate logistic spatial model. In this model, both response variables are binary and their correlation is modeled through random effects that follow Conditionally Autoregressive (CAR) dis-

tribution. In the following sections, we will provide a brief review of the CAR model, then present the proposed model.

4.1 Conditionally Autoregressive Model (CAR)

Areal data (e.g. health outcomes) are collected over geographic regions such as counties, census tracts, zip codes, and so on. Conditional Autoregressive Model (CAR), pioneered by Besag (1974), is one of the most popular models to analyze such data. The multivariate conditionally autoregressive (MCAR) model is used as a prior for the spatial random effects in our model. In what follows, we introduce both univariate CAR and MCAR models.

Univariate CAR

Consider a univariate spatially random variable ϕ_i observed at n areal locations, and define $\Phi = (\phi_1, ..., \phi_n)'$. Under the MRF (Markov Random Field) assumption, we specify the *full conditional* distributions as

$$p(\phi_i|\phi_j, j \neq i, \tau_i^{-1}) = N(\alpha \sum_{i \sim j} b_{ij}\phi_j, \tau_i^{-1}), i, j = 1, ..., n, \tag{4.1}$$

where $i \sim j$ denotes that region j is a neighbor of region i. From Hammersley-Clifford Theorem and Brook's Lemma (see, e.g. Banerjee, Carlin, & Gelfand, 2014, section 4.2), the full conditional distributions in (4.1) uniquely determine the joint distribution,

$$\Phi \sim N(0, [D_\tau(I - \alpha B)]^{-1}), \tag{4.2}$$

where B is an $n \times n$ matrix with $b_{ii} = 0$, and $D_\tau = Diag(\tau_i)$. Usually we assume that $D_\tau = \tau D$, where D is an $n \times n$ diagonal matrix. α is a smoothing parameter, and is

often interpreted as measuring spatial association.

Example of Univariate CAR: IAR

From formulation (4.2), we can choose α,D and B to obtain various CAR model structures. The most popular one is the *pairwise difference* formulation, also known as the *intrinsic autoregressive* (IAR) model. In IAR model, we set the smoothing parameter $\alpha = 1$, and $D = Diag(m_i)$, where m_i is the number of neighbors of region i, and $B = D^{-1}W$, where W denotes the adjacency matrix of the map (i.e., $w_{ii} = 0$, and $w_{ii'} = 1$ if $i \sim i'$, and 0 otherwise). Then formulation (4.2) becomes

$$\Phi \sim N(0, [\tau(D - W)]^{-1}), \tag{4.3}$$

Model (4.3) is simple and easy to fit, but has two major drawbacks. First, $\tau(D - W)$ is singular, and thus (4.3) is improper. Second, it contains no parameter to control the strength of spatial dependence among regions.

Multivariate CAR

Next consider the multivariate case:

$$p(\nu_i|\nu_{j\neq i}, \Gamma_i^{-1}) = MVN(R_i \sum_{i\sim j} B_{ij}\nu_j, \Gamma_i^{-1}), i, j = 1, ..., n, \tag{4.4}$$

where $\nu_i = (\phi_{i1}, \phi_{i2}, ..., \phi_{ip})'$ is a p-dimensional vector, and Γ_i^{-1},R_i, and B_{ij} are $p \times p$ matrices. For example, this model might be appropriate for a data set on p types of cancer over n counties. Banerjee et al. (2014) proved that full conditional distributions in (4.4) uniquely determine the joint distribution

$$\nu \sim N(0, [\Gamma(I - B_R)]^{-1}), \tag{4.5}$$

where $\nu' = (\nu_1', \nu_2', ..., \nu_n')$, B_R is $np \times np$ with $(B_R)_{ij} = R_i B_{ij}$, $(B_R)_{ii} = 0$, and Γ is an $np \times np$ block diagonal matrix with $p \times p$ diagonal entries Γ_i.

To obtain a proper joint distribution (4.5), we need to make sure that $\Gamma(I - B_R)$ is positive definite and symmetric. However, it is difficult.
To simply the formulation, we often assume that $R_i = \alpha I_{p\times p}$ for $i = 1, ..., n$ and $\Gamma = D \otimes \Lambda$. Under these assumptions, (4.5) becomes

$$\nu \sim N(0, [(D(I - \alpha B) \otimes \Lambda)]^{-1}), \tag{4.6}$$

where Λ is a $p \times p$ positive definite and symmetric matrix.

4.2 The Proposed Model

Suppose we want to model two binary variables in the framework of logistic regression. The data are collected on m locations with different sample size (n_i , $i = 1, ..., m$) in each location. The total sample size is $N = n_1 + n_2 + \cdots + n_m$.

$$\begin{array}{rcl} y_{ijk} & \sim & Bernoulli(\Pi_{ijk}), \\ logit(\pi_{ijk}) & = & \boldsymbol{x}_{ij}\boldsymbol{\beta}_k + u_{ik}, \end{array} , \tag{4.7}$$

for $i = 1, ..., m$, $j = 1, ..., n_i$ and $k = 1, 2$. Note that here we assume all individuals in the same location share the same spatial random effect, i.e. u_{ik} does not change over subscription j; the predictor variables are the same for both response variables.
We can rewrite the model in a matrix form:

$$\begin{array}{rcl} \boldsymbol{Y}_k & \sim & Bernoulli(\boldsymbol{\Pi}_k) \\ logit(\boldsymbol{\Pi}_k) & = & \boldsymbol{X}\boldsymbol{\beta}_k + \boldsymbol{Z}\boldsymbol{U}_k \end{array} , \; for\ k = 1, 2, \tag{4.8}$$

where $\boldsymbol{Y}_k$ is a $N \times 1$ column vector of the binary response variable; $\boldsymbol{\Pi}_k$ is a $N \times 1$ column vector of the corresponding probabilities of success; $\boldsymbol{X}$ is a $N \times P$ matrix of the p explanatory variables; $\boldsymbol{\beta}_k$ are the fixed effects; $\boldsymbol{Z}$ is a $N \times m$ design matrix for the random effects ; $\boldsymbol{U}_k$ is a $m \times 1$ vector of spatial random effects; and logit function is defined as usual: $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$.

More specifically, the design matrix for the random effects is:

$$\begin{bmatrix} \mathbf{1}_{n_1} & & & & \\ & \ddots & & & \\ & & \mathbf{1}_{n_i} & & \\ & & & \ddots & \\ & & & & \mathbf{1}_{n_m} \end{bmatrix} \tag{4.9}$$

where $\mathbf{1}_{n_i}$ is a $n_i \times 1$ column vector with all elements being 1.

Furthermore, we can eliminate subscription k by stacking variables and using one equation:

$$\begin{array}{rcl} \boldsymbol{Y} & \sim & Bernoulli(\boldsymbol{\Pi}) \\ logit(\boldsymbol{\Pi}) & = & \boldsymbol{X}^*\boldsymbol{\beta} + \boldsymbol{Z}^*\boldsymbol{U} \end{array}, \tag{4.10}$$

where $\boldsymbol{Y} = [\boldsymbol{Y}_1^T \ \ \boldsymbol{Y}_2^T]^T$, a $2N \times 1$ column vector; $\boldsymbol{\Pi}$ are the corresponding probabilities of success; $\boldsymbol{X}^* = I_2 \otimes \boldsymbol{X}$, a $2N \times 2P$ matrix; $\boldsymbol{\beta} = [\boldsymbol{\beta}_1^T \ \ \boldsymbol{\beta}_2^T]^T$, a $2p \times 1$ column vector; $\boldsymbol{Z}^* = I_2 \otimes \boldsymbol{Z}$, a $2N \times 2m$ matrix; $\boldsymbol{U} = [\boldsymbol{U}_1^T \ \ \boldsymbol{U}_2^T]^T$, a $2m \times 1$ column vector.

Next we impose prior distributions to the fixed effects $\boldsymbol{\beta}$ and random effects $\boldsymbol{U}$:

$$\begin{array}{rcl} \boldsymbol{U} & \sim & N(0, \Lambda \otimes Q^{-1}),\ Q = M - \alpha A \\ \boldsymbol{\beta} & \sim & N(0, \sigma_\beta^2 I_{2p}) \end{array}, \tag{4.11}$$

where $\sigma_\beta^2 = 10^6$; A is the adjacency matrix and M is a diagonal matrix with each entry being the number of neighbors of the corresponding site; Λ is the covariance matrix for

the pair of random effects in each location. We use the result of Huang, Wand et al. (2013) for the prior distribution of Λ:

$$\begin{array}{rcl} \Lambda|a_1, a_2 & \sim & Inverse - Wishart(\nu + 1, 2\nu \text{ diag } (1/a_1, 1/a_2)) \\ a_1, a_2 & \sim & Inverse - Gamma(1/2, 1/G_a^2) \end{array} . \tag{4.12}$$

where $G_a^2 = 10^5$.

