SAMPLE
Small Area Methods for Poverty and Living Condition Estimates

SAMPLE DELIVERABLE 22

SOFTWARE ON SMALL AREA ESTIMATION

Grant agreement No:	SSH - CT - 2007 – 217565
Project Acronym:	SAMPLE
Project Full title:	Small Area Methods for Poverty and Living Conditions Estimates
Funding Scheme:	Collaborative Project - Small or medium scale focused research project
Deliverable n.	22
Deliverable name:	Software on Small Area Estimation
WP no.:	2
Lead beneficiary:	Number
Nature:	Software
Dissemination level:	PU
Due delivery date from Annex I:	28 February 2011
Actual delivery date:	28 February 2011
Project co-ordinator name:	Mrs. Monica Pratesi
Title:	Associate Professor of Statistics - University of Pisa
Organization:	Department of Statistics and Mathematics Applied to Economics of the University of Pisa (UNIPI-DSMAE)
Tel:	+39-050-2216252, +39-050-2216492
Fax:	+39-050-2216375
E-mail:	coordinator@sample-project.eu
Project website address:	www.sample-project.eu

SOFTWARE ON SMALL AREA ESTIMATION

Project Acronym:	SAMPLE
Project Full title:	Small Area Methods for Poverty and Living Conditions Estimates
Project/Contract No:	EU-FP7-SSH-2007-1
Grant agreement No:	217565
Work Package 2:	Small area estimation of poverty and inequality indicators
Document title:	Software on small area estimation
Date:	February 28th, 2011
Type of document:	Deliverable D22
Status:	final
Editors:	I. Molina, D. Morales, M. Pratesi and N. Tzavidis
Authors:	Nicola Salvati (salvati@ec.unipi.it), DSMAE
	Caterina Giusti (caterina.giusti@ec.unipi.it), DSMAE
	Stefano Marchetti (s.marchetti@ds.unifi.it), DSMAE
	Monica Pratesi (m.pratesi@ec.unipi.it), DSMAE
	Nikos Tzavidis (n.tzavidis@soton.ac.uk), Southampton
	Isabel Molina Peralta (isabel.molina@uc3m.es), UC3M
	Domingo Morales (d.morales@umh.es), UMH
	María Dolores Esteban (md.esteban@umh.es), UMH
	Laureano Santamaría (l.santamaria@umh.es), UMH
	Yolanda Marhuenda (y.marhuenda@umh.es), UMH
	Agustín Pérez (agustin.perez@umh.es), UMH
	Maria Chiara Pagliarella (mc.pagliarella@unicas.it), UMH
	Ray Chambers (ray@uow.edu.au), CSSM
	J.N.K. Rao (jrao@math.carleton.ca), Carleton University
	Caterina Ferretti (ferretti@ds.unifi.it), University of Florence

Contents

Prologue

This report contains a description of the software developments on small area estimation carried out by the partners of WP2 in the SAMPLE project. The target of the report is to give a guide for users of the developed software. For the sake of completeness, we have also paid special attention to introduce the programmed statistical methodology and to illustrate how to use the software in applications to data.

The manuscript is organized in 28 chapters. Chapters 1-14 contains the statistical methodology, the the software description of the R function and the examples of usage. Chapters 15-28 contains the codes of the R functions

Chapter 1 introduces the basic Fay-Herriot (FH) model and describes the corresponding software. As some of the models proposed by the SAMPLE project are in fact generalization of the FH model, this chapter will be of great interest for users.

Chapter 2 introduces an area-level spatial model and describe two fitting methods and two bootstrap procedures to estimate the mean squared error (MSE) of the empirical best linear unbiased estimator. An example data set is given to illustrate how to use the programmed R functions. The software in chapters 15-16 has been programmed by the UC3M team.

Chapter 3 deals with area-level time models. Two models are presented. The second one contains time random effects following an auto-regressive process AR(1) and the first one is a simplification where these effects are independent. Theoretical descriptions are presented as well as an example of use. The software in chapter 17 has been programmed by the UMH team.

Chapter 4 presents some area-level partitioned time models. The considered models can ne applied to populations that can be divided in two parts with different statistical behaviors. The models in this chapter generalize those one appearing in Chapter 3. The software in chapter 18 has been programmed by the UMH team.

Chapter 5 contains some area-level spatio-temporal models with random effects that take into account for time and space variability between areas. The software in chapter 19 has been programmed by the UMH-UC3M teams.

Chapter 6 gives the software for fitting two unit-level time models. The software to calculate the Empirical Best Linear Unbiased Predictors (EBLUP) and to estimate their mean squared errors is also given. The software in chapters 20 has been programmed by the UMH team.

Chapter 7 introduces a methodology for obtaining small area estimation of a domain mean by using the M-quantile regression approach. Chapter 8 deals with the nonparametric M-quantile small area estimation of a domain mean. Chapter 9 do the same but using the M-quantile geographically weighted regression approach where the regression parameters may vary between specified locations. Chapters 7-9 also discuss the estimation of the domain means and the estimation of the corresponding MSEs. These chapters present software descriptions and examples of usage.

Chapter 10 introduces an estimators of the small area cumulative distribution function (cdf) by using the Chambers-Dunstan estimator of the cdf and the linear M-quantile small area model. Chapter 11 deals with the nonparametric M-quantile estimation of a cdf. Chapter 12 is devoted to the M-quantile estimation of poverty indicators and the bootstrap-based estimation of the mean square error.

The software in chapters 21-26 has been programmed by the UNIPI-DSMAE and the Southampton teams.

Chapters 13 and 14 deals with the Empirical Best (EB) prediction of poverty measures with unit level models and with the Fast EB method for estimation of fuzzy poverty measures.

The software in chapters 27 and 28 has been programmed by UC3M and by University Florence.

Chapter 1

Fay-Herriot model

1.1 Methodology

1.1.1 Model and small area EB estimator

Fay-Herriot (FH) models were introduced by Fay & Herriot (1979) to obtain small area estimators of median income in small places in the U.S. These models are well known in the literature of small area estimation (SAE) and are the basic tool when auxiliary data at the unit level are not available or there are confidentiality reasons preventing their use, and therefore only aggregated data at the small area level are available.

The basic Fay-Herriot model, assuming normality, is defined as

$$\begin{array}{ll} (i) & Y_i|\theta_i \overset{ind}{\sim} N(\theta_i, D_i),\ i = 1, \ldots, m; \\ (ii) & \theta_i \overset{ind}{\sim} N(\mathbf{x}_i'\beta, A),\ i = 1, \ldots, m. \end{array} \tag{1.1}$$

Component (ii) of (1.1) is the linking model and component (i) is the sampling model. Marginally,

$$Y_i \overset{ind}{\sim} N(\mathbf{x}_i'\beta, D_i + A),\ i = 1, \ldots, m. \tag{1.2}$$

In matrix notation, (1.2) may be written as $\mathbf{Y} \sim N\{\mathbf{X}\beta, \Sigma(A)\}$, where $\mathbf{Y} = (Y_1, \ldots, Y_m)'$, $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)'$ and $\Sigma(A) = diag(A + D_1, \ldots, A + D_m)$.

The best (or Bayes) estimator of θ_i under squared error loss is given by

$$\begin{aligned} \hat{\theta}_i^B = E(\theta_i|Y_i) &= Y_i - \frac{D_i}{A + D_i}(Y_i - \mathbf{x}_i'\beta) \\ &= \{1 - B_i(A)\}Y_i + B_i(A)\mathbf{x}_i'\beta, \end{aligned} \tag{1.3}$$

noting that $\theta_i|Y_i \overset{ind}{\sim} N\{\hat{\theta}_i^B, g_{1i}(A)\}$, where

$$g_{1i}(A) = D_i\{1 - B_i(A)\} \tag{1.4}$$

and $B_i(A) = D_i/(A + D_i)$. Expression (1.3) shows that $\hat{\theta}_i^B$ is a convex combination of the direct estimator Y_i and the regression synthetic estimator $\mathbf{x}_i'\beta$.

In practice, the model parameters β and A are unknown. For a given A, the maximum likelihood (ML) estimator of β is obtained from (1.2) as

$$\begin{aligned}\tilde{\beta}(A) &= \{\mathbf{X}'\Sigma^{-1}(A)\mathbf{X}\}^{-1}\mathbf{X}'\Sigma^{-1}(A)\mathbf{Y} \\ &= \left\{\sum_{j=1}^{m}(A+D_j)^{-1}\mathbf{x}_j\mathbf{x}_j'\right\}^{-1}\sum_{j=1}^{m}(A+D_j)^{-1}\mathbf{x}_jY_j. \qquad (1.5)\end{aligned}$$

Note that $\tilde{\beta}(A)$ is also the weighted least squares (WLS) estimator of β without normality assumption. Substituting $\tilde{\beta}(A)$ for β in (1.3), we get the first-step empirical best (or empirical Bayes) estimator, $\tilde{\theta}_i^{EB}(A)$, of θ_i which is also equal to the best linear unbiased prediction (BLUP) estimator of θ_i without normality assumption.

Several estimators, $\hat{A}$, of A have been proposed in the literature including moment estimators without normality assumption, ML and restricted (or residual) ML estimator (REML) estimator; see Section 1.1.2. Substituting $\hat{A}$ for A in the first-step empirical best (EB) estimator, $\tilde{\theta}_i^{EB}(A)$, we get the final EB estimator $\hat{\theta}_i^{EB}$:

$$\begin{aligned}\hat{\theta}_i^{EB} = \tilde{\theta}_i^{EB}(\hat{A}) &= \{1-B_i(\hat{A})\}Y_i + B_i(\hat{A})\mathbf{x}_i'\hat{\beta} \\ &= Y_i - B_i(\hat{A})(Y_i - \mathbf{x}_i'\hat{\beta}), \qquad (1.6)\end{aligned}$$

where $\hat{\beta} = \tilde{\beta}(\hat{A})$. Without the normality assumption, the estimator (1.6) is also the empirical BLUP (EBLUP) estimator.

1.1.2 Fitting methods

In this section, we give the formulae for an estimator of A based on moment method, ML and REML methods. A moment estimator, due to Fay and Herriot (1979), is given by $\hat{A}_{FH} = \max(0, A_{FH}^*)$ with A_{FH}^* obtained iteratively as the solution of the following non-linear equation in A:

$$\sum_{j=1}^{m}(A+D_j)^{-1}\{Y_j - \mathbf{x}_j'\tilde{\beta}(A)\}^2 = m-p. \qquad (1.7)$$

This equation can be solved using an iterative method such as the Fisher-scoring algorithm. For this, let us define

$$s(A) = \sum_{j=1}^{m}(A+D_j)^{-1}\{Y_j - \mathbf{x}_j'\tilde{\beta}(A)\}^2 - m - p.$$

By a first order Taylor expansion of $s(\hat{A}_{FH})$ around the true A, we get

$$0 = s(\hat{A}_{FH}) \approx s(A) + s'(A)(\hat{A}_{FH} - A). \qquad (1.8)$$

The Fisher-scoring algorithm replaces in this equation, the derivative $s'(A)$ by its expectation $E[-s'(A)]$, which in this case is equal to

$$E[-s'(A)] = \sum_{j=1}^{m}(A+D_j)^{-1}.$$

Then, solving for $\hat{A}_{FH}$ in (1.8), we get

$$\hat{A}_{FH} = A + \left\{E[-s'(A)]\right\}^{-1} s(A).$$

This algorithm starts with an initial value $\hat{A}_{FH}^{(0)}$, and then in each iteration it updates the estimate using the updating equation

$$\hat{A}_{FH}^{(k+1)} = \hat{A}_{FH}^{(k)} + \left\{ E[-s'(A)]\big|_{A=\hat{A}_{FH}^{(k)}} \right\}^{-1} s(\hat{A}_{FH}^{(k)}),$$

In the function `fitFH`, the starting value is set to $\hat{A}_{FH}^{(0)} = \text{median}(D_i)$. It stops either when the number of iterations $k >$ MAXITER where MAXITER can be chosen by the user (default is 500), or when

$$\left| \frac{\hat{A}_{FH}^{(k+1)} - \hat{A}_{FH}^{(k)}}{\hat{A}_{FH}^{(k)}} \right| < 0.0001.$$

Convergence of the iteration is generally rapid.

Assuming normality, A and β can be estimated by ML or REML procedures. In fact, under regularity conditions, the estimators derived from these two methods (and using the Normal likelihood) remain consistent at order $O_p(m^{-1/2})$ even without the Normality assumption, for details see Jiang (1996). ML estimators of A and β are obtained by maximizing the log-likelihood, given by

$$\ell(A, \beta; \mathbf{Y}) = c - \frac{1}{2}\log|\Sigma(A)| - \frac{1}{2}(\mathbf{Y} - \mathbf{X}\beta)'\Sigma^{-1}(A)(\mathbf{Y} - \mathbf{X}\beta), \tag{1.9}$$

where c denotes a constant. Taking derivative of ℓ with respect to β and equating to zero, we obtain the maximum likelihood equation for β, which gives the WLS estimator (1.5). The maximum likelihood equation for A is obtained taking derivative of ℓ with respect to A and equating to zero, and is given by

$$\sum_{j=1}^{m} (A + D_j)^{-2}\{Y_j - \mathbf{x}_j'\tilde{\beta}(A)\}^2 = \sum_{j=1}^{m} (A + D_j)^{-1}. \tag{1.10}$$

Again, Fisher-scoring algorithm may be used to solve this equation. Let us denote

$$s_{ML}(A) = \sum_{j=1}^{m} (A + D_j)^{-2}\{Y_j - \mathbf{x}_j'\tilde{\beta}(A)\}^2 - \sum_{j=1}^{m} (A + D_j)^{-1}.$$

The Fisher information for A is obtained by taking expectation of the negative derivative of $s_{ML}(A)$, and is given by

$$I_{ML}(A) = E\left\{-s_{ML}'(A)\right\} = \frac{1}{2}\sum_{j=1}^{m} (A + D_j)^{-2}.$$

Finally, the updating equation for the ML estimator of A is

$$\hat{A}_{ML}^{(k+1)} = \hat{A}_{ML}^{(k)} + \left\{ I_{ML}(\hat{A}_{ML}^{(k)}) \right\}^{-1} s_{ML}(\hat{A}_{ML}^{(k)}).$$

Initial value of A and stopping criterion are set the same as in the FH method described before. Let $\hat{A}^*_{ML}$ be the estimate obtained in the last iteration of the algorithm. Then, the final ML estimate is $\hat{A}_{ML} = \max(0, \hat{A}^*_{ML})$.

The REML estimator of A is obtained by maximizing the joint p.d.f. of a transformation $\mathbf{F}'\mathbf{Y}$ of the data $\mathbf{Y}$, where $\mathbf{F}$ is an $m \times p$ matrix satisfying $\mathbf{F}'\mathbf{X} = \mathbf{0}$. Then, the REML estimator maximizes the following function that does not depend on β,

$$\ell_R(A;\mathbf{Y}) = c - \frac{1}{2}\log|\mathbf{F}'\Sigma(A)\mathbf{F}| - \frac{1}{2}\mathbf{Y}'\mathbf{F}\left(\mathbf{F}'\Sigma(A)\mathbf{F}\right)^{-1}\mathbf{F}'\mathbf{Y}.$$

It holds that

$$\mathbf{F}\left(\mathbf{F}'\Sigma(A)\mathbf{F}\right)^{-1}\mathbf{F}' = \mathbf{P}(A),$$

where

$$\mathbf{P}(A) = \Sigma^{-1}(A) - \Sigma^{-1}(A)\mathbf{X}\left(\mathbf{X}'\Sigma^{-1}(A)\mathbf{X}\right)^{-1}\mathbf{X}'\Sigma^{-1}(A).$$

Using this relation, we obtain

$$\ell_R(A;\mathbf{Y}) = c - \frac{1}{2}\log|\mathbf{F}'\Sigma(A)\mathbf{F}| - \frac{1}{2}\mathbf{Y}'\mathbf{P}(A)\mathbf{Y}.$$

The score is obtained by taking derivative of ℓ_R with respect to A, and is given by

$$\begin{aligned}
s_R(A) &= -\frac{1}{2}\text{trace}\{\mathbf{P}(A)\} + \frac{1}{2}\mathbf{Y}'\mathbf{P}^2(A)\mathbf{Y} \\
&= -\sum_{j=1}^{m}(A+D_j)^{-1} - \text{trace}\{\left(\mathbf{X}'\Sigma^{-1}(A)\mathbf{X}\right)^{-1}\mathbf{X}'\Sigma^{-2}(A)\mathbf{X}\} \\
&\quad + \frac{1}{2}\left\{\mathbf{Y} - \mathbf{X}\tilde{\beta}(A)\right\}'\Sigma^{-2}(A)\left\{\mathbf{Y} - \mathbf{X}\tilde{\beta}(A)\right\} \\
&= \sum_{j=1}^{m}(A+D_j)^{-1} - \sum_{j=1}^{m}\frac{\mathbf{x}_j'\left\{\sum_{k=1}^{m}(A+D_k)^{-1}\mathbf{x}_k\mathbf{x}_k'\right\}^{-1}\mathbf{x}_j}{(A+D_j)^2} - \sum_{j=1}^{m}\frac{\{Y_j - \mathbf{x}_j'\tilde{\beta}(A)\}^2}{(A+D_j)^2}.
\end{aligned}$$

The REML estimator of A is obtained by solving the non-linear equation $s_R(A) = 0$. Again, application of Fisher-scoring algorithm requires also the Fisher information for A, which is given by

$$\begin{aligned}
\mathcal{I}_R(A) &= E\left\{-s_R'(A)\right\} = \frac{1}{2}\text{trace}\{\mathbf{P}^2(A)\} \\
&= \frac{1}{2}\text{trace}\{\Sigma(A)^{-2}\} - \text{trace}\left[\{\mathbf{X}'\Sigma^{-1}(A)\mathbf{X}\}^{-1}\mathbf{X}'\Sigma^{-3}(A)\mathbf{X}\right] \\
&\quad + \frac{1}{2}\text{trace}\left\{\left(\left[\{\mathbf{X}'\Sigma^{-1}(A)\mathbf{X}\}^{-1}\mathbf{X}'\Sigma^{-3}(A)\mathbf{X}\right]\right)^2\right\}.
\end{aligned}$$

Finally, the updating equation is

$$\hat{A}^{(k+1)}_{REML} = \hat{A}^{(k)}_{REML} + \left\{\mathcal{I}_R(\hat{A}^{(k)}_{REML})\right\}^{-1} s_R(\hat{A}^{(k)}_{REML}).$$

Initial value $\hat{A}^{(0)}$ and stopping criterion are set the same as in the FH and ML methods. Again, if $\hat{A}^*_{REML}$ is the last value obtained in the iteration, then REML estimate is finally $\hat{A}_{REML} = \max(0, \hat{A}^*_{REML})$.

The estimator of β is obtained by replacing in (1.5), A by an estimator $\hat{A}$, that is, $\hat{\beta} = \tilde{\beta}(\hat{A})$. Similarly, the EB estimator is obtained as in (1.6). Function `fitFH` delivers, together with estimates of model coefficients β, their asymptotic standard errors given by the diagonal elements of the Fisher information in each case, the Z statistics obtained by dividing the estimates by their standard errors, and the p-values of the significance tests. Since for large m, it holds

$$\hat{\beta} \sim N(\beta, \mathcal{I}^{-1}(\beta)),$$

where $\mathcal{I}(\beta)$ is the Fisher information, then the Z statistic for a coefficient β_j is

$$Z_j = \hat{\beta}_j / \sqrt{\hat{var}(\hat{\beta}_j)}, \quad j = 1, \ldots, p,$$

where $\hat{var}(\hat{\beta}_j)$ is the estimated asymptotic variance of $\hat{\beta}_j$, given by the j-th element in the diagonal of $\mathcal{I}^{-1}(\hat{\beta})$. Finally, for the test

$$H_0 : \beta_j = 0 \quad \text{versus} \quad H_1 : \beta_j \neq 0,$$

p-values are obtained as

$$\text{p-value} = 2P(Z > |Z_j|),$$

where Z is a standard normal random variable.

Three different goodness of fit measures are also delivered by function `fitFH`. The first one is the estimated log-likelihood $\ell(\hat{A}, \hat{\beta}; \mathbf{Y})$, obtained by replacing the obtained estimates $\hat{A}$ and $\hat{\beta}$ in (1.9). The second is AIC, given in this case by

$$\text{AIC} = -2\ell(\hat{A}, \hat{\beta}; \mathbf{Y}) + 2(p+1).$$

Finally, the BIC is obtained as

$$\text{BIC} = -2\ell(\hat{A}, \hat{\beta}; \mathbf{Y}) + (p+1)\log(m).$$

1.1.3 Mean squared error of the EB estimator

In practical applications, the EB estimator $\hat{\theta}_i^{EB}$ should be accompanied with its estimated MSE. Under model (1.1), the MSE of the best estimator $\hat{\theta}_i^{B}$ is given by

$$MSE(\hat{\theta}_i^B) = E(\hat{\theta}_i^B - \theta_i)^2 = E\{V(\theta_i|Y_i)\} = E\{g_{1i}(A)\} = g_{1i}(A),$$

showing that a large reduction in MSE over $MSE(Y_i) = E[E\{(Y_i - \theta_i)^2|\theta_i\}] = E(D_i) = D_i$ is obtained when $1 - B_i(A) = A/(A + D_i)$ is small. Under normality of random effects and errors, the MSE of the EB can be decomposed as

$$\begin{aligned} \text{MSE}(\hat{\theta}_i^{EB}) &= \text{MSE}[\tilde{\theta}_i^{EB}(A)] \quad + E\{[\tilde{\theta}_i^{EB}(\hat{A}) - \tilde{\theta}_i^{EB}(A)]^2\} \\ &= [g_{1i}(A) + g_{2i}(A)] + g_{3i}(A), \end{aligned} \tag{1.11}$$

where $g_{1i}(A)$ is $O(1)$ for large m, $g_{2i}(A)$ is due to the estimation of β and is $O(m^{-1})$, and the last term measures the uncertainty of the EB estimator arising from the estimation of A and is of lower order (Prasad & Rao, 1990). The last two terms on the right hand side of (1.11) are given by

$$\begin{aligned} g_{2i}(A) &= \{B_i(A)\}^2 \mathbf{x}_i' \left\{ \sum_{j=1}^{m} (A+D_j)^{-1} \mathbf{x}_j \mathbf{x}_j' \right\}^{-1} \mathbf{x}_i \\ &=: \{B_i(A)\}^2 \tilde{h}_{ii} \end{aligned} \tag{1.12}$$

and

$$g_{3i}(A) = \{B_i(A)\}^2 (A+D_i)^{-1} \bar{V}(\hat{A}), \tag{1.13}$$

where $\bar{V}(\hat{A})$ is the asymptotic variance (as $m \to \infty$) of an estimator $\hat{A}$ of A. Note that $g_{3i}(A)$ depends on the choice of $\hat{A}$.

Using the REML estimator $\hat{A}_{REML}$, a nearly unbiased estimator of $MSE(\hat{\theta}_i^{EB})$ is given by

$$mse_{REML}(\hat{\theta}_i^{EB}) = g_{1i}(\hat{A}_{REML}) + g_{2i}(\hat{A}_{REML}) + 2g_{3i}(\hat{A}_{REML}), \tag{1.14}$$

where $g_{1i}(A)$ is given by (1.5) and

$$\bar{V}(\hat{A}_{REML}) = \frac{2}{\sum_{j=1}^{m} (A+D_j)^{-2}}; \tag{1.15}$$

see Datta and Lahiri (2000).

For the Fay–Herriot (FH) estimator $\hat{A}_{FH}$, we need the following expression for its bias to terms of order $O(m^{-1})$:

$$b_{FH}(A) = \frac{2\left[m \sum_{j=1}^{m} (A+D_j)^{-2} - \left\{\sum_{j=1}^{m} (A+D_j)^{-1}\right\}^2\right]}{\left\{\sum_{j=1}^{m} (A+D_j)^{-1}\right\}^3}. \tag{1.16}$$

Note that the bias of $\hat{A}_{REML}$ is zero if terms of order $o(m^{-1})$ are ignored.

A nearly unbiased estimator of $MSE(\hat{\theta}_i^{EB})$ using $\hat{A}_{FH}$ is given by

$$mse_{FH}(\hat{\theta}_i^{EB}) = g_{1i}(\hat{A}_{FH}) + g_{2i}(\hat{A}_{FH}) + 2g_{3i}(\hat{A}_{FH}) - b_{FH}(\hat{A}_{FH})\{B_i(\hat{A}_{FH})\}^2, \tag{1.17}$$

where, in $g_{3i}(\hat{A}_{FH})$, the asymptotic variance is

$$\bar{V}(\hat{A}_{FH}) = \frac{2m}{\left\{\sum_{j=1}^{m} (A+D_j)^{-1}\right\}^2}; \tag{1.18}$$

see Datta et al. (2005).

1.2 The Software: description of R functions

This section describes the implemented R functions that fit the basic Fay-Herriot model (1.1), give the small area EB estimates and analytical estimates of the MSE of the EB estimator. In the rest of this section we describe briefly these R functions. An example showing the use of these functions is provided in Section 1.3 and full R codes are included in Appendix 1.

1.2.1 fitFH

R function `fitFH` fits the basic Fay-Herriot model (1.1). This function is defined as

```
fitFH<-function(X,y,Dvec,method="REML",MAXITER=500)
```

Arguments of this function are:

`X`: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`y`: vector containing the direct estimates of the response variable for the m areas.

`Dvec`: vector containing the m sampling variances $D_1, \dots, D_m$ of direct estimators.

`method`: type of fitting method, to be chosen between REML or FH methods. Default is REML method.

`MAXITER`: maximum number of iterations allowed in the Fisher-scoring algorithm. Default is 500 iterations.

The function returns a list with the following objects:

`convergence`: a logical value equal to TRUE if Fisher-scoring algorithm converges in less than MAXITER iterations.

`modelcoefficients`: data.frame in the shape of a table with estimated model coefficients in first column, their (asymptotic) standard errors in second column, Z statistics in third column and p-values of the significance of each coefficient in last column.

`variance`: estimated random effects variance A.

`goodnessoffit`: a vector containing three basic goodness-of-fit measures, loglikelihood, AIC and BIC.

`EBpredictor`: a vector of size m with the values of the EB predictor for the m areas.

1.2.2 MSE.FHmodel

R function `MSE.FHmodel` gives analytical MSE estimates of EB predictors for the m small areas, when EB predictors are obtained from the basic Fay-Herriot model (1.1). This function is defined as

```
MSE.FHmodel<-function(X,Dvec,A,method="REML")
```

Arguments of this function are:

`X:` matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`Dvec:` vector containing the m sampling variances $D_1, \ldots, D_m$ of direct estimators.

`A:` estimated random effects variance A obtained using the fitting method specified in `method`:.

`method:` Type of fitting method, to be chosen between REML or FH methods. Default is REML method.

The function returns:

`mse:` a vector with the estimated MSEs of the EB small area estimators.

1.3 Examples of usage of R functions

This section shows how to use the R functions described in Section 1.2 to produce small area EB estimators along with their corresponding estimated MSEs, based on the basic Fay-Herriot model (1.1).

1.3.1 Example data set

We consider the data set on milk expenditure used initially by Arora and Lahiri (1997) and later by You and Chapman (2006). This data set comes from the Consumer Expenditure Survey conducted by the U.S. Bureau of the Census and is used by the Bureau of Labor Statistics to compute the monthly Consumer Price Index (CPI) numbers. Here the small areas are the $m = 43$ publication areas of the CPI throughout the U.S. and Y_i is the i-the area direct estimator of the average expenditure on fresh whole milk for the year 1989. As explanatory variables in the Fay-Herriot model, we used indicators for the 4 major areas created by You and Chapman (2006). Table 1.1 lists the full data including Small area, sample size n_i, direct estimate y_i, standard error of direct estimate y_i (SD), coefficient of variation of y_i (CV) and Major area.

Small area	n_i	y_i	SD	CV	Major area
1	191	1.099	0.163	0.148	1
2	633	1.075	0.08	0.074	1
3	597	1.105	0.083	0.075	1
4	221	0.628	0.109	0.174	1
5	195	0.753	0.119	0.158	1
6	191	0.981	0.141	0.144	1
7	183	1.257	0.202	0.161	1
8	188	1.095	0.127	0.116	2
9	204	1.405	0.168	0.12	2
10	188	1.356	0.178	0.131	2
11	149	0.615	0.1	0.163	2

12	290	1.46	0.201	0.138	2
13	250	1.338	0.148	0.111	2
14	194	0.854	0.143	0.167	2
15	184	1.176	0.149	0.127	3
16	193	1.111	0.145	0.131	3
17	218	1.257	0.135	0.107	3
18	266	1.43	0.172	0.12	3
19	214	1.278	0.137	0.107	3
20	213	1.292	0.163	0.126	3
21	196	1.002	0.125	0.125	3
22	95	1.183	0.247	0.209	3
23	195	1.044	0.14	0.134	3
24	187	1.267	0.171	0.135	3
25	479	1.193	0.106	0.089	3
26	230	0.791	0.121	0.153	4
27	186	0.795	0.121	0.152	4
28	199	0.759	0.259	0.341	4
29	238	0.796	0.106	0.133	4
30	207	0.565	0.089	0.158	4
31	165	0.886	0.225	0.254	4
32	153	0.952	0.205	0.215	4
33	210	0.807	0.119	0.147	4
34	383	0.582	0.067	0.115	4
35	255	0.684	0.106	0.155	4
36	226	0.787	0.126	0.16	4
37	224	0.44	0.092	0.209	4
38	212	0.759	0.132	0.174	4
39	211	0.77	0.1	0.13	4
40	179	0.8	0.113	0.141	4
41	312	0.756	0.083	0.11	4
42	241	0.865	0.121	0.14	4
43	205	0.64	0.129	0.202	4

Table 1.1: Data on Milk expenditure.