The optimal q-density for $\boldsymbol{\beta}$ and $\boldsymbol{U}$:

$$\begin{aligned} \log(q^*(\boldsymbol{\beta}, \boldsymbol{U})) &\propto E[\boldsymbol{Y}^T(\boldsymbol{X}^*\boldsymbol{\beta} + \boldsymbol{Z}^*\boldsymbol{U}) - \mathbf{1}^T \log(1 + exp(\boldsymbol{X}^*\boldsymbol{\beta} + \boldsymbol{Z}^*\boldsymbol{U})) \\ &\quad -\tfrac{1}{2}\boldsymbol{U}^T\Lambda^{-1} \otimes Q\boldsymbol{U} - \tfrac{1}{2}\boldsymbol{\beta}^T\sigma_\beta^{-2}I_{2p}\boldsymbol{\beta}] \\ &= \boldsymbol{Y}^T\boldsymbol{C}\boldsymbol{\nu} - \mathbf{1}^T \log(1 + exp(\boldsymbol{C}\boldsymbol{\nu})) - \tfrac{1}{2}\boldsymbol{\nu}^T \begin{bmatrix} \sigma_\beta^{-2}I_{2p} & \mathbf{0} \\ \mathbf{0} & E(\Lambda^{-1} \otimes Q) \end{bmatrix} \boldsymbol{\nu} \\ &\geq \boldsymbol{Y}^T\boldsymbol{C}\boldsymbol{\nu} + [\boldsymbol{\nu}^T\boldsymbol{C}^T \text{ diag } (\lambda(\boldsymbol{\xi}))\boldsymbol{\nu}\boldsymbol{C} - \tfrac{1}{2}\mathbf{1}^T\boldsymbol{C}\boldsymbol{\nu}] - \tfrac{1}{2}\boldsymbol{\nu}^T \begin{bmatrix} \sigma_\beta^{-2}I_{2p} & \mathbf{0} \\ \mathbf{0} & E(\Lambda^{-1} \otimes Q) \end{bmatrix} \boldsymbol{\nu} \\ &= -\tfrac{1}{2}\boldsymbol{\nu}^T \left\{ \begin{bmatrix} \sigma_\beta^{-2}I_{2p} & \mathbf{0} \\ \mathbf{0} & E(\Lambda^{-1}) \otimes Q \end{bmatrix} - 2\boldsymbol{C}^T \text{ diag } (\lambda(\boldsymbol{\xi}))\boldsymbol{C} \right\} \boldsymbol{\nu} + (\boldsymbol{Y}^T - \tfrac{1}{2}\mathbf{1}^T)\boldsymbol{C}\boldsymbol{\nu} \end{aligned} , \tag{4.13}$$

where $\boldsymbol{C} = [\boldsymbol{X}^* \ \boldsymbol{Z}]$; $\boldsymbol{\nu} = [\boldsymbol{\beta}^T \ \boldsymbol{U}^T]^T$; $E_q(\Lambda^{-1})$ is the mean matrix of q-density of Λ^{-1}. We also use the facts that:$-\log(1 + e^x) = \max_{\xi \in \mathbb{R}} \left\{ \lambda(\xi)x^2 - \frac{1}{2}x + \psi(\xi) \right\}$, where $\lambda(\xi) = -\tanh(\xi/2)/(4\xi)$ and $\psi(\xi) = \xi/2 - \log(1+e^\xi) + \xi \tanh(\xi/2)$; $E(\Lambda^{-1} \otimes Q) = E(\Lambda^{-1}) \otimes Q$. Hence the optimal q-density for $\boldsymbol{\beta}$ and $\boldsymbol{U}$ is multivariate normal distribution

$$
\begin{aligned}
q^*(\boldsymbol{\beta}, \boldsymbol{U}; \boldsymbol{\xi}) &\sim N(\boldsymbol{\mu}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})}, \boldsymbol{\Sigma}_{q((\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi}))}) \\
\boldsymbol{\Sigma}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})} &= \left\{ \begin{bmatrix} \sigma_\beta^{-2} I_{2p} & \mathbf{0} \\ \mathbf{0} & E_q(\Lambda^{-1}) \otimes Q \end{bmatrix} - 2\boldsymbol{C}^T \operatorname{diag}(\lambda(\boldsymbol{\xi}))\boldsymbol{C} \right\}^{-1} . \qquad (4.14) \\
\boldsymbol{\mu}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})} &= \boldsymbol{\Sigma}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})} \boldsymbol{C}^T (\boldsymbol{Y} - \tfrac{1}{2}\mathbf{1})
\end{aligned}
$$

The optimal q-density for Λ:

$$
\begin{aligned}
\log q^*(\Lambda) &\propto \log P(\boldsymbol{U}|\Lambda) + \log P(\Lambda|a_1, a_2) \\
&= \log|\Lambda \otimes Q^{-1}|^{-\frac{1}{2}} - \tfrac{1}{2}\boldsymbol{U}^T(\Lambda^{-1} \otimes Q)\boldsymbol{U} - \tfrac{\nu+1+p+1}{2}\log|\Lambda| - \tfrac{1}{2}tr(S_0\Lambda^{-1}) \\
&= \log(|Q^{-1}|^2|\Lambda|^n)^{-\frac{1}{2}} - \tfrac{1}{2}tr(E([\boldsymbol{U}_1\ \boldsymbol{U}_2]^T Q[\boldsymbol{U}_1\ \boldsymbol{U}_2])\Lambda^{-1}) - \tfrac{\nu+1+p+1}{2}\log|\Lambda| \\
&\quad -\tfrac{1}{2}tr(E(S_0)\Lambda^{-1}) \\
&= -\tfrac{\nu+1+n+p+1}{2}\log|\Lambda| - \tfrac{1}{2}tr(E(S_0 + S_1)\Lambda^{-1})
\end{aligned} \qquad (4.15)
$$

where $S_1 = [\boldsymbol{U}_1\ \boldsymbol{U}_2]^T Q[\boldsymbol{U}_1\ \boldsymbol{U}_2]$.

Next,

$$
\begin{aligned}
E(S_1) &= \begin{bmatrix} E(\boldsymbol{U}_1^T Q \boldsymbol{U}_1) & E(\boldsymbol{U}_1^T Q \boldsymbol{U}_2) \\ E(\boldsymbol{U}_2^T Q \boldsymbol{U}_1) & E(\boldsymbol{U}_2^T Q \boldsymbol{U}_2) \end{bmatrix} \\
&= \begin{bmatrix} E(\boldsymbol{U}_1^T Q \boldsymbol{U}_1) & E(\boldsymbol{U}_1^T Q \boldsymbol{U}_2) \\ E(\boldsymbol{U}_2^T Q \boldsymbol{U}_1) & E(\boldsymbol{U}_2^T Q \boldsymbol{U}_2) \end{bmatrix} \\
&= \begin{bmatrix} tr(Q\boldsymbol{\Sigma}_{11}) + \boldsymbol{\mu}_1^T Q \boldsymbol{\mu}_1 & tr(Q\boldsymbol{\Sigma}_{21}) + \boldsymbol{\mu}_1^T Q \boldsymbol{\mu}_2 \\ tr(Q\boldsymbol{\Sigma}_{12}) + \boldsymbol{\mu}_2^T Q \boldsymbol{\mu}_1 & tr(Q\boldsymbol{\Sigma}_{22}) + \boldsymbol{\mu}_2^T Q \boldsymbol{\mu}_2 \end{bmatrix} \\
E(S_0) &= 2\nu \ \text{diag}\ (\mu_{1/a_1},\ \mu_{1/a_2})
\end{aligned}
\tag{4.16}
$$

where $\mu_{1/a_1},\ \mu_{1/a_2}$ are the mean of q-density of $a_1,\ a_2$; $\boldsymbol{\mu}_1 = E(U_1)\ , \boldsymbol{\mu}_2 = E(U_2)$; $\boldsymbol{\Sigma}_{11} = Var(\boldsymbol{U}_1)$, $\boldsymbol{\Sigma}_{22} = Var(\boldsymbol{U}_2)$, $\boldsymbol{\Sigma}_{12} = \boldsymbol{\Sigma}_{21}^T = Cov(\boldsymbol{U}_1, \boldsymbol{U}_2)$. So the optimal q-density for Λ is Inverse-Wishart distribution with degree freedom df=$\nu+n+1$ and scale matrix $S_\Lambda = E(S_0 + S_1)$.

The optimal q-density for a_1 and a_2:

$$
a_1,\ a_2 \sim \text{Inverse-Gamma}\left(\frac{\nu+2}{2},\ \nu\left(E_q(\Lambda^{-1})\right)_{rr} + 1/G_a^2\right),\ r = 1, 2. \tag{4.17}
$$

where $E_q(\Lambda^{-1})$ is the mean matrix of q-density of Λ^{-1}.

The algorithm is described as following:

Initialize: $E_q(\Lambda^{-1})$ (2×2 matrix, positive definite), $\boldsymbol{\xi}$ ($2N\times1$ vector; all entries positive), $\mu_{1/a_1},\ \mu_{1/a_2}$ (two positive real numbers)

Cycle:

Update $\boldsymbol{\Sigma}_{q(\boldsymbol{\beta},\boldsymbol{U},\boldsymbol{\xi})}$ and $\boldsymbol{\mu}_{q(\boldsymbol{\beta},\boldsymbol{U},\boldsymbol{\xi})}$ for $q^*(\boldsymbol{\beta}, \boldsymbol{U})$:

Update $\boldsymbol{\xi} \leftarrow \sqrt{\text{diag}\left\{\boldsymbol{C}(\boldsymbol{\Sigma}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})} + \boldsymbol{\mu}_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})}\boldsymbol{\mu}^T_{q(\boldsymbol{\beta},\boldsymbol{U};\boldsymbol{\xi})})\boldsymbol{C}^T\right\}}$

Update $E(S_1)$ and $E(S_0)$

Update $E_q(\Lambda^{-1}) = \frac{df}{2}[E(S_0) + E(S_1)]^{-1}$

Update $\mu_{1/a_1},\ \mu_{1/a_2}$

repeat until the increase in $\underline{p}(y;\theta)$ is negligible.

Here T is a predefined number of iterations.

4.3 Simulation Studies

This simulation is based on a 9×9 square lattice. So there are 81 sites, with 50 observations in the first 40 locations, and 100 obs in the other 41 locations. This is unbalanced design and total number of observations is N=6100. The data has two binary response variables, only one covariate. Both responses share the same covariate. Parameters: $\Lambda = \begin{bmatrix} 1 & 1 \\ & \\ 1 & 2 \end{bmatrix}$ $\boldsymbol{\beta}_1 = [2\ 3]^T$, $\boldsymbol{\beta}_2 = [1\ 4]^T$; smoothing parameter α is fixed at 0.9. Number of iterations is 5000. The algorithm was implemented in the **R** computing environment.

Figure 4.1 shows the values of $\log\underline{p}(y;\theta)$ along with the iteration number. We can see that the convergence is quite rapid. Within two minutes and only about 70 iterations, the increase in the log of the lower bound is smaller than 0.001 and the loop converges. It is known that for a complicated model like this (about 200 parameters), the MCMC usually needs to collect thousands samples to ensure the accuracy, which can take hours.