1.3.2 Example of R code for running function fitFH

Here we include an example of R code used to read the data set in Table 1.1 and run function `fitFH` using that data. R code includes suitable comments explaining what is each line doing.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
```

```
data<-read.table("MilkData.txt",header=TRUE)
attach(data)

# Create the auxiliary variables, which are the indicators
# of 4 Major Areas
# Create these indicators and put them in the columns of matrix X.

m<-dim(data)[1]
M<-length(unique(MajorArea))
X<-matrix(0,nr=m,nc=M)
for (i in 1:4) {X[,i]<-as.numeric(MajorArea==i)}

# Load file where function is located
source("Fitting_FHModel.R")

# Call the function using REML method and put the output
# in the list results.
results<-fitFH(X,yi,SD^2,method="REML")

# Print function output
print(results)

# Fit function using FH method.
# Include the function output in an object call results.
# Now call the function using FH method and put the output in
# list results.
results<-fitFH(X,yi,SD^2,method="FH")
print(results)
```

1.3.3 Output of function fitFH

Output of function `fitFH` when setting

```
method="REML"
```

is given below:

```
------------------------------------------------------
$convergence
[1] TRUE

$modelcoefficients
```

```
  beta.REML std.errorbeta   tvalue       pvalue
1  0.968189    0.06936237 13.95842 2.795652e-44
2  1.100970    0.07614518 14.45882 2.205456e-47
3  1.195135    0.06094029 19.61158 1.231550e-85
4  0.726888    0.04301468 16.89860 4.606911e-64

$variance
[1] 0.01855048

$goodnessoffit
   loglike        AIC        BIC
 12.677463 -15.354927  -6.548926

$EBpredictor
           [,1]
 [1,] 1.0219708
 [2,] 1.0476021
 [3,] 1.0679516
 [4,] 0.7608159
 [5,] 0.8461566
 [6,] 0.9743727
 [7,] 1.0584532
 [8,] 1.0977764
 [9,] 1.2215462
[10,] 1.1951466
[11,] 0.7852141
[12,] 1.2139470
[13,] 1.2096603
[14,] 0.9834961
[15,] 1.1864247
[16,] 1.1556980
[17,] 1.2263414
[18,] 1.2856494
[19,] 1.2363250
[20,] 1.2349603
[21,] 1.0903013
[22,] 1.1923057
[23,] 1.1216465
[24,] 1.2230299
[25,] 1.1938055
```

```
[26,] 0.7627197
[27,] 0.7649553
[28,] 0.7338445
[29,] 0.7699297
[30,] 0.6134414
[31,] 0.7695564
[32,] 0.7958257
[33,] 0.7723190
[34,] 0.6102299
[35,] 0.7001781
[36,] 0.7592790
[37,] 0.5298859
[38,] 0.7434468
[39,] 0.7548997
[40,] 0.7701921
[41,] 0.7481165
[42,] 0.8040778
[43,] 0.6810868
------------------------------------------------------
```

Output of function `fitFH` when setting

```
method="FH"
```

is given below:

```
------------------------------------------------------
$convergence
[1] TRUE

$modelcoefficients
    beta.FH std.errorbeta   tvalue       pvalue
1 0.9679012    0.06695896 14.45514 2.326569e-47
2 1.0973513    0.07400814 14.82744 9.737434e-50
3 1.1946922    0.05925359 20.16236 2.096386e-90
4 0.7257494    0.04150620 17.48532 1.853544e-68

$variance
[1] 0.01642027

$goodnessoffit
```

```
  loglike       AIC       BIC
 12.76205 -15.52410  -6.71810

$EBpredictor
           [,1]
 [1,] 1.0179759
 [2,] 1.0449639
 [3,] 1.0644808
 [4,] 0.7706920
 [5,] 0.8525124
 [6,] 0.9738262
 [7,] 1.0508569
 [8,] 1.0961652
 [9,] 1.2105053
[10,] 1.1856404
[11,] 0.7975687
[12,] 1.2021499
[13,] 1.2004587
[14,] 0.9889713
[15,] 1.1867450
[16,] 1.1579920
[17,] 1.2242232
[18,] 1.2786804
[19,] 1.2335659
[20,] 1.2318601
[21,] 1.0959551
[22,] 1.1922126
[23,] 1.1259974
[24,] 1.2206948
[25,] 1.1936875
[26,] 0.7602435
[27,] 0.7623581
[28,] 0.7322880
[29,] 0.7674591
[30,] 0.6173102
[31,] 0.7649969
[32,] 0.7893148
[33,] 0.7693760
[34,] 0.6128615
[35,] 0.7009616
```

```
[36,] 0.7568908
[37,] 0.5371932
[38,] 0.7418816
[39,] 0.7532513
[40,] 0.7675187
[41,] 0.7470595
[42,] 0.7993630
[43,] 0.6831609
--------------------------------------------------
```

1.3.4 Example of R code for running function MSE.FHmodel

Here we include an example of R code used to read the data set in Table 1.1, fit the Fay-Herriot model to that data set using function `fitFH` which gives also the small area EB estimators, and finally run function `MSE.FHmodel` to obtain analytical MSE estimators of EB predictors. R code includes suitable comments explaining what is each line doing.

```
# Set path where data set and functions are
setwd("Path")

# Read data set
data<-read.table("MilkData.txt",header=TRUE)
attach(data)

# The auxiliary variables are indicators of 4 Major Areas
# Create these indicators

m<-dim(data)[1]
M<-length(unique(MajorArea))
X<-matrix(0,nr=m,nc=M)
for (i in 1:4) {X[,i]<-as.numeric(MajorArea==i)}

# Load files where R functions are
source("Fitting_FHModel.R")
source("MSE_FHModel.R")

# Fit FH model using FH method
results<-fitFH(X,yi,SD^2,method="FH")

# Compute estimated MSEs of EB estimators
mse<-MSE.FHmodel(X,SD^2,results$variance,method="REML")
```

```
mse
```

1.3.5 Output of function MSE.FHmodel

Output of function `MSE.FHmodel` when setting

```
method="REML"
```

is given below:

```
-------------------------------------------------------------
 [1] 0.012757016 0.005314467 0.005632201 0.008323471 0.009283520
 [6] 0.011178151 0.014867661 0.010252709 0.013470878 0.014094867
[11] 0.007558331 0.015325273 0.012038943 0.011640323 0.011466957
[16] 0.011181888 0.010423673 0.012914352 0.010580560 0.012385543
[21] 0.009599980 0.015890239 0.010810965 0.012857861 0.007864537
[26] 0.008855177 0.008855177 0.015041525 0.007569376 0.005975211
[31] 0.014211799 0.013571948 0.008691546 0.003833361 0.007569376
[36] 0.009253152 0.006264329 0.009709449 0.007020470 0.008185874
[41] 0.005391054 0.008855177 0.009484220
-------------------------------------------------------------
```

Chapter 2

Area-level spatial model

2.1 Methodology

2.1.1 Model and small area EB estimator

The Fay-Herriot model defined as (1.1) can be also expressed as

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{v} + \mathbf{e}, \tag{2.1}$$

where $\mathbf{y} = (y_1, \ldots, y_m)^T$ is the vector of direct estimators for the m small areas, $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)^T$ is a $m \times p$ matrix containing in the columns the values of p explanatory variables for the m areas, $\mathbf{v} = (v_1, \ldots, v_m)^T$ is a vector of independent and identically distributed variables (called random effects) with $\mathbf{v} \sim N(\mathbf{0}, \sigma_v^2 I_m)$ and $\mathbf{e} = (e_1, \ldots, e_m)^T$ is the vector of independent sampling errors, independent of $\mathbf{v}$, with $\mathbf{e} \sim N(\mathbf{0}, \Psi)$, where the covariance matrix $\Psi = \text{diag}(\psi_1, \ldots, \psi_m)$ is known. Here, the target quantity is the vector $\theta = \mathbf{X}\beta + \mathbf{v} = (\theta_1, \ldots, \theta_m)^T$, which usually contains the true means of the target variable in the m areas.

Model (2.1) can be extended to allow for spatially correlated area effects as follows. Let $\mathbf{v}$ be the result of a SAR process with unknown autoregression parameter ρ and proximity matrix $\mathbf{W}$ (Anselin, 1988; Cressie, 1993), i.e.,

$$\mathbf{v} = \rho\mathbf{W}\mathbf{v} + \mathbf{u}. \tag{2.2}$$

We assume that the matrix $(\mathbf{I}_m - \rho\mathbf{W})$ is non-singular. Then $\mathbf{v}$ can be expressed as

$$\mathbf{v} = (\mathbf{I}_m - \rho\mathbf{W})^{-1}\mathbf{u}. \tag{2.3}$$

Here, $\mathbf{u} = (u_1, \ldots, u_m)^T$ is a vector with mean $\mathbf{0}$ and covariance matrix $\sigma_u^2\mathbf{I}_m$, where $\mathbf{I}_m$ denotes the $m \times m$ identity matrix and σ_u^2 is an unknown parameter. We consider that the proximity matrix $\mathbf{W}$ is defined in row standardized form; that is, $\mathbf{W}$ is row stochastic. Then, $\rho \in (-1, 1)$ is called spatial autocorrelation parameter (Banerjee et al., 2004). Hereafter, the vector of variance components will be denoted $\omega = (\omega_1, \omega_2)^T = (\sigma_u^2, \rho)^T$. Equation (2.3) implies that $\mathbf{v}$ has mean vector $\mathbf{0}$ and covariance matrix equal to

$$\mathbf{G}(\omega) = \sigma_u^2[(\mathbf{I}_m - \rho\mathbf{W})^T(\mathbf{I}_m - \rho\mathbf{W})]^{-1}. \tag{2.4}$$

Since $\mathbf{e}$ is independent of $\mathbf{v}$, the covariance matrix of $\mathbf{y}$ is equal to

$$\mathbf{V}(\omega) = \mathbf{G}(\omega) + \Psi.$$

Combining (2.1) and (2.3), the model is

$$\mathbf{y} = \mathbf{X}\beta + (\mathbf{I}_m - \rho\mathbf{W})^{-1}\mathbf{u} + \mathbf{e} \tag{2.5}$$

Under model (2.5), the Spatial BLUP of the quantity of interest $\theta_i = \mathbf{x}_i^T\beta + v_i$ is

$$\tilde{\theta}_i(\omega) = \mathbf{x}_i^T\tilde{\beta}(\omega) + \mathbf{b}_i^T\mathbf{G}(\omega)\mathbf{V}^{-1}(\omega)[\mathbf{y} - \mathbf{X}\tilde{\beta}(\omega)], \tag{2.6}$$

where $\tilde{\beta}(\omega) = [\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{y}$ is the generalized least squares estimator of the regression parameter β and $\mathbf{b}_i^T$ is the $1 \times m$ vector $(0,\dots,0,1,0,\dots,0)$ with 1 in the i-th position. The Spatial BLUP $\tilde{\theta}_i(\omega)$ depends on the unknown vector of variance components $\omega = (\sigma_u^2, \rho)^T$. The two stage estimator $\tilde{\theta}_i(\hat{\omega})$ obtained by replacing ω in expression (2.6) by a consistent estimator $\hat{\omega} = (\hat{\sigma}_u^2, \hat{\rho})^T$ is called Spatial EBLUP (Singh et al., 2005; Petrucci & Salvati, 2006).

2.1.2 Fitting methods

A maximum likelihood estimator (MLE) of $\omega = (\sigma_u^2, \rho)^T$ is obtained maximizing the log-likelihood of ω given the data vector $\mathbf{y}$,

$$\ell(\omega;\mathbf{y}) = c - \frac{1}{2}\log|\mathbf{V}(\omega)| - \frac{1}{2}(\mathbf{y} - \mathbf{X}\beta)^T\mathbf{V}^{-1}(\omega)(\mathbf{y} - \mathbf{X}\beta),$$

where c denotes a constant. In practice, an iterative algorithm such as the Fisher-scoring algorithm must be applied to maximize the likelihood. Let $\mathbf{S}(\omega) = (S_{\sigma_u^2}, S_\rho)^T$ be the scores or derivatives of the log-likelihood with respect to σ_u^2 and ρ, and let $\mathcal{I}(\omega)$ be the Fisher information matrix obtained from $\ell(\omega;\mathbf{y})$, with elements

$$\mathcal{I}(\omega) = \begin{pmatrix} \mathcal{I}_{\sigma_u^2,\sigma_u^2} & \mathcal{I}_{\sigma_u^2,\rho^2} \\ \mathcal{I}_{\rho,\sigma_u^2} & \mathcal{I}_{\rho,\rho} \end{pmatrix}.$$

Then the Fisher-scoring algorithm starts with an initial estimate $\omega^{(0)} = (\sigma_u^{2(0)}, \rho^{(0)})^T$ and then at each iteration k, this estimate is updated with the equation

$$\omega^{(k+1)} = \omega^{(k)} + \mathcal{I}^{-1}(\omega^{(k)})\mathbf{S}(\omega^{(k)}).$$

The ML equation for β obtained by equating the corresponding score to zero yields

$$\tilde{\beta}(\omega) = [\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{y}. \tag{2.7}$$

Let us denote

$$\mathbf{C}(\rho) = (\mathbf{I}_m - \rho\mathbf{W})^T(\mathbf{I}_m - \rho\mathbf{W})$$

and

$$\mathbf{P}(\omega) = \mathbf{V}^{-1}(\omega) - \mathbf{V}^{-1}(\omega)\mathbf{X}\left[\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X}\right]^{-1}\mathbf{X}^T\mathbf{V}^{-1}(\omega).$$

Then the derivative of $\mathbf{C}(\rho)$ with respect to ρ is

$$\frac{\partial \mathbf{C}(\rho)}{\partial \rho} = -\mathbf{W} - \mathbf{W}^T + 2\rho \mathbf{W}^T \mathbf{W}$$

and the derivatives of $\mathbf{V}(\omega)$ with respect to σ_u^2 and ρ are respectively given by

$$\frac{\partial \mathbf{V}(\omega)}{\partial \sigma_u^2} = \mathbf{C}^{-1}(\rho), \quad \frac{\partial \mathbf{V}(\omega)}{\partial \rho} = -\sigma_u^2 \mathbf{C}^{-1}(\rho)\frac{\partial \mathbf{C}(\rho)}{\partial \rho}\mathbf{C}^{-1}(\rho) \triangleq \mathbf{A}(\omega).$$

The scores associated to σ_u^2 and ρ, after replacing (2.7), are given by

$$\begin{aligned} S_{\sigma_u^2} &= -\frac{1}{2}\operatorname{trace}\left\{\mathbf{V}^{-1}(\omega)\mathbf{C}^{-1}(\rho)\right\} + \frac{1}{2}\mathbf{y}^T\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\mathbf{P}(\omega)\mathbf{y}, \\ S_\rho &= -\frac{1}{2}\operatorname{trace}\left\{\mathbf{V}^{-1}(\omega)\mathbf{A}^{-1}(\omega)\right\} + \frac{1}{2}\mathbf{y}^T\mathbf{P}(\omega)\mathbf{A}(\omega)\mathbf{P}(\omega)\mathbf{y}. \end{aligned}$$

The elements of the Fisher information matrix are

$$\begin{aligned} &I_{\sigma_u^2,\sigma_u^2} = \frac{1}{2}\operatorname{trace}\left\{\mathbf{V}^{-1}(\omega)\mathbf{C}^{-1}(\rho)\mathbf{V}^{-1}(\omega)\mathbf{C}^{-1}(\rho)\right\}, \\ &I_{\sigma_u^2,\rho} = I_{\rho,\sigma_u^2} = \frac{1}{2}\operatorname{trace}\left\{\mathbf{V}^{-1}(\omega)\mathbf{A}(\omega)\mathbf{V}^{-1}(\omega)\mathbf{C}^{-1}(\rho)\right\}, \\ &I_{\rho,\rho} = \frac{1}{2}\operatorname{trace}\left\{\mathbf{V}^{-1}(\omega)\mathbf{A}(\omega)\mathbf{V}^{-1}(\omega)\mathbf{A}(\omega)\right\}. \end{aligned}$$

In the function `fitSpatialFH`, the starting value of σ_u^2 is set to $\sigma_u^{2(0)} = \operatorname{median}(\psi_i)$. For ρ, we take $\rho^{(0)} = 0.5$. The algorithm stops either when the number of iterations $k >$ MAXITER where MAXITER can be chosen by the user (default is 500), or when

$$\max\left\{\left|\frac{\sigma_u^{2(k+1)} - \sigma_u^{2(k)}}{\sigma_u^{2(k)}}\right|, \left|\frac{\rho^{(k+1)} - \rho^{(k)}}{\rho^{(k)}}\right|\right\} < 0.0001.$$

A restricted maximum likelihood estimator (RMLE) of ω is obtained by maximizing the restricted likelihood, which is the joint p.d.f. of a transformation of the response $\mathbf{y}$, that eliminates the vector of coefficients β. Let $\mathbf{F}$ be an $m \times p$ matrix satisfying $\mathbf{F}^T\mathbf{X} = \mathbf{0}$. Then, the restricted log-likelihood is the logarithm of the joint p.d.f. of the transformed data $\mathbf{F}^T\mathbf{y}$ and is given by

$$\ell_R(\omega;\mathbf{y}) = c - \frac{1}{2}\log|\mathbf{F}^T\mathbf{V}(\omega)\mathbf{F}| - \frac{1}{2}\mathbf{y}^T\mathbf{F}(\mathbf{F}^T\mathbf{V}(\omega)\mathbf{F})^{-1}\mathbf{F}^T\mathbf{y},$$

where

$$\mathbf{F}\left[\mathbf{F}^T\mathbf{V}(\omega)\mathbf{F}\right]^{-1}\mathbf{F}^T = \mathbf{P}(\omega),$$

so that the restricted log-likelihood becomes

$$\ell_R(\omega;\mathbf{y}) = c - \frac{1}{2}\log|\mathbf{F}^T\mathbf{V}(\omega)\mathbf{F}| - \frac{1}{2}\mathbf{y}^T\mathbf{P}(\omega)\mathbf{y}.$$

Using the following properties of the matrix $\mathbf{P}(\omega)$,

$$\mathbf{P}(\omega)\mathbf{V}(\omega)\mathbf{P}(\omega)=\mathbf{P}(\omega),\quad \frac{\partial \mathbf{P}(\omega)}{\partial \omega_j}=-\mathbf{P}(\omega)\frac{\partial \mathbf{V}(\omega)}{\partial \omega_j}\mathbf{P}(\omega),$$

we obtain the scores corresponding to this restricted log-likelihood,

$$S^R_{\sigma_u^2}=-\frac{1}{2}\operatorname{trace}\left\{\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\right\}+\frac{1}{2}\mathbf{y}^T\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\mathbf{P}(\omega)\mathbf{y},$$
$$S^R_{\rho}=-\frac{1}{2}\operatorname{trace}\left\{\mathbf{P}(\omega)\mathbf{A}(\omega)\right\}+\frac{1}{2}\mathbf{y}^T\mathbf{P}(\omega)\mathbf{A}(\omega)\mathbf{P}(\omega)\mathbf{y},$$

Finally, the elements of the Fisher information obtained from ℓ_R are

$$I^R_{\sigma_u^2,\sigma_u^2}=\frac{1}{2}\operatorname{tr}\{\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\},$$
$$I^R_{\sigma_u^2,\rho}=I^R_{\rho,\sigma_u^2}=\frac{1}{2}\operatorname{tr}\{\mathbf{P}(\omega)\mathbf{A}(\omega)\mathbf{P}(\omega)\mathbf{C}^{-1}(\rho)\},$$
$$I^R_{\rho,\rho}=\frac{1}{2}\operatorname{tr}\{\mathbf{P}(\omega)\mathbf{A}(\omega)\mathbf{P}(\omega)\mathbf{A}(\omega)\}.$$

Starting values and stopping criterion are set the same as in the case of ML estimates.

2.1.3 Mean squared error of the Spatial EBLUP

Again, under normality of random effects and errors, the MSE of the Spatial EBLUP can be decomposed as

$$\begin{aligned}\mathrm{MSE}[\tilde{\theta}_i(\hat{\omega})] &= \mathrm{MSE}[\tilde{\theta}_i(\omega)] + E\{[\tilde{\theta}_i(\hat{\omega})-\tilde{\theta}_i(\omega)]^2\}\\ &= [g_{1i}(\omega)+g_{2i}(\omega)] + g_{3i}(\omega),\end{aligned}\tag{2.8}$$

where the first two terms on the right hand side are easily calculated due to the linearity of the Spatial BLUP $\tilde{\theta}_i(\omega)$ in the data vector $\mathbf{y}$. They are given by

$$g_{1i}(\omega) = \mathbf{b}_i^T[\mathbf{G}(\omega)-\mathbf{G}(\omega)\mathbf{V}^{-1}(\omega)\mathbf{G}(\omega)]\mathbf{b}_i,\tag{2.9}$$
$$g_{2i}(\omega) = \mathbf{b}_i^T[\mathbf{I}_m-\mathbf{G}(\omega)\mathbf{V}^{-1}(\omega)]\mathbf{X}(\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X})^{-1}\mathbf{X}^T[\mathbf{I}_m-\mathbf{V}^{-1}(\omega)\mathbf{G}(\omega)]\mathbf{b}_i.\tag{2.10}$$

However, for the last term $g_{3i}(\omega)=E\{[\tilde{\theta}_i(\hat{\omega})-\tilde{\theta}_i(\omega)]^2\}$, an exact analytical expression does not exist due to the non-linearity of the EBLUP $\tilde{\theta}_i(\hat{\omega})$ in $\mathbf{y}$. Under the basic Fay-Herriot model (2.1) with independent random effects v_i (diagonal covariance matrix $\mathbf{V}$), Prasad & Rao (1990) obtained an approximation up to $o(m^{-1})$ terms of $g_{3i}(\omega)$ through Taylor linearization, see Section 1.1.3. Their formula can be taken as a naive approximation of the true $g_{3i}(\omega)$ under model (2.1)–(2.2). Straightforward application of this formula to model (2.1)–(2.2) yields

$$g_{3i}^{PR}(\omega)=\operatorname{trace}\left\{\mathbf{L}_i(\omega)\mathbf{V}(\omega)\mathbf{L}_i^T(\omega)I^{-1}(\omega)\right\},$$

where

$$\mathbf{L}_i(\omega)=\begin{pmatrix}\mathbf{b}_i^T\left[\mathbf{C}^{-1}(\rho)\mathbf{V}^{-1}(\omega)-\sigma_u^2\mathbf{C}^{-1}(\rho)\mathbf{V}^{-1}(\omega)\mathbf{C}^{-1}(\rho)\mathbf{V}^{-1}(\omega)\right]\\ \mathbf{b}_i^T\left[\mathbf{A}(\omega)\mathbf{V}^{-1}(\omega)-\sigma_u^2\mathbf{C}^{-1}(\rho)\mathbf{V}^{-1}(\omega)\mathbf{A}(\omega)\mathbf{V}^{-1}(\omega)\right]\end{pmatrix}.$$

Then the full MSE can be approximated by

$$\mathrm{MSE}^{PR}[\tilde{\theta}_i(\hat{\omega})] = g_{1i}(\omega) + g_{2i}(\omega) + g_{3i}^{PR}(\omega). \tag{2.11}$$

Singh et al. (2005) arrived to the same formula (2.11) for the true MSE under a Fay-Herriot model with random effects following a SAR process. However, this formula is not accounting for the extra uncertainty of the Spatial EBLUP $\tilde{\theta}_i(\hat{\omega})$ due to the estimation of the autocorrelation parameter ρ.

As to MSE estimation, when $\hat{\omega}$ is obtained by REML method, Singh et al. (2005) derived the following MSE estimator

$$\mathrm{mse}^{SSK}[\tilde{\theta}_i(\hat{\omega})] = g_{1i}(\hat{\omega}) + g_{2i}(\hat{\omega}) + 2g_{3i}^{PR}(\hat{\omega}) - g_{4i}(\hat{\omega}). \tag{2.12}$$

Here, $g_{4i}(\omega)$ is given by

$$g_{4i}(\omega) = \frac{1}{2}\sum_{k=1}^{2}\sum_{\ell=1}^{2} \mathbf{b}_i^T \Psi \mathbf{V}^{-1}(\omega)\frac{\partial^2 \mathbf{V}(\omega)}{\partial\omega_k\partial\omega_\ell}\mathbf{V}^{-1}(\omega)\Psi \mathcal{I}_{k\ell}^{-1}(\omega)\mathbf{b}_i,$$

where the second order derivatives of $\mathbf{V}(\omega)$ are given by

$$\frac{\partial^2\mathbf{V}(\omega)}{\partial(\sigma_u^2)^2} = 0_{m\times m};$$
$$\frac{\partial^2\mathbf{V}(\omega)}{\partial\sigma_u^2\partial\rho} = \frac{\partial^2\mathbf{V}(\omega)}{\partial\sigma_u^2\partial\rho} = -\mathbf{C}^{-1}(\omega)\frac{\partial\mathbf{C}(\rho)}{\partial\rho}\mathbf{C}^{-1}(\omega)$$
$$\frac{\partial^2\mathbf{V}(\omega)}{\partial\partial\rho^2} = 2\sigma_u^2\mathbf{C}^{-1}(\omega)\frac{\partial\mathbf{C}(\rho)}{\partial\rho}\mathbf{C}^{-1}(\omega)\frac{\partial\mathbf{C}(\rho)}{\partial\rho}\mathbf{C}^{-1}(\omega) - 2\sigma_u^2\mathbf{C}^{-1}(\omega)\mathbf{W}^T\mathbf{W}\mathbf{C}^{-1}(\omega).$$

When $\hat{\omega}$ is obtained by ML, their estimator is

$$\mathrm{mse}_{ML}^{SSK}[\tilde{\theta}_i(\hat{\omega})] = g_{1i}(\hat{\omega}) + g_{2i}(\hat{\omega}) + 2g_{3i}^{PR}(\hat{\omega}) - g_{4i}(\hat{\omega}) - \mathbf{b}_{ML}^T(\hat{\omega})\nabla g_{1i}(\hat{\omega}), \tag{2.13}$$

where $\nabla g_{1i}(\omega) = \partial g_{1i}(\omega)/\partial\omega$ is the gradient of $g_{1i}(\omega)$ and $\mathbf{b}_{ML}(\hat{\omega})$ is the bias of the ML estimator $\hat{\omega}$ up to order $o(m^{-1})$. This bias is equal to $\mathbf{b}_{ML}(\hat{\omega}) = \mathcal{I}^{-1}(\hat{\omega})\mathbf{h}(\hat{\omega})/2$ with $\mathbf{h}(\hat{\omega}) = (h_1(\hat{\omega}), h_2(\hat{\omega}))^T$ and

$$h_k(\omega) = \mathrm{trace}\left\{\left[\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X}\right]^{-1}\frac{\partial\left[\mathbf{X}^T\mathbf{V}^{-1}(\omega)\mathbf{X}\right]}{\partial\omega_k}\right\}, \quad k = 1, 2.$$

2.1.4 Parametric bootstrap mean squared error

Here we propose to use the parametric bootstrap of González-Manteiga et al. (2008) extended to the FH model with spatial correlation (2.1)–(2.2). The final MSE estimate obtained by this procedure is expected to be consistent if the model parameter estimates are consistent. This could be seen by the method of imitation as in González-Manteiga et al. (2008), using the asymptotic formula of the MSE obtained by Singh et al. (2005). This extended parametric bootstrap works as follows:

1) Fit model (2.5) to the initial data $\mathbf{y} = (y_1, \ldots, y_m)^T$, obtaining estimates $\hat{\omega} = (\hat{\sigma}_u^2, \hat{\rho})^T$ and $\hat{\beta} = \tilde{\beta}(\hat{\omega})$.

2) Generate a vector $\mathbf{t}_1^*$ whose elements are m independent copies of a $N(0,1)$. Construct bootstrap vectors $\mathbf{u}^* = \hat{\sigma}_u \mathbf{t}_1^*$ and $\mathbf{v}^* = (\mathbf{I}_m - \hat{\rho}\mathbf{W})^{-1}\mathbf{u}^*$, and calculate the bootstrap quantity of interest $\theta^* = \mathbf{X}\hat{\beta} + \mathbf{v}^*$, by regarding $\hat{\beta}$ and $\hat{\omega}$ as the true values of the parameters.

3) Generate a vector $\mathbf{t}_2^*$ with m independent copies of a $N(0,1)$, independently of the generation of $\mathbf{t}_1^*$, and construct the vector of random errors $\mathbf{e}^* = \Psi^{1/2}\mathbf{t}_2^*$.

4) Obtain bootstrap data $\mathbf{y}^*$ directly applying the model, $\mathbf{y}^* = \theta^* + \mathbf{e}^* = \mathbf{X}\hat{\beta} + \mathbf{v}^* + \mathbf{e}^*$.

5) Regarding $\hat{\beta}$ and $\hat{\omega}$ as the true values of β and ω, fit model (2.5) to bootstrap data $\mathbf{y}^*$, obtaining estimates of the "true" $\hat{\beta}$ and $\hat{\omega}$ based on bootstrap data $\mathbf{y}^*$: first, calculate the estimator of $\hat{\beta}$ calculated at the "true" $\hat{\omega}$,

$$\tilde{\beta}^*(\hat{\omega}) = \left[\mathbf{X}^T\mathbf{V}^{-1}(\hat{\omega})\mathbf{X}\right]^{-1}\mathbf{X}^T\mathbf{V}^{-1}(\hat{\omega})\mathbf{y}^*;$$

next, obtain the estimator $\hat{\omega}^*$ based on $\mathbf{y}^*$, and finally, the estimator of $\hat{\beta}$ calculated at $\hat{\omega}^*$, that is, $\tilde{\beta}^*(\hat{\omega}^*)$.

6) Calculate the bootstrap Spatial BLUP from bootstrap data $\mathbf{y}^*$ and regarding $\hat{\omega}$ as the true value of ω,

$$\tilde{\theta}_i^*(\hat{\omega}) = \mathbf{x}_i^T\tilde{\beta}^*(\hat{\omega}) + \mathbf{b}_i^T\mathbf{G}(\hat{\omega})\mathbf{V}(\hat{\omega})^{-1}[\mathbf{y}^* - \mathbf{X}\tilde{\beta}^*(\hat{\omega})].$$

Calculate also the bootstrap Spatial EBLUP using $\hat{\omega}^*$ in place of the "true" $\hat{\omega}$,

$$\tilde{\theta}_i^*(\hat{\omega}^*) = \mathbf{x}_i^T\tilde{\beta}^*(\hat{\omega}^*) + \mathbf{b}_i^T\mathbf{G}(\hat{\omega}^*)\mathbf{V}^{-1}(\hat{\omega}^*)[\mathbf{y}^* - \mathbf{X}\tilde{\beta}^*(\hat{\omega}^*)].$$

7) Repeat steps 2)–6) B times. In b-th bootstrap replication, let $\theta_i^{*(b)}$ be the quantity of interest for i-th area, $\hat{\omega}^{*(b)}$ the bootstrap estimate of ω, $\tilde{\theta}_i^{*(b)}(\hat{\omega})$ the bootstrap Spatial BLUP and $\tilde{\theta}_i^{*(b)}(\hat{\omega}^{*(b)})$ the bootstrap Spatial EBLUP for i-th area.

8) A parametric bootstrap estimator of $g_{3i}(\omega)$ is

$$g_{3i}^{PB}(\hat{\omega}) = B^{-1}\sum_{b=1}^{B}\left[\tilde{\theta}_i^{*(b)}(\hat{\omega}^{*(b)}) - \tilde{\theta}_i^{*(b)}(\hat{\omega})\right]^2.$$

Similarly, a naive parametric bootstrap estimator of the full MSE is

$$\mathrm{mse}^{naPB}[\tilde{\theta}_i(\hat{\omega})] = B^{-1}\sum_{b=1}^{B}\left[\tilde{\theta}_i^{*(b)}(\hat{\omega}^{*(b)}) - \theta_i^{*(b)}\right]^2. \tag{2.14}$$

Another MSE estimate can be obtained as in Pfeffermann & Tiller (2006), by adding the analytical estimates $g_{1i}(\hat{\omega})$ and $g_{2i}(\hat{\omega})$, the bootstrap estimate $g_{3i}^{PB}(\hat{\omega})$, and a bootstrap bias correction of $g_{1i}(\hat{\omega}) + g_{2i}(\hat{\omega})$. The final estimator obtained in this way is

$$\mathrm{mse}^{bcPB}[\tilde{\theta}_i(\hat{\omega})] = 2\left[g_{1i}(\hat{\omega}) + g_{2i}(\hat{\omega})\right] - B^{-1}\sum_{b=1}^{B}\left[g_{1i}(\hat{\omega}^{*(b)}) + g_{2i}(\hat{\omega}^{*(b)})\right] + g_{3i}^{PB}(\hat{\omega}). \tag{2.15}$$

2.1.5 Nonparametric bootstrap

This section describes a nonparametric bootstrap for MSE estimation, in which bootstrap random effects $\{u_1^*,\dots,u_m^*\}$ and bootstrap random errors $\{e_1^*,\dots,e_m^*\}$ are obtained by resampling respectively from the empirical distribution of predicted random effects $\{\hat{u}_1,\dots,\hat{u}_m\}$ and of residuals $\{\hat{r}_1,\dots,\hat{r}_m\}$, where $r_i = y_i - \tilde{\theta}_i(\hat{\omega})$, $i = 1,\dots,m$, both previously standardized. This method avoids the need of distributional assumptions; therefore, it is expected to be more robust to non-normality of any of the random components of the model.

Under model (2.1)–(2.2), the BLUPs of **u** and **v** are respectively

$$\tilde{\mathbf{v}}(\omega) = \mathbf{G}(\omega)\mathbf{V}^{-1}(\omega)[\mathbf{y}-\mathbf{X}\tilde{\beta}(\omega)], \quad \tilde{\mathbf{u}}(\omega) = (\mathbf{I}-\rho\mathbf{W})\tilde{\mathbf{v}}(\omega),$$

and the covariance matrix of $\tilde{\mathbf{u}}(\omega)$ is

$$\mathbf{V}_{\mathbf{u}}(\omega) = (\mathbf{I}-\rho\mathbf{W})\mathbf{G}(\omega)\mathbf{P}(\omega)\mathbf{G}(\omega)(\mathbf{I}-\rho\mathbf{W}^T).$$

Moreover, consider the vector of residuals

$$\tilde{\mathbf{r}}(\omega) = \mathbf{y}-\mathbf{X}\tilde{\beta}(\omega)-\tilde{\mathbf{v}}(\omega) = (y_1-\tilde{\theta}_1(\omega),\dots,y_m-\tilde{\theta}_m(\omega))^T.$$

It is easy to see that the covariance matrix of $\tilde{\mathbf{r}}(\omega)$ is

$$\mathbf{V}_{\mathbf{r}}(\omega) = \Psi\mathbf{P}(\omega)\Psi.$$

The covariance matrices $\mathbf{V}_{\mathbf{u}}(\omega)$ and $\mathbf{V}_{\mathbf{r}}(\omega)$ are not diagonal; hence, the elements of the vectors $\tilde{\mathbf{u}}(\omega)$ and $\tilde{\mathbf{r}}(\omega)$ are correlated. Indeed, both $\tilde{\mathbf{u}}(\omega)$ and $\tilde{\mathbf{r}}(\omega)$ lie in a space of dimension $m-p$. Since the methods that resample from the empirical distribution work well under an ideally *iid* setup, when applying these methods, a previous standardization step is crucial. Here we propose to transform both $\hat{\mathbf{u}} = \tilde{\mathbf{u}}(\hat{\omega})$ and $\hat{\mathbf{r}} = \tilde{\mathbf{r}}(\hat{\omega})$ to make them as close as possible to vectors with uncorrelated and unit variance elements. We describe the standardization method only for $\hat{\mathbf{u}}$, since for $\hat{\mathbf{r}}$ the process is analogous. Let us consider the estimated covariance matrix $\hat{\mathbf{V}}_{\mathbf{u}} = \mathbf{V}_{\mathbf{u}}(\hat{\omega})$. The method works by carrying out the spectral decomposition of $\hat{\mathbf{V}}_{\mathbf{u}}$,

$$\hat{\mathbf{V}}_{\mathbf{u}} = \mathbf{Q}_{\mathbf{u}}\Delta_{\mathbf{u}}\mathbf{Q}_{\mathbf{u}}^T,$$

where $\Delta_{\mathbf{u}}$ is a diagonal matrix with the $m-p$ non-zero eigenvalues of $\hat{\mathbf{V}}_{\mathbf{u}}$ and $\mathbf{Q}_{\mathbf{u}}$ is the matrix with the corresponding eigenvectors in the columns. Then we take the matrix $\hat{\mathbf{V}}_{\mathbf{u}}^{-1/2} = \mathbf{Q}_{\mathbf{u}}\Delta_{\mathbf{u}}^{-1/2}\mathbf{Q}_{\mathbf{u}}^T$. Squaring this matrix gives a generalized inverse of $\hat{\mathbf{V}}_{\mathbf{u}}$. With the obtained square root, we transform $\hat{\mathbf{u}}$ as

$$\hat{\mathbf{u}}^S = \hat{\mathbf{V}}_{\mathbf{u}}^{-1/2}\hat{\mathbf{u}}.$$

The covariance matrix of $\hat{\mathbf{u}}^S$ is then $Var(\hat{\mathbf{u}}^S) = \mathbf{Q}_{\mathbf{u}}\mathbf{Q}_{\mathbf{u}}^T$, which is close to an identity matrix. Observe that in the transformation

$$\hat{\mathbf{u}}^S = \mathbf{Q}_{\mathbf{u}}\Delta_{\mathbf{u}}^{-1/2}\mathbf{Q}_{\mathbf{u}}^T\hat{\mathbf{u}},$$

the vector $\mathbf{Q}_\mathbf{u}^T\hat{\mathbf{u}}$ contains the coordinates of $\hat{\mathbf{u}}$ in its principal components, which are uncorrelated with covariance matrix $\Delta_\mathbf{u}$. Then multiplying by $\Delta_\mathbf{u}^{-1/2}$, these coordinates are standardized to have unit variance. Finally, this standardized vector in the space of the principal components is returned to the original space by multiplying by $\mathbf{Q}_\mathbf{u}$. Thus, the transformed vector $\hat{\mathbf{u}}^S$ contains the coordinates of the vector $\Delta_\mathbf{u}^{-1/2}\mathbf{Q}_\mathbf{u}^T\hat{\mathbf{u}}$, with standard elements, in the original space. The eigenvalues, which are the variances of the uncorrelated principal components, collect better the variability than the diagonals of $\hat{\mathbf{V}}_\mathbf{u}$. Indeed, simulations were also carried out standardizing simply by taking $\hat{u}_i^S = \hat{u}_i/\sqrt{v_{ii}}$, where v_{ii} is the i-the diagonal element of $\hat{\mathbf{V}}_\mathbf{u}$, but the resulting nonparametric bootstrap did not work well.

The final nonparametric bootstrap procedure works by replacing steps 2) and 3) of the parametric bootstrap by the new steps 2') and 3') below:

2') With the estimates $\hat{\omega} = (\hat{\sigma}_u^2, \hat{\rho})^T$ and $\hat{\beta} = \tilde{\beta}(\hat{\omega})$ obtained in step 1), calculate predictors of $\mathbf{v}$ and $\mathbf{u}$ as follows

$$\hat{\mathbf{v}} = \mathbf{G}(\hat{\omega})\mathbf{V}(\hat{\omega})^{-1}(\mathbf{y} - \mathbf{X}\hat{\beta}), \quad \hat{\mathbf{u}} = (\mathbf{I} - \hat{\rho}\mathbf{W})\hat{\mathbf{v}} = (\hat{u}_1, \ldots, \hat{u}_m)^T.$$

Then take $\hat{\mathbf{u}}^S = \hat{\mathbf{V}}_\mathbf{u}^{-1/2}\hat{\mathbf{u}} = (\hat{u}_1^S, \ldots, \hat{u}_m^S)^T$, where $\hat{\mathbf{V}}_\mathbf{u}^{1/2}$ is the square root of the generalized inverse of $\hat{\mathbf{V}}_\mathbf{u}$ obtained by the spectral decomposition. It is convenient to re-scale the elements $\hat{u}_i^S$ so that they have sample mean exactly equal to zero and sample variance $\hat{\sigma}_u^2$. This is achieved by the transformation

$$\hat{u}_i^{SS} = \frac{\hat{\sigma}_u(\hat{u}_i^S - m^{-1}\sum_{j=1}^m \hat{u}_j^S)}{\sqrt{m^{-1}\sum_{d=1}^m(\hat{u}_d^S - m^{-1}\sum_{j=1}^m \hat{u}_j^S)^2}}, \quad i = 1, \ldots, m.$$

Construct the vector $\mathbf{u}^* = (u_1^*, \ldots, u_m^*)^T$, whose elements are obtained by extracting a simple random sample with replacement of size m from the set $\{\hat{u}_1^{SS}, \ldots, \hat{u}_m^{SS}\}$. Then obtain $\mathbf{v}^* = (\mathbf{I} - \hat{\rho}\mathbf{W})^{-1}\mathbf{u}^*$ and calculate the bootstrap quantity of interest $\theta^* = \mathbf{X}\hat{\beta} + \mathbf{v}^* = (\theta_1^*, \ldots, \theta_m^*)^T$

3') Compute the vector of residuals $\hat{\mathbf{r}} = \mathbf{y} - \mathbf{X}\hat{\beta} - \hat{\mathbf{v}} = (\hat{r}_1, \ldots, \hat{r}_m)^T$. Standardize the residuals by $\hat{\mathbf{r}}^S = \hat{\mathbf{V}}_\mathbf{r}^{-1/2}\hat{\mathbf{r}} = (\hat{r}_1^S, \ldots, \hat{r}_m^S)^T$, where $\hat{\mathbf{V}}_\mathbf{r} = \Psi\mathbf{P}(\hat{\omega})\Psi$ is the estimated covariance matrix and $\hat{\mathbf{V}}_\mathbf{r}^{-1/2}$ is a root square of the generalized inverse derived from the spectral decomposition of $\hat{\mathbf{V}}_\mathbf{r}$. Again, re-standardize these values

$$\hat{r}_i^{SS} = \frac{\hat{r}_i^S - m^{-1}\sum_{j=1}^m \hat{r}_j^S}{\sqrt{m^{-1}\sum_{d=1}^m(\hat{r}_d^S - m^{-1}\sum_{j=1}^m \hat{r}_j^S)^2}}, \quad i = 1, \ldots, m.$$

Construct $\mathbf{r}^* = (r_1^*, \ldots, r_m^*)^T$ by extracting a simple random sample with replacement of size m from the set $\{\hat{r}_1^{SS}, \ldots, \hat{r}_m^{SS}\}$. Then take $\mathbf{e}^* = (e_1^*, \ldots, e_m^*)^T$, where $e_i^* = \psi_i^{1/2} r_i^*$, $i = 1, \ldots, m$.

This procedure yields naive and bias-corrected nonparametric bootstrap estimators analogous to (2.14) and (2.15). They are respectively denoted as $\mathrm{mse}^{naNPB}[\tilde{\theta}_i(\hat{\omega})]$ and $\mathrm{mse}^{bcNPB}[\tilde{\theta}_i(\hat{\omega})]$.

2.2 The Software: description of R functions

This section describes the implemented R functions that fit the area-level spatial model with random effects following a SAR process as defined in (2.5), give the spatial EBLUP small area estimators and provide analytical, parametric bootstrap and nonparametric bootstrap MSE estimators. A brief description of these R functions is given in the rest of this section. An example showing the use of these functions is provided in the next section and full R codes are included in Appendix 2.

2.2.1 fitSpatialFH

R function `fitSpatialFH` fits the area-level spatial model with random effects following a SAR process, as defined in (2.5), using REML fitting method. The function is defined as