Figures 4.2 and 4.3 show the trace plots of the estimates of the parameters of interest. When the loop stops, the estimates are quite close to the true values. As comparison, the traditional MCMC is performed using **BUGS** software (Lunn & Spiegelhalter,

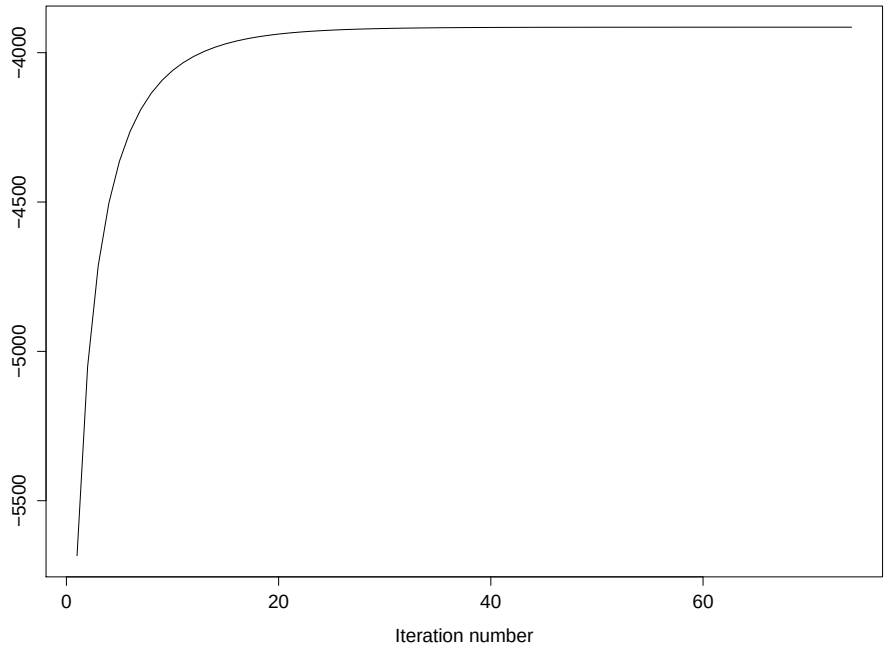


Figure 4.1: Successive values of log of the lower bound

2000). The result is summarized in Table 4.1. We can see that the results of both algorithms are quite similar. But for the MCMC, we collected 100,000 samples with 5000 as burn-in. And this takes about 10 hours. We achieve the similar accuracy using the Variational Bayes algorithm within several minutes.

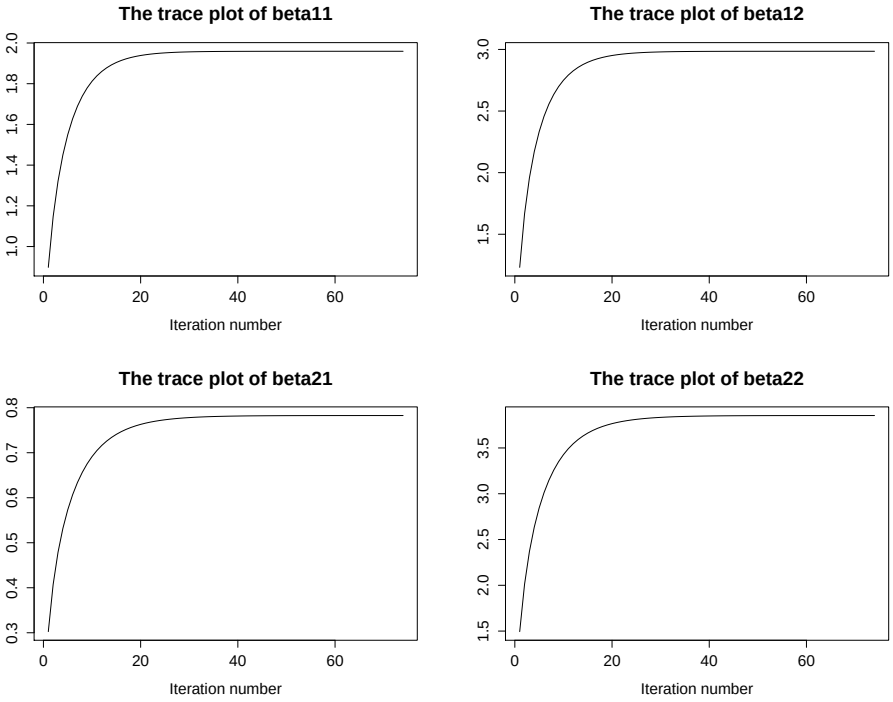


Figure 4.2: Trace Plots for the estimates of regression coefficients

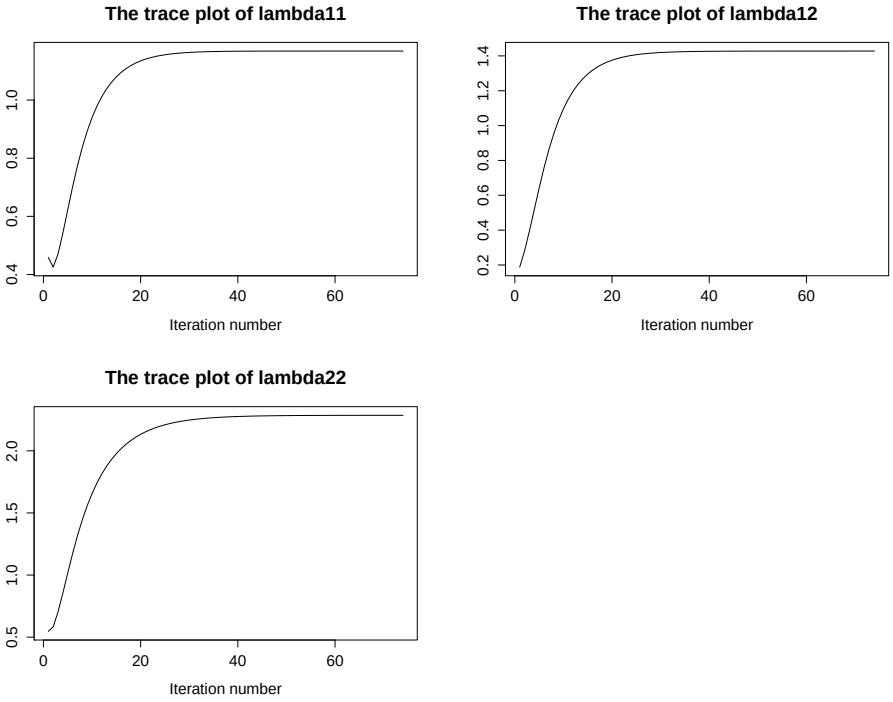


Figure 4.3: Trace Plots for the estimates of covariance matrix Λ

Table 4.1: Summary of MCMC

	mean	sd	2.5%	25%	50%	75%	97.5%
beta1[1]	2.02	0.06	1.90	1.98	2.02	2.06	2.14
beta1[2]	3.01	0.08	2.85	2.96	3.01	3.07	3.18
beta2[1]	0.84	0.05	0.74	0.80	0.84	0.87	0.94
beta2[2]	3.90	0.11	3.69	3.82	3.89	3.97	4.11
lambda[1,1]	1.22	0.27	0.77	1.02	1.19	1.38	1.83
lambda[1,2]	1.39	0.30	0.89	1.18	1.36	1.57	2.07
lambda[2,1]	1.39	0.30	0.89	1.18	1.36	1.57	2.07
lambda[2,2]	2.30	0.48	1.52	1.97	2.25	2.59	3.38

Chapter 5

Conclusions

Throughout Chapter 1 to 3, we considered the following model:

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + u_t \ , \quad t = 1, \ldots, N \ , \tag{5.1}$$

where y_t is the dependent variable, $\mathbf{x}_t$ is a $(p+1) \times 1$ vector of independent variables, the error term u_t follows a stationary autoregressive series with order k,

$$u_t = \rho_1 u_{t-1} + \cdots + \rho_k u_{t-k} + e_t \ , \tag{5.2}$$

and the errors e_t are independently and identically distributed with mean zero and finite variance.

In Chapter 1 and 2, we briefly discussed the main algorithm for analyzing this model, i.e. the double bootstrap method proposed by McKnight et al. (2000). In Chapter 2, based on their algorithm, we developed the **R** package `DBfit`. The instructions of this package are demonstrated using a real data example and simulation data.

We also found the issue of overestimating the autoregressive parameter ρ in the original algorithm. And we provided a tentative solution to that issue. A simulation

study indicated that this solution worked well as expected. In the future, instead of using the current solution, we will study the first order differencing to address the .99 issue. Furthermore, large simulation studies with different combinations of the sample size and different predefined ρ showed that this **R** package worked as well as the original Fortran version software.

In Chapter 3, we developed a robust rank-based version of the double bootstrap method that is resistant to outliers. This algorithm used some functions from the `Rfit` R package which was developed by Kloke and McKean. The robustness of this new version was presented in analyzing a simulated data where the value of a observation was changed intentionally. The results fully stated the advantages of this new version in the present of outliers. Also, as what we did in Chapter 2, several simulation studies under different settings of the sample size and ρ were performed to further validate the robust version. The results of the simulation studies are quite satisfactory. Lastly, we studied the contaminated normal model using both OLS version and rank-based version. Not surprisingly, the results showed that the rank-based algorithm is superior to the OLS version in that it not only yields valid estimates, but it has much higher efficiency.

In Chapter 4, we utilized the variational approximation method to fit the multivariate mixed logistic regression, in which the random effects are assumed to come from the Multivariate Conditionally Autoregressive (MCAR) process. The spatial processes were proved to be quite useful to model some diseases across the areas. The model is as following:

$$\begin{array}{rcl} \boldsymbol{Y} & \sim & Bernoulli(\boldsymbol{\Pi}) \\ logit(\boldsymbol{\Pi}) & = & \boldsymbol{X}^*\boldsymbol{\beta} + \boldsymbol{Z}^*\boldsymbol{U} \end{array}, \tag{5.3}$$

A detailed derivation of the q-densities that approximate the true posterior distribution

were presented and a loop algorithm was provided. As a comparison, the traditional Monte Carlo Markov Chain algorithm was also developed using WinBUGS, since the MCMC is the most commonly used procedure in the Bayesian framework. These two algorithms were then compared through simulation studies. The result showed the speedup and accuracy of our proposed Variational Bayes algorithm for analyzing the multivariate mixed logistic model.

In the future, we need to check the power of the significance tests of $\boldsymbol{\beta}$ for both versions of the package `DBfit`. We are still validating the proposed confidence interval of ρ by McKnight et al. (2000). In the same time, we have also proposed several types of the confidence interval of ρ and we are comparing all of them in the simulation study. Our goal is to dramatically reduce the computation time without losing too much accuracy and efficiency.

Appendix A

Functions in R Package DBfit

```
boot1 <-
  function(y,phi1,arp,nbs,x,allb,method){

    upper <- .99
    lower <- -.99
    for(j in 1:arp){
      if(phi1[j] < lower){phi1[j] <- lower}
      if(phi1[j] > upper){phi1[j] <- upper}
    }

    icent <- 0
    phia <- phi1
    xcopy <- x
    xcpy <- x
    n <- length(y)
    yy <- y
    p <-length(x[1,])
    adj <- (n - arp - p)/(n - arp - 2*p)

    ypart <- nurho(y,phi1)
    n1 <- n - arp

    xpart <- wrho(x,phi1)
    ehat <- ypart - xpart%*%allb

    ehat <- (ehat - mean(ehat))*sqrt(adj)

    #     bootstrap loop

    ind <- 1:n1
    bsr1 <- matrix(rep(0,nbs*arp),ncol=arp)

    for(nbk in 1:nbs){
      ystar <- yy[1:arp]
      ind2 <- sample(ind,n1,replace=TRUE)
      estar <- ehat[ind2]

      for(i in (arp+1):n){
        ypart <- 0
        for(k in 1:arp){
          ypart <- ypart + phia[k]*ystar[i-k]
```

```
        }
        xpart <- 0
        for(j in 1:p){
          for(k in 1:arp){
            if(k == 1){
              xpart <- xpart + allb[j]*(xcopy[i,j]-phia[k]*xcopy[i-k,j])
            } else {
              xpart <- xpart - allb[j]*phia[k]*xcopy[i-k,j]
            }
          }
        }
        ystar[i] <- ypart + xpart + estar[i-arp]
      }
      if(icent == 1){
        avey <- mean(ystar)
        ystar <- ystar - avey
      }
      dfit1 <- durbin1fit(ystar,xcpy,arp,method=method)
      adjphi1 <- dfit1$coef[2:(arp+1)]
      for(k in 1:arp){
        if(adjphi1[k] < lower){adjphi1[k] <- lower}
        if(adjphi1[k] > upper){adjphi1[k] <- upper}
        bsr1[nbk,k] <- phi1[k] - adjphi1[k]
      }
    }

    bias1 <- apply(bsr1,2,mean)
    return(bias1)
  }

boot2 <-
  function(y,xcopy,phi1,beta,nbs,method){
    #
    #   beta includes intercept

    arp <- length(phi1)
    phi <- phi1
    p <- length(beta)
    n <- length(y)
    zn1 <- n-p-arp
    zn2 <- n-2*p-arp
    if(zn1 < 0){zn1 <- 1}
    if(zn2 < 0){zn2 <- 1}
    adj <- zn1/zn2
    icent <- 1

    n1 <- n - arp
    ones <- rep(1,n)
    proj1 <- ones%*%t(ones)/n
    x2 <- xcopy[,2:p]
    xbar <- apply(x2,2,mean)
    xc <- xcopy[,2:p] - proj1%*%xcopy[,2:p]
    x <- xc
    p <- p - 1
    ehat <- rep(0,n1)

    for(i in (arp+1):n){
      ypart <- y[i]
      for(k in 1:arp){ypart <- ypart - phi[k]*y[i-k]}
      xpart <- 0
      for(j in 1:(p+icent)){
        for(k in 1:arp){
          if(k == 1){
            xpart <- xpart + beta[j]*(xcopy[i,j] - phi[k]*xcopy[i-k,j])
          } else {
```

```
            xpart <- xpart - beta[j]*phi[k]*xcopy[i-k,j]
          }
        }
      }
      ehat[i-arp] <- ypart - xpart
    }
    ehat <- (ehat - mean(ehat))*sqrt(adj)
    sigehat <- sd(ehat)*sqrt((n-arp)/(n-arp-1))

    oldb <- beta
    pp1 <- p + icent
    bsbeta <- matrix(rep(0,pp1^2),ncol=pp1)


    ind <- 1:n1
    ii <- sample(ind,1,replace=TRUE)
    allbeta <- c()
    rhostar <- c()
    MSEstar <- c()
    for(nbk in 1:nbs){
      ind2 <- sample(ind,n1,replace=TRUE)
      ystar <- y[ii:(ii+arp-1)]
      estar <- ehat[ind2]

      for(i in (arp+1):n){
        ypart <- 0
        for(k in 1:arp){ypart <- ypart + phi[k]*ystar[i-k]}
        xpart <- 0
        for(j in 1:pp1){
          for(k in 1:arp){
            if(k == 1){
              xpart <- xpart + beta[j]*(xcopy[i,j]-phi[k]*xcopy[i-k,j])
            } else {
              xpart <- xpart - beta[j]*phi[k]*xcopy[i-k,j]
            }
          }
        }
        ystar[i] <- ypart + xpart + estar[i-arp]
      }

      avey <- mean(ystar)
      sigestar <- sd(estar*(n/(n-1)))
      MSEstar <- c(MSEstar, sigestar ^ 2)
      ystar <- ystar - avey

      d2fit <- durbin2fit(ystar,x,phi,method=method)
      dum3 <- d2fit$beta
      d1 <- avey - t(xbar)%*%dum3
      dum3 <- c(d1,dum3)

      uhat <- y - xcopy %*% dum3
      u.y <- tail(uhat, length(uhat)-1)
      u.x <- head(uhat, length(uhat)-1)
      ufit <- lm(u.y ~ u.x)
      rhotmp <- ufit$coef[2:(arp+1)]
      rhostar <- rbind(rhostar, rhotmp)

      allbeta <- rbind(allbeta, dum3)

      for(i in 1:pp1){
        for(j in 1:pp1){
          bsbeta[i,j] <- bsbeta[i,j]+(dum3[i]-oldb[i])*(dum3[j]-oldb[j])/sigestar^2
        }
      }
    }
```

```
172
173     bsbeta <- bsbeta / nbs
174
175     list(betacov = bsbeta, allbeta = allbeta, rhostar = rhostar, MSEstar = MSEstar)
176   }
177
178 dbfit.default <-
179   function(x, y, arp, nbs=500, nbscov=500, conf=0.95, CritVal="z", correction=TRUE,
        method="OLS",mod=1, ...)
180   {
181     x <- as.matrix(x)
182     y <- as.numeric(y)
183     est <- simula(x=x, y=y, arp=arp, nbs=nbs, nbscov=nbscov, conf=conf, CritVal=CritVal
          ,method=method,mod=mod, ...)
184     if ((est$adjar >= 0.99) && correction){
185       est <- simulacorrection(x=x, y=y, arp=arp, nbs=nbs, nbscov=nbscov, conf=conf,
            CritVal=CritVal,method=method, ...)
186     }
187     ## est$fitted.values <- as.vector(x %*% est$coefficients)
188     ## est$residuals <- y - est$fitted.values
189     est$call <- match.call()
190     class(est) <- "dbfit"
191     est
192   }
193
194 dbfit.formula <-
195   function(formula, data=list(),arp,nbs=500,nbscov=500,conf=0.95,CritVal='z',correction
        =TRUE,method="OLS",...)
196   {
197     mf <- model.frame(formula=formula, data=data)
198     x <- model.matrix(attr(mf, "terms"), data=mf)
199     y <- model.response(mf)
200     est <- dbfit.default(x=x, y=y,arp=arp,nbs=nbs,nbscov=nbscov,conf=conf,
201                          CritVal=CritVal,correction=correction,method=method, ...)
202     est$call <- match.call()
203     est$formula <- formula
204     est
205   }
206
207 dbfit <-
208   function(x, ...) UseMethod("dbfit")
209
210 durbin1fit <-
211   function(y,x,arp,method){
212
213     xy <- durbin1xy(y,x,arp)
214
215     m <- length(xy[1,])
216     y2 <- xy[,m]
217     x2 <- xy[,1:(m-1)]
218     if(method=="OLS"){
219       fit <- lm(y2 ~ x2)
220     } else if (method=="RANK") {
221       fit <- suppressWarnings(rfit(y2 ~ x2))
222     }
223     return(fit)
224   }
225
226 durbin1xy <-
227   function(y,x,arp){
228
229     lagy <- lagmat(y,arp)
230     y2 <- lagy[,1]
231     part1 <- lagy[,2:(arp+1)]
232     n <- length(y)
```

```
    p <- length(x[1,])

    s1 <- arp + 1
    s2 <- n
    part2 <- lagx(x[,2:p],s1,s2)
    allx <- cbind(part1,part2)

    for(j in 1:arp){
      s1 <- s1 -1
      s2 <- s2 -1
      part3 <- lagx(x[,2:p],s1,s2)
      allx <- cbind(allx,part3)
    }

    durbin1xy <- cbind(allx,y2)
    return(durbin1xy)
  }

durbin2fit <-
  function(yc,xc,adjphi,method){
    p<-ncol(xc)
    arp <- length(adjphi)
    n <- length(yc)

    nuy <- nurho(yc,adjphi)
    wx <- wrho(xc,adjphi)

    if(method=="OLS"){
      fitls <- lm(nuy ~ wx - 1)
    } else if (method=="RANK") {
      fitls <- rfit(nuy ~ wx - 1)
    }

    resd <- fitls$resid
    beta <- fitls$coef
    n1 <- length(resd)

    sigma2 <- var(resd)
    sigma2 <- (n1/(n1 - 2*p -1))*sigma2
    sigma <- sqrt(sigma2)

    list(beta=beta,sigma=sigma)
  }

hmdesign2 <-
  function (n1, n2)
  {
    n = n1 + n2
    c1 = rep(1, n)
    c2 = 1:n
    c3 = c(rep(0, n1), rep(1, n2))
    c4 = c(rep(0, (n1 + 1)), 1:(n2 - 1))
    hmdesign2 = cbind(c1, c2, c3, c4)
    hmdesign2
  }

hmmat <-
  function(vecss,k){

    n <- sum(vecss)
    xmat <- matrix(rep(0,n*2*k),ncol=(2*k))

    ic <- 1
    ir <- 1
```

```
    for(i in 1:k){
      ni <- vecss[i]
      xmat[ir:(ir+ni-1),ic] <- 1
      if(i==1){
        xmat[ir:(ir+ni-1),ic+1] <- 1:ni
      } else {
        xmat[ir:(ir+ni-1),ic+1] <- 0:(ni-1)
      }
      if(i > 1){
        ics <- 1
        for(j in 1:(i-1)){
          last <- xmat[ir-1,ics+1]
          xmat[ir:(ir+ni-1),ics] <- 1
          xmat[ir:(ir+ni-1),ics+1] <- last + (1:ni)
          ics <- ics+2
        }
      }
      ic <- ic + 2
      ir <- ir + ni
    }
    return(xmat)
  }

hypothmat <-
  function(sfit,mmat,n,p){

    q <- length(mmat[,1])
    bpart <- mmat%*%sfit$coefficients
    varpart <- mmat%*%sfit$betacov%*%t(mmat)
    tst <- t(bpart)%*%solve(varpart)%*%bpart
    # pv <- 1 - pchisq(tst,q)
    pvf <- 1 - pf(tst/q,q,n-p)
    hypothmat <- c(tst,pvf)
    return(hypothmat)
  }

lagmat <-
  function (x, p)
  {
    n <- length(x)
    xmat <- matrix(ncol = p, nrow = n - p)
    resp <- x[(p + 1):n]
    for (j in 1:p) {
      xmat[, j] <- x[(p - j + 1):(n - j)]
    }
    lagmat <- cbind(resp, xmat)
    return(lagmat)
  }

lagx <-
  function(x,s1,s2){
    lagx <- x[s1:s2,]
    return(lagx)
  }

nurho <-
  function(yc,adjphi){

    arp <- length(adjphi)
    lagdata <- lagmat(yc,arp)
    yresp <- lagdata[,1]
    ylag <- lagdata[,2:(arp+1)]
    ylag <- as.matrix(ylag)
    nurho <- yresp - ylag%*%adjphi
    return(nurho)
```

```
  }

print.dbfit <-
  function(x, ...)
  {
    cat("Call:\n")
    print(x$call)
    cat("\nCoefficients:\n")
    print(x$coefficients)
  }

print.summary.dbfit <-
  function(x, ...)
  {
    cat("Call:\n")
    print(x$call)

    cat("\nInitial rho:\n")
    print(x$rho1)

    cat("\nFinal rho:\n")
    print(c(x$adjar, x$CI_rho))

    cat("\nNonstationarity flag:\n")
    print(x$flag99)

    printCoefmat(x$tab, P.value=TRUE, has.Pvalue=TRUE)
  }

rhoci2 <-
  function(n,rho,cv){

    rat <- (1-rho)/(1+rho)
    u1 <- exp(-cv*(2/sqrt(n-3)))
    u2 <- exp(cv*(2/sqrt(n-3)))
    ub <- (1-rat*u1)/(1+rat*u1)
    lb <- (1-rat*u2)/(1+rat*u2)
    rhoci <- c(rho,lb,ub)
    return(rhoci)
  }

simpgen1hm2 <-
  function (n1, n2, rho, beta = c(0, 0, 0, 0))
  {
    n <- n1 + n2
    nstop <- 500 + n
    err <- rnorm(1)
    for (i in 2:nstop) {
      err[i] <- rho * err[i - 1] + rnorm(1)
    }
    errs <- err[501:nstop]
    xmat <- hmdesign2(n1, n2)
    y <- xmat %*% beta + errs
    mat <- cbind(xmat, y)
    return(mat)
  }

simula <-
  function(x,y,arp,nbs,nbscov, conf, CritVal,method,mod) {
    upper <- .99
    lower <- -.99
    n <- length(y)
    p <- length(x[1,])
    df <- n - p - arp
    icent <- 0
```

```
428     if (p > 1) {
429       icent <- 1
430     }
431
432     xcopy <- x
433
434     ones <- rep(1,n)
435     proj1 <- ones %*% t(ones) / n
436     x2 <- x[,2:p]
437     xbar <- apply(x2,2,mean)
438     xc <- x[,2:p] - proj1 %*% x[,2:p]
439     x <- xc
440     p <- p - 1
441
442     ##   durb1aa
443     pcent <- p + icent
444
445     dfit <- durbin1fit(y,xcopy,arp,method=method)
446     adjphi <- dfit$coef[2:(arp + 1)]
447     for (j in 1:arp) {
448       if (adjphi[j] < lower) {
449         adjphi[j] <- lower
450       }
451       if (adjphi[j] > upper) {
452         adjphi[j] <- upper