```
fitSpatialFH<-function(X,y,Dvec,W,method="REML",MAXITER=500)
```

Arguments of this function are:

`X`: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`y`: vector containing the direct estimates of the response variable for the m areas.

`Dvec`: vector containing the m sampling variances $D_1, \dots, D_m$ of direct estimators.

`W`: $m \times m$ proximity matrix with rows adding to one.

`method`: type of fitting method, to be chosen between REML or ML methods. Default is REML method.

`MAXITER`: maximum number of iterations allowed to the Fisher-scoring algorithm. Default is 500 iterations.

The function returns a list with the following objects:

`convergence`: a logical value equal to TRUE if Fisher-scoring algorithm converges in less than MAXITER iterations.

`modelcoefficients`: data.frame in the shape of a table with the estimated model coefficients in first column, their asymptotic standard errors in second column, the Z statistics in third column and the p-values of the significance of each coefficient in last column.

`variance`: estimated random effects variance σ_u^2.

`spatialcorr`: estimated spatial correlation parameter ρ.

`goodnessoffit`: a vector containing three different goodness-of-fit measures, namely the loglikelihood, the AIC and the BIC.

`EBpredictor`: a vector of size m with the values of the EB predictor for the m areas.

2.2.2 MSE.SpatialFHmodel

R function `MSE.SpatialFHmodel` gives the analytical MSE estimates of the Spatial EBLUPs for the m small areas obtained from the Spatial Fay-Herriot model (2.5). This function is defined as

```
MSE.SpatialFHmodel<-function(X,Dvec,A,rho,W,method="REML")
```

Arguments of this function are:

`X`: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`Dvec`: vector containing the m sampling variances $D_1, \ldots, D_m$ of direct estimators.

`A`: estimated random effects variance σ_u^2 obtained using the fitting method specified in `method`.

`rho`: estimated spatial autocorrelation parameter obtained using the fitting method specified in `method`.

`W`: proximity matrix with rows adding to one.

`method`: type of fitting method. Currently only REML method is available.

The function returns:

`mse`: a vector with the analytical MSE estimates of the Spatial EBLUP small area estimators.

2.2.3 PBMSE.SpatialFHmodel

R function `PBMSE.SpatialFHmodel` gives parametric bootstrap MSE estimates of the Spatial EBLUPs for the m small areas obtained from the Spatial Fay-Herriot model (2.5). This function is defined as

```
PBMSE.SpatialFHmodel<-function(X,Dvec,beta,A,rho,W,n.boot,method="REML",
seed=Sys.time())
```

Arguments of this function are:

`X`: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`Dvec`: vector containing the m sampling variances $D_1, \ldots, D_m$ of direct estimators.

`beta`: estimated regression coefficients β obtained using the fitting method specified in `method`.

`A`: estimated random effects variance σ_u^2 obtained using the fitting method specified in `method`.

`rho`: estimated spatial autocorrelation parameter obtained using the fitting method specified in `method`.

`W`: proximity matrix with rows adding to one.

`n.boot`: number of bootstrap replicates.

`method`: type of fitting method. Currently only REML method is available.

`seed`: Seed to be used in the generation of samples for the parametric bootstrap method. Default is the system time.

The function returns a data frame containing the following two vectors:

`PBmse`: naive parametric bootstrap MSE estimates of the Spatial EBLUP small area estimators.

`bcPBmse`: bias-corrected parametric bootstrap MSE estimates of the Spatial EBLUP small area estimators.

2.2.4 NPBMSE.SpatialFHmodel

R function `NPBMSE.SpatialFHmodel` gives nonparametric bootstrap MSE estimates of the Spatial EBLUPs for the m small areas obtained from the Spatial Fay-Herriot model (2.5). This function is defined as

```
NPBMSE.SpatialFHmodel<-function(X,y,Dvec,W,n.boot,method="REML",
seed=Sys.time())
```

Arguments of this function are:

`X`: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $m \times p$, where m is the number of areas or sample size. The elements in first column might be equal to 1 if the model includes an intercept.

`y`: vector containing the direct estimates of the response variable for the m areas.

`W`: proximity matrix with rows adding to one.

`n.boot`: number of bootstrap replicates.

`method`: type of fitting method. Currently only REML method is available.

`seed`: Seed to be used in the generation of samples for the non parametric bootstrap method. Default is the system time.

The function returns a data frame containing the following two vectors:

`NPBmse`: naive nonparametric bootstrap MSE estimates of the Spatial EBLUP small area estimators.

`bcNPBmse`: bias-corrected nonparametric bootstrap MSE estimates of the Spatial EBLUP small area estimators.

2.3 Examples of usage of R functions

This section shows how to use the R functions described in Section 1.2 to produce small area EB estimators along with their corresponding estimated MSE, based on the basic Fay-Herriot model (1.1).

2.3.1 Example data set

The example data set used to show the use of functions `fitSpatialFH` and `MSE.SpatialFHmodel` has been obtained drawing a sample from the Italian Agricultural Census of year 2000, see Molina et al. (2008) for a description of the data generation. The small areas are $m = 274$ municipalities of Tuscany and a proximity matrix was obtained from the neighborhood structure of these municipalities, with one assigned to the pairs of municipalities that share a common edge. Data set for the first ten municipalities is shown below. The response variable with the direct estimates for the m municipalities is grapehect, the auxiliary variables are area and workdays, and var is the sampling error of direct estimators.

```
       grapehect       area  workdays          var
1      30.9477582  203.93775  73.95884 2.153054e+01
2      57.2161416  187.22505 148.21444 2.014299e+02
3      73.7540672  590.73016 171.34921 2.818218e+00
4      66.2420341  318.32333 105.86333 2.138089e+01
5      36.9317990  217.25789  87.83813 6.424393e+01
6      78.5339314 1562.02985 202.68657 1.629171e-01
7      50.4567217  101.47826  93.48161 1.473757e+01
8      41.9724571  147.48728  85.71624 1.950657e+01
9     111.5708810  274.27101 233.63077 3.293872e+02
10     10.2326214  145.89855  32.67029 6.659302e-03
...
```

The proximity matrix for the first 10 municipalities is

```
     V1        V2        V3        V4        V5        V6        V7
1  0.00 0.3333333 0.3333333 0.0000000 0.0000000 0.0000000 0.0000000
2  0.25 0.0000000 0.0000000 0.2500000 0.0000000 0.0000000 0.2500000
3  0.50 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
4  0.00 0.3333333 0.0000000 0.0000000 0.3333333 0.0000000 0.3333333
5  0.00 0.0000000 0.0000000 0.1428571 0.0000000 0.1428571 0.1428571
6  0.00 0.0000000 0.0000000 0.0000000 0.5000000 0.0000000 0.0000000
7  0.00 0.2000000 0.0000000 0.2000000 0.2000000 0.0000000 0.0000000
8  0.20 0.2000000 0.2000000 0.0000000 0.0000000 0.0000000 0.2000000
9  0.00 0.0000000 0.0000000 0.0000000 0.1111111 0.1111111 0.0000000
10 0.00 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
```

```
          V8        V9       V10 ...
1  0.3333333 0.0000000 0.0000000
2  0.2500000 0.0000000 0.0000000
3  0.5000000 0.0000000 0.0000000
4  0.0000000 0.0000000 0.0000000
5  0.0000000 0.1428571 0.0000000
6  0.0000000 0.5000000 0.0000000
7  0.2000000 0.0000000 0.0000000
8  0.0000000 0.0000000 0.0000000
9  0.0000000 0.0000000 0.1111111
10 0.0000000 0.2500000 0.0000000
...
```

2.3.2 Example of R code for running function fitSpatialFH

Here we include an example of R code used to read the data set from the Italian Agriculture Census listed in Section 2.3.1 and run function `fitFH` using that data. R code includes suitable comments explaining what is each line doing.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
data<-read.table("SimulSpatData_ItAgricSurvey2000.txt",header=TRUE)
attach(data)

# Load file where function is located
source("Fitting_SpatialFHModel.R")

# Read file with proximity matrix and assign it to a matrix called prox.
prox<-as.matrix(read.table("ProximitySpatData_ItAgricSurvey2000.txt"))

# Define matrix with area-level auxiliary variables
X<-as.matrix(cbind(area,workdays))

# Fit function using REML method and put the output in the list
resultsSp.

resultsSp<-fitSpatialFH(X,grapehect,var,prox,method="ML")
print(resultsSp)
```

2.3.3 Output of function fitSpatialFH

Output of function `fitSpatialFH` when setting

`method="ML"`

is given below. Spatial EB predictors are given only for the first 10 small areas for brevity.

```
------------------------------------------------------
$convergence
[1] TRUE

$modelcoefficients
               Bstim std.errorbeta    tvalue       pvalue
area     -0.01241993   0.002095302 -5.927514 3.075547e-09
workdays  0.49861050   0.012732816 39.159485 0.000000e+00

$variance
[1] 67.69242

$spatialcorr
[1] 0.6550672

$EBpredictor
            [,1]
V1    31.2591747
V2    71.9504581
V3    73.8753880
V4    62.2488593
V5    39.6819534
V6    78.5365740
V7    50.1555221
V8    41.2015557
V9   109.2337422
V10   10.2332637
...
------------------------------------------------------
```

2.3.4 Example of R code for running function MSE.SpatialFHmodel

Here we include an example of R code used to read the data set from the Italian Agriculture Census in Section 2.3.1, fit the Spatial Fay-Herriot model to that data set using function `fitSpatialFH` which gives also the small area EB estimators, and finally run function `MSE.SpatialFHmodel` to obtain analytical MSE estimates of Spatial EBLUPs for the $m = 274$ municipalities. R code includes suitable comments explaining what is each line doing.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
data<-read.table("SimulSpatData_ItAgricSurvey2000.txt",header=TRUE)
attach(data)

# Read proximity matrix and assign it to a matrix called prox
prox<-as.matrix(read.table("ProximitySpatData_ItAgricSurvey2000.txt"))

# Create matrix with area-level auxiliary variables
X<-as.matrix(cbind(area,workdays))

# Load files where functions are
source("Fitting_SpatialFHModel.R")
source("MSE_SpatialFHModel.R")

# Fit the Spatial Fay-Herriot model with REML method
results<-fitSpatialFH(X,grapehect,var,prox)

# Obtain analytical MSE estimates of Spatial EBLUPs
mse<-MSE.SpatialFHmodel(X,var,results$variance,results$spatialcorr,prox)
mse
```

2.3.5 Output of function MSE.SpatialFHmodel

Output of function `MSE.SpatialFHmodel` when setting

```
method="REML"
```

is given below for the first 10 small areas:

```
-----------------------------------------------------------------------
[1] 1.661451e+01 5.180644e+01 2.721105e+00 1.692434e+01 3.133056e+01
[6] 1.626345e-01 1.226893e+01 1.508771e+01 4.901339e+01 6.658713e-03 ...
-----------------------------------------------------------------------
```

2.3.6 Example of R code for running function PBMSE.SpatialFHmodel

Here we include an example of R code used to read the data set from the Italian Agriculture Census in Section 2.3.1, fit the Spatial Fay-Herriot model to that data set using function `fitSpatialFH` which gives also the small area EB estimators, and finally run function `PBMSE.SpatialFHmodel` to obtain parametric bootstrap MSE estimates of Spatial EBLUPs for the $m = 274$ municipalities. R code includes suitable comments explaining what is each line doing.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
data<-read.table("SimulSpatData_ItAgricSurvey2000.txt",header=TRUE)
attach(data)

# Read proximity matrix
prox<-as.matrix(read.table("ProximitySpatData_ItAgricSurvey2000.txt"))

# Create the matrix with area-level auxiliary variables
X<-as.matrix(cbind(area,workdays))

# Load files with needed functions
source("Fitting_SpatialFHModel.R")
source("PBMSE_SpatialFHModel.R")

# Fit the Spatial Fay-Herriot model by REML method
results<-fitSpatialFH(X,grapehect,var,prox)

# Take the estimated model coefficients
coef<-results$modelcoefficients$beta

# Take the estimated random effects variance
A<-results$variance

# Take the estimated spatial autocorrelation parameter
rho<-results$spatialcorr

# Obtain the naive and bias-corrected parametric bootstrap mse estimates
PBmse<-PBMSE.SpatialFHmodel(X,var,coef,A,rho,prox,100)
PBmse$PBmse
```

2.3.7 Output of function PBMSE.SpatialFHmodel

Output of function `PBMSE.SpatialFHmodel` is given below for the first 10 small areas:

```
-------------------------------------------------------------------
[1] 1.88266e+01 4.48179e+01 2.33316e+00 1.65272e+01 3.86748e+01
[6] 1.58263e-01 1.11911e+01 1.24805e+01 4.63717e+01 4.57704e-03 ...
-------------------------------------------------------------------
```

2.3.8 Example of R code for running function NPBMSE.SpatialFHmodel

Here we include an example of R code used to read the data set from the Italian Agriculture Census in Section 2.3.1 and run function `NPBMSE.SpatialFHmodel` to obtain nonparametric bootstrap MSE estimates of Spatial EBLUPs for the $m = 274$ municipalities. R code includes suitable comments explaining what is each line doing.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
data<-read.table("SimulSpatData_ItAgricSurvey2000.txt",header=TRUE)
attach(data)

# Read proximity matrix
prox<-as.matrix(read.table("ProximitySpatData_ItAgricSurvey2000.txt"))

# Create the matrix with area-level auxiliary variables
X<-as.matrix(cbind(area,workdays))

# Load file with required function
source("NPBMSE_SpatialFHModel.R")

# Obtain the naive and bias-corrected parametric bootstrap mse estimates
NPBmse<-NPBMSE.SpatialFHmodel(X,grapehect,var,prox,100,seed=1111)
```

2.3.9 Output of function NPBMSE.SpatialFHmodel

Output of function `NPBMSE.SpatialFHmodel` is given below for the first 10 small areas:

```
NPBmse$NPBmse
-------------------------------------------------------------------
  [1] 1.953809e+01 4.751716e+01 2.375469e+00 1.460335e+01 3.886240e+01
  [6] 1.609646e-01 9.748048e+00 1.319229e+01 4.455835e+01 8.025153e-03
```

```
-----------------------------------------------------------------

NPBmse$bcNPBmse
-----------------------------------------------------------------
  [1] 1.660721e+01 5.232989e+01 2.721583e+00 1.696373e+01 3.131900e+01
  [6] 1.626388e-01 1.228415e+01 1.511206e+01 4.886813e+01 6.658720e-03
-----------------------------------------------------------------
```

Chapter 3

Area-level time models

3.1 Area-level model with independent time effects

3.1.1 The methodology

Consider the model

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{dt} + e_{dt}, \quad d = 1,\ldots,D, \quad t = 1,\ldots,m_d, \tag{3.1}$$

where y_{dt} is a direct estimator of the indicator of interest for area d and time instant t, and $\mathbf{x}_{dt}$ is a vector containing the aggregated (population) values of p auxiliary variables. The index d is used for domains and the index t for time instants. We assume that the vectors u_{dt}'s are $N(0,\sigma_u^2)$, the errors e_{dt}'s are independent $N(0,\sigma_{dt}^2)$ with known variances σ_{dt}^2, and the u_{dt}'s are independent of the e_{dt}'s. Model (3.1) can be alternatively written in the form

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \mathbf{e}, \tag{3.2}$$

where $\mathbf{y} = \underset{1\le d\le D}{\mathrm{col}}(\mathbf{y}_d), \mathbf{y}_d = \underset{1\le t\le m_d}{\mathrm{col}}(y_{dt}), \mathbf{u} = \underset{1\le d\le D}{\mathrm{col}}(\mathbf{u}_d), \mathbf{u}_d = \underset{1\le t\le m_d}{\mathrm{col}}(u_{dt}), \mathbf{e} = \underset{1\le d\le D}{\mathrm{col}}(\mathbf{e}_d), \mathbf{e}_d = \underset{1\le t\le m_d}{\mathrm{col}}(e_{dt})$, $\mathbf{X} = \underset{1\le d\le D}{\mathrm{col}}(\mathbf{X}_d), \mathbf{X}_d = \underset{1\le t\le m_d}{\mathrm{col}}(\mathbf{x}_{dt}), \mathbf{x}_{dt} = \underset{1\le i\le p}{\mathrm{col}'}(x_{dti}), \beta = \underset{1\le i\le p}{\mathrm{col}}(\beta_i), \mathbf{Z} = \mathbf{I}_M, M = \sum_{d=1}^D m_d$. We assume that $\mathbf{u} \sim N(\mathbf{0},\mathbf{V}_u)$ and $\mathbf{e} \sim N(\mathbf{0},\mathbf{V}_e)$ are independent with covariance matrices

$$\mathbf{V}_u = \sigma_u^2\mathbf{I}_M, \quad \mathbf{I}_M = \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{I}_{m_d}), \quad \mathbf{V}_e = \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{V}_{ed}), \quad \mathbf{V}_{ed} = \underset{1\le t\le m_d}{\mathrm{col}}(\sigma_{dt}^2),$$

and known variances σ_{dt}^2. The BLUE of β and the BLUP of $\mathbf{u}$ are

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y} \qquad \text{and} \qquad \widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\beta}),$$

where

$$var(\mathbf{y}) = \mathbf{V} = \sigma_u^2 \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{I}_{m_d}) + \mathbf{V}_e = \underset{1\le d\le D}{\mathrm{diag}}(\sigma_u^2\mathbf{I}_{m_d} + \mathbf{V}_{ed}) = \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{V}_d).$$

The estimator $\widehat{\beta}$ and the predictor $\widehat{\mathbf{u}}$ are calculated by applying the formulas

$$\widehat{\beta} = \left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)^{-1}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{y}_d\right) \qquad \text{and} \qquad \widehat{\mathbf{u}} = \sigma_u^2 \underset{1\le d\le D}{\mathrm{col}}\left(\mathbf{V}_d^{-1}(\mathbf{y}_d - \mathbf{X}_d\widehat{\beta})\right).$$

The Henderson 3 estimators of σ_u^2 is

$$\widehat{\sigma}_{uH}^2 = \frac{\mathbf{y}'\mathbf{P}_2\mathbf{y} - (M-p)}{\text{tr}\{\mathbf{P}_2\}},$$

where $\mathbf{Q}_2 = \sum_{d=1}^D \left(\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)^{-1}$ and

$$\begin{aligned}
\mathbf{P}_2 &= \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_{ed}^{-1}) - \underset{1\le d\le D}{\text{col}}(\mathbf{V}_{ed}^{-1}\mathbf{X}_d)\mathbf{Q}_2 \underset{1\le d\le D}{\text{col}'}(\mathbf{X}_d'\mathbf{V}_{ed}^{-1}),\\
\text{tr}\{\mathbf{P}_2\} &= \sum_{d=1}^D\sum_{t=1}^{m_d}\sigma_{dt}^{-2} - \sum_{d=1}^D \text{tr}\{\mathbf{X}_d'\mathbf{V}_{ed}^{-2}\mathbf{X}_d\mathbf{Q}_2\},\\
\mathbf{y}'\mathbf{P}_2\mathbf{y} &= \sum_{d=1}^D\sum_{t=1}^{m_d}\sigma_{dt}^{-2}y_{dt}^2 - \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)\mathbf{Q}_2\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)'.
\end{aligned}$$

The REML estimators are calculated by using the Fisher-scoring algorithm with the updating formula

$$\theta^{k+1} = \theta^k + F^{-1}(\theta^k)S(\theta^k),$$

where $\theta = \sigma_u^2$. The Henderson 3 estimator $\widehat{\sigma}_{uH}^2$ is used as seed of the Fisher-scoring algorithm. The REML score and Fisher amount of information are

$$S = S(\theta) = -\frac{1}{2}\text{tr}(\mathbf{P}) + \frac{1}{2}\mathbf{y}'\mathbf{P}^2\mathbf{y} \quad \text{and} \quad F = F(\theta) = \frac{1}{2}\text{tr}(\mathbf{P}^2),$$

where $\mathbf{Q} = \left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)^{-1}$ and

$$\begin{aligned}
\mathbf{P} &= \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_d^{-1}) - \underset{1\le d\le D}{\text{col}}(\mathbf{V}_d^{-1}\mathbf{X}_d)\mathbf{Q}\underset{1\le d\le D}{\text{col}'}(\mathbf{X}_d'\mathbf{V}_d^{-1}),\\
\text{tr}(\mathbf{P}) &= \sum_{d=1}^D \text{tr}(\mathbf{V}_d^{-1}) - \sum_{d=1}^D \text{tr}(\mathbf{X}_d'\mathbf{V}_d^{-2}\mathbf{X}_d\mathbf{Q}),\\
\text{tr}(\mathbf{P}^2) &= \sum_{d=1}^D \text{tr}(\mathbf{V}_d^{-2}) - 2\sum_{d=1}^D \text{tr}(\mathbf{X}_d'\mathbf{V}_d^{-3}\mathbf{X}_d\mathbf{Q})\\
&+ \text{tr}\left\{\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-2}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-2}\mathbf{X}_d\right)\mathbf{Q}\right\},\\
\mathbf{y}'\mathbf{P}^2\mathbf{y} &= \sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-2}\mathbf{y}_d - 2\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-2}\mathbf{y}_d\right)\\
&+ \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-2}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)'.
\end{aligned}$$

The REML estimator of β is

$$\widehat{\beta}_{REML} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators of σ_u^2 and β are

$$\hat{\sigma}_u^2 \sim N(\theta, F^{-1}(\sigma_u^2)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for σ_u^2 and β_i are

$$\hat{\sigma}_u^2 \pm z_{\alpha/2}\nu^{1/2}, \quad \hat{\beta}_i \pm z_{\alpha/2} q_{ii}^{1/2},\ i = 1, \dots, p,$$

where $\hat{\sigma}_u^2 = \sigma_u^{2,(\kappa)}$, $\nu = F^{-1}(\sigma_u^{2,(\kappa)})$, $(\mathbf{X}'\mathbf{V}^{-1}(\sigma_u^{2,(\kappa)})\mathbf{X})^{-1} = (q_{ij})_{i,j=1,\dots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantile of the standard normal distribution $N(0,1)$. Observed $\hat{\beta}_i = \beta_0$, the p-value for testing the hypothesis $H_0 : \beta_i = 0$ is

$$p = 2P_{H_0}(\hat{\beta}_i > |\beta_0|) = 2P(N(0,1) > |\beta_0|/\sqrt{q_{ii}}).$$

We are interested in predicting $\mu_{dt} = \mathbf{x}_{dt}\beta + u_{dt}$ with the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. If we do not take into account the error e_{dt}, this is equivalent to predict $y_{dt} = \mathbf{a}'\mathbf{y}$, where $\mathbf{a} = \underset{1\le\ell\le D}{\text{col}}(\underset{1\le k\le m_\ell}{\text{col}}(\delta_{d\ell}\delta_{tk}))$ is a vector having one "1" in the cell $t + \sum_{\ell=1}^{d-1} m_\ell$ and "0"'s in the remaining cells. The total $\overline{Y}_{dt}$ is estimated with $\widehat{\overline{Y}}_{dt}^{eblup} = \widehat{\mu}_{dt}$. The mean squared error of $\widehat{\overline{Y}}_{dt}^{eblup}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\sigma_u^2) + g_2(\sigma_u^2) + g_3(\sigma_u^2),$$

and the estimator of $MSE(\widehat{\overline{Y}}_{dt}^{eblup})$ is

$$mse(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\hat{\sigma}_u^2) + g_2(\hat{\sigma}_u^2) + 2g_3(\hat{\sigma}_u^2),$$

where

$$\begin{aligned}
g_1(\sigma_u^2) &= \frac{\sigma_u^2\sigma_{dt}^2}{\sigma_u^2+\sigma_{dt}^2}, \\
g_2(\sigma_u^2) &= \left[\mathbf{a}_d'\mathbf{X}_d - \sigma_u^2\mathbf{a}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d + \sigma_u^4\mathbf{a}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q} \\
&\cdot \left[\mathbf{X}_d'\mathbf{a}_d - \sigma_u^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\mathbf{a}_d + \sigma_u^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\mathbf{V}_d^{-1}\mathbf{a}_d\right], \quad \mathbf{a}_d = \underset{1\le k\le m_d}{\text{col}}(\delta_{tk}), \\
g_3(\sigma_u^2) &= qF^{-1}(\sigma_u^2), \quad q = \frac{1}{\sigma_u^2+\sigma_{dt}^2} - \frac{2\sigma_u^2}{(\sigma_u^2+\sigma_{dt}^2)^2} + \frac{\sigma_u^4}{(\sigma_u^2+\sigma_{dt}^2)^3}
\end{aligned}$$

and F is the REML Fisher amount of information calculated by the updating equation of the Fisher-scoring algorithm.

3.1.2 The Software: description of R functions

This section describes the R functions that have been implemented for fitting the area-level model with independent time effects (3.1). An example of how to use these functions is given in the next section and the related codes are listed in Appendix 3.1.

The developed R software contains a series of functions that return, as final output, the EBLUP estimates of interest. We recall that R functions are objects with the form

$$name \ <- \ function \ (arg_1, arg_2, ...) \ \{expression\}.$$

R functions allows to define a dependent variable *name* as output of a given procedure, when inputs variables are *arguments*. The *expression* within curly brackets contains the needed calculations to obtain *name* from *arguments*. The function codes appearing in *expression* are listed in Appendix A.

A brief descriptions of programmed R functions is given in the next subsections. The functions can be used for calculating the Henderson 3 and the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

H3area

Function **H3area** calculates the unbiased Henderson 3 estimator of σ_u^2 and has the form

$$H3.area \ <- \ function \ (X,\ ydt,\ D,\ md,\ sigma2edt).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the known error variances σ_{ed}^2, with size M.

The function returns the value of the Henderson 3 estimate of $\hat{\sigma}_u^2$.

REMLarea.indep

Function **REMLarea.indep** calculates the estimate of σ_u^2 and the Fisher amount of information F for the Restricted Maximum Likelihood (REML) method. The function is

$$REMLarea.indep \ <- \ function \ (X, ydt, D, md, sigma2edt, sigma.0, MAXITER = 500).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigma.0: Henderson 3 estimate of σ^2_u obtained as output of **H3area**. It is used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 500.

The function returns a list of three elements. First element **sigma.f** is the REML estimate of σ^2_u, second element **Fsig** is the estimated Fisher amount of information F and third element **Q** is the inverse matrix appearing in the expression of $\hat{\beta}$.

BETA.U.area.indep

Function **BETA.U.area.indep** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area.indep <- function\ (X,\ ydt,\ D,\ md,\ sigma2edt,\ sigmau).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the variable of interest for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmau: estimated value of σ^2_u, calculated by the function **REMLarea.indep**.

The function returns an array with the following elements:

beta: vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

u: vector containing the predicted random effects $\widehat{\mathbf{u}}$, with size M.

mse.area.indep

Function **mse.area.indep** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.area.indep \; <\!- \; function \; (X,\; D,\; md,\; sigma2edt,\; sigmau,\; Fsig).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmau: estimated value of σ^2_u, calculated by the function **REMLarea.indep**.

Fsig: estimated Fisher amount of information, calculated by the function **REMLarea.indep**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}_{dt}^{eblup})$, with size M.

Interval.indep

Function **Interval.indep** calculates the asymptotic confidence intervals for σ^2_u and β_i. The function is

$$Interval.indep \; <\!- \; function \; (fit,\; conf = 0.95).$$

The arguments are:

fit: returned object, obtained by applying the function **REMLarea.indep**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ^2_u and β_i respectively.

pvalue

Function **pvalue** calculates the asymptotic p-value of test statistics $\hat{\beta}_i$ for the null hypothesis $H_0 : \beta_i = 0$. The function is

$$pvalue \; <\!- \; function \; (beta0,\; fit).$$

The arguments are:

beta0: observed value of $\hat{\beta}_i$, calculated by the function **BETA.U.area.indep**.

fit: returned object, obtained by applying the function **REMLarea.indep**.

This function returns the vector **pval** containing the asymptotic p-values for hypotheses $H_0 : \beta_i = 0$, $i = 1, \dots, p$, with size p.

3.2 Area-level model with correlated time effects

3.2.1 The methodology

Consider the model

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{dt} + e_{dt}, \quad d = 1,\ldots,D, \quad t = 1,\ldots,m_d, \tag{3.3}$$

where y_{dt} is a direct estimator of the indicator of interest for area d and time instant t, and $\mathbf{x}_{dt}$ is a vector containing the aggregated (population) values of p auxiliary variables. The index d is used for domains and the index t for time instants. We further assume that the random vectors $(u_{d1},\ldots,u_{dm_d}), d = 1,\ldots,D$, follow i.i.d. AR(1) processes with variance and auto-correlation parameters σ_u^2 and ρ respectively, the errors e_{dt}'s are independent $N(0,\sigma_{dt}^2)$ with known variances σ_{dt}^2, and the u_{dt}'s are independent of the e_{dt}'s.

In matrix notation the model is

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \mathbf{e}, \tag{3.4}$$

where $\mathbf{y} = \underset{1\le d\le D}{\text{col}}(\mathbf{y}_d), \mathbf{y}_d = \underset{1\le t\le m_d}{\text{col}}(y_{dt}), \mathbf{u} = \underset{1\le d\le D}{\text{col}}(\mathbf{u}_d), \mathbf{u}_d = \underset{1\le t\le m_d}{\text{col}}(u_{dt}), \mathbf{e} = \underset{1\le d\le D}{\text{col}}(\mathbf{e}_d), \mathbf{e}_d = \underset{1\le t\le m_d}{\text{col}}(e_{dt})$, $\mathbf{X} = \underset{1\le d\le D}{\text{col}}(\mathbf{X}_d), \mathbf{X}_d = \underset{1\le t\le m_d}{\text{col}}(\mathbf{x}_{dt}), \mathbf{x}_{dt} = \underset{1\le i\le p}{\text{col}'}(x_{dti}), \beta = \underset{1\le i\le p}{\text{col}}(\beta_i), \mathbf{Z} = \mathbf{I}_{M\times M}$ and $M = \sum_{d=1}^D m_d$. In this notation, $\mathbf{u} \sim N(\mathbf{0},\mathbf{V}_u)$ and $\mathbf{e} \sim N(\mathbf{0},\mathbf{V}_e)$ are independent with covariance matrices

$$\mathbf{V}_u = \sigma_u^2\Omega(\rho), \quad \Omega(\rho) = \underset{1\le d\le D}{\text{diag}}(\Omega_d(\rho)), \quad \mathbf{V}_e = \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_{ed}), \quad \mathbf{V}_{ed} = \underset{1\le t\le m_d}{\text{diag}}(\sigma_{dt}^2),$$

where the σ_{dt}^2 are known and

$$\Omega_d = \Omega_d(\rho) = \frac{1}{1-\rho^2}\begin{pmatrix} 1 & \rho & \ldots & \rho^{m_d-2} & \rho^{m_d-1} \\ \rho & 1 & \ddots & & \rho^{m_d-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \rho^{m_d-2} & & \ddots & 1 & \rho \\ \rho^{m_d-1} & \rho^{m_d-2} & \ldots & \rho & 1 \end{pmatrix}_{m_d\times m_d}.$$

If the variance components are known, then the BLUE of β and the BLUP of $\mathbf{u}$ are

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y} \qquad \text{and} \qquad \widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\beta}),$$

where

$$var(\mathbf{y}) = \mathbf{V} = \sigma_u^2 \underset{1\le d\le D}{\text{diag}}(\Omega_d(\rho)) + \mathbf{V}_e = \underset{1\le d\le D}{\text{diag}}(\sigma_u^2\Omega_d(\rho) + \mathbf{V}_{ed}) = \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_d).$$

The estimator $\widehat{\beta}$ and the predictor $\widehat{\mathbf{u}}$ are calculated by applying the formulas

$$\widehat{\beta} = \left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)^{-1}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{y}_d\right), \quad \widehat{\mathbf{u}} = \sigma_u^2 \underset{1\le d\le D}{\text{col}}\left(\Omega_d(\rho)\mathbf{V}_d^{-1}(\mathbf{y}_d - \mathbf{X}_d\widehat{\beta})\right).$$

If the variance components are unknown, their REML estimators are calculated by using the Fisher-scoring algorithm with the updating formula

$$\theta^{k+1} = \theta^k + \mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k),$$

where $\theta = (\theta_1, \theta_2) = (\sigma_u^2, \rho)$. As seeds we use $\rho = 0$ and $\sigma_u^{2(0)} = \widehat{\sigma}_{uH}^2$, where $\widehat{\sigma}_{uH}^2$ is the Henderson 3 estimator of σ_u^2 under the model restricted to $\rho = 0$. The REML scores and components of the Fisher information matrix are

$$S_a = -\frac{1}{2}\text{tr}(\mathbf{PV}_a) + \frac{1}{2}\mathbf{y}'\mathbf{PV}_a\mathbf{Py}, \quad F_{ab} = \frac{1}{2}\text{tr}(\mathbf{PV}_a\mathbf{PV}_b), \quad a,b = 1,2.$$

where $\mathbf{V}_1 = \frac{\partial \mathbf{V}}{\partial \sigma_u^2} = \underset{1\le d\le D}{\text{diag}}\,(\Omega_d(\rho))$, $\mathbf{V}_2 = \frac{\partial \mathbf{V}}{\partial \rho} = \sigma_u^2 \underset{1\le d\le D}{\text{diag}}\,(\dot{\Omega}_d(\rho))$, $\mathbf{Q} = \left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)^{-1}$,

$$\begin{aligned}
\mathbf{P} &= \underset{1\le d\le D}{\text{diag}}\,(\mathbf{V}_d^{-1}) - \underset{1\le d\le D}{\text{col}}(\mathbf{V}_d^{-1}\mathbf{X}_d)\mathbf{Q}\underset{1\le d\le D}{\text{col}'}(\mathbf{X}_d'\mathbf{V}_d^{-1}),\\
\mathbf{PV}_a &= \underset{1\le d\le D}{\text{diag}}\,(\mathbf{V}_d^{-1}\mathbf{V}_{ad}) - \underset{1\le d\le D}{\text{col}}(\mathbf{V}_d^{-1}\mathbf{X}_d)\mathbf{Q}\underset{1\le d\le D}{\text{col}'}(\mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}),\\
\text{tr}(\mathbf{PV}_a) &= \sum_{d=1}^D \text{tr}(\mathbf{V}_d^{-1}\mathbf{V}_{ad}) - \sum_{d=1}^D \text{tr}(\mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{X}_d\mathbf{Q}),\\
\text{tr}(\mathbf{PV}_a\mathbf{PV}_b) &= \sum_{d=1}^D \text{tr}(\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{V}_{bd}) - 2\sum_{d=1}^D \text{tr}(\mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{V}_{bd}\mathbf{V}_d^{-1}\mathbf{X}_d\mathbf{Q})\\
&+ \text{tr}\left\{\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{bd}\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\right\}.\\
\mathbf{y}'\mathbf{PV}_a\mathbf{Py} &= \sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{y}_d - \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)'\\
&- \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{y}_d\right)\\
&+ \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{X}_d'\mathbf{V}_d^{-1}\mathbf{V}_{ad}\mathbf{V}_d^{-1}\mathbf{X}_d\right)\mathbf{Q}\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_d^{-1}\mathbf{X}_d\right)'.
\end{aligned}$$

Finally, the derivative of matrix $\Omega_d(\rho)$ with respect to ρ is

$$\dot{\Omega}_d(\rho) = \frac{1}{1-\rho^2}\begin{pmatrix} 0 & 1 & \dots & \dots & (m_d-1)\rho^{m_d-2}\\ 1 & 0 & \ddots & & (m_d-2)\rho^{m_d-3}\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ (m_d-2)\rho^{m_d-3} & & \ddots & 0 & 1\\ (m_d-1)\rho^{m_d-2} & \dots & \dots & 1 & 0 \end{pmatrix} + \frac{2\rho\Omega_d(\rho)}{(1-\rho^2)^2}.$$

The REML estimator of β is calculated by applying the formula

$$\widehat{\beta}_{REML} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators of θ and β are

$$\hat{\theta} \sim N_2(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_i are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2},\ a=1,2, \quad \hat{\beta}_i \pm z_{\alpha/2}q_{ii}^{1/2},\ i=1,\dots,p,$$

where $\hat{\theta}=\theta^{\kappa}$, $\mathbf{F}^{-1}(\theta^{\kappa})=(\nu_{ab})_{a,b=1,2}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^{\kappa})\mathbf{X})^{-1}=(q_{ij})_{i,j=1,\dots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantile of the standard normal distribution $N(0,1)$. Observed $\hat{\beta}_i=\beta_0$, the p-value for testing the hypothesis $H_0:\beta_i=0$ is

$$p = 2P_{H_0}(\hat{\beta}_i > |\beta_0|) = 2P(N(0,1) > |\beta_0|/\sqrt{q_{ii}}).$$

We are interested in predicting $\mu_{dt}=\mathbf{x}_{dt}\beta+u_{dt}$ with the EBLUP $\widehat{\mu}_{dt}=\mathbf{x}_{dt}\widehat{\beta}+\widehat{u}_{dt}$. If we do not take into account the error, e_{dt}, this is equivalent to predict $y_{dt}=\mathbf{a}'\mathbf{y}$, where $\mathbf{a}=\underset{1\le\ell\le D}{\mathrm{col}}(\underset{1\le k\le m_\ell}{\mathrm{col}}(\delta_{d\ell}\delta_{tk}))$ is a vector having one 1 in the position $t+\sum_{\ell=1}^{d-1}m_\ell$ and 0's in the remaining cells. To estimate $\overline{Y}_{dt}$ we use $\widehat{\overline{Y}}_{dt}^{eblup}=\widehat{\mu}_{dt}$. The mean squared error of $\widehat{\overline{Y}}_{dt}^{eblup}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\theta)+g_2(\theta)+g_3(\theta),$$

and the estimator of $MSE(\widehat{\overline{Y}}_{dt}^{eblup})$ is

$$mse(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\hat{\theta})+g_2(\hat{\theta})+2g_3(\hat{\theta}),$$

where $\theta=(\sigma_u^2,\rho)$, $\mathbf{a}_d=\underset{1\le k\le m_d}{\mathrm{col}}(\delta_{tk})$. The expressions for g_1-g_3 are

$$\begin{aligned}
g_1(\theta) &= \sigma_u^2\mathbf{a}_d'\Omega_d\mathbf{a}_d-\sigma_u^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d,\\
g_2(\theta) &= \left[\mathbf{a}_d'\mathbf{X}_d-\sigma_u^2\mathbf{a}_d'\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d+\sigma_u^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q}\\
&\cdot \left[\mathbf{X}_d'\mathbf{a}_d-\sigma_u^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{a}_d+\sigma_u^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right],\\
g_3(\theta) &\approx \mathrm{tr}\left\{\begin{pmatrix} q_{11} & q_{12}\\ q_{21} & q_{22}\end{pmatrix}\begin{pmatrix} F_{11} & F_{12}\\ F_{21} & F_{22}\end{pmatrix}^{-1}\right\},
\end{aligned}$$

where F_{ab} is the element of the REML Fisher information matrix.

$$\begin{aligned}
q_{11} &= \mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d-2\sigma_u^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d+\sigma_u^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d,\\
q_{12} &= \sigma_u^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d-\sigma_u^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d-\sigma_u^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\\
&+ \sigma_u^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d,\\
q_{22} &= \sigma_u^4\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d-2\sigma_u^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d+\sigma_u^8\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d.
\end{aligned}$$

3.2.2 The Software: description of R functions

This section describes the R functions that have been implemented for fitting the area-level model with time correlated effects (4.2). A brief descriptions of programmed R functions is given in the next subsections and the related codes are listed in Appendix B. The function can be used for calculating the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

REMLarea.autocorr

Function **REMLarea.autocorr** calculates the estimate of σ_u^2, the correlation coefficient ρ and the Fisher information matrix F for the Restricted Maximum Likelihood (REML) method. The function is

$$REMLarea.autocorr \ <- \ function \ (X, ydt, D, md, sigma2edt, sigma.0, MAXITER = 500).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ_{ed}^2, with size M.

sigma.0: Henderson 3 estimate of σ_u^2 obtained as output of **H3area** (see Section 3.1.2). It is used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations for the Fisher-scoring algorithm. Default value is 500.

The function returns a list of three elements. First element **theta.f** is the vector containing the REML estimates of σ_u^2 and ρ, second element **Fsig** is the estimated Fisher information matrix F and third element **Q** is the inverse matrix appearing in the expression of $\hat{\beta}$.

BETA.U.area.autocorr

Function **BETA.U.area.autocorr** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area.autocorr \ <- \ function \ (X,\ ydt,\ D,\ md,\ sigma2edt,\ sigmau,\ rho).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the variable of interest for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmau: estimated value of σ^2_u, calculated by the function **REMLarea.autocorr**.

rho: estimated value of ρ, calculated by the function **REMLarea.autocorr**.

The function returns an array with the following elements:

beta: vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

u: vector containing the predicted random effects $\widehat{\mathbf{u}}$, with size M.

mse.area.autocorr

Function **mse.area.autocorr** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.area.autocorr \ <- \ function\ (X,\ D,\ md,\ sigma2edt,\ sigmau,\ rho,\ Fsig).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmau: estimated value of σ^2_u, calculated by the function **REMLarea.autocorr**.

rho: estimated value of ρ, calculated by the function **REMLarea.autocorr**.

Fsig: estimated Fisher amount of information, calculated by the function **REMLarea.autocorr**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}^{eblup}_{dt})$, with size M.

Interval.autocorr

Function **Intervalo.autocorr** calculates the asymptotic confidence intervals for σ_u^2 and β_i. The function is

$$Interval.autocorr \ <- \ function\ (fit,\ conf = 0.95).$$

The arguments are:

fit: returned object, obtained by applying the function **REMLarea.autocorr**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ_u^2 and β_i respectively.

3.3 Examples of usage of R functions

This section demonstrates how the R routines described in chapters 3.1 and 3.2 can be applied to produce EBLUP estimates with their corresponding mean squared errors.

3.3.1 Example data set

Table 3.3.1.1 presents the data sets used in the example. There are 10 domains (areas), 3 time periods, 2 independent variables $X1$ and $X2$ for each domain and time period. Dependent variables is labeled by Y and the error variances by Var. There are are 30 observations. The file *dataExample.txt* contains the data, which should be sorted by domains and time periods.

Domain	*Time*	*ones*	$X1$	$X2$	Y	Var
11	1	1	0.414586518	0.188617372	0.07099622	0.000675053
11	2	1	0.410214326	0.196059524	0.072288837	0.000751033
11	3	1	0.430158306	0.203894195	0.082660682	0.001154806
12	1	1	0.416790574	0.191078693	0.146031284	0.001563715
12	2	1	0.39922138	0.186915951	0.082499785	0.000930011
12	3	1	0.430940532	0.187417706	0.078653447	0.001043772
21	1	1	0.419196102	0.148285289	0.301060532	0.001080062
21	2	1	0.402663325	0.148051767	0.241819175	0.000959909
21	3	1	0.399746159	0.144933539	0.237411987	0.001252906
22	1	1	0.385341527	0.150625093	0.305820376	0.001155839
22	2	1	0.381030859	0.154456747	0.263558025	0.001045748
22	3	1	0.371354065	0.151396774	0.285240927	0.00138958
31	1	1	0.405546324	0.163096412	0.179595403	0.000279276
31	2	1	0.418451155	0.160096774	0.171986758	0.000268098
31	3	1	0.422061454	0.152057299	0.160000387	0.000288642
32	1	1	0.38297778	0.172748583	0.179514009	0.000248195
32	2	1	0.392681888	0.163776148	0.183061881	0.000272611
32	3	1	0.401865322	0.159065292	0.188520565	0.000326651

Table 3.3.1.1. Data set *dataExample*.

Domain	*Time*	*ones*	$X1$	$X2$	Y	*Var*
41	1	1	0.449063901	0.127355842	0.353368205	0.001370052
41	2	1	0.452458366	0.123193508	0.249208975	0.001059773
41	3	1	0.45310808	0.129271701	0.317849199	0.001248913
42	1	1	0.438922319	0.131555109	0.335358187	0.001132905
42	2	1	0.426280132	0.140372377	0.322904936	0.001211021
42	3	1	0.441941803	0.12645344	0.353882106	0.001397947
51	1	1	0.375257147	0.168546499	0.208500731	0.002511227
51	2	1	0.384939766	0.152016046	0.329334406	0.00405833
51	3	1	0.389536232	0.170682902	0.335023497	0.00482337
52	1	1	0.350720286	0.151162941	0.204206531	0.002439803
52	2	1	0.373013185	0.133152685	0.389675949	0.004361985
52	3	1	0.381826881	0.145456107	0.452922541	0.005475034

Table 3.3.1.1. Data set *dataExample* (continuation).

3.3.2 Example of R code

An R code for reading the data file and applying the above described functions is needed. The file *Example.R* contains this code and, for this example, is located in the folder *C:/Areatimemodel*. It is important to take care on where to put this file and all the function *R* files. Note that under *Windows system*, the folder of the file is set by default installation. Otherwise the user can type the complete path. But under *Linux* the folder is the same than the one used to execute R.

The R file *Example.R* contains a program with the instructions for fitting the area level model to data in file *dataExample.txt*. First step is to open the *R* files containing all the above described R functions, i.e. *H3.R, REMLindep.R, REMLautocorr.R, EstimationBETAindep.R, EstimationBETAautocorr.R, EstimationMSEindep.R, EstimationMSEautocorr.R, ICindep.R, ICautocorr.R* and *pvalue*. Second step is to read the data file *dataExample.txt*. Third step is to run the application. The program creates several *txt* files in folder *C:/Areatimemodel*. The new files contain the output of the program in what follows the code in *Example.R* is listed.