453       }
454     }
455     rho1 <- adjphi
456     #     return(adjphi)
457
458     ybar <- mean(y)
459     yc <- y - ybar
460     d2fit <- durbin2fit(yc,xc,adjphi,method=method)
461
462     beta <- d2fit$beta
463     b0 <- ybar - t(xbar) %*% beta
464
465     allb <- c(b0,beta)
466
467
468     ####
469     cnt <- 0
470     ic <- 0
471     adjar <- adjphi
472     #   beginning of bs loop
473     while (ic == 0) {
474       holdr <- adjar
475
476       cnt <- cnt + 1
477       np <- p + icent
478       bsbias <- boot1(y,adjar,arp,nbs,xcopy,allb,method=method)
479
480       for (k in 1:arp) {
481         if (k == 1) {
482           hild <- 0
483         }
484         adjar[k] <- adjphi[k] + bsbias[k]
485         if (adjar[k] < lower) {
486           adjar[k] <- lower
487         }
488         if (adjar[k] > upper) {
489           adjar[k] <- upper
490         }
491       }
492       diff <- adjar - holdr
```

```
493
494       d2fit2 <- durbin2fit(yc,xc,adjar,method=method)
495       beta2 <- d2fit2$beta
496       b02 <- ybar - t(xbar) %*% beta2
497
498       allb <- c(b02,beta2)
499       check1 <- 0
500       metric <- 0
501
502       for (k in 1:arp) {
503         if (diff[k] < 0) {
504           check1 <- check1 + 1
505         }
506         metric <- metric + diff[k] ^ 2
507       }
508       metric <- sqrt(metric)
509       if (check1 == arp) {
510         adjar <- holdr
511       }
512
513       if (((metric <= .01) &&
514            (abs(diff[1]) <= .01)) | (cnt > 8)) {
515         ic <- 1
516       }
517     }
518
519     ############
520
521     d2fit <- durbin2fit(yc,xc,adjar,method=method)
522
523     beta <- d2fit$beta
524     b0 <- ybar - t(xbar) %*% beta
525     sigd2 <- d2fit$sigma
526     mse <- c(sigd2 ^ 2)
527
528     allb <- c(b0,beta)
529
530
531
532     ##  bs cov mat
533     bscov <- boot2(y,xcopy,adjar,allb,nbscov,method=method)
534     betacov <- bscov$betacov * mse
535     sesbeta <- diag(betacov) ^ (1 / 2)
536     tees <- allb / sesbeta
537     pvals <- 2 * (1 - pt(abs(tees),df))
538     tabbeta <- cbind(allb,sesbeta,tees,pvals)
539     colnames(tabbeta) <- c("beta","SE","t-ratio","p-value")
540     rname <- c()
541     for (j in 1:p) {
542       rname <- c(rname,paste("beta_",j))
543     }
544     rownames(tabbeta) <- c("Intercept",rname)
545
546     ### .99 flag ###
547     flag99 <- 0
548     if (adjar >= 0.99) {
549       flag99 <- 1
550     }
551
552     ### rho CI (for .99 cases, probably should not provide CI of rho) ###
553     #     if (CritVal == "z") {
554     #       cv <- abs(qnorm((1 - conf) / 2))
555     #     }else if (CritVal == "t") {
556     #       cv <- abs(qt((1 - conf) / 2,n - 13))
```

```
557     #       ### need to replace 13 later and this should not be an option so should be
            determined and removed from args
558     #     }
559
560     rhostar <- bscov$rhostar
561     rhobias <- adjar - rho1
562     k.multi <- (mean(rhostar) + rhobias) / mean(rhostar) * mod
563     # k.multi <- (rho1 + rhobias) / rho1
564     rhostar <- rhostar * k.multi
565     MSEstar <- bscov$MSEstar
566
567     ### (a) ###
568     rhocov1 <- matrix(rep(0,arp^2),ncol=arp)
569     for(nbk in 1:nbscov){
570       rhotmp <- rhostar[nbk,]
571       for(i in 1:arp){
572         for(j in 1:arp){
573           rhocov1[i,j] <- rhocov1[i,j]+(rhotmp[i]-adjar[i])*(rhotmp[j]-adjar[j])
574         }
575       }
576     }
577     rhocov1 <- rhocov1 / nbscov
578     serho1 <- diag(rhocov1) ^ (1 / 2)
579     rho_CI_1 <- c(adjar - qt(0.975, df) * serho1, adjar + qt(0.975, df) * serho1)
580
581     ### (b) ###
582     rhocov2 <- matrix(rep(0,arp^2),ncol=arp)
583     for(nbk in 1:nbscov){
584       rhotmp <- rhostar[nbk,]
585       MSEtmp <- MSEstar[nbk]
586       for(i in 1:arp){
587         for(j in 1:arp){
588           rhocov2[i,j] <- rhocov2[i,j]+(rhotmp[i]-adjar[i])*(rhotmp[j]-adjar[j]) /
                MSEtmp
589         }
590       }
591     }
592     rhocov2 <- rhocov2 * mse / nbscov
593     serho2 <- diag(rhocov2) ^ (1 / 2)
594     rho_CI_2 <- c(adjar - qt(0.975, df) * serho2, adjar + qt(0.975, df) * serho2)
595     ### (c) ###
596     rho_CI_3 <- quantile(rhostar, probs = c(0.025,0.975))
597     ### residuals and fitted.values
598     ypart <- nurho(y, adjar)
599     xpart <- wrho(xcopy, adjar)
600     ehat <- ypart - xpart %*% allb
601     fitted.values <- y[2:n] - ehat
602
603
604     list(
605       coefficients = allb, rho1 = rho1, adjar = adjar, mse = mse, rho_CI_1 = rho_CI_1,
            rho_CI_2 = rho_CI_2,
606       rho_CI_3 = rho_CI_3, betacov = betacov,
607       tabbeta = tabbeta, flag99 = flag99, residuals = ehat, fitted.values = fitted.
            values
608     )
609   }
610
611 simulacorrection <-
612   function(x,y,arp,nbs,nbscov,conf,CritVal,method) {
613     upper <- .99
614     lower <- -.99
615     n <- length(y)
616     p <- length(x[1,])
617     df <- n - p - arp
```

```
618     icent <- 0
619     if (p > 1) {
620       icent <- 1
621     }
622
623     xcopy <- x
624
625     ones <- rep(1,n)
626     proj1 <- ones %*% t(ones) / n
627     x2 <- x[,2:p]
628     xbar <- apply(x2,2,mean)
629     xc <- x[,2:p] - proj1 %*% x[,2:p]
630     x <- xc
631     p <- p - 1
632
633     ##   Durbin stage 1
634     pcent <- p + icent
635
636     dfit <- durbin1fit(y,xcopy,arp,method=method)
637     adjphi <- dfit$coef[2:(arp + 1)]
638     for (j in 1:arp) {
639       if (adjphi[j] < lower) {
640         adjphi[j] <- lower
641       }
642       if (adjphi[j] > upper) {
643         adjphi[j] <- upper
644       }
645     }
646
647     ## Durbin stage 2
648     ybar <- mean(y)
649     yc <- y - ybar
650     d2fit <- durbin2fit(yc,xc,adjphi,method=method)
651
652     beta <- d2fit$beta
653     b0 <- ybar - t(xbar) %*% beta
654
655     allb <- c(b0,beta)
656
657     adjar <- adjphi
658     rho1 <- adjphi
659
660
661     ### calc init sse ###
662     #      ypart <- nurho(y,adjar)
663     #      xpart <- wrho(xcopy,adjar)
664     #      ehat <- ypart - xpart%*%allb
665     #      init_sse<-sum(ehat^2)
666     #      ehat2<-y-xcopy%*%allb
667     #      init_sse2<-sum(ehat2^2)
668     #      ### collect ###
669     #      coll.bias<-c()
670     #      coll.sse<-init_sse
671     #      coll.sse_without_rho<-init_sse2
672     #      coll.CI_adj_rho<-rhoci2(n,init_rho)
673     #      coll.beta<-allb
674     #    no need of bs loop
675
676
677     holdr <- adjar
678
679     ## only perform bootstrap once
680     np <- p + icent
681     bsbias <- boot1(y,adjar,arp,nbs,xcopy,allb,method=method)
682
```

```
683     for (k in 1:arp) {
684       if (k == 1) {
685         hild <- 0
686       }
687       adjar[k] <- adjphi[k] + bsbias[k]
688       if (adjar[k] < lower) {
689         adjar[k] <- lower
690       }
691       if (adjar[k] > upper) {
692         adjar[k] <- upper
693       }
694     }
695     diff <- adjar - holdr
696
697     d2fit2 <- durbin2fit(yc,xc,adjar,method=method)
698     beta2 <- d2fit2$beta
699     b02 <- ybar - t(xbar) %*% beta2
700
701     allb <- c(b02,beta2)
702
703     ### sse ###
704     ypart <- nurho(y,adjar)
705     xpart <- wrho(xcopy,adjar)
706     ehat <- ypart - xpart %*% allb
707     sse <- sum(ehat ^ 2)
708     ehat2 <- y - xcopy %*% allb
709     sse2 <- sum(ehat2 ^ 2)
710     ### CI ###
711     if (CritVal == "z") {
712       cv <- abs(qnorm((1 - conf) / 2))
713     }else if (CritVal == "t") {
714       cv <- abs(qt((1 - conf) / 2,n - 13))### need to replace 13 later
715     }
716
717
718
719     CI_adj_rho <- rhoci2(n,adjar,cv)
720     adj_mid <- (CI_adj_rho[2] + CI_adj_rho[3]) / 2
721     CI_init_rho <- rhoci2(n,rho1,cv)
722     init_mid <- (CI_init_rho[2] + CI_init_rho[3]) / 2
723     if (adj_mid <= 0.95) {
724       adjar <- adj_mid
725     } else {
726       adjar <- init_mid
727     }
728
729     # SE_rho <- summary(dfit)$coeff[2:(arp + 1),2]
730     # CI_rho <- c(adjar - cv * SE_rho, adjar + cv * SE_rho)
731     #
732     # if (CI_rho[2] > 0.99) {CI_rho[2] <- 0.99}
733     # if (CI_rho[1] < -0.99) {CI_rho[1] <- -0.99}
734     ### collection  ####
735     #         coll.bias<-c(coll.bias,diff)
736     #         coll.sse<-c(coll.sse,sse)
737     #         coll.sse_without_rho<-c(coll.sse_without_rho,sse2)
738     #         coll.CI_adj_rho<-rbind(coll.CI_adj_rho,CI_adj_rho)
739     #         coll.beta<-rbind(coll.beta,allb)
740
741     d2fit <- durbin2fit(yc,xc,adjar,method=method)
742
743     beta <- d2fit$beta
744     b0 <- ybar - t(xbar) %*% beta
745     sigd2 <- d2fit$sigma
746     mse <- c(sigd2 ^ 2)
747
```

```
748     allb <- c(b0,beta)
749
750     ####   CHECK BEFORE PUTTING OTHER BS'S IN
751     ##     list(allb=allb,adjar=adjar)
752
753     ##  bs cov mat
754     bscov <- boot2(y,xcopy,adjar,allb,nbscov,method=method)
755     betacov <- bscov$betacov * mse
756     sesbeta <- diag(betacov) ^ (1 / 2)
757     tees <- allb / sesbeta
758     pvals <- 2 * (1 - pt(abs(tees), df))
759     tabbeta <- cbind(allb, sesbeta, tees, pvals)
760     colnames(tabbeta) <- c("beta","SE","t-ratio","p-value")
761     rname <- c()
762     for (j in 1:p) {
763       rname <- c(rname, paste("beta_", j))
764     }
765     rownames(tabbeta) <- c("Intercept", rname)
766     flag99 <- 1
767
768     ypart <- nurho(y, adjar)
769     xpart <- wrho(xcopy, adjar)
770     ehat <- ypart - xpart %*% allb
771     fitted.values <- y[2:n] - ehat
772     rho_CI_1 <- c(NA,NA)
773     rho_CI_2 <- c(NA,NA)
774     rho_CI_3 <- c(NA,NA)
775     list(
776       coefficients = allb, rho1 = rho1, adjar = adjar, mse = mse, rho_CI_1 = rho_CI_1,
              rho_CI_2 = rho_CI_2,
777       rho_CI_3 = rho_CI_3, betacov = betacov, tabbeta = tabbeta,
778       flag99 = flag99, residuals = ehat, fitted.values = fitted.values
779     )
780     #      list(coefficients=allb,adjar=adjar,init_rho=init_rho,
781     #           bias=coll.bias,SSE=coll.sse,SSE_WO_rho=coll.sse_without_rho,
782     #           coll.CI_adj_rho=coll.CI_adj_rho,coll.beta=coll.beta)
783     #     result<-cbind(coll.CI_adj_rho,c(0,coll.bias),coll.sse,coll.sse_without_rho)
784     #     rownames(result)<-NULL
785     #     colnames(result)<-c("rho","lb","ub","bias","SSE","SSE W/O rho")
786     #       list(coll=result,rho_F=adjar)
787   }
788
789 summary.dbfit <-
790   function(object, ...)
791   {
792     res <- list(call = object$call, tab = object$tabbeta, rho1 = unname(object$rho1),
              adjar = unname(object$adjar),
793               flag99 = object$flag99)
794     class(res) <- "summary.dbfit"
795     res
796   }
797
798 wrho <-
799   function(xc,adjphi){
800
801     arp <- length(adjphi)
802     n <- length(xc[,1])
803
804     s1 <- arp+1
805     s2 <- n
806
807     x1 <- lagx(xc,s1,s2)
808
809     for(j in 1:arp){
810       s1 <- s1 - 1
```

```
811       s2 <- s2 - 1
812       xt <- lagx(xc,s1,s2)*adjphi[j]
813       x1 <- x1 - xt
814     }
815     wrho <- x1
816     return(wrho)
817   }
```