```
####################################################################
###
###                       Area-level time models
###                           SAMPLE project
###
### Author: Agustin Perez Martin
### File name: Example.R
### Updated: November 25th, 2009
###
####################################################################

### Establishing the folder where data and routine files are located.
setwd("C:/Areatimemodel/")
```

```
### Call functions: H3area, REMLarea.indep, BETA.U.area.indep,
###                 mse.area.indep, Interval.indep,
###                 REMLarea.autocorr, BETA.U.area.autocorr,
###                 mse.area.autocorr, Interval.autocorr and p-value
source("H3.R")
source("REMLindep.R")
source("EstimationBETAindep.R")
source("EstimationMSEindep.R")
source("ICindep.R")
source("REMLautocorr.R")
source("EstimationBETAautocorr.R")
source("EstimationMSEautocorr.R")
source("ICautocorr.R")
source("pvalue.R")


### Reading data
data <- read.table(file = "dataExample.txt", header = T)

X <- as.matrix(data[,3:(ncol(data)-2)])
ydt <- data[,ncol(data)-1]
D <- length(unique(data[,1]))
md <- rep(length(unique(data[,2])), D)
sigma2edt <- data[,ncol(data)]


### Calculating H3 and REML variance estimates
sigma.0 <- H3area(X, ydt, D, md, sigma2edt)

### Model with independent time effects
### Arguments: (X, ydt, D, md, sigma2edt, sigma.0, MAXITER = 500)
fit0 <- REMLarea.indep(X, ydt, D, md, sigma2edt, sigma.0=sigma.0)
sigmau.hat <- fit0[[1]]
if(sigmau.hat<0) {
    write.table(data.frame(sigmau.hat),
    file="VAR.NEGATIVE.indep.txt", append=TRUE)
    sigmau.hat <- 0
}
### Arguments: (X, ydt, D, md, sigma2edt, sigmau)
```

```
beta.u.hat <- BETA.U.area.indep(X, ydt, D, md, sigma2edt, sigmau.hat)
beta.hat0 <- beta.u.hat[1:ncol(X),]
Int0 <- Interval.indep(fit0, 0.90)
                beta.hat0-Int0[[2]]
                beta.hat0+Int0[[2]]
(pvalue0 <- pvalue(beta.hat0, fit0))
                pvalue0>0.1
udt.hat0 <- beta.u.hat[-(1:ncol(X)),]

### EBLUP of the population parameter
mudt.hat.0 <- as.vector(X%*%beta.hat0 + udt.hat0)
sqrt.mse.0 <- sqrt(mse.area.indep(X, D,md,sigma2edt,sigmau.hat,
fit0[[2]]))
residuals.0 <- ydt-mudt.hat.0
Henderson3 <- sigma.0

######################################################################

### Reading data
data <- read.table(file = "dataExample.txt", header = T)

X <- as.matrix(data[,3:(ncol(data)-2)])
ydt <- data[,ncol(data)-1]
D <- length(unique(data[,1]))
md <- rep(length(unique(data[,2])), D)
sigma2edt <- data[,ncol(data)]


### Calculating H3 and REML variance estimates
sigma.0 <- H3area(X, ydt, D, md, sigma2edt)


### Model with time correlated effects
### Arguments: (X, ydt, D, md, sigma2edt, sigma.0, MAXITER = 500)
fit1 <- REMLarea.autocorr(X, ydt, D, md, sigma2edt, sigma.0=sigma.0)
sigmau.hat <- fit1[[1]][1]
rho.hat <- fit1[[1]][2]
if(sigmau.hat<0) {
    write.table(data.frame(sigmau.hat),
    file="VAR.NEGATIVE.autocorr.txt", append=TRUE, col.names=FALSE)
    sigmau.hat <- 0
}
if(rho.hat < -1 || rho.hat > 1 ) {
    write.table(data.frame(rho.hat),
    file="COEFFICIENT OF CORRELATION OUT OF RANGE.txt",
    append=TRUE, col.names=FALSE)
    if(rho.hat < -1) {
        rho.hat <- -0.8
    }
    else rho.hat <- 0.8
}

### Arguments: (X, ydt, D, md, sigma2edt, sigmau)
beta.u.hat <- BETA.U.area.autocorr(X, ydt, D, md, sigma2edt,
 sigmau.hat, rho.hat)
beta.hat1 <- beta.u.hat[1:ncol(X),]
Int1 <- Interval.autocorr(fit1, 0.90)
                beta.hat1-Int1[[3]]
                beta.hat1+Int1[[3]]
(pvalue1 <- pvalue(beta.hat1, fit1))
                pvalue1>0.1
udt.hat1 <- beta.u.hat[-(1:ncol(X)),]

### EBLUP of the population parameter
mudt.hat.1 <- as.vector(X%*%beta.hat1 + udt.hat1)
sqrt.mse.1 <- sqrt(mse.area.autocorr(X, D, md, sigma2edt, sigmau.hat,
 rho.hat, fit1[[2]]))
residuals.1 <- ydt-mudt.hat.1



### Create .txt files in the folder that contains for
### the resulting output
write.table(data.frame(data[,1:2], Direct=ydt, EBLUP.0=mudt.hat.0,
EBLUP.1 = mudt.hat.1, sqrt.mse.direct=sqrt(sigma2edt),sqrt.mse.0,
sqrt.mse.1),
file="EBLUP_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(names(data)[3:(ncol(data)-2)], beta.hat0,
Std.error.beta.hat0=Int0[[2]], beta.hat1,
Std.error.beta.hat1=Int1[[3]]),
file="beta_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(data[,1:2], udt.hat0, udt.hat1),
file="u_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(data[,1:2], residuals.0, residuals.1),
file="res_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(names(data)[3:(ncol(data)-2)],
beta.hat0, pvalue0,
beta.hat1, pvalue1),
file="pvalue_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(Henderson3),
file="H3_Example.txt",
row.names=FALSE, sep="\t")

rm(list=ls(all=TRUE))
```

3.3.3 Outputs

Outputs of model 3.1 are labeled with "0" and outputs of model 4.2 with "1". The resulting outputs appear in the files *EBLUP Example.txt, beta Example.txt, u Example.txt, res Example.txt* and *H3 Example.txt* they are:

```
"EBLUP Example.txt" output:
"Domain" "Time"     "Direct"      "EBLUP.0"           "EBLUP.1"
11        1   0.07099622 0.0810870361440193 0.0885760701808574
11        2   0.072288837 0.0714371458327979 0.0762137925944318
11        3   0.082660682 0.052765034270926 0.0538737359735373
12        1   0.146031284 0.109117506760301 0.102837121921340
12        2   0.082499785 0.0971554778459163 0.106764441585287
12        3   0.078653447 0.0843765271001174 0.0885295192441997
21         1   0.301060532 0.274500665705542 0.266447830656762
21        2   0.241819175 0.251428302623275 0.255877074238007
```

```
21         3   0.237411987 0.258537796783850 0.260107687775899
22         1   0.305820376 0.283098865615287 0.28319090899208
22         2   0.263558025 0.257285589812057 0.265416656672475
22          3   0.285240927 0.276171257611288 0.278843232596284
31          1   0.179595403 0.184281701245608 0.178946303365822
31         2   0.171986758 0.178743608216205 0.170751458760549
31         3   0.160000387 0.176197813417279 0.176946498998694
32         1   0.179514009 0.179403322361097 0.175353458075361
32         2   0.183061881 0.18819267495888 0.187602783590969
32         3   0.188520565 0.196045641946532 0.195173080779298
41         1   0.353368205 0.328880532968824 0.313891050763258
41         2   0.249208975 0.287780626070448 0.304526684502343
41         3   0.317849199 0.308344519206265 0.297811346284671
42         1   0.335358187 0.317365511683804 0.327043967075657
42         2   0.322904936 0.297114134229847 0.305165260878979
42         3   0.353882106 0.333682212237881 0.343986387001928
51         1   0.208500731 0.202743936123168 0.217921552265458
51         2   0.329334406 0.272296339000462 0.283812864935672
51         3   0.335023497 0.209031824984797 0.218745541751390
52         1   0.204206531 0.260791713818310 0.26913502405639
52         2   0.389675949 0.348727254435842 0.342316109985649
52         3   0.452922541 0.311715605292666 0.304711309262514
```

```
 "sqrt.mse.direct"     "sqrt.mse.0"      "sqrt.mse.1"
0.0259817820789876 0.0222492368527262  0.02529723350447
0.0274049812990266 0.0233016417373966  0.0252815935554673
0.0339824366401234 0.0276158146322022  0.0291731520035163
0.0395438364350249 0.0281434204926076  0.0291793759628180
0.0304960817155254 0.0243856072652821  0.0261038788228251
0.0323074604387284 0.0257724969473051  0.028042744686898
0.0328642967367324 0.0248947518737948  0.0270197642584938
0.0309823982286717 0.0241802664409422  0.0257100094090509
0.0353964122475711 0.0259738102143437  0.0276420314540668
0.0339976322704979 0.0257435562716122  0.0273014024355766
0.0323380271507091 0.0251764189235333  0.0260644239979016
0.0372770707003649 0.0275971919369438  0.0287455620155905
0.0167115528901416 0.0155811019725347  0.0218917272635606
0.016373698421554 0.0153266667694242  0.0222328564459353
0.0169894673253754 0.0158295168167019  0.0219648135137690
0.0157542057876619 0.0149352292491623  0.0217904387722267
```

```
0.0165109357699677 0.0154673378691320  0.0222683939673919
0.0180734888718255 0.0166336789434665  0.0221892735173991
0.0370142134861731 0.0278643356937977  0.0285147125903474
0.0325541548807522 0.0263980146010989  0.0268870621864523
0.0353399632144687 0.0273638999999272  0.0282672691197796
0.0336586541620428 0.0260350821153883  0.0277529991368032
0.0347997270104235 0.0258673070510146  0.0262767739803480
0.0373891294362412 0.0277340978267927  0.0290881039736661
0.0501121442367017 0.0301782165013877  0.0317869641656460
0.0637050233498113 0.0314629079368649  0.0317167152903283
0.0694504859594229 0.0314451804811292  0.0329894046030183
0.04939436202645 0.0328545769793037  0.0344369451171986
0.0660453253455534 0.0342088762681202  0.0338925210978683
0.0739934726850957 0.032842871228608   0.0336642969125901


"beta Example.txt" output:

"names.data..3..ncol.data....2.."
           "beta.hat0"   "Std.error.beta.hat0"
"ones" 1.10551525071017 0.289971666456405
"X1" -0.669666738329566 0.595923529114958
"X2" -3.87902568140871 0.666469527559167

          "beta.hat1"     "Std.error.beta.hat1"
"ones"  0.996423126478698 0.344512091727891
"X1"    -0.488073608903706 0.698080746499988
"X2"    -3.63831553531472 0.843460606017086

"u Example.txt" output:
"Domain" "Time"     "udt.hat0"             "udt.hat1"
11         1 -0.0151417833538597    -0.0192488034769234
11         2 0.00114871351187552    -0.0066681353538353
11         3 0.0262233120003018    0.0092309435331532
12          1 0.0239121976242283    0.00504305226140131
12         2 -0.0159627192542945    -0.0147500568838680
12         3 -0.00555408840689845   -0.0156781552255283
21         1 0.0249095455482918    0.0141319291469034
21         2 -0.0101401062182701    -0.00535766612788023
21         3 -0.0170798275905686    -0.0138959417024322
```

```
22         1 0.0199126224457154    0.0228644281268613
22         2 0.00607571993117639    0.0169270187938618
22         3 0.00661144680352257    0.0144974596439298
31         1 -0.0169974947363216    -0.0261441556468742
31         2 -0.0255293245181901    -0.0389541223494046
31         3 -0.0568427521514347    -0.0602471372373828
32         1 0.000451742320188551   -0.00563446795630265
32         2 -0.0190646925190417    -0.0212933730780450
32         3 -0.0233354166433626    -0.0263804646846378
41         1 0.0181050219383043    4.90153569384333e-06
41         2 -0.0368675251357491    -0.0228466003977432
41         3 0.00770892659885395    -0.00713044634013867
42         1 0.0160875850499344    0.0234862376843940
42         2 0.0215725644741984    0.0275172167976236
42         3 0.0146368285366828    0.0233409064848382
51         1 0.00232211321187295    0.0178768815245654
51         2 0.0142365922610651    0.0283510210380592
51         3 0.0264593924546214    0.0334430237828832
52         1 -0.0234928966839586    -0.00613231011438557
52         2 0.00950921134206837    0.0124023572829343
52         3 0.0261250911590495    0.0238630203744320

"res Example.txt" output:
"Domain" "Time"     "residuals.0"          "residuals.1"
11         1 -0.0100908161440193   -0.0175798501808574
11         2 0.000851691167202129  -0.00392495559443180
11         3 0.029895647729074   0.0287869460264627
12         1 0.0369137772396989   0.0431941620786601
12         2 -0.0146556928459163   -0.0242646565852869
12         3 -0.0057230801001174   -0.0098760722441997
21         1 0.0265598662944575   0.0346127013432377
21         2 -0.00960912762327454  -0.0140578992380065
21         3 -0.0211258097838496   -0.0226957007758992
22         1 0.0227215103847131   0.0226294670079202
22         2 0.00627243518794296   -0.00185863167247524
22         3 0.0090696693887124   0.00639769440371557
31         1 -0.00468629824560846  0.000649099634178196
31         2 -0.00675685021620515  0.00123529923945057
31         3 -0.0161974264172793   -0.0169461119986940
32         1 0.000110686638902596  0.00416055092463880
```

```
32         2 -0.00513079395887975  -0.00454090259096904
32         3 -0.00752507694653232  -0.00665251577929787
41         1 0.0244876720311756   0.0394771542367417
41         2 -0.0385716510704477   -0.0553177095023425
41         3 0.00950467979373487   0.0200378527153294
42         1 0.0179926753161957   0.00831421992434317
42         2 0.0257908017701525   0.0177396751210214
42         3 0.020199893762119   0.00989571899807246
51         1 0.00575679487683178   -0.00942082126545776
51         2 0.0570380669995378   0.0455215410643282
51         3 0.125991672015203   0.116277955248610
52         1 -0.0565851828183103   -0.06492849305639
52         2 0.0409486945641577   0.0473598390143509
52         3 0.141206935707334   0.148211231737486


"pvalue Example.txt" output:
"names.data..3..ncol.data....2.."
           "beta.hat0"           "pvalue0"
"ones" 1.10551525071017 3.58748940379529e-10
"X1" -0.669666738329566 0.0645448111231833
"X2" -3.87902568140871 1.03371377940706e-21

           "beta.hat1"           "pvalor1"
"ones" 0.996423126478698 1.96135137530811e-06
"X1" -0.488073608903706 0.250133966757249
"X2" -3.63831553531472 1.29192448422908e-12

"H3 Example.txt" output:
0.00101519275290970
```

Chapter 4

Area-level partitioned time models

4.1 Partitioned Fay-Herriot model 1

4.1.1 The methodology

Consider the model (model 1)

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{dt} + e_{dt}, \quad d = 1,\dots,D = D_A + D_B, \quad t = 1,\dots,m_d, \tag{4.1}$$

where y_{dt} is a direct estimator of the indicator of interest for domain d and time instant t, and $\mathbf{x}_{dt}$ is a vector containing the aggregated (population) values of p auxiliary variables. The index d is used for domains and the index t for time instants. We assume that the random effects u_{dt}'s are all independent and i.i.d. $N(0,\sigma_A^2)$ if $d \le D_A$ and i.i.d. $N(0,\sigma_B^2)$ if $d > D_A$. We also assume that the errors e_{dt}'s are independent $N(0,\sigma_{dt}^2)$ with known σ_{dt}^2's. Finally, we assume that the u_{dt}'s and the e_{dt}'s are mutually independent. In matrix notation the model is

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \mathbf{e},$$

where vectors $\mathbf{y}$, $\mathbf{u}$ and $\mathbf{e}$ can be decomposed in the form $\mathbf{v} = (\mathbf{v}_A', \mathbf{v}_B')'$, with $\mathbf{v}_A = \underset{d\le D_A}{\text{col}}(\mathbf{v}_d)$, $\mathbf{v}_B = \underset{d>D_A}{\text{col}}(\mathbf{v}_d)$ and $\mathbf{v}_d = \underset{1\le t\le m_d}{\text{col}}(v_{dt})$, matrix $\mathbf{X}$ can be similarly decomposed in the form $\mathbf{X} = (\mathbf{X}_A', \mathbf{X}_B')'$, with $\mathbf{X}_A = \underset{d\le D_A}{\text{col}}(\mathbf{X}_d)$, $\mathbf{X}_B = \underset{d>D_A}{\text{col}}(\mathbf{X}_d)$, $\mathbf{X}_d = \underset{1\le t\le m_d}{\text{col}}(\mathbf{x}_{dt})$, $\mathbf{x}_{dt} = \underset{1\le j\le p}{\text{col}'}(x_{dtj})$. The rest of the terms are $\beta = \beta_{p\times 1}$, $\mathbf{Z} = \mathbf{I}_M$, $M = M_A + M_B$, $M_A = \sum_{d\le D_A} m_d$ and $M_B = \sum_{d>D_A} m_d$, where $\mathbf{I}_M$ denotes the identity $M\times M$ matrix. In this notation, $\mathbf{u} \sim N(\mathbf{0}, \mathbf{V}_u)$ and $\mathbf{e} \sim N(\mathbf{0}, \mathbf{V}_e)$ are independent with covariance matrices

$$\mathbf{V}_u = \text{var}(\mathbf{u}) = \text{diag}(\sigma_A^2\mathbf{I}_{M_A}, \sigma_B^2\mathbf{I}_{M_B}), \quad \mathbf{V}_e = \text{var}(\mathbf{e}) = \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_{ed}), \quad \mathbf{V}_{ed} = \underset{1\le t\le m_d}{\text{diag}}(\sigma_{dt}^2).$$

The covariance matrix of vector $\mathbf{y}$ is $\mathbf{V} = \text{var}(\mathbf{y}) = \text{diag}(\mathbf{V}_A, \mathbf{V}_B)$, where $\mathbf{V}_A = \underset{d\le D_A}{\text{diag}}(\mathbf{V}_d)$, $\mathbf{V}_B = \underset{d>D_A}{\text{diag}}(\mathbf{V}_d)$, $\mathbf{V}_d = \sigma_A^2\mathbf{I}_{m_d} + \mathbf{V}_{ed}$ if $d \le D_A$ and $\mathbf{V}_d = \sigma_B^2\mathbf{I}_{m_d} + \mathbf{V}_{ed}$ if $d > D_A$.

If $\sigma_A^2 > 0$ and $\sigma_B^2 > 0$ are known, the best linear unbiased estimator (BLUE) of β is

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$$

and the best linear unbiased predictor (BLUP) of $\mathbf{u}$ is

$$\widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\beta}) = \operatorname{diag}\left(\sigma_A^2\mathbf{I}_{M_A},\sigma_B^2\mathbf{I}_{M_B}\right)\underset{1\le d\le D}{\operatorname{col}}(\mathbf{V}_d^{-1})(\mathbf{y}-\mathbf{X}\widehat{\beta}),$$

so that

$$\widehat{\mathbf{u}}_d = \begin{cases} \sigma_A^2\mathbf{V}_d^{-1}(\mathbf{y}_d-\mathbf{X}_d\widehat{\beta}), & d=1,\dots,D_A, \\ \sigma_B^2\mathbf{V}_d^{-1}(\mathbf{y}_d-\mathbf{X}_d\widehat{\beta}), & d=D_A+1,\dots,D, \end{cases}$$

or equivalently

$$\hat{u}_{dt} = \left[\frac{\sigma_A^2}{\sigma_A^2+\sigma_{dt}^2}I_{\{d\le D_A\}}(d)+\frac{\sigma_B^2}{\sigma_B^2+\sigma_{dt}^2}I_{\{d>D_A\}}(d)\right](y_{dt}-\mathbf{x}_{dt}\hat{\beta}),\ d=1,\dots,D,t=1,\dots,m_d.$$

By substituting σ_A^2 and σ_B^2 by suitable estimators, we obtain the empirical BLUE (EBLUE) of β and the empirical BLUP (EBLUP) of $\mathbf{u}$. They will be denoted in the same manner as their corresponding non-empirical versions, i.e. $\widehat{\beta}$ and $\widehat{\mathbf{u}}$.

The loglikelihood of the restricted (residual) maximum likelihood method is

$$\begin{aligned} l_{reml} &= l_{reml}(\sigma_A^2,\sigma_B^2) = -\frac{M-p}{2}\log 2\pi+\frac{1}{2}\log|\mathbf{X}'\mathbf{X}|-\frac{1}{2}\log|\mathbf{V}_A|-\frac{1}{2}\log|\mathbf{V}_B| \\ &- \frac{1}{2}\log|\mathbf{X}_A'\mathbf{V}_A^{-1}\mathbf{X}_A+\mathbf{X}_B'\mathbf{V}_B^{-1}\mathbf{X}_B|-\frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{y}, \end{aligned}$$

where

$$\mathbf{P}=\mathbf{V}^{-1}-\mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}, \quad \mathbf{PVP}=\mathbf{P}, \quad \mathbf{PX}=\mathbf{0}.$$

Let $\theta=(\theta_1,\theta_2)=(\sigma_A^2,\sigma_B^2)$, then

$$\mathbf{V}_1=\frac{\partial\mathbf{V}}{\partial\sigma_A^2}=\operatorname{diag}\big(\mathbf{I}_{M_A},\underset{d>D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d})\big), \quad \mathbf{V}_2=\frac{\partial\mathbf{V}}{\partial\sigma_B^2}=\operatorname{diag}\big(\underset{d\le D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d}),\mathbf{I}_{M_B}\big)$$

and

$$\mathbf{P}_a=\frac{\partial\mathbf{P}}{\partial\theta_a}=-\mathbf{P}\frac{\partial\mathbf{V}}{\partial\theta_a}\mathbf{P}=-\mathbf{P}\mathbf{V}_a\mathbf{P}, \quad a=1,2.$$

By taking partial derivatives of l_{reml} with respect to θ_a, we get the scores

$$S_a=\frac{\partial l_{reml}}{\partial\theta_a}=-\frac{1}{2}\operatorname{tr}(\mathbf{P}\mathbf{V}_a)+\frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{y}, \quad a=1,2.$$

By taking again partial derivatives with respect to θ_a and θ_b, taking expectations and changing the sign, we get the Fisher information matrix components

$$F_{ab}=\frac{1}{2}\operatorname{tr}(\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{V}_b), \quad a,b=1,2.$$

To calculate the REML estimate of θ we apply the Fisher-scoring algorithm with the updating fórmula

$$\theta^{k+1}=\theta^k+\mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k),$$

where **S** and **F** are the column vector of scores and the Fisher information matrix respectively. As seeds we use $\sigma_A^{2(0)} = \sigma_B^{2(0)} = \widehat{\sigma}_{uH}^2$, where $\widehat{\sigma}_{uH}^2$ is the Henderson 3 estimator under model with $\sigma_A^2 = \sigma_B^2$ as follow:

$$\widehat{\sigma}_{uH}^2 = \frac{\mathbf{y}'\mathbf{P}_2\mathbf{y} - (M-p)}{\text{tr}\{\mathbf{P}_2\}},$$

where $\mathbf{Q}_2 = \sum_{d=1}^D \left(\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)^{-1}$ and

$$\begin{aligned}
\mathbf{P}_2 &= \underset{1\le d\le D}{\text{diag}}\,(\mathbf{V}_{ed}^{-1}) - \underset{1\le d\le D}{\text{col}}(\mathbf{V}_{ed}^{-1}\mathbf{X}_d)\mathbf{Q}_2\underset{1\le d\le D}{\text{col}'}(\mathbf{X}_d'\mathbf{V}_{ed}^{-1}),\\
\text{tr}\{\mathbf{P}_2\} &= \sum_{d=1}^D\sum_{t=1}^{m_d}\sigma_{dt}^{-2} - \sum_{d=1}^D \text{tr}\{\mathbf{X}_d'\mathbf{V}_{ed}^{-2}\mathbf{X}_d\mathbf{Q}_2\},\\
\mathbf{y}'\mathbf{P}_2\mathbf{y} &= \sum_{d=1}^D\sum_{t=1}^{m_d}\sigma_{dt}^{-2}y_{dt}^2 - \left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)\mathbf{Q}_2\left(\sum_{d=1}^D \mathbf{y}_d'\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right)'.
\end{aligned}$$

The REML (EBLUE-REML) estimator of β is

$$\widehat{\beta} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators of θ and β are

$$\hat{\theta} \sim N_2(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_j are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2},\ a=1,2, \quad \hat{\beta}_j \pm z_{\alpha/2}q_{jj}^{1/2},\ j=1,\dots,p,$$

where $\hat{\theta} = \theta^\kappa$, $\mathbf{F}^{-1}(\theta^\kappa) = (\nu_{ab})_{a,b=1,2}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^\kappa)\mathbf{X})^{-1} = (q_{ij})_{i,j=1,\dots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantile of the standard normal distribution $N(0,1)$. If β_{j0} is the observed value of $\hat{\beta}_j$, then the p-value for testing the hypothesis $H_0: \beta_j = 0$ is

$$p = 2P_{H_0}(\hat{\beta}_j > |\beta_{j0}|) = 2P(N(0,1) > |\beta_{j0}|/\sqrt{q_{jj}}).$$

We are interested in predicting the value of $\mu_{dt} = \mathbf{x}_{dt}\beta + u_{dt}$ by using the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. If we do not take into account the error, e_{dt}, this is equivalent to predict $y_{dt} = \mathbf{a}'\mathbf{y}$, where $\mathbf{a} = \underset{1\le \ell\le D}{\text{col}}(\underset{1\le k\le m_\ell}{\text{col}}(\delta_{d\ell}\delta_{tk}))$ is a vector having one 1 in the position $t + \sum_{\ell=1}^{d-1} m_\ell$ and 0's in the remaining cells. To estimate $\overline{Y}_{dt}$ we use $\widehat{\overline{Y}}_{dt}^{eblup} = \widehat{\mu}_{dt}$. The mean squared error of $\widehat{\overline{Y}}_{dt}^{eblup}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\theta) + g_2(\theta) + g_3(\theta),$$

and the estimator of $MSE(\widehat{\overline{Y}}_{dt}^{eblup})$ is

$$mse(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\hat{\theta}) + g_2(\hat{\theta}) + 2g_3(\hat{\theta}),$$

where $\theta = (\sigma_A^2, \sigma_B^2)$, $\mathbf{a}_d = \underset{1\leq k\leq m_d}{\text{col}}(\delta_{tk})$. The expressions for $g_1 - g_3$ are

$$
\begin{aligned}
g_1(\theta) &= \begin{cases} \sigma_A^2 - \sigma_A^4 \mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{a}_d = \frac{\sigma_A^2 \sigma_{dt}^2}{\sigma_A^2 + \sigma_{dt}^2} & \text{if } d \leq D_A \\ \sigma_B^2 - \sigma_B^4 \mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{a}_d = \frac{\sigma_B^2 \sigma_{dt}^2}{\sigma_B^2 + \sigma_{dt}^2} & \text{if } d > D_A \end{cases} \\
g_2(\theta) &= \left[\mathbf{a}_d'\mathbf{X}_d - \sigma_A^2 \mathbf{a}_d' \mathbf{V}_{ed}^{-1} \mathbf{X}_d + \sigma_A^4 \mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{V}_{ed}^{-1} \mathbf{X}_d\right] \mathbf{Q} \\
&\cdot \left[\mathbf{X}_d'\mathbf{a}_d - \sigma_A^2 \mathbf{X}_d' \mathbf{V}_{ed}^{-1} \mathbf{a}_d + \sigma_A^4 \mathbf{X}_d' \mathbf{V}_{ed}^{-1} \mathbf{V}_d^{-1} \mathbf{a}_d\right] \quad \text{if } d \leq D_A \\
&= \left[\mathbf{a}_d'\mathbf{X}_d - \sigma_B^2 \mathbf{a}_d' \mathbf{V}_{ed}^{-1} \mathbf{X}_d + \sigma_B^4 \mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{V}_{ed}^{-1} \mathbf{X}_d\right] \mathbf{Q} \\
&\cdot \left[\mathbf{X}_d'\mathbf{a}_d - \sigma_B^2 \mathbf{X}_d' \mathbf{V}_{ed}^{-1} \mathbf{a}_d + \sigma_B^4 \mathbf{X}_d' \mathbf{V}_{ed}^{-1} \mathbf{V}_d^{-1} \mathbf{a}_d\right] \quad \text{if } d > D_A \\
g_3(\theta) &= \begin{cases} q_{11} F_{11}^{-1}, & \text{if } d \leq D_A, \\ q_{22} F_{22}^{-1}, & \text{if } d > D_A, \end{cases}
\end{aligned}
$$

where F_{ab} is the REML Fisher amount of information appearing in the updating equation of the Fisher-scoring algorithm and

$$
\begin{aligned}
q_{11} &= \left[\mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{a}_d - 2\sigma_A^2 \mathbf{a}_d' \mathbf{V}_d^{-2} \mathbf{a}_d + \sigma_A^4 \mathbf{a}_d' \mathbf{V}_d^{-3} \mathbf{a}_d\right] I_{\{d\leq D_A\}}(d) = \frac{\sigma_{dt}^4}{(\sigma_A^2 + \sigma_{dt}^2)^3} I_{\{d\leq D_A\}}(d) \\
q_{22} &= \left[\mathbf{a}_d' \mathbf{V}_d^{-1} \mathbf{a}_d - 2\sigma_B^2 \mathbf{a}_d' \mathbf{V}_d^{-2} \mathbf{a}_d + \sigma_B^4 \mathbf{a}_d' \mathbf{V}_d^{-3} \mathbf{a}_d\right] I_{\{d> D_A\}}(d) = \frac{\sigma_{dt}^4}{(\sigma_B^2 + \sigma_{dt}^2)^3} I_{\{d> D_A\}}(d).
\end{aligned}
$$

4.1.2 The Software: description of R functions for model 1

This section describes the R functions that have been implemented for fitting the partitioned area-level model with independent time effects (4.1). An example of how to use these functions is given in the next section and the related codes are listed in Appendix 18.1.

The developed R software contains a series of functions that return, as final output, the EBLUP estimates of interest. We recall that R functions are objects with the form

$$name \ {<}{-} \ function \ (arg_1, arg_2, ...) \ \{expression\}.$$

R functions allows to define a dependent variable *name* as output of a given procedure, when inputs variables are *arguments*. The *expression* within curly brackets contains the needed calculations to obtain *name* from *arguments*. The function codes appearing in *expression* are listed in Appendix A 18.1.

A brief descriptions of programmed R functions is given in the next subsections. The functions can be used for calculating the Henderson 3 and the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

H3area

Function **H3area** calculates the unbiased Henderson 3 estimator of σ_u^2 and has the form

$$H3.area \ {<}{-} \ function \ (X, \ ydt, \ D, \ md, \ sigma2edt).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the known error variances σ^2_{ed}, with size M.

REMLarea

Function **REMLarea** calculates the estimate of σ^2_A, σ^2_B and the Fisher amount of information F for the Restricted Maximum Likelihood (REML) method. The function is

$$REMLarea <- function\ (X,\ ydt,\ D,\ Da,\ Db,\ md,\ sigma2edt,\ sigma.0,\ \textit{MAXITER} = 100).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigma.0: Henderson 3 estimate of σ^2_u obtained as output of **H3area**. It is used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 100.

The function returns a list of six elements. First two elements are **sigma.fa** and **sigma.fb** the REML estimates of σ^2_u, the third element **F** is the estimated Fisher information matrix F, the forth element **ITER** is the indicator of the maximum iteration when the algorithm stop the loop without convergence, the fifth element **Q** is the inverse matrix appearing in the expression of $\hat{\beta}$, and the last one element **Bad** is the sum of iterations of the algorithm which stopped without convergence.

BETA.U.area

Function **BETA.U.area** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area <- function\ (X,\ ydt,\ D,\ Da,\ Db,\ md,\ sigma2edt,\ sigmaua,\ sigmaub).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the variable of interest for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmaua: estimated value of σ^2_A, calculated by the function **REMLarea**.

sigmaub: estimated value of σ^2_B, calculated by the function **REMLarea**.

The function returns an array with the following elements:

beta: vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

u: vector containing the predicted random effects $\widehat{\mathbf{u}}$, with size M.

mse.area

Function **mse.area** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.area <- function\ (X,\ D,\ Da,\ Db,\ md,\ sigma2edt,\ sigmaua,\ sigmaub,\ F11,\ F22).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmaua: estimated value of σ^2_A, calculated by the function **REMLarea**.

sigmaub: estimated value of σ^2_B, calculated by the function **REMLarea**.

F11: element of Fisher information matrix F11, calculated by the function **REMLarea**.

F22: element of Fisher information matrix F22, calculated by the function **REMLarea**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}^{eblup}_{dt})$, with size M.

Interval

Function **Interval** calculates, within the same file, the asymptotic confidence intervals for σ^2_A, σ^2_B and β_i and the asymptotic p-value of test statistics $\hat{\beta}_i$ for the null hypothesis $H_0 : \beta_i = 0$.

The first function is

$$Interval\ <-\ function\ (Fisher,\ conf = 0.95).$$

The arguments are:

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ^2_A, σ^2_B and β_i respectively.

The second function is

$$pvalueBeta\ <-\ function\ (beta0.hat,\ Fisher).$$

The arguments are:

beta0.hat: observed value of $\hat{\beta}_i$, calculated by the function **BETA.U.area**.

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea**.

This function returns the vector **2*p.beta** containing the asymptotic p-values for hypotheses H_0 : $\beta_i = 0, i = 1, \ldots, p$, with size p.

4.2 Partitioned Fay-Herriot model 2

4.2.1 The methodology

Let us consider the model (model 2)

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{dt} + e_{dt}, \quad d = 1,\dots,D = D_A + D_B, \quad t = 1,\dots,m_d, \tag{4.2}$$

where y_{dt} is a direct estimator of the indicator of interest for area d and time instant t, and $\mathbf{x}_{dt}$ is a vector containing the aggregated (population) values of p auxiliary variables. The index d is used for domains and the index t for time instants. We assume that the random vectors $(u_{d1},\dots,u_{dm_d})$, $d \le D_A$, follow i.i.d. first order auto-regressive processes with variance and auto-correlation parameters σ_A^2 and ρ respectively; in short, $(u_{d1},\dots,u_{dm_d}) \sim_{iid} AR1(\sigma_A^2,\rho)$, $d \le D_A$. We further assume that $(u_{d1},\dots,u_{dm_d}) \sim_{iid} AR1(\sigma_B^2,\rho)$, $d > D_A$, and that the errors e_{dt}'s are independent $N(0,\sigma_{dt}^2)$ with known σ_{dt}^2's. Finally we assume that the $(u_{d1},\dots,u_{dm_d})$'s and the e_{dt}'s are mutually independent.

In matrix notation the model is

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \mathbf{e},$$

where vectors $\mathbf{y}$, $\mathbf{u}$ and $\mathbf{e}$ can be decomposed in the form $\mathbf{v} = (\mathbf{v}_A', \mathbf{v}_B')'$, with $\mathbf{v}_A = \underset{d\le D_A}{\mathrm{col}}(\mathbf{v}_d)$, $\mathbf{v}_B = \underset{d>D_A}{\mathrm{col}}(\mathbf{v}_d)$ and $\mathbf{v}_d = \underset{1\le t\le m_d}{\mathrm{col}}(v_{dt})$, matrix $\mathbf{X}$ can be similarly decomposed in the form $\mathbf{X} = (\mathbf{X}_A', \mathbf{X}_B')'$, with $\mathbf{X}_A = \underset{d\le D_A}{\mathrm{col}}(\mathbf{X}_d)$, $\mathbf{X}_B = \underset{d>D_A}{\mathrm{col}}(\mathbf{X}_d)$ and $\mathbf{X}_d = \underset{1\le t\le m_d}{\mathrm{col}'}(\mathbf{x}_{dt})$, $\beta = \beta_{p\times 1}$, $\mathbf{Z} = \mathbf{I}_M$ and $M = \sum_{d=1}^{D} m_d$. In this notation, $\mathbf{u} \sim N(\mathbf{0}, \mathbf{V}_u)$ and $\mathbf{e} \sim N(\mathbf{0}, \mathbf{V}_e)$ are independent with covariance matrices

$$\mathbf{V}_u = \mathrm{var}(\mathbf{u}) = \mathrm{diag}(\sigma_A^2\Omega_A, \sigma_B^2\Omega_B), \quad \mathbf{V}_e = \mathrm{var}(\mathbf{e}) = \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{V}_{ed})$$

where $\Omega_A = \underset{d\le D_A}{\mathrm{diag}}(\Omega_d)$, $\Omega_B = \underset{d>D_A}{\mathrm{diag}}(\Omega_d)$, $\mathbf{V}_{ed} = \underset{1\le t\le m_d}{\mathrm{diag}}(\sigma_{dt}^2)$ and

$$\Omega_d = \Omega_d(\rho) = \frac{1}{1-\rho^2}\begin{pmatrix} 1 & \rho & \dots & \rho^{m_d-2} & \rho^{m_d-1} \\ \rho & 1 & \ddots & & \rho^{m_d-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \rho^{m_d-2} & & \ddots & 1 & \rho \\ \rho^{m_d-1} & \rho^{m_d-2} & \dots & \rho & 1 \end{pmatrix}_{m_d\times m_d}.$$

The covariance matrix of vector $\mathbf{y}$ is $\mathbf{V} = \mathrm{var}(\mathbf{y}) = \mathrm{diag}(\mathbf{V}_A, \mathbf{V}_B)$, where $\mathbf{V}_A = \underset{d\le D_A}{\mathrm{diag}}(\mathbf{V}_d)$, $\mathbf{V}_B = \underset{d>D_A}{\mathrm{diag}}(\mathbf{V}_d)$, $\mathbf{V}_d = \sigma_A^2\Omega_d + \mathbf{V}_{ed}$ if $d \le D_A$ and $\mathbf{V}_d = \sigma_B^2\Omega_d + \mathbf{V}_{ed}$ if $d > D_A$.

If $\sigma_A^2 > 0$, $\sigma_B^2 > 0$ and ρ are known, the best linear unbiased estimator (BLUE) of β is

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$$

and the best linear unbiased predictor (BLUP) of $\mathbf{u}$ is

$$\widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y} - \mathbf{X}\widehat{\beta}) = \mathrm{diag}\left(\sigma_A^2\underset{d\le D_A}{\mathrm{diag}}(\Omega_d), \sigma_B^2\underset{d>D_A}{\mathrm{diag}}(\Omega_d)\right)\underset{1\le d\le D}{\mathrm{col}}(\mathbf{V}_d^{-1})(\mathbf{y} - \mathbf{X}\widehat{\beta}),$$

so that

$$\widehat{\mathbf{u}}_d = \begin{cases} \sigma_A^2 \Omega_d \mathbf{V}_d^{-1}(\mathbf{y}_d - \mathbf{X}_d\widehat{\beta}), & d = 1,\dots,D_A, \\ \sigma_B^2 \Omega_d \mathbf{V}_d^{-1}(\mathbf{y}_d - \mathbf{X}_d\widehat{\beta}), & d = D_A+1,\dots,D. \end{cases}$$

The loglikelihood of the restricted (residual) maximum likelihood method is

$$\begin{aligned} l_{reml} &= l_{reml}(\sigma_A^2,\sigma_B^2,\rho) = -\frac{M-p}{2}\log 2\pi + \frac{1}{2}\log|\mathbf{X}'\mathbf{X}| - \frac{1}{2}\log|\mathbf{V}_A| - \frac{1}{2}\log|\mathbf{V}_B| \\ &- \frac{1}{2}\log|\mathbf{X}_A'\mathbf{V}_A^{-1}\mathbf{X}_A + \mathbf{X}_B'\mathbf{V}_B^{-1}\mathbf{X}_B| - \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{y}, \end{aligned}$$

where

$$\mathbf{P} = \mathbf{V}^{-1} - \mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}, \quad \mathbf{PVP} = \mathbf{P}, \quad \mathbf{PX} = \mathbf{0}.$$

Let $\theta = (\theta_1,\theta_2,\theta_3) = (\sigma_A^2,\sigma_B^2,\rho)$, then

$$\begin{aligned} \mathbf{V}_1 &= \frac{\partial \mathbf{V}}{\partial \sigma_A^2} = \text{diag}\left(\underset{d\le D_A}{\text{diag}}(\Omega_d(\rho)), \underset{d>D_A}{\text{diag}}(\mathbf{0}_{m_d\times m_d})\right) \\ \mathbf{V}_2 &= \frac{\partial \mathbf{V}}{\partial \sigma_B^2} = \text{diag}\left(\underset{d\le D_A}{\text{diag}}(\mathbf{0}_{m_d\times m_d}), \underset{d>D_A}{\text{diag}}(\Omega_d(\rho))\right), \\ \mathbf{V}_3 &= \frac{\partial \mathbf{V}}{\partial \rho} = \text{diag}\left(\sigma_A^2 \underset{d\le D_A}{\text{diag}}(\dot{\Omega}_d(\rho)), \sigma_B^2 \underset{d>D_A}{\text{diag}}(\dot{\Omega}_d(\rho))\right), \end{aligned}$$

where $\dot{\Omega}(\rho) = \partial\Omega(\rho)/\partial\rho$. Then

$$\mathbf{P}_a = \frac{\partial \mathbf{P}}{\partial \theta_a} = -\mathbf{P}\frac{\partial \mathbf{V}}{\partial \theta_a}\mathbf{P} = -\mathbf{P}\mathbf{V}_a\mathbf{P}, \quad a = 1,2,3.$$

By taking partial derivatives of l_{reml} with respect to θ_a, we get the scores

$$S_a = \frac{\partial l_{reml}}{\partial \theta_a} = -\frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a) + \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{y}, \quad a = 1,2,3.$$

By taking again partial derivatives with respect to θ_a and θ_b, taking expectations and changing the sign, we get the Fisher information matrix components

$$F_{ab} = \frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{V}_b), \quad a,b = 1,2,3.$$

To calculate the REML estimate we apply the Fisher-scoring algorithm with the updating fórmula

$$\theta^{k+1} = \theta^k + \mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k),$$

where **S** and **F** are the column vector of scores and the Fisher information matrix respectively. As seeds we use $\rho^{(0)} = 0$, and $\sigma_A^{2(0)} = \sigma_B^{2(0)} = \widehat{\sigma}_{uH}^2$, where $\widehat{\sigma}_{uH}^2$ is the Henderson 3 estimator under model with $\rho = 0$ and $\sigma_A^2 = \sigma_B^2$. The REML estimator of β is

$$\widehat{\beta}_{reml} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators of θ and β are

$$\hat{\theta} \sim N_3(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_j are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2},\ a=1,2,3, \quad \hat{\beta}_j \pm z_{\alpha/2}q_{jj}^{1/2},\ j=1,\dots,p,$$

where $\hat{\theta}=\theta^{\kappa}$, $\mathbf{F}^{-1}(\theta^{\kappa})=(\nu_{ab})_{a,b=1,2,3}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^{\kappa})\mathbf{X})^{-1}=(q_{ij})_{i,j=1,\dots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantile of the standard normal distribution $N(0,1)$). Observed $\hat{\beta}_j=\beta_0$, the p-value for testing the hypothesis $H_0:\ \beta_j=0$ is

$$p = 2P_{H_0}(\hat{\beta}_j > |\beta_0|) = 2P(N(0,1) > \beta_0/\sqrt{q_{jj}}).$$

We are interested in predicting the value of $\mu_{dt}=\mathbf{x}_{dt}\beta+u_{dt}$ by using the EBLUP $\widehat{\mu}_{dt}=\mathbf{x}_{dt}\widehat{\beta}+\widehat{u}_{dt}$. If we do not take into account the error, e_{dt}, this is equivalent to predict $y_{dt}=\mathbf{a}'\mathbf{y}$, where $\mathbf{a}=\underset{1\le\ell\le D}{\text{col}}(\underset{1\le k\le m_\ell}{\text{col}}(\delta_{d\ell}\delta_{tk}))$ is a vector having one 1 in the position $t+\sum_{\ell=1}^{d-1}m_\ell$ and 0's in the remaining cells. To estimate $\overline{Y}_{dt}$ we use $\widehat{\overline{Y}}_{dt}^{eblup}=\widehat{\mu}_{dt}$. The mean squared error of $\widehat{\overline{Y}}_{dt}^{eblup}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\theta)+g_2(\theta)+g_3(\theta),$$

and the estimator of $MSE(\widehat{\overline{Y}}_{dt}^{eblup})$ is

$$mse(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\hat{\theta})+g_2(\hat{\theta})+2g_3(\hat{\theta}),$$

where $\theta=(\sigma_A^2,\sigma_B^2,\rho)$, $\mathbf{a}_d=\underset{1\le k\le m_d}{\text{col}}(\delta_{tk})$. The expressions for g_1-g_3 are

$$\begin{aligned}
g_1(\theta) &= \begin{cases} \sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{a}_d-\sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d & \text{if } d\le D_A \\ \sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{a}_d-\sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d & \text{if } d> D_A \end{cases}\\
g_2(\theta) &= \left[\mathbf{a}_d'\mathbf{X}_d-\sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d+\sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q}\\
&\cdot\ \left[\mathbf{X}_d'\mathbf{a}_d-\sigma_A^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{a}_d+\sigma_A^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] \quad \text{if } d\le D_A\\
&= \left[\mathbf{a}_d'\mathbf{X}_d-\sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d+\sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q}\\
&\cdot\ \left[\mathbf{X}_d'\mathbf{a}_d-\sigma_B^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{a}_d+\sigma_B^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] \quad \text{if } d> D_A\\
g_3(\theta) &= \begin{cases} \text{tr}\left\{\begin{pmatrix} q_{11} & q_{13}\\ q_{31} & q_{33}\end{pmatrix}\begin{pmatrix} F_{11} & F_{13}\\ F_{31} & F_{33}\end{pmatrix}^{-1}\right\} & \text{if } d\le D_A\\ \text{tr}\left\{\begin{pmatrix} q_{22} & q_{23}\\ q_{32} & q_{33}\end{pmatrix}\begin{pmatrix} F_{22} & F_{23}\\ F_{32} & F_{33}\end{pmatrix}^{-1}\right\} & \text{if } d> D_A\end{cases}
\end{aligned}$$

where F_{ab} are the REML Fisher amount of information calculated by the updating equation of the Fisher-scoring algorithm and

$$\begin{aligned}
q_{11} &= \left[\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - 2\sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d + \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d\le D_A\}}(d),\\
q_{22} &= \left[\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - 2\sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d + \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d> D_A\}}(d),\\
q_{33} &= \left[\sigma_A^4\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - 2\sigma_A^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d + \sigma_A^8\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d\le D_A\}}(d)\\
&+ \left[\sigma_B^4\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - 2\sigma_B^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d + \sigma_B^8\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d> D_A\}}(d),\\
q_{13} &= \left[\sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\right.\\
&+ \left.\sigma_A^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d\le D_A\}}(d) \quad \text{where} \quad q_{31}=q_{13},\\
q_{23} &= \left[\sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\right.\\
&+ \left.\sigma_B^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] I_{\{d> D_A\}}(d) \quad \text{where} \quad q_{32}=q_{23}.
\end{aligned}$$

4.2.2 The Software: description of R functions for model 2

This section describes the R functions that have been implemented for fitting the partitioned area-level model with correlated time effect (4.2). A brief descriptions of programmed R functions is given in the next subsections and the related R codes are listed in Appendix B 18.2. The functions can be used for calculating the Henderson 3 and the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

H3area

Function **H3area** calculates the unbiased Henderson 3 estimator of σ_u^2 and has the form

$$H3.area\ <\!-\ function\ (X,\ ydt,\ D,\ md,\ sigma2edt).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the known error variances σ_{ed}^2, with size M.

The function returns the value of the Henderson 3 estimate of $\hat{\sigma}_u^2$.

REMLarea.corr

Function **REMLarea.corr** calculates the estimate of σ_A^2, σ_B^2, ρ and the Fisher amount of information F for the Restricted Maximum Likelihood (REML) method. The function is

$$REMLarea.corr <- function\ (X,\ ydt,\ D,\ Da,\ Db,\ md,\ mda,\ mdb,\ sigma2edt,$$

$$sigma.0,\ MAXITER = 100).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

mdb: vector containing the time instants totals m_d within each domain, with size D_B.

sigma2edt: vector containing the error variances σ_{ed}^2, with size M.

sigma.0: Henderson 3 estimate of σ_u^2 obtained as output of **H3area**. It is used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 100.

The function returns a list of four elements. The first element is **theta.f**, a vector containing the REML estimates of σ_A^2, σ_B^2 and ρ, the second **Fsig** is the estimated Fisher information matrix F, the third element **ITER** is the indicator of the maximum iteration when the algorithm stop the loop without convergence and the last one element **Bad** is the sum of iterations of the algorithm which stopped without convergence.

BETA.U.area.corr

Function **BETA.U.area.corr** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area.corr \ <- \ function \ (X, \ ydt, \ D, \ Da, \ Db, \ md, \ mda, \ mdb, \ sigma2edt,$$

$$sigmaua, sigmaub, rho).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the variable of interest for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

mdb: vector containing the time instants totals m_d within each domain, with size D_B.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmaua: estimated value of σ^2_A, calculated by the function **REMLarea.corr**.

sigmaub: estimated value of σ^2_B, calculated by the function **REMLarea.corr**.

rho: estimated value of ρ, calculated by the function **REMLarea.corr**.

The function returns an array with the following elements:

beta: vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

u: vector containing the predicted random effects $\widehat{\mathbf{u}}$, with size M.

mse.area.corr

Function **mse.area.corr** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.area.corr \ <- \ function \ (X,\ D,\ Da,\ Db,\ md,\ mda,\ mdb,\ sigma2edt,\ sigmaua,$$
$$sigmaub,\ rho,\ F11,\ F22,\ F33,\ F13,\ F23).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

mdb: vector containing the time instants totals m_d within each domain, with size D_B.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmaua: estimated value of σ^2_A, calculated by the function **REMLarea.corr**.

sigmaub: estimated value of σ^2_B, calculated by the function **REMLarea.corr**.

rho: estimated value of ρ, calculated by the function **REMLarea.corr**.

F11, F22, F33, F13, F23: elements of Fisher information matrix F, calculated by the function **REMLarea.corr**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}_{dt}^{eblup})$, with size M.

Interval.corr

Function **Interval.corr** calculates, within the same file, the asymptotic confidence intervals for σ^2_A, σ^2_B, ρ and β_i and the asymptotic p-value of test statistics $\hat{\beta}_i$ for the null hypothesis $H_0 : \beta_i = 0$.

The first function is

$$Interval.corr \ <- \ function \ (Fisher,\ conf = 0.95).$$

The arguments are:

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea.corr**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma.std.err**, **rho.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ_A^2, σ_B^2, ρ and β_i respectively.

The second function is

$$pvalueBeta <- function\ (beta0.hat,\ Fisher).$$

The arguments are:

beta0.hat: observed value of $\hat{\beta}_i$, calculated by the function **BETA.U.area.corr**.

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea.corr**.

This function returns the vector **2*p.beta** containing the asymptotic p-values for hypotheses H_0 : $\beta_i = 0, i = 1,\ldots,p$, with size p.

4.3 Partitioned Fay-Herriot model 3

4.3.1 The methodology

Consider the model (model 3)

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{dt} + e_{dt}, \quad d = 1,\ldots,D = D_A + D_B, \quad t = 1,\ldots,m_d, \tag{4.3}$$

where y_{dt} is a direct estimator of the indicator of interest for area d and time instant t, and $\mathbf{x}_{dt}$ is a vector containing the aggregated (population) values of p auxiliary variables. The index d is used for domains and the index t for time instants. We assume that the random vectors $(u_{d1},\ldots,u_{dm_d})$, $d \le D_A$, follow i.i.d. first order auto-regressive processes with variance and auto-correlation parameters σ_A^2 and ρ_A respectively; in short, $(u_{d1},\ldots,u_{dm_d}) \sim_{iid} AR1(\sigma_A^2,\rho_A)$, $d \le D_A$. We further assume that $(u_{d1},\ldots,u_{dm_d}) \sim_{iid} AR1(\sigma_B^2,\rho_B)$, $d > D_A$, and that the errors e_{dt}'s are independent $N(0,\sigma_{dt}^2)$ with known σ_{dt}^2's. Finally we assume that the $(u_{d1},\ldots,u_{dm_d})$'s and the e_{dt}'s are mutually independent.

In matrix notation the model is

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \mathbf{e},$$

where vectors $\mathbf{y}$, $\mathbf{u}$ and $\mathbf{e}$ can be decomposed in the form $\mathbf{v} = (\mathbf{v}_A', \mathbf{v}_B')'$, with $\mathbf{v}_A = \underset{d\le D_A}{\mathrm{col}}(\mathbf{v}_d)$, $\mathbf{v}_B = \underset{d>D_A}{\mathrm{col}}(\mathbf{v}_d)$ and $\mathbf{v}_d = \underset{1\le t\le m_d}{\mathrm{col}}(v_{dt})$, matrix $\mathbf{X}$ can be similarly decomposed in the form $\mathbf{X} = (\mathbf{X}_A', \mathbf{X}_B')'$, with $\mathbf{X}_A = \underset{d\le D_A}{\mathrm{col}}(\mathbf{X}_d)$, $\mathbf{X}_B = \underset{d>D_A}{\mathrm{col}}(\mathbf{X}_d)$ and $\mathbf{X}_d = \underset{1\le t\le m_d}{\mathrm{col}'}(\mathbf{x}_{dt})$, $\beta = \beta_{p\times 1}$, $\mathbf{Z} = \mathbf{I}_M$ and $M = \sum_{d=1}^D m_d$. In this notation, $\mathbf{u} \sim N(\mathbf{0}, \mathbf{V}_u)$ and $\mathbf{e} \sim N(\mathbf{0}, \mathbf{V}_e)$ are independent with covariance matrices

$$\mathbf{V}_u = \mathrm{var}(\mathbf{u}) = \mathrm{diag}(\sigma_A^2\Omega_A, \sigma_B^2\Omega_B), \quad \mathbf{V}_e = \mathrm{var}(\mathbf{e}) = \underset{1\le d\le D}{\mathrm{diag}}(\mathbf{V}_{ed})$$

where $\Omega_A = \underset{d\le D_A}{\operatorname{diag}}(\Omega_d)$, $\Omega_B = \underset{d>D_A}{\operatorname{diag}}(\Omega_d)$, $\mathbf{V}_{ed} = \underset{1\le t\le m_d}{\operatorname{diag}}(\sigma_{dt}^2)$ and

$$\Omega_d = \Omega_d(\rho) = \frac{1}{1-\rho^2}\begin{pmatrix} 1 & \rho & \dots & \rho^{m_d-2} & \rho^{m_d-1} \\ \rho & 1 & \ddots & & \rho^{m_d-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \rho^{m_d-2} & & \ddots & 1 & \rho \\ \rho^{m_d-1} & \rho^{m_d-2} & \dots & \rho & 1 \end{pmatrix}_{m_d\times m_d}, \quad \rho=\rho_A,\rho_B.$$

The covariance matrix of vector $\mathbf{y}$ is $\mathbf{V}=\text{var}(\mathbf{y})=\text{diag}(\mathbf{V}_A,\mathbf{V}_B)$, where $\mathbf{V}_A = \underset{d\le D_A}{\operatorname{diag}}(\mathbf{V}_d)$, $\mathbf{V}_B = \underset{d>D_A}{\operatorname{diag}}(\mathbf{V}_d)$, $\mathbf{V}_d = \sigma_A^2\Omega_d + \mathbf{V}_{ed}$ if $d\le D_A$ and $\mathbf{V}_d = \sigma_B^2\Omega_d+\mathbf{V}_{ed}$ if $d>D_A$.

If $\sigma_A^2>0$, ρ_A, $\sigma_B^2>0$ and ρ_B are known, the best linear unbiased estimator (BLUE) of β is

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$$

and the best linear unbiased predictor (BLUP) of $\mathbf{u}$ is

$$\widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\beta}) = \text{diag}\left(\sigma_A^2\underset{d\le D_A}{\operatorname{diag}}(\Omega_d), \sigma_B^2\underset{d>D_A}{\operatorname{diag}}(\Omega_d)\right)\underset{1\le d\le D}{\operatorname{col}}(\mathbf{V}_d^{-1})(\mathbf{y}-\mathbf{X}\widehat{\beta}),$$

so that

$$\widehat{\mathbf{u}}_d = \begin{cases} \sigma_A^2\Omega_d\mathbf{V}_d^{-1}(\mathbf{y}_d-\mathbf{X}_d\widehat{\beta}), & d=1,\dots,D_A, \\ \sigma_B^2\Omega_d\mathbf{V}_d^{-1}(\mathbf{y}_d-\mathbf{X}_d\widehat{\beta}), & d=D_A+1,\dots,D. \end{cases}$$

The loglikelihood of the restricted (residual) maximum likelihood method is

$$\begin{aligned} l_{reml} &= l_{reml}(\sigma_A^2,\rho_A,\sigma_B^2,\rho_B) = -\frac{M-p}{2}\log 2\pi + \frac{1}{2}\log|\mathbf{X}'\mathbf{X}| - \frac{1}{2}\log|\mathbf{V}_A| - \frac{1}{2}\log|\mathbf{V}_B| \\ &- \frac{1}{2}\log|\mathbf{X}_A'\mathbf{V}_A^{-1}\mathbf{X}_A + \mathbf{X}_B'\mathbf{V}_B^{-1}\mathbf{X}_B| - \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{y}, \end{aligned}$$

where

$$\mathbf{P} = \mathbf{V}^{-1} - \mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}, \quad \mathbf{PVP}=\mathbf{P}, \quad \mathbf{PX}=\mathbf{0}.$$

Let $\theta=(\theta_1,\theta_2,\theta_3,\theta_4)=(\sigma_A^2,\rho_A,\sigma_B^2,\rho_B)$, then

$$\begin{aligned} \mathbf{V}_1 &= \frac{\partial\mathbf{V}}{\partial\sigma_A^2} = \text{diag}\left(\underset{d\le D_A}{\operatorname{diag}}(\Omega_d(\rho_A)), \underset{d>D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d})\right) \\ \mathbf{V}_2 &= \frac{\partial\mathbf{V}}{\partial\rho_A} = \text{diag}\left(\sigma_A^2\underset{d\le D_A}{\operatorname{diag}}(\dot\Omega_d(\rho_A)), \underset{d>D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d})\right), \\ \mathbf{V}_3 &= \frac{\partial\mathbf{V}}{\partial\sigma_B^2} = \text{diag}\left(\underset{d\le D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d}), \underset{d>D_A}{\operatorname{diag}}(\Omega_d(\rho_B))\right), \\ \mathbf{V}_4 &= \frac{\partial\mathbf{V}}{\partial\rho_B} = \text{diag}\left(\underset{d\le D_A}{\operatorname{diag}}(\mathbf{0}_{m_d\times m_d}), \sigma_B^2\underset{d>D_A}{\operatorname{diag}}(\dot\Omega_d(\rho_B))\right). \end{aligned}$$

where $\dot{\Omega}(\rho) = \partial\Omega(\rho)/\partial\rho$. Then

$$\mathbf{P}_a = \frac{\partial \mathbf{P}}{\partial \theta_a} = -\mathbf{P}\frac{\partial \mathbf{V}}{\partial \theta_a}\mathbf{P} = -\mathbf{P}\mathbf{V}_a\mathbf{P}, \quad a = 1,2,3,4.$$

By taking partial derivatives of l_{reml} with respect to θ_a, we get the scores

$$S_a = \frac{\partial l_{reml}}{\partial \theta_a} = -\frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a) + \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{y}, \quad a = 1,2,3,4.$$

By taking again partial derivatives with respect to θ_a and θ_b, taking expectations and changing the sign, we get the Fisher information matrix components

$$F_{ab} = \frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{V}_b), \quad a,b = 1,2,3,4.$$

To calculate the REML estimate we apply the Fisher-scoring algorithm with the updating fórmula

$$\theta^{k+1} = \theta^k + \mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k),$$

where $\mathbf{S}$ and $\mathbf{F}$ are the column vector of scores and the Fisher information matrix respectively. As seeds we use $\rho_A^{(0)} = \rho_B^{(0)} = 0$, and $\sigma_A^{2(0)} = \sigma_B^{2(0)} = \widehat{\sigma}_{uH}^2$, where $\widehat{\sigma}_{uH}^2$ is the Henderson 3 estimator under model with $\rho_A = \rho_B = 0$ and $\sigma_A^2 = \sigma_B^2$. The REML estimator of β is

$$\widehat{\beta}_{reml} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of of the REML estimators of θ and β are

$$\hat{\theta} \sim N_4(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_j are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2}, \ a = 1,2,3,4, \quad \hat{\beta}_j \pm z_{\alpha/2}q_{jj}^{1/2}, \ j = 1,\ldots,p,$$

where $\hat{\theta} = \theta^\kappa$, $\mathbf{F}^{-1}(\theta^\kappa) = (\nu_{ab})_{a,b=1,2,3,4}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^\kappa)\mathbf{X})^{-1} = (q_{ij})_{i,j=1,\ldots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantile of the standard normal distribution $N(0,1)$). Observed $\hat{\beta}_j = \beta_0$, the p-value for testing the hypothesis $H_0: \beta_j = 0$ is

$$p = 2P_{H_0}(\hat{\beta}_j > |\beta_0|) = 2P(N(0,1) > \beta_0/\sqrt{q_{jj}}).$$

We are interested in predicting the value of $\mu_{dt} = \mathbf{x}_{dt}\beta + u_{dt}$ by using the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. If we do not take into account the error, e_{dt}, this is equivalent to predict $y_{dt} = \mathbf{a}'\mathbf{y}$, where $\mathbf{a} = \underset{1\le\ell\le D}{\text{col}}(\underset{1\le k\le m_\ell}{\text{col}}(\delta_{d\ell}\delta_{tk}))$ is a vector having one 1 in the position $t + \sum_{\ell=1}^{d-1} m_\ell$ and 0's in the remaining cells. To estimate $\overline{Y}_{dt}$ we use $\widehat{\overline{Y}}_{dt}^{eblup} = \widehat{\mu}_{dt}$. The mean squared error of $\widehat{\overline{Y}}_{dt}^{eblup}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\theta) + g_2(\theta) + g_3(\theta),$$

and the estimator of $MSE(\widehat{\overline{Y}}_{dt}^{eblup})$ is

$$mse(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\hat{\theta}) + g_2(\hat{\theta}) + 2g_3(\hat{\theta}),$$

where $\theta = (\sigma_A^2, \rho_A, \sigma_B^2, \rho_B)$,$\mathbf{a}_d = \underset{1\le k\le m_d}{\text{col}}(\delta_{tk})$. The expressions for $g_1 - g_3$ are

$$\begin{aligned}
g_1(\theta) &= \begin{cases} \sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{a}_d - \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d & \text{if } d \le D_A \\ \sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{a}_d - \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d & \text{if } d > D_A \end{cases} \\
g_2(\theta) &= \left[\mathbf{a}_d'\mathbf{X}_d - \sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d + \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q} \\
&\cdot\ \left[\mathbf{X}_d'\mathbf{a}_d - \sigma_A^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{a}_d + \sigma_A^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] \quad \text{if } d \le D_A \\
&= \left[\mathbf{a}_d'\mathbf{X}_d - \sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d + \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_{ed}^{-1}\mathbf{X}_d\right]\mathbf{Q} \\
&\cdot\ \left[\mathbf{X}_d'\mathbf{a}_d - \sigma_B^2\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{a}_d + \sigma_B^4\mathbf{X}_d'\mathbf{V}_{ed}^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right] \quad \text{if } d > D_A \\
g_3(\theta) &= \begin{cases} \text{tr}\left\{\begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22}\end{pmatrix}\begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22}\end{pmatrix}^{-1}\right\} & \text{if } d \le D_A \\ \text{tr}\left\{\begin{pmatrix} q_{33} & q_{34} \\ q_{43} & q_{44}\end{pmatrix}\begin{pmatrix} F_{33} & F_{34} \\ F_{43} & F_{44}\end{pmatrix}^{-1}\right\} & \text{if } d > D_A \end{cases}
\end{aligned}$$

where F_{ab} are the REML Fisher amount of information calculated by the updating equation of the Fisher-scoring algorithm and

$$\begin{aligned}
q_{11} &= \left[\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - 2\sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d + \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d\le D_A\}}(d), \\
q_{12} &= \left[\sigma_A^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_A^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\right. \\
&+ \left.\sigma_A^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d\le D_A\}}(d) \quad \text{where} \quad q_{21} = q_{12}, \\
q_{22} &= \left[\sigma_A^4\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_A^6\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_A^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\right. \\
&+ \left.\sigma_A^8\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d\le D_A\}}(d), \\
q_{33} &= \left[\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - 2\sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d + \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d>D_A\}}(d), \\
q_{34} &= \left[\sigma_B^2\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_B^4\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d\right. \\
&+ \left.\sigma_B^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d>D_A\}}(d) \quad \text{where} \quad q_{43} = q_{34}, \\
q_{44} &= \sigma_B^4\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d - \sigma_B^6\mathbf{a}_d'\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d - \sigma_B^6\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{a}_d \\
&+ \left.\sigma_B^8\mathbf{a}_d'\Omega_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\dot{\Omega}_d\mathbf{V}_d^{-1}\Omega_d\mathbf{a}_d\right]I_{\{d>D_A\}}(d).
\end{aligned}$$

4.3.2 The Software: description of R functions for model 3

This section describes the R functions that have been implemented for fitting the partitioned area-level model with correlated time effect (4.3). A brief descriptions of programmed R functions is given in the next subsections and the related R codes are listed in Appendix C 18.3. The functions can be used for calculating the Henderson 3 and the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

H3area

Function **H3area** calculates the unbiased Henderson 3 estimator of σ_u^2 and has the form

$$H3.area \ <- \ function\ (X,\ ydt,\ D,\ md,\ sigma2edt).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma2edt: vector containing the known error variances σ_{ed}^2, with size M.

The function returns the value of the Henderson 3 estimate of $\hat{\sigma}_u^2$.

REMLarea.2corr

Function **REMLarea.2corr** calculates the estimate of σ_A^2, σ_B^2, ρ_A, ρ_B and the Fisher amount of information F for the Restricted Maximum Likelihood (REML) method. The function is

$$REMLarea.2corr \ <- \ function\ (X,\ ydt,\ D,\ Da,\ Db,\ md,\ mda,\ mdb,\ sigma2edt,$$

$$sigma.0,\ MAXITER = 100).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

mdb: vector containing the time instants totals m_d within each domain, with size D_B.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigma.0: Henderson 3 estimate of σ^2_u obtained as output of **H3area**. It is used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 100.

The function returns a list of four elements. The first element is **theta.f**, a vector containing the REML estimates of σ^2_A, σ^2_B, ρ_A and ρ_B the second **Fsig** is the estimated Fisher information matrix F, the third element **ITER** is the indicator of the maximum iteration when the algorithm stop the loop without convergence and the last one element **Bad** is the sum of iterations of the algorithm which stopped without convergence.

BETA.U.area.2corr

Function **BETA.U.area.2corr** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area.2corr \ <- \ function\ (X,\ ydt,\ D,\ Da,\ Db,\ md,\ mda,\ sigma2edt,$$

$$sigmaua,\ sigmaub,\ rhoa,\ rhob).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

ydt: vector containing the direct estimates of the variable of interest for area d and time instant t, with size M.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

sigma2edt: vector containing the error variances σ^2_{ed}, with size M.

sigmaua: estimated value of σ^2_A, calculated by the function **REMLarea.2corr**.

sigmaub: estimated value of σ_B^2, calculated by the function **REMLarea.2corr**.

rhoa: estimated value of ρ_A, calculated by the function

rhob: estimated value of ρ_B, calculated by the function **REMLarea.2corr**.

The function returns an array with the following elements:

beta: vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

u: vector containing the predicted random effects $\widehat{\mathbf{u}}$, with size M.

mse.area.2corr

Function **mse.area.2corr** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.area.2corr \ <- \ function\ (X,\ D,\ Da,\ Db,\ md,\ mda,\ mdb,\ sigma2edt,\ sigmaua,\ sigmaub,$$
$$rhoa,\ rhob,\ F11,\ F22,\ F12,\ F33,\ F44,\ F34).$$

The arguments are:

X: matrix containing the aggregated (population) values of p auxiliary variables, with dimension $M \times p$. First column elements should be equal to 1 if the model includes intercept.

D: total number of domains.

Da: total number of domains of the group A.

Db: total number of domains of the group B.

md: vector containing the time instants totals m_d within each domain, with size D.

mda: vector containing the time instants totals m_d within each domain, with size D_A.

mdb: vector containing the time instants totals m_d within each domain, with size D_B.

sigma2edt: vector containing the error variances σ_{ed}^2, with size M.

sigmaua: estimated value of σ_A^2, calculated by the function **REMLarea.2corr**.

sigmaub: estimated value of σ_B^2, calculated by the function **REMLarea.2corr**.

rhoa: estimated value of ρ_A, calculated by the function **REMLarea.2corr**.

rhob: estimated value of ρ_B, calculated by the function **REMLarea.2corr**.

F11, F22, F12, F33, F44, F34: elements of Fisher information matrix F, calculated by the function **REMLarea.2corr**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}_{dt}^{eblup})$, with size M.

Interval.2corr

Function **Interval.2corr** calculates, within the same file, the asymptotic confidence intervals for σ_A^2, σ_B^2, ρ_A, ρ_B and β_i and the asymptotic p-value of test statistics $\hat{\beta}_i$ for the null hypothesis $H_0 : \beta_i = 0$.

The first function is

$$Interval \ <\!- \ function \ (Fisher, \ conf = 0.95).$$

The arguments are:

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea.2corr**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma.std.err**, **rho.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ_A^2, σ_B^2, ρ_A, ρ_B and β_i respectively.

The second function is

$$pvalueBeta.2corr \ <\!- \ function \ (beta0.hat, \ Fisher).$$

The arguments are:

beta0.hat: observed value of $\hat{\beta}_i$, calculated by the function **BETA.U.area.2corr**.

Fisher: returned object of Fisher information matrix, obtained by applying the function **REMLarea.2corr**.

This function returns the vector **2*p.beta** containing the asymptotic p-values for hypotheses H_0 : $\beta_i = 0, i = 1, \dots, p$, with size p.

4.4 Examples of usage of R functions

This section demonstrates how the R routines described in chapters 4.1, 4.2 and 4.3 can be applied to produce EBLUP estimates with their corresponding mean squared errors.

4.4.1 Example data set

Table 4.4.1.1 presents the data sets used in the example. There are 10 domains (areas) crossed with variable *Sex* which divide the population in two groups, 3 time periods, a intercept variable *ones* and 2 independent variables $X1$ and $X2$ for each domain and time period. Dependent variables is labeled by Y and the error variances by Var. There are are 30 observations. The file *dataExample.txt* contains the data, which should be sorted by domains/sex and time periods.

Domain	*Sex*	*Time*	*ones*	*X*1	*X*2	*Y*	*Var*
1	1	1	1	0.783265583	0.331728042	0.099870796	0.014167965
1	1	2	1	0.216346833	0.196779310	0.255527036	0.012960672
1	1	3	1	0.056667147	0.203072229	0.758100324	0.004162126
2	1	1	1	0.342828496	0.789455882	0.531886402	0.015905454
2	1	2	1	0.596317580	0.021936772	0.975476720	0.007200226
2	1	3	1	0.537585866	0.543745488	0.562169587	0.010016244
3	1	1	1	0.210260343	0.214442971	0.720967375	0.016778616
3	1	2	1	0.832804962	0.120835233	0.494603026	0.020104690
3	1	3	1	0.188368681	0.300660778	0.924517608	0.016954615
4	1	1	1	0.537603798	0.221765139	0.533091894	0.005465959
4	1	2	1	0.200491580	0.021550740	0.517926099	0.002823810
4	1	3	1	0.007613312	0.706056104	0.038331778	0.000019083
5	1	1	1	0.459343496	0.696134155	0.399109186	0.011460628
5	1	2	1	0.438793922	0.821961781	0.943433632	0.008220732
5	1	3	1	0.384939092	0.622613717	0.254052932	0.000868833
1	2	1	1	0.375006267	0.217942523	0.305351622	0.007523092
1	2	2	1	0.717422790	0.817325369	0.060847784	0.019176240
1	2	3	1	0.745012109	0.771316745	0.344023489	0.002657316

Table 4.4.1.1. Data set *dataExample*.

Domain	*Sex*	*Time*	*ones*	*X*1	*X*2	*Y*	*Var*
2	2	1	1	0.323252837	0.888016118	0.389850382	0.009978866
2	2	2	1	0.488759367	0.551428405	0.105179702	0.000799896
2	2	3	1	0.282042241	0.636340640	0.163296232	0.004229634
3	2	1	1	0.860611106	0.586243775	0.899385519	0.009776561
3	2	2	1	0.651417737	0.175238106	0.019623444	0.005936166
3	2	3	1	0.455091459	0.407611037	0.724542241	0.001704073
4	2	1	1	0.507312619	0.842438438	0.474116093	0.000830603
4	2	2	1	0.227729281	0.458520990	0.208451174	0.021446166
4	2	3	1	0.491374145	0.753102269	0.994509978	0.016204204
5	2	1	1	0.392654380	0.603564240	0.546367453	0.023036316
5	2	2	1	0.815092048	0.533703922	0.277008991	0.011246273
5	2	3	1	0.755942844	0.559191967	0.228748126	0.018568897

Table 4.4.1.1. Data set *dataExample* (continuation).

4.4.2 Example of R code

An R code for reading the data file and applying the above described functions is needed. The file *Example.R* contains this code and, for this example, is located in the folder *C:/PartFHModel*. It is important to take care on where to put this file and all the function *R* files. Note that under *Windows system*, the folder of the file is set by default installation. Otherwise the user can type the complete path. But under *Linux* the folder is the same than the one used to execute R.

The R file *Example.R* contains a program with the instructions for fitting the area level model to data in file *dataExample.txt*. First step is to open the *R* files containing all the above described R functions, i.e. *H3.R, REML.R, REMLcorr.R, REML2corr.R, EstimationBETA.R, EstimationBETAcorr.R, EstimationBETA2corr.R, EstimationMSE.R, EstimationMSEcorr.R, EstimationMSE2corr.R, IC.R, ICcorr.R* and

IC2corr.R. Second step is to read the data file *dataExample.txt*. Third step is to run the application. The program creates several *txt* files in folder *C:/PartFHModel*. The new files contain the output of the program in what follows the code in *Example.R* is listed.