./code.R

Appendix B

Reference Manual of R Package

`DBfit`

`DBfit-package` *A Double Bootstrap Method for Analyzing Linear Models With Autoregressive Errors*

Description

Computes the double bootstap as discussed in McKnight, McKean, and Huitema (2000). The double bootstrap method provides a better fit for a linear model with autoregressive errors than ARIMA when the sample size is small.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Joseph W. McKean and Shaofeng Zhang

Maintainer: Joseph W. McKean <joseph.mckean@wmich.edu> and Shaofeng Zhang <shaofeng.zhang@wmich.edu>

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

`boot1` *First Boostrap Procedure For parameter estimations*

Description

Function performing the first bootstrap procedure to yield the parameter estimates

Usage

```
boot1(y, phi1, arp, nbs, x, allb, method)
```

Arguments

`y`	the response variable
`phi1`	the Durbin two-stage estimate of the autoregressive parameter rho
`arp`	the order of autoregressive errors
`nbs`	the bootstrap size
`x`	the original design matrix (including intercept), without centering
`allb`	all the Durbin two-stage estimates of the regression coefficients
`method`	If "OLS", uses the ordinary least square; If "RANK", uses the rank-based fit

Value

An estimate of the bias is returned

Note

This function is for internal use. The main function for users is `dbfit`.