```
########################################################################
###
###          Example of usage for Partitioned Fay-Herriot
###                          Models
###
### Author: Maria Chiara Pagliarella
### File name: Example.R
### Updated: December, 2010
###
########################################################################

rm(list=ls(all=TRUE))

### Establishing the folder where data and routine files are located.
setwd("C:/PartFHModel/")

### Call functions: H3area, REMLarea, BETA.U.area, mse.area,
###                 Interval, pvalueBeta
###                 REMLarea.corr, BETA.U.area.corr, mse.area.corr,
###                 Interval.corr, pvalueBeta.corr
###                 REMLarea.2corr, BETA.U.area.2corr, mse.area.2corr,
###                 Interval.2corr and pvalueBeta.2corr
source("H3.R")
source("REML.R")
source("EstimationBETA.R")
source("EstimationMSE.R")
source("IC.R")
source("REMLcorr.R")
source("EstimationBETAcorr.R")
source("EstimationMSEcorr.R")
source("ICcorr.R")
source("REML2corr.R")
source("EstimationBETA2corr.R")
source("EstimationMSE2corr.R")
source("IC2corr.R")
```

```
## Reading data
data <- read.table(file = "dataExample.txt", header = T)

X <- as.matrix(data[,4:(ncol(data)-2)])
p <- ncol(X)
ydt <- data[,ncol(data)-1]
D <- length(unique(data[,1]))*2
Da <- Db <- D/2
md <- rep(length(unique(data[,3])), D)
M <- sum(md)
sigma2edt <- data[,ncol(data)]


## Calculating H3 and REML variance estimates
sigma.0 <- H3area(X, ydt, D, md, sigma2edt)

## Partitioned Fay-Herriot Model 1 with independent time effects
## Arguments: (X, ydt, D, Da, Db, md, sigma2edt, sigma.0, MAXITER=100)
fit1 <- REMLarea(X, ydt, D, Da, Db, md, sigma2edt, sigma.0 = sigma.0,
                 MAXITER = 100)
sigmaua.hat <- fit1[[1]]
sigmaub.hat <- fit1[[2]]
if(sigmaua.hat<0) {
    write.table(data.frame(sigmaua.hat), file="VAR_A.NEGATIVE.txt",
    append=TRUE)
    sigmaua.hat <- 0
}
if(sigmaub.hat<0) {
    write.table(data.frame(sigmaub.hat), file="VAR_B.NEGATIVE.txt",
    append=TRUE)
    sigmaub.hat <- 0
}


### Arguments: (X, ydt, D, Da, Db, md, sigma2edt, sigmaua, sigmaub)
beta.u.hat <- BETA.U.area(X, ydt, D, Da, Db, md, sigma2edt,
                sigmaua=sigmaua.hat, sigmaub=sigmaub.hat)
beta0.hat <- beta.u.hat[1:p]
beta1.hat <- beta0.hat
```

```
### Confidence intervals for beta and sigma_a/sigma_b
### with level of precision 90%
Int <- Interval(fit1, 0.90)
             beta0.hat-Int[[4]]
             beta0.hat+Int[[4]]

MinSigma1 <- c(Int[[8]], Int[[11]], Int[[14]])
MaxSigma1 <- c(Int[[9]], Int[[12]], Int[[15]])
TestSigma1 <- c(Int[[10]], Int[[13]], Int[[16]])
Int.sigma <- data.frame(MinSigma1,MaxSigma1,TestSigma1)
row.names(Int.sigma)<-c("Sigma.A","Sigma.B","Sigma.Dif")
Int.sigma

### P-value of beta
(pvalueB <- pvalueBeta(beta0.hat, fit1))
                pvalueB>0.10
udt.hat <- beta.u.hat[-(1:p)]

### EBLUP of the population parameter
mudt.hat <- as.vector( X %*% beta0.hat + udt.hat)

sqrt.mse <- sqrt(mse.area(X, D, Da, Db, md, sigma2edt,
            sigmaua=sigmaua.hat, sigmaub=sigmaub.hat,
            F11=fit1[[3]][1,1], F22=fit1[[3]][2,2]))

residual <- ydt - mudt.hat
Henderson3 <- sigma.0

####################################################################

### Reading data
data <- read.table(file = "dataExample.txt", header = T)

X <- as.matrix(data[,4:(ncol(data)-2)])
p <- ncol(X)
ydt <- data[,ncol(data)-1]
D <- length(unique(data[,1]))*2
Da <- Db <- D/2
md <- rep(length(unique(data[,3])), D)
M <- sum(md)
```

```
mda <- rep(length(unique(data[,3])), Da)
Ma <- sum(mda)
mdb <- rep(length(unique(data[,3])), Db)
Mb <- sum(mdb)
sigma2edt <- data[,ncol(data)]

### Calculating H3 and REML variance estimates
sigma.0 <- H3area(X, ydt, D, md, sigma2edt)

### Partitioned Fay-Herriot Model 2 with only one
### factor of correlation rho
###Arguments: (X, ydt, D, Da, Db, md, mda, mdb, sigma2edt,
               sigma.fa, sigma.fb, rho=0.2, MAXITER = 100)
fit1 <- REMLarea.corr(X, ydt, D, Da, Db, md, mda, mdb, sigma2edt,
        sigma.fa=sigma.0, sigma.fb=sigma.0, rho=0.2, MAXITER = 100)

sigmaua.hat <- fit1[[1]][1]
sigmaub.hat <- fit1[[1]][2]
rho.hat <- fit1[[1]][3]

### Arguments: (X, ydt, D, Da, Db, md, mda, sigma2edt,
###             sigmaua, sigmaub, rho)
### Note that Omega.a is calculate with index mda, and
### Omega.b is calculated without index mdb, but only with md
beta.u.hat <- BETA.U.area.corr(X, ydt, D, Da, Db, md, mda, sigma2edt,
              sigmaua=sigmaua.hat, sigmaub=sigmaub.hat, rho=rho.hat)
beta0.hat <- beta.u.hat[1:p]
beta2.hat <- beta0.hat

### Confidence intervals for beta, sigma_a/sigma_b and rho
### with confidence level 90%

Int.corr <- Interval.corr(fit1, 0.90)
            beta0.hat-Int.corr[[5]]
            beta0.hat+Int.corr[[5]]

MinSigma2 <- c(Int.corr[[9]], Int.corr[[12]], Int.corr[[15]])
MaxSigma2 <- c(Int.corr[[10]], Int.corr[[13]], Int.corr[[16]])
TestSigma2 <- c(Int.corr[[11]], Int.corr[[14]], Int.corr[[17]])
Int.corr.sigma <- data.frame(MinSigma2,MaxSigma2,TestSigma2)
```

```
row.names(Int.corr.sigma)<-c("Sigma.A","Sigma.B","Sigma.Dif")
Int.corr.sigma

MinRho2 <- c(Int.corr[[18]])
MaxRho2 <- c(Int.corr[[19]])
TestRho2 <- c(Int.corr[[20]])
Int.corr.rho <- data.frame(MinRho2,MaxRho2,TestRho2)
Int.corr.rho


### P-value of beta
(pvalueB.corr <- pvalueBeta.corr(beta0.hat, fit1))
                 pvalueB.corr>0.10
udt.hat.corr <- beta.u.hat[-(1:p)]

### EBLUP of the population parameter
mudt.hat.corr <- as.vector(X %*% beta0.hat + udt.hat.corr)

sqrt.mse.corr <- sqrt(mse.area.corr(X, D, Da, Db, md, mda, mdb,
   sigma2edt, sigmaua=sigmaua.hat, sigmaub=sigmaub.hat, rho=rho.hat,
   F11=fit1[[2]][1,1], F22=fit1[[2]][2,2], F13=fit1[[2]][1,3],
   F23=fit1[[2]][2,3], F33=fit1[[2]][3,3]))

residual.corr <- ydt - mudt.hat.corr

#####################################################################

### Reading data
data <- read.table(file = "dataExample.txt", header = T)

X <- as.matrix(data[,4:(ncol(data)-2)])
p <- ncol(X)
ydt <- data[,ncol(data)-1]
D <- length(unique(data[,1]))*2
Da <- Db <- D/2
md <- rep(length(unique(data[,3])), D)
M <- sum(md)
mda <- rep(length(unique(data[,3])), Da)
Ma <- sum(mda)
mdb <- rep(length(unique(data[,3])), Db)
```

```
Mb <- sum(mdb)
sigma2edt <- data[,ncol(data)]

### Calculating H3 and REML variance estimates
sigma.0 <- H3area(X, ydt, D, md, sigma2edt)

### Partitioned Fay-Herriot Model 3 with two
### factor of correlation rho_a and rho_b
###Arguments: (X, ydt, D, Da, Db, md, mda, mdb, sigma2edt, sigma.fa,
###            sigma.fb, rhoa, rhob, MAXITER = 100)
fit1 <- REMLarea.2corr(X, ydt, D, Da, Db, md, mda, mdb, sigma2edt,
sigma.fa=sigma.0, sigma.fb=sigma.0, rhoa=0.1, rhob=0.3, MAXITER = 100)

sigmaua.hat <- fit1[[1]][1]
sigmaub.hat <- fit1[[1]][3]
rhoa.hat <- fit1[[1]][2]
rhob.hat <- fit1[[1]][4]

### Arguments: (X, ydt, D, Da, Db, md, mda, sigma2edt, sigmaua,
### sigmaub, rhoa, rhob)
### Take care that Omega.a is calculate with index mda and Omega.b is
### calculated without index mdb, only with md
beta.u.hat <- BETA.U.area.2corr(X, ydt, D, Da, Db, md, mda, sigma2edt,
sigmaua=sigmaua.hat, sigmaub=sigmaub.hat, rhoa=rhoa.hat, rhob=rhob.hat)
beta0.hat <- beta.u.hat[1:p]
beta3.hat <- beta0.hat

### Confidence intervals for beta, sigma_a/sigma_b,
### rho_a and rho_b with level of precision 90%

Int.2corr <- Interval.2corr(fit1, 0.90)
            beta0.hat-Int.2corr[[7]]
            beta0.hat+Int.2corr[[7]]

MinSigma3 <- c(Int.2corr[[11]], Int.2corr[[14]], Int.2corr[[17]])
MaxSigma3 <- c(Int.2corr[[12]], Int.2corr[[15]], Int.2corr[[18]])
TestSigma3 <- c(Int.2corr[[13]], Int.2corr[[16]], Int.2corr[[19]])
Int.2corr.sigma <- data.frame(MinSigma3,MaxSigma3,TestSigma3)
row.names(Int.2corr.sigma)<-c("Sigma.A","Sigma.B","Sigma.Dif")
Int.2corr.sigma
```

```
MinRho3 <- c(Int.2corr[[20]], Int.2corr[[23]], Int.2corr[[26]])
MaxRho3 <- c(Int.2corr[[21]], Int.2corr[[24]], Int.2corr[[27]])
TestRho3 <- c(Int.2corr[[22]], Int.2corr[[25]], Int.2corr[[28]])
Int.2corr.rho <- data.frame(MinRho3,MaxRho3,TestRho3)
row.names(Int.2corr.rho)<-c("Rho.A","Rho.B","Rho.Dif")
Int.2corr.rho

### P-value of beta
(pvalueB.2corr <- pvalueBeta.2corr(beta0.hat, fit1))
                 pvalueB.2corr>0.10
udt.hat.2corr <- beta.u.hat[-(1:p)]

### EBLUP of the population parameter
mudt.hat.2corr <- as.vector(X %*% beta0.hat + udt.hat.2corr)

sqrt.mse.2corr <- sqrt(mse.area.2corr(X, D, Da, Db, md, mda, mdb,
  sigma2edt, sigmaua=sigmaua.hat, sigmaub=sigmaub.hat, rhoa=rhoa.hat,
  rhob=rhob.hat, F11=fit1[[2]][1,1], F22=fit1[[2]][2,2],
  F12=fit1[[2]][1,2], F33=fit1[[2]][3,3], F44=fit1[[2]][4,4],
  F34=fit1[[2]][3,4]))

residual.2corr <- ydt - mudt.hat.2corr

### Create .txt files in the folder that contains for
### the resulting output

write.table(data.frame(data[,1:3],
Direct=ydt, EBLUP1=mudt.hat, EBLUP2=mudt.hat.corr,
EBLUP3=mudt.hat.2corr,
MSE.Direct=sqrt(sigma2edt), MSE.EBLUP1=sqrt.mse,
MSE.EBLUP2=sqrt.mse.corr, MSE.EBLUP3=sqrt.mse.2corr),
file="EBLUP_Example.txt", row.names=FALSE, sep="\t")

write.table(data.frame(names(data)[4:(ncol(data)-2)],
beta1=beta1.hat, Std.error.beta1=Int[[4]],
beta2=beta2.hat, Std.error.beta2=Int.corr[[5]],
beta3=beta3.hat, Std.error.beta3=Int.corr[[7]]),
file="beta_Example.txt",
row.names=FALSE, sep="\t")
```

```
write.table(data.frame(data[,1:3], udt1=udt.hat,
udt2=udt.hat.corr, udt3=udt.hat.2corr),
file="u_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(data[,1:3], res1=residual,
res2=residual.corr, res3=residual.2corr),
file="res_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(Int.sigma,
Int.corr.sigma,Int.2corr.sigma),
file="IntervalSigma_Example.txt",
row.names=c("Sigma.A","Sigma.B","Sigma.Dif"), sep="\t")

write.table(data.frame(Int.corr.rho,
Int.2corr.rho),
file="IntervalRho_Example.txt",
row.names=c("Rho.A","Rho.B","Rho.Dif"), sep="\t")

write.table(data.frame(names(data)[4:(ncol(data)-2)],
beta1=beta1.hat, pvalueB1=pvalueB,
beta2=beta2.hat, pvalueB2=pvalueB.corr,
beta3=beta3.hat, pvalueB3=pvalueB.2corr),
file="pvalueB_Example.txt",
row.names=FALSE, sep="\t")

write.table(data.frame(Henderson3),
file="H3_Example.txt",
row.names=FALSE, sep="\t")

rm(list=ls(all=TRUE))
```

4.4.3 Outputs

Outputs of model 4.1.1 are labeled with ”1”, outputs of model 4.2.1 are labeled with ”2” and outputs of model 4.3.1 are labeled with ”3”. The resulting outputs appear in the files *EBLUP_Example.txt, beta_Example.txt, u_Example.txt, res_Example.txt, pvalueB_Example.txt, IntervalSigma_Example.txt, IntervalRho_Example.txt* and *H3_Example.txt* they are:

```
"EBLUP_Example.txt" output:
"Domain" "Sex" "Time" "Direct"     "EBLUP1"      "EBLUP2"      "EBLUP3"
1      1      1    0.099870796  0.152614368  0.127916117  0.114977231
1      1      2    0.255527036  0.290432840  0.272156258  0.285802917
1      1      3    0.758100324  0.746876535  0.752181931  0.744941436
2      1      1    0.531886402  0.513658094  0.515653727  0.542515091
2      1      2    0.975476720  0.939148577  0.953496558  0.953680974
2      1      3    0.562169587  0.549501831  0.550903074  0.570483977
3      1      1    0.720967375  0.685926683  0.698545296  0.700841842
3      1      2    0.494603026  0.493947008  0.481668287  0.541926025
3      1      3    0.924517608  0.852726890  0.882428250  0.876267170
4      1      1    0.533091894  0.530599817  0.530694729  0.528425152
4      1      2    0.517926099  0.518707153  0.518305720  0.514850823
4      1      3    0.038331778  0.038426115  0.038376233  0.038381315
5      1      1    0.399109186  0.402338902  0.395498090  0.428220755
5      1      2    0.943433632  0.895286392  0.918590095  0.883639181
5      1      3    0.254052932  0.255985290  0.254571503  0.257710756
1      2      1    0.305351622  0.320926649  0.321256497  0.333622361
1      2      2    0.060847784  0.120481639  0.125071712  0.129789433
1      2      3    0.344023489  0.345624042  0.346288016  0.343575087
2      2      1    0.389850382  0.391358287  0.394615443  0.394891700
2      2      2    0.105179702  0.108260763  0.108494570  0.110501895
2      2      3    0.163296232  0.176252387  0.177623855  0.187675335
3      2      1    0.899385519  0.851253098  0.853542941  0.845253727
3      2      2    0.019623444  0.049665972  0.042584135  0.025930608
3      2      3    0.724542241  0.719676099  0.720056244  0.722114156
4      2      1    0.474116093  0.473430662  0.473562030  0.473242084
4      2      2    0.208451174  0.260754741  0.242366051  0.163032156
4      2      3    0.994509978  0.904156184  0.903908084  0.867485580
5      2      1    0.546367453  0.524947338  0.523377246  0.495941273
5      2      2    0.277008991  0.294427599  0.293347978  0.279322437
5      2      3    0.228748126  0.263874382  0.263763777  0.259101937


"MSE.Direct"     "MSE.EBLUP1"     "MSE.EBLUP2"     "MSE.EBLUP3"
0.119029262      0.180635453      0.116359596      0.116243999
0.113844949      0.180155009      0.114699285      0.110389357
0.064514544      0.179133305      0.064181651      0.064062787
0.126116828      0.181146006      0.122989431      0.123179741
0.084854143      0.179088827      0.083873601      0.084110071
```