`boot2` *First Boostrap Procedure For parameter estimations*

Description

Function performing the second bootstrap procedure to yield the inference of the regression coefficients

Usage

```
boot2(y, xcopy, phi1, beta, nbs, method)
```

Arguments

`y`	the response variable
`xcopy`	the original design matrix (including intercept), without centering
`phi1`	the estimate of the autoregressive parameter rho from the first bootstrap procedure
`beta`	the estimates of the regression coefficients from the first bootstrap procedure
`nbs`	the bootstrap size
`method`	If "OLS", uses the ordinary least square; If "RANK", uses rank-based fit

Value

`betacov`	the estimate of var-cov matrix of betas
`allbeta`	the estimates of betas inside of the second bootstrap, not the final estimates of betas. The final estimates of betas are still from `boot1`.
`rhostar`	the estimates of rho inside of the second bootstrap, not the final estimates of rho. The final estimate(s) of rho are still from `boot1`.
`MSEstar`	MSE used inside of the second bootstrap.

Note

This function is for internal use. The main function for users is `dbfit`

`dbfit` *The main function for the double bootstrap method*

Description

This function is used to implement the double bootstrap method. It is used to yield estimates of both regression coefficients and autoregressive parameters(rho), and also the inference of them. However, the inference of rho is still under development.

Usage

```
dbfit.default(x, y, arp, nbs = 500, nbscov = 500, conf = 0.95, CritVal = "z", correction = TRUE, me
```

Arguments

`x`	the design matrix, including intercept, i.e. the first column being ones.
`y`	the response variable.
`arp`	the order of autoregressive errors.
`nbs`	the bootstrap size for the first bootstrap procedure. Default is 500.
`nbscov`	the bootstrap size for the second bootstrap procedure. Default is 500.
`conf`	the confidence level of CI for the 0.99 correction, default is 0.95. For internal use of testing. To be deleted when the pkg is done.
`CritVal`	the critical value of CI for the 0.99 correction, default is "z" (z-score). For internal use of testing. To be deleted when the pkg is done.
`correction`	logical. If TRUE, uses the correction for cases that the estimate of rho is 0.99. Default is TRUE.
`method`	the method to be used for fitting. If "OLS", uses the ordinary least square `lm`; If "RANK", uses the rank-based fit `rfit`.
`mod`	a number to modify the multiplier in rho inference. To be deleted when the development of rho inference is done.
`...`	additional arguments to be passed to fitting routines

Details

Computes the double bootstrap as discussed in McKnight, McKean, and Huitema (2000). For details, see the references.

Value

`coefficients`	the estimates of regression coefficients based on the first bootstrap procedure
`rho1`	the Durbin two-stage estimate of the autoregressive parameter rho
`adjar`	the estimates of regression coefficients based on the first bootstrap procedure
`mse`	the mean square error
`rho_CI_1`	for the development of the inference of rho; the first type of CI for rho, only one of the three types should be kept when the development is done. For .99 cases, the CI of rho is not calculated.

`rho_CI_2`	for the development of the inference of rho; the second type of CI for rho, only one of the three types should be kept when the development is done. For .99 cases, the CI of rho is not calculated.
`rho_CI_3`	for the development of the inference of rho; the third type of CI for rho, only one of the three types should be kept when the development is done. For .99 cases, the CI of rho is not calculated.
`betacov`	the estimate of the variance-covariance matrix of betas
`tabbeta`	a table of point estimates, SE's, test statistics and p-values.
`flag99`	an indicator; if 1, it indicates the original fit yields an estimate of rho to be 0.99. When the correction is requested (default), the correction procedure kicks in, and the final estimates of rho is corrected.
`residuals`	the residuals, that is response minus fitted values.
`fitted.values`	the fitted mean values.

Author(s)

Joseph W. McKean and Shaofeng Zhang

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

See Also

dbfit.formula

Examples

```
#need to make sure that the dependent package \pkg{Rfit} is installed
data(testdata)
y<-testdata[,5]
x<-testdata[,1:4]
fit1<-dbfit(x,y,1) # OLS fit, default
fit2<-dbfit(x,y,1,method="RANK") # rank-based fit
```

`durbin1fit` *Durbin stage 1 fit*

Description

Function implements the Durbin stage 1 fit

Usage

```
durbin1fit(y, x, arp, method)
```

Arguments

y	the response variable in stage 1, not the original response variable
x	the model matrix in stage 1, not the original design matrix
arp	the order of autoregressive errors.
method	the method to be used for fitting. If "OLS", uses the ordinary least square; If "RANK", uses the rank-based fit.

Note

This function is for internal use. The main function for users is `dbfit`.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

---

durbin1xy *Creating New X and Y for Durbin Stage 1*

Description

Functions provides the tranformed reponse variable and model matrix for Durbin stage 1 fit. Note that they are different from the orginal ones. For details, see the reference.

Usage

```
durbin1xy(y, x, arp)
```

Arguments

y	the orginal response variable
x	the orginal design matrix with first column of all one's (corresponding to the intercept)
arp	the order of autoregressive errors.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

`durbin2fit` *Durbin stage 2 fit*

Description

Function implements the Durbin stage 1 fit

Usage

```
durbin2fit(yc, xc, adjphi, method)
```

Arguments

`yc`	a transformed reponse variable
`xc`	a transformed design matrix
`adjphi`	the Durbin stage 1 estimate(s) of the autoregressive parameters rho
`method`	the method to be used for fitting. If "OLS", uses the ordinary least square; If "RANK", uses the rank-based fit.

Value

`beta`	the estimates of regression coefficients
`sigma`	the estimate of standard deviation of the white noise

Note

This function is for internal use. The main function for users is `dbfit`.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

`hmdesign2` *the Two-Phase Design Matrix*

Description

Returns the design matrix for a two-phase intervention model.

Usage

```
hmdesign2(n1, n2)
```

Arguments

`n1`	number of obs in phase 1
`n2`	number of obs in phase 2

Details

It returns a matrix of 4 columns. As discussed in Huitema, Mckean, & Mcknight (1999), in two-phase design: beta0 = intercept, beta1 = slope for Phase 1, beta2 = level change from Phase 1 to Phase 2, and beta3 slope change from Phase 1 to Phase 2.

References

Huitema, B. E., Mckean, J. W., & Mcknight, S. (1999). Autocorrelation effects on least- squares intervention analysis of short time series. Educational and Psychological Measurement, 59 (5), 767-786.

Examples

```
n1 <- 15
n2 <- 15
hmdesign2(n1, n2)
```

---

hmmat *K-Phase Design Matrix*

Description

Returns the design matrix for a general k-phase intervention model

Usage

```
hmmat(vecss, k)
```

Arguments

vecss	a vector of length k with each element being the number of observations in each phase
k	number of phases

Details

It returns a matrix of 2*k columns. The design can be unbalanced, i.e. each phase can have different observations.

References

Huitema, B. E., Mckean, J. W., & Mcknight, S. (1999). Autocorrelation effects on least- squares intervention analysis of short time series. Educational and Psychological Measurement, 59 (5), 767-786.

See Also

hmdesign2

Examples

```
# a three-phase design matrix
hmmat(c(10,10,10),3)
```

`hypothmat` *General Linear Tests of the regression coefficients*

Description

Performs general linear tests of the regressio coefficients.

Usage

```
hypothmat(sfit, mmat, n, p)
```

Arguments

`sfit`	the result of a call to dbfit.
`mmat`	a full row rank q*(p+1) matrix, where q is the row number of the matrix and p is number of independent variables.
`n`	total number of observations.
`p`	number of independent variables.

Details

This functions performs the general linear F-test of the form H0: Mb = 0 vs HA: Mb != 0.

Value

`tst`	the test statistic
`pvf`	the p-value of the F-test

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

Examples

```
data(testdata)
y<-testdata[,5]
x<-testdata[,1:4]
fit1<-dbfit(x,y,1) # OLS fit, default
# a test that H0: b1 = b3 vs HA: b1 != b3
mat<-matrix(c(1,0,0,-1),nrow=1)
hypothmat(sfit=fit1,mmat=mat,n=40,p=4)
```

lagx *Lag Functions*

Description

For preparing the transformed x and y in the Durbin stage 1 fit

Usage

```
lagx(x, s1, s2)
lagmat(x, p)
```

Note

These function are for internal use.

nurho *Creating a new response variable for Durbin stage 2*

Description

It returns a new response variable (vector) for Durbin stage 2.

Usage

```
nurho(yc, adjphi)
```

Arguments

yc	the centered response variable y
adjphi	(initial) estimate of rho in Durbin stage 1

Details

see reference.

Note

This function is for internal use. The main function for users is dbfit.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

print.dbfit *DBfit Internal Print Functions*

Description

These functions print the output in a user-friendly manner using the internal R function `print`.

Usage

```
## S3 method for class 'dbfit'
print(x, ...)
## S3 method for class 'summary.dbfit'
print(x, ...)
```

Arguments

`x`	An object to be printed
`...`	additional arguments to be passed to `print`

See Also

`dbfit`, `summary.dbfit`

rhoci2 *A fisher type CI of the autoregressive parameter rho*

Description

This function returns a Fisher type CI for rho, which is later used to correct the .99 cases.

Usage

```
rhoci2(n, rho, cv)
```

Arguments

`n`	total number of observations
`rho`	final estimate of rho, usually .99.
`cv`	critical value for CI

Details

see reference.

Note

This function is for internal use.

References

Shaofeng Zhang (2017). Ph.D. Dissertation. Rao, C. R. (1952). Advanced statistical methods in biometric research. p. 231

simpgen1hm2 *Simulation Data Generating Function*

Description

Generates the simulation data for a two-phase intervention model.

Usage

```
simpgen1hm2(n1, n2, rho, beta = c(0, 0, 0, 0))
```

Arguments

n1	number of obs in phase 1
n2	number of obs in phase 2
rho	pre-defined autoregressive parameter(s)
beta	pre-defined regression coefficients

Details

This function is used for simulations when developing the package. With pre-defined sample sizes in both phases and parameters, it returns a simulation data. Then use the double bootstrap method `dbfit` to fit the data and collect results. This is usually repeated many times (5000) to check the performance of the method.

Value

mat	a matrix containing the simulation data. The last column is the response variable. All other columns make up the design matrix.

See Also

hmdesign2

Examples

```
n1 <- 15
n2 <- 15
rho <- 0.6
beta <- c(0,0,0,0)
dat <- simpgen1hm2(n1, n2, rho, beta)
dat
```

simula *Work Horse Function to implement the Double Bootstrap method*

Description

simula is the original work horse function to implement the DB method. However, when this function returns an estimate of rho to be .99, another work horse function simulacorrection kicks in.

Usage

```
simula(x, y, arp, nbs, nbscov, conf, CritVal, method, mod)
```

Arguments

x	the design matrix, including intercept, i.e. the first column being ones.
y	the response variable.
arp	the order of autoregressive errors.
nbs	the bootstrap size for the first bootstrap procedure. Default is 500.
nbscov	the bootstrap size for the second bootstrap procedure. Default is 500.
conf	the confidence level of CI for the 0.99 correction, default is 0.95. For internal use of testing. To be deleted when the pkg is done.
CritVal	the critical value of CI for the 0.99 correction, default is "z" (z-score). For internal use of testing. To be deleted when the pkg is done.
method	the method to be used for fitting. If "OLS", uses the ordinary least square lm; If "RANK", uses the rank-based fit rfit.
mod	a number to modify the multiplier in rho inference. To be deleted when the development of rho inference is done.

Details

see dbfit.

Note

Users should use dbfit to perform the analysis.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

See Also

dbfit.

`simulacorrection` *Work Horse Function to Implement the Double Bootstrap Method For .99 Cases*

Description

When function `simula` returns an estimate of rho to be .99, this function kicks in and ouputs a corrected estimate of rho.

Usage

```
simulacorrection(x, y, arp, nbs, nbscov, conf, CritVal, method)
```

Arguments

`x`	the design matrix, including intercept, i.e. the first column being ones.
`y`	the response variable.
`arp`	the order of autoregressive errors.
`nbs`	the bootstrap size for the first bootstrap procedure. Default is 500.
`nbscov`	the bootstrap size for the second bootstrap procedure. Default is 500.
`conf`	the confidence level of CI for the 0.99 correction, default is 0.95. For internal use of testing. To be deleted when the pkg is done.
`CritVal`	the critical value of CI for the 0.99 correction, default is "z" (z-score). For internal use of testing. To be deleted when the pkg is done.
`method`	the method to be used for fitting. If "OLS", uses the ordinary least square `lm`; If "RANK", uses the rank-based fit `rfit`.

Details

If 0.99 problem is detected, then construct Fisher CI for both initial estimate (in Durbin stage 1) and first bias-corrected estimate (perform only one bootstrap, instead of a loop); if the midpoint of latter is smaller than 0.95, then this midpoint is the final estimate for rho; otherwise the midpoint of the former CI is the final estimate.

By default, when function `simula` returns an estimate of rho to be .99, this function kicks in and ouputs a corrected estimate of rho. However, users can turn the auto correction off by setting correction="FALSE" in `dbfit`. Users are encouraged to investigate why the stationarity assumption is violated based on their experience of time series analysis and knowledge of the data.

Note

Users should use `dbfit` to perform the analysis.

References

Shaofeng Zhang (2017). Ph.D. Dissertation.

See Also

`dbfit`.

`summary.dbfit` *Summarize the double bootstrap (DB) fit*

Description

It summarizes the DB fit in a way that is similar to OLS `lm`.

Usage

```
summary.dbfit(object, ...)
```

Arguments

`object`	a result of the call to `rfit`
`...`	additional arguments to be passed

Value

`call`	the call to `rfit`
`tab`	a table of point estimates, standard errors, t-ratios and p-values
`rho1`	the Durbin two-stage estimate of rho
`adjar`	the DB (final) estimate of rho
`flag99`	an indicator; if 1, it indicates the original fit yields an estimate of rho to be 0.99.

Examples

```
data(testdata)
y<-testdata[,5]
x<-testdata[,1:4]
fit1<-dbfit(x,y,1) # OLS fit, default
summary(fit1)
```

`testdata` *testdata*

Description

This data serves as a test data.

Usage

```
data("testdata")
```

Format

A data frame with 40 observations. First 4 columns make up the design matrix. Last column is the response variable.

Examples

```
data(testdata)
## maybe str(testdata) ; plot(testdata) ...
```

wrho *Creating a new design matrix for Durbin stage 2*

Description

It returns a new design matrix for Durbin stage 2.

Usage

```
wrho(xc, adjphi)
```

Arguments

xc	centered design matrix, no column of ones
adjphi	(initial) estimate of rho in Durbin stage 1

Details

see reference.

Note

This function is for internal use. The main function for users is `dbfit`.

References

McKnight, S. D., McKean, J. W., and Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. Psychological methods, 5 (1), 87. Shaofeng Zhang (2017). Ph.D. Dissertation.

References

Banerjee, S., Carlin, B. P., & Gelfand, A. E. (2014). *Hierarchical modeling and analysis for spatial data*. Crc Press.

Bernal, J. L., Cummins, S., & Gasparrini, A. (2016). Interrupted time series regression for the evaluation of public health interventions: a tutorial. *International journal of epidemiology*, (p. dyw098).

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, (pp. 192–236).

Box, G. E., & Tiao, G. C. (1975). Intervention analysis with applications to economic and environmental problems. *Journal of the American Statistical association*, *70*(349), 70–79.

Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). *Time series analysis: forecasting and control*. San Francisco:Holden-Day.

Cochrane, D., & Orcutt, G. H. (1949). Application of least squares regression to relationships containing auto-correlated error terms. *Journal of the American statistical association*, *44*(245), 32–61.

Durbin, J. (1960). Estimation of parameters in time-series regression models. *Journal of the Royal Statistical Society. Series B (Methodological)*, (pp. 139–153).

Hall, P. (1988). Theoretical comparison of bootstrap confidence intervals. *The Annals of Statistics*, (pp. 927–953).

Hettmansperger, T. P., & McKean, J. W. (2011). *Robust nonparametric statistical methods*. CRC Press.

Hodges Jr, J. L., & Lehmann, E. L. (1956). The efficiency of some nonparametric competitors of the t-test. *The Annals of Mathematical Statistics*, (pp. 324–335).

Huang, A., Wand, M. P., et al. (2013). Simple marginally noninformative prior distributions for covariance matrices. *Bayesian Analysis*, *8*(2), 439–452.

Huitema, B. E., McKean, J. W., & McKnight, S. (1999). Autocorrelation effects on least-squares intervention analysis of short time series. *Educational and Psychological Measurement*, *59*(5), 767–786.

Jaeckel, L. A. (1972). Estimating regression coefficients by minimizing the dispersion of the residuals. *The Annals of Mathematical Statistics*, (pp. 1449–1458).

Johnston, D. W., & Johnston, M. (2013). Useful theories should apply to individuals. *British journal of health psychology*, *18*(3), 469–473.

Johnston, J., & Dinardo, J. (1984). *Econometric methods*, vol. 972. Wiley Online Library.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, *37*(2), 183–233.

Kloke, J., & McKean, J. W. (2014). *Nonparametric statistical methods using R*. CRC Press.

Kloke, J. D., & McKean, J. W. (2012). Rfit: Rank-based estimation for linear models. *The R Journal*, *4*(2), 57–64.

Lunn, T. A. B. N., D.J., & Spiegelhalter, D. (2000). Winbugs — a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing*, *10*, 325–337.

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, (pp.

50–60).

McGrory, C. A., & Titterington, D. (2007). Variational approximations in bayesian model selection for finite mixture distributions. *Computational Statistics & Data Analysis*, *51*(11), 5352–5367.

McKean, J., J.W. (1975). Tests of hypotheses based on ranks in the general linear model. ph.d.dissertation.

McKean, J. W., & Hettmansperger, T. P. (2016). Rank-based analysis of linear models and beyond: A review. In *Robust Rank-Based and Nonparametric Methods*, (pp. 1–24). Springer.

McKnight, S. D., McKean, J. W., & Huitema, B. E. (2000). A double bootstrap method to analyze linear models with autoregressive error terms. *Psychological methods*, *5*(1), 87.

Prais, S. J., & Winsten, C. B. (1954). Trend estimators and serial correlation. Tech. rep., Cowles Commission discussion paper Chicago.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
URL https://www.R-project.org/

Rao, C. R. (1952). Advanced statistical methods in biometric research.

Stine, R. A. (1987). Estimating properties of autoregressive forecasts. *Journal of the American statistical association*, *82*(400), 1072–1078.

Titterington, D. (2004). Bayesian methods for neural networks and related models. *Statistical Science*, (pp. 128–139).

Tukey, J. W. (1960). A survey of sampling from contaminated distributions. *Contributions to probability and statistics*, *2*, 448–485.

Wand, M. P., Ormerod, J. T., Padoan, S. A., Fuhrwirth, R., et al. (2011). Mean field variational bayes for elaborate distributions. *Bayesian Analysis*, *6*(4), 847–900.

Wang, B., Titterington, D., et al. (2006). Convergence properties of a general algorithm for calculating variational bayesian estimates for a normal mixture model. *Bayesian Analysis*, *1*(3), 625–650.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, *1*(6), 80–83.

Winn, J., & Bishop, C. M. (2005). Variational message passing. *Journal of Machine Learning Research*, *6*(Apr), 661–694.