```
0.100081188     0.179178693     0.098325969     0.098358056
0.129532299     0.181587150     0.126224460     0.125856788
0.141791008     0.184115882     0.137650861     0.136412299
0.130209887     0.181585600     0.126311857     0.126494619
0.073932122     0.178953199     0.073232087     0.073262760
0.053139537     0.179359694     0.060558383     0.052976042
0.004368394     0.180324717     0.016337274     0.004368301
0.107054323     0.179527043     0.107076234     0.104725932
0.090668250     0.179059981     0.090808401     0.089053306
0.029475968     0.179951862     0.034754711     0.029432297
0.086735762     0.180397793     0.085082127     0.085186164
0.138478300     0.184902570     0.130758919     0.133274843
0.051549158     0.180339436     0.051187048     0.051451955
0.099894276     0.181048526     0.097884799     0.098806111
0.028282436     0.180699037     0.028211816     0.028353783
0.065035634     0.180197817     0.064203930     0.064895732
0.098876495     0.181011824     0.096891708     0.097553387
0.077046519     0.180271503     0.075890337     0.078185615
0.041280422     0.180484442     0.041054896     0.041257536
0.028820180     0.180692136     0.028767854     0.028777266
0.146445095     0.185612374     0.136567553     0.139181069
0.127295735     0.183071018     0.120545008     0.120796946
0.151777192     0.186221157     0.141603731     0.140107524
0.106048445     0.181379300     0.102659785     0.103839843
0.136267739     0.184334269     0.128480688     0.129830889

"beta_Example.txt" output:
"names.data..4..ncol.data....2.."
            "beta1"         "Std.error.beta1"
"ones"      0.556234163     0.257665646
"X1"       -0.056247243     0.401666172
"X2"       -0.150277601     0.354324034


            "beta2"         "Std.error.beta2"
"ones"      0.512872052     0.310712624
"X1"       -0.052549289     0.468731475
"X2"       -0.113831846     0.417125007
            "beta3"         "Std.error.beta3"
"ones"      0.571756002     0.823584675
```

```
"X1"      -0.118129914     0.416182186
"X2"      -0.286495353     0.303293162
"u_Example.txt" output:
"Domain" "Sex" "Time"   "udt1"          "udt2"          "udt3"
1         1     1       -0.309711971    -0.306034669    -0.269213132
1         1     2       -0.224060887    -0.206947170    -0.204019695
1         1     3       0.224346950     0.265403784     0.238058769
2         1     1       0.095344624     0.110662289     0.237432830
2         1     2       0.419752239     0.474457675     0.458652700
2         1     3       0.105218159     0.128176330     0.218013502
3         1     1       0.173745060     0.221132715     0.215360790
3         1     2       0.002714656     0.026314441     0.103167935
3         1     3       0.352270526     0.413679610     0.412901060
4         1     1       0.037930718     0.071317310     0.083710922
4         1     2       -0.023011318    0.018422519     -0.027046940
4         1     3       -0.411275403    -0.393724075    -0.330193535
5         1     1       -0.023445085    -0.013993552    0.110166161
5         1     2       0.487255622     0.522341779     0.599206097
5         1     3       -0.185032215    -0.167199005    -0.090196488
1         2     1       -0.181462566    -0.147100443    -0.131394663
1         2     2       -0.272573774    -0.257062627    -0.123057557
1         2     3       -0.052793615    -0.039633770    0.080805964
2         2     1       -0.013244863    -0.000185388    0.115734019
2         2     2       -0.337614695    -0.315923412    -0.245535330
2         2     3       -0.268489933    -0.247991248    -0.168454406
3         2     1       0.431525245     0.452628602     0.543117758
3         2     2       -0.443593377    -0.416108700    -0.418668569
3         2     3       0.250294385     0.277498042     0.320896736
4         2     1       0.072331063     0.083245218     0.202769575
4         2     2       -0.213764843    -0.206344698    -0.250458073
4         2     3       0.488734864     0.502584415     0.569535864
5         2     1       0.081501087     0.099843734     0.143487848
5         2     2       -0.135756138    -0.115939063    -0.043243118
5         2     3       -0.165806053    -0.145730162    -0.063148702

"res_Example.txt" output:
"Domain" "Sex" "Time"   "res1"          "res2"           "res3"
1         1     1       -0.052743572    -0.028045322    -0.015106436
1         1     2       -0.034905805    -0.016629222    -0.030275881
1         1     3       0.011223789     0.005918394     0.013158888
```

```
2          1    1        0.018228308     0.016232675     -0.010628689
2          1    2        0.036328143     0.021980161     0.021795745
2          1    3        0.012667755     0.011266513     -0.008314390
3          1    1        0.035040692     0.022422080     0.020125534
3          1    2        0.000656019     0.012934740     -0.047322999
3          1    3        0.071790718     0.042089358     0.048250438
4          1    1        0.002492077     0.002397165     0.004666742
4          1    2        -0.000781054    -0.000379622    0.003075276
4          1    3        -0.000094337    -0.000044455    -0.000049537
5          1    1        -0.003229716    0.003611097     -0.029111569
5          1    2        0.048147240     0.024843536     0.059794451
5          1    3        -0.001932358    -0.000518571    -0.003657824
1          2    1        -0.015575027    -0.015904874    -0.028270739
1          2    2        -0.059633855    -0.064223928    -0.068941649
1          2    3        -0.001600552    -0.002264527    0.000448402
2          2    1        -0.001507905    -0.004765061    -0.005041318
2          2    2        -0.003081061    -0.003314867    -0.005322193
2          2    3        -0.012956155    -0.014327623    -0.024379103
3          2    1        0.048132421     0.045842579     0.054131792
3          2    2        -0.030042528    -0.022960691    -0.006307164
3          2    3        0.004866141     0.004485997     0.002428085
4          2    1        0.000685431     0.000554063     0.000874009
4          2    2        -0.052303567    -0.033914877    0.045419018
4          2    3        0.090353795     0.090601895     0.127024398
5          2    1        0.021420115     0.022990208     0.050426181
5          2    2        -0.017418609    -0.016338987    -0.002313446
5          2    3        -0.035126255    -0.035015650    -0.030353810
```

```
"pvalueB_Example.txt" output:
"names.data..4..ncol.data....2.."
          "beta1"           "pvalueB1"
"ones"    0.556234163       0.000384036
"X1"      -0.056247243      0.817830105
"X2"      -0.150277601      0.485412886

          "beta2"           "pvalueB2"
"ones"    0.512872052       0.006626632
"X1"      -0.052549289      0.853696648
"X2"      -0.113831846      0.653522287
```

```
         "beta3"         "pvalueB3"
"ones"   0.571756002     0.002727025
"X1"     -0.118129914    0.604153562
"X2"     -0.286495353    0.197796129

"IntervalSigma_Example.txt" output:
              "MinSigma1"     "MaxSigma1"     "TestSigma1"
"Sigma.A"     0.022699847     0.143719103     FALSE
"Sigma.B"     0.025342077     0.149932097     FALSE
"Sigma.Dif"   -0.091272781    0.082417556     TRUE

              "MinSigma2"     "MaxSigma2"     "TestSigma2"
"Sigma.A"     0.053112308     0.283172506     FALSE
"Sigma.B"     0.022878204     0.147375206     FALSE
"Sigma.Dif"   -0.049557312    0.215588716     TRUE

              "MinSigma3"     "MaxSigma3"     "TestSigma3"
"Sigma.A"     0.025191372     0.214743291     FALSE
"Sigma.B"     0.003598742     0.104415376     FALSE
"Sigma.Dif"   -0.043800767    0.175721313     TRUE

"IntervalRho_Example.txt" output:
            "MinRho2"         "MaxRho2"       "TestRho2"
"Rho.A"       -0.565132787    0.309142370     TRUE
"Rho.B"       -0.565132787    0.309142370     TRUE
"Rho.Dif"     -0.565132787    0.309142370     TRUE

            "MinRho3"         "MaxRho3"       "TestRho3"
"Rho.A"       0.357545902     1.027632142     FALSE
"Rho.B"       -1.060453795    -0.105485577    FALSE
"Rho.Dif"     0.695466037     1.855651378     FALSE

"H3_Example.txt" output:
"Henderson3"
0.0747886698641487
```

Chapter 5

Area-level spatio-temporal models

5.1 Model A

Let y_{dt} be a direct estimator of the target population parameter and let $\mathbf{x}_{dt}$ be a vector containing the aggregated values of p auxiliary variables. Subindexes d and t are used for domains and time instants respectively. Let us consider the model

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{1d} + u_{2dt} + e_{dt}, \quad d = 1,\dots,D, \quad t = 1,\dots,T, \tag{5.1}$$

where $\{u_{1d}\}$, $\{u_{2dt}\}$ y $\{e_{dt}\}$ are independent with distributions $\{u_{1d}\}_{d=1}^{D} \sim SAR(1)$, $\{u_{2dt}\}$ i.i.d $N(0,\sigma_2^2)$ and $e_{dt} \sim N(0,\sigma_{dt}^2)$. Model (5.1) can alternatively written as

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{e}, \tag{5.2}$$

where

- $\mathbf{y} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(y_{dt}))$, $\quad \mathbf{e} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(e_{dt}))$,
- $\mathbf{u}_1 = \underset{1\le d\le D}{\text{col}}(u_{1d}), \mathbf{u}_2 = \underset{1\le d\le D}{\text{col}}(\mathbf{u}_{2d}), \mathbf{u}_{2d} = \underset{1\le t\le T}{\text{col}}(u_{2dt})$,
- $\mathbf{X} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(\mathbf{x}_{dt})), \mathbf{x}_{dt} = \underset{1\le j\le p}{\text{col}'}(x_{dtj}), \beta = \underset{1\le j\le p}{\text{col}}(\beta_j)$,
- $\mathbf{Z}_1 = \underset{1\le d\le D}{\text{diag}}(\mathbf{1}_T), \mathbf{Z}_2 = \mathbf{I}_{M\times M}, M = DT$.

We assume that $\mathbf{u}_1 \sim N(\mathbf{0},\mathbf{V}_{u_1})$, $\mathbf{u}_2 \sim N(\mathbf{0},\mathbf{V}_{u_2})$ and $\mathbf{e} \sim N(\mathbf{0},\mathbf{V}_e)$ are independent with covariance matrices

$$\begin{aligned}
\mathbf{V}_{u_1} &= \sigma_1^2\Omega_1(\rho_1), \quad \Omega_1(\rho_1) = \left[(\mathbf{I}_D-\rho_1\mathbf{W})'(\mathbf{I}_D-\rho_1\mathbf{W})\right]^{-1} \triangleq \mathbf{C}^{-1}(\rho_1),\\
\mathbf{V}_{u_2} &= \sigma_2^2\mathbf{I}_{DT},\\
\mathbf{V}_e &= \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_{ed}), \quad \mathbf{V}_{ed} = \underset{1\le t\le T}{\text{diag}}(\sigma_{dt}^2),
\end{aligned}$$

and known σ_{dt}^2's. We assume that the rows of the proximity matrix $\mathbf{W}$ are stochastic vectors, i.e. with components summing up to one. The vector $\mathbf{u}_1$ is distributed according to a stochastic process SAR(1) and the variables $\mathbf{u}_{2dt}$ are i.i.d. normal. Therefore, the variance of $\mathbf{y}$ is

$$\text{var}(\mathbf{y}) = \mathbf{V} = \mathbf{Z}_1\mathbf{V}_{u_1}\mathbf{Z}_1' + \mathbf{Z}_2\mathbf{V}_{u_2}\mathbf{Z}_2' + \mathbf{V}_e = \mathbf{Z}_1\mathbf{V}_{u_1}\mathbf{Z}_1' + \underset{1\le d\le D}{\text{diag}}\,(\sigma_2^2\mathbf{I}_T + \mathbf{V}_{ed}).$$

The BLU estimators and predictors of β and $\mathbf{u}$ are

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y} \qquad \text{and} \qquad \widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y} - \mathbf{X}\widehat{\beta}),$$

where $\mathbf{V}_u = \text{diag}(\mathbf{V}_{u_1}, \mathbf{V}_{u_2})$ and $\mathbf{Z} = (\mathbf{Z}_1, \mathbf{Z}_2)$.

Let us define $\theta = (\theta_1, \theta_2, \theta_3) = (\sigma_1^2, \rho_1, \sigma_2^2)$, $\mathbf{P} = \mathbf{V}^{-1} - \mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}$ and

$$\mathbf{P}_a = \frac{\partial \mathbf{P}}{\partial \theta_a} = -\mathbf{P}\frac{\partial \mathbf{V}}{\partial \theta_a}\mathbf{P} = -\mathbf{P}\mathbf{V}_a\mathbf{P}, \quad a = 1,2,3.$$

The updating formula of the Fisher-scoring algorithm for calculating the RENL estimates of the variance components is

$$\theta^{k+1} = \theta^k + \mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k).$$

where the components of the score vector are

$$S_a = -\frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a) + \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{y}, \quad a = 1,2,3.$$

and the elements of the Fisher information matrix are

$$F_{ab} = \frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{V}_b), \quad a,b = 1,2,3.$$

This algorithm requires starting values of θ (seeds). We may obtain seeds by considering the model without $\mathbf{u}_1$ and with $\rho_2 = 0$. For this last model we might consider the Henderson 3 estimator $\widehat{\sigma}_{u_2H}^2$ of the only remaining variance σ_2^2. Therefore, we might propose the following seeds: $\sigma_1^{2(0)} = \sigma_2^{2(0)} = \frac{1}{2}\widehat{\sigma}_{u_2H}^2$, $\rho_1^{(0)} = 0.3$.

The REML estimator of β is

$$\widehat{\beta} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators θ and β are

$$\hat{\theta} \sim N_2(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_j are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2},\ a = 1,\ldots,3, \quad \hat{\beta}_j \pm z_{\alpha/2}q_{jj}^{1/2},\ j = 1,\ldots,p,$$

where $\hat{\theta} = \theta^\kappa$, $\mathbf{F}^{-1}(\theta^\kappa) = (\nu_{ab})_{a,b=1,\ldots,3}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^\kappa)\mathbf{X})^{-1} = (q_{ij})_{i,j=1,\ldots,p}$, κ is the final iteration of the Fisher-scoring algorithm and z_α is the α-quantil of the standard normal distribution $N(0,1)$. Observed $\hat{\beta}_j = \beta_0$, the p-value for testing the test of hypothesis $H_0: \beta_j = 0$ is

$$p = 2P_{H_0}(\hat{\beta}_j > |\beta_0|) = 2P(N(0,1) > \beta_0/\sqrt{q_{jj}}).$$

5.2 Model B

Let y_{dt} be a direct estimator of the characteristic of interest and let $\mathbf{x}_{dt}$ be a vector containing the aggregated values of p of auxiliary variables. The subindex d is used for domains and the subindex t for time instants. Let us consider the model

$$y_{dt} = \mathbf{x}_{dt}\beta + u_{1d} + u_{2dt} + e_{dt}, \quad d = 1,\dots,D, \quad t = 1,\dots,T, \tag{5.3}$$

where $\{u_{1d}\}$, $\{u_{2dt}\}$ and $\{e_{dt}\}$ are independent with distributions $\{u_{1d}\}_{d=1}^{D} \sim SAR(1)$, $\{u_{2dt}\}_{t=1}^{T}$ i.i.d $AR(1)$ and $e_{dt} \sim N(0,\sigma_{dt}^2)$.

The model (5.3) can be alternatively written in the form

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{e}, \tag{5.4}$$

where

- $\mathbf{y} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(y_{dt}))$, $\quad \mathbf{e} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(e_{dt}))$,
- $\mathbf{u}_1 = \underset{1\le d\le D}{\text{col}}(u_{1d}), \mathbf{u}_2 = \underset{1\le d\le D}{\text{col}}(\mathbf{u}_{2d}), \mathbf{u}_{2d} = \underset{1\le t\le T}{\text{col}}(u_{2dt})$,
- $\mathbf{X} = \underset{1\le d\le D}{\text{col}}(\underset{1\le t\le T}{\text{col}}(\mathbf{x}_{dt})), \mathbf{x}_{dt} = \underset{1\le j\le p}{\text{col}'}(x_{dtj}), \beta = \underset{1\le j\le p}{\text{col}}(\beta_j)$,
- $\mathbf{Z}_1 = \underset{1\le d\le D}{\text{diag}}(\mathbf{1}_T), \mathbf{Z}_2 = \mathbf{I}_{M\times M}, M = DT$.

We assume that $\mathbf{u}_1 \sim N(\mathbf{0},\mathbf{V}_{u_1})$, $\mathbf{u}_2 \sim N(\mathbf{0},\mathbf{V}_{u_2})$ and $\mathbf{e} \sim N(\mathbf{0},\mathbf{V}_e)$ are independent with covariance matrices

$$\begin{aligned}
\mathbf{V}_{u_1} &= \sigma_1^2\Omega_1(\rho_1), \quad \Omega_1(\rho_1) = \left[(\mathbf{I}_D-\rho_1\mathbf{W})'(\mathbf{I}_D-\rho_1\mathbf{W})\right]^{-1} \triangleq \mathbf{C}^{-1}(\rho_1),\\
\mathbf{V}_{u_2} &= \sigma_2^2\Omega_2(\rho_2), \quad \Omega_2(\rho_2) = \underset{1\le d\le D}{\text{diag}}(\Omega_{2d}(\rho_2)),\\
\mathbf{V}_e &= \underset{1\le d\le D}{\text{diag}}(\mathbf{V}_{ed}), \quad \mathbf{V}_{ed} = \underset{1\le t\le T}{\text{diag}}(\sigma_{dt}^2),\\
\Omega_{2d} &= \Omega_{2d}(\rho_2) = \frac{1}{1-\rho_2^2}\begin{pmatrix} 1 & \rho_2 & \dots & \rho_2^{T-2} & \rho_2^{T-1}\\ \rho_2 & 1 & \ddots & & \rho_2^{T-2}\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ \rho_2^{T-2} & & \ddots & 1 & \rho_2\\ \rho_2^{T-1} & \rho_2^{T-2} & \dots & \rho_2 & 1 \end{pmatrix}_{T\times T},
\end{aligned}$$

where the σ_{dt}^2's are known. We assume that the rows of matrix $\mathbf{W}$ are stochastic vectors, i.e. their components sum up to one. The vector $\mathbf{u}_1$ is distributed as a SAR(1) stochastic process and the vectors $\mathbf{u}_{2d}$ are independent with homogeneous AR(1) distributions (they all have the same variance and autocorrelation parameters). The variance of $\mathbf{y}$ is

$$\text{var}(\mathbf{y}) = \mathbf{V} = \mathbf{Z}_1\mathbf{V}_{u_1}\mathbf{Z}_1' + \mathbf{Z}_2\mathbf{V}_{u_2}\mathbf{Z}_2' + \mathbf{V}_e = \mathbf{Z}_1\mathbf{V}_{u_1}\mathbf{Z}_1' + \underset{1\le d\le D}{\text{diag}}(\sigma_2^2\Omega_{2d}(\rho_2) + \mathbf{V}_{ed}).$$

The BLU estimator and predictor of β and $\mathbf{u}$ are

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y} \qquad \text{and} \qquad \widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\beta}),$$

where $\mathbf{V}_u = \text{diag}(\mathbf{V}_{u_1}, \mathbf{V}_{u_2})$ and $\mathbf{Z} = (\mathbf{Z}_1, \mathbf{Z}_2)$.

Let us define $\theta = (\theta_1, \theta_2, \theta_3, \theta_4) = (\sigma_1^2, \rho_1, \sigma_2^2, \rho_2)$, $\mathbf{P} = \mathbf{V}^{-1} - \mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}$ and Then

$$\mathbf{P}_a = \frac{\partial \mathbf{P}}{\partial \theta_a} = -\mathbf{P}\frac{\partial \mathbf{V}}{\partial \theta_a}\mathbf{P} = -\mathbf{P}\mathbf{V}_a\mathbf{P}, \quad a = 1, \dots, 4.$$

The updating formula of the Fisher-scoring algorithm for calculating the REML estimates of the variance components is

$$\theta^{k+1} = \theta^k + \mathbf{F}^{-1}(\theta^k)\mathbf{S}(\theta^k).$$

where the components of the score vector are

$$S_a = \frac{\partial l_{REML}}{\partial \theta_a} = -\frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a) + \frac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{y}, \quad a = 1, \dots, 4.$$

and the elements of the Fisher information matrix are

$$F_{ab} = \frac{1}{2}\text{tr}(\mathbf{P}\mathbf{V}_a\mathbf{P}\mathbf{V}_b), \quad a, b = 1, \dots, 4.$$

We can take the reduced model without $\mathbf{u}_1$ and with $\rho_2 = 0$ as a reference for obtaining seeds for the Fisher-scoring algorithm. For the mentioned reduced model, it is easy to calculate the Henderson 3 estimator $\widehat{\sigma}_{u_2H}^2$ of the only remaining variance σ_2^2. Therefore, a possible set of algorithm seeds is $\sigma_1^{2(0)} = \sigma_2^{2(0)} = \frac{1}{2}\widehat{\sigma}_{u_2H}^2$, $\rho_1^{(0)} = \rho_2^{(0)} = 0.3$.

The REML estimator of β is

$$\widehat{\beta} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-1}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}.$$

The asymptotic distributions of the REML estimators of θ and β are

$$\hat{\theta} \sim N_2(\theta, \mathbf{F}^{-1}(\theta)), \quad \hat{\beta} \sim N_p(\beta, (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}).$$

Asymptotic confidence intervals at the level $1-\alpha$ for θ_a and β_j are

$$\hat{\theta}_a \pm z_{\alpha/2}\nu_{aa}^{1/2}, \; a = 1, \dots, 4, \quad \hat{\beta}_j \pm z_{\alpha/2}q_{jj}^{1/2}, \; j = 1, \dots, p,$$

where $\hat{\theta} = \theta^\kappa$, $\mathbf{F}^{-1}(\theta^\kappa) = (\nu_{ab})_{a,b=1,\dots,4}$, $(\mathbf{X}'\mathbf{V}^{-1}(\theta^\kappa)\mathbf{X})^{-1} = (q_{ij})_{i,j=1,\dots,p}$, κ is the last iteration in the Fisher-scoring algorithm and z_α is the α-quantil of the standard normal distribution $N(0,1)$. If we observe $\hat{\beta}_j = \beta_0$, the p-value for testing $H_0: \beta_j = 0$ is

$$p = 2P_{H_0}(\hat{\beta}_j > |\beta_0|) = 2P(N(0,1) > \beta_0/\sqrt{q_{jj}}).$$

5.3 The Software: description of R functions

This section describes the R functions implemented for fitting the area-level spatio-temporal models with random effects following a SAR process defined in (5.1) and (5.3) with uncorrelated and correlated time effects (models A and B, respectively). An example showing the use of these functions is given in the next section and full R codes are included in Appendix INDICAR REFERENCIA. The R package MASS must be loaded to use the following software.

5.3.1 fitSpatioTemporalFH

This function fits the area-level spatio-temporal model with random effects following a SAR process, using REML fitting method, for models A and B defined in Sections 5.1 and 5.2 respectively. The function is defined as

```
FitSpatioTemporalFH <- function(model,X,y,nD,nT,sigma2dt,theta0,
                                       W,MAXITER,PRECISION,confidence)
```

The arguments are:

- `model`: type of model to be chosen between ”A” or ”B”.
- `X`: matrix with dimension $(\texttt{nD} \times \texttt{nT}) \times p$ containing the aggregated (population) values of p auxiliary variables. The elements in first column might be equal to 1 if the model includes an intercept, in this case p is the number of auxiliary variables plus 1. This matrix is ordered by area and time instant.
- `y`: column vector with size $(\texttt{nD} \times \texttt{nT})$ containing the direct estimates of the response variable for each area $d = 1,...,\texttt{nD}$ and time instant $t = 1,...,\texttt{nT}$. This vector is ordered by area and time instant.
- `nD`: total number of areas.
- `nT`: total number of time instants.
- `sigma2dt`: column vector with size $(\texttt{nD} \times \texttt{nT})$ containing the sampling variances σ^2_{dt} of each area $d = 1,...,\texttt{nD}$ and time instant $t = 1,...,\texttt{nT}$. This vector is ordered by area and time instant.
- `theta0`: vector with the values of the parameters $\theta = (\sigma_1^2, \rho_1, \sigma_2^2)$ for model A or $\theta = (\sigma_1^2, \rho_1, \sigma_2^2, \rho_2)$ for model B.
- `W`: $\texttt{nD} \times \texttt{nD}$ proximity matrix with rows adding to 1 or 0, if there are no spatial influence in any area. This matrix is ordered by area.
- `MAXITER`: maximum number of iterations allowed for the Fisher-scoring algorithm.
- `PRECISION`: convergence tolerance limit for the Fisher-scoring algorithm.

- `confidence`: value of the confidence interval of level $1-\alpha$ to calculate the confidence intervals of θ and β and p-values of the auxiliary variables.

The function returns a list with the following objects:

- `model`: type of model ”A” or ”B”.
- `convergence`: a logical value equal to TRUE if Fisher-scoring algorithm converges in less than MAXITER iterations.
- `iterations`: number of iterations performed by the Fisher-scoring algorithm.
- `validtheta`: a logical value equal to TRUE if parameters $\theta = (\sigma_1^2, \rho_1, \sigma_2^2)$ for model A or $\theta = (\sigma_1^2, \rho_1, \sigma_2^2, \rho_2)$ for model B are valids, that is, $\sigma_1^2 > 0, \rho_1 \in [-1,1], \sigma_2^2 > 0$ and $\rho_2 \in [-1,1]$.
- `theta`: data.frame in the shape of a table with the estimated parameters θ in first column and their asymptotic standard erros in second column.
- `beta`: data.frame in the shape of a table with the estimated model coefficients in first column, their asymptotic standard errors in second column, the Z statistics in third column and the p-values of the significance of each coefficient in last column.
- `goodnessoffit`: a vector containing three different goodness-of-fit measures, namely the log-likehood, the AIC and the BIC.
- `estimates`: a column vector with size $(\texttt{nD} \times \texttt{nT})$ with the values of the SEBA or SEBB estimates for the `nD` areas and `nT` time instants.

5.3.2 BootMSE.SpatioTemporalFH

This function gives parametric bootstrap MSE estimates of the spatio-temporal Fay-Herriot models A and B. The function is defined as

```
BootMSE.SpatioTemporalFH <- function(model,nB,Xdt,nD,nT,sigma2dt,beta,
                                     theta,rho1_0_b,rho2_0_b,
                                     W,MAXITER,PRECISION,confidence)
```

The arguments are:

- `model`: type of model to be chosen between ”A” or ”B”.
- `nB`: total number of bootstrap replicates.
- `Xdt`: matrix with dimension $(\texttt{nD} \times \texttt{nT}) \times p$ containing the aggregated (population) values of p auxiliary variables. The elements in first column might be equal to 1 if the model includes an intercept, in this case p is the number of auxiliary variables plus 1. This matrix is ordered by area and time instant.

- `nD`: total number of areas.
- `nT`: total number of time instants.
- `sigma2dt`: column vector with size ($\texttt{nD} \times \texttt{nT}$) containing the sampling variances σ^2_{dt} of each area $d = 1, ..., \texttt{nD}$ and time instant $t = 1, ..., \texttt{nT}$. This vector is ordered by area and time instant.
- `beta`: vector with size p containing the estimated regression coefficients β, obtained by applying the function `fitSpatioTemporalFH` to the original sample.
- `theta`: vector with the values of the estimated parameters $\theta = (\sigma_1^2, \rho_1, \sigma_2^2)$ for model A or $\theta = (\sigma_1^2, \rho_1, \sigma_2^2, \rho_2)$ for model B, obtained by applying the function `fitSpatioTemporalFH` to the original sample.
- `rho1_0_b`: initial value of parameter ρ_1 to fit the bootstrap samples by applying the funtion `fitSpatioTemporalFH`.
- `rho2_0_b`: initial value of parameter ρ_2 to fit the bootstrap samples by applying the funtion `fitSpatioTemporalFH`.
- `W`: $\texttt{nD} \times \texttt{nD}$ proximity matrix with rows adding to 1 or 0, if there are no spatial influence in any area. This matrix is ordered by area.
- `MAXITER`: maximum number of iterations allowed for the Fisher-scoring algorithm.
- `PRECISION`: convergence tolerance limit for the Fisher-scoring algorithm.
- `confidence`: value of the confidence interval of level $1 - \alpha$ to calculate the confidence intervals of θ and β and p-values of the auxiliary variables.

The function returns:

- `msedt`: a vector with size $\texttt{nD} \times \texttt{nT}$ containing the parametric bootstrap mean squared errors for the `nD` areas and `nT` time instants.

5.3.3 Auxiliary functions

In this section are described three auxiliary functions used by the previous ones.

Henderson

This function calculates the Henderson 3 estimator of σ^2. The function is defined as

```
Henderson <- function(X,y,sigma2dt)
```

The arguments are:

- `X`: matrix with dimension $(\texttt{nD} \times \texttt{nT}) \times p$ containing the aggregated (population) values of p auxiliary variables. The elements in first column might be equal to 1 if the model includes an intercept, in this case p is the number of auxiliary variables plus 1. This matrix is ordered by area and time instant.
- `y`: column vector with size $(\texttt{nD} \times \texttt{nT})$ containing the direct estimates of the response variable for each area $d = 1,...,\texttt{nD}$ and time instant $t = 1,...,\texttt{nT}$. This vector is ordered by area and time instant.
- `sigma2dt`: column vector with size $(\texttt{nD} \times \texttt{nT})$ containing the sampling variances σ_{dt}^2 of each area $d = 1,...,\texttt{nD}$ and time instant $t = 1,...,\texttt{nT}$. This vector is ordered by area and time instant.

The function returns:

- `sigma2`: value of the Henderson estimate of σ^2.

MessageErrorFitting

This function prints an error message according to the values obtained by de Fisher-scoring algorithm. The function is defined as

```
MessageErrorFitting <- function(model,nsample,convergence,niter,
                                validtheta,theta)
```

The arguments are:

- `model`: type of model to be chosen between ”A” or ”B”.
- `nsample`: number of the sample.
- `convergence`: a logical value equal to TRUE if Fisher-scoring algorithm converges in less than MAXITER iterations. This argument is calculated by the function `FitSpatioTemporalFH`.
- `niter`: number of iterations the performed by the Fisher-scoring algorithm. This argument is calculated by the function `FitSpatioTemporalFH`.
- `validtheta`: a logical value equal to TRUE if the values of `theta` are valids. This argument is calculated by the function `FitSpatioTemporalFH`.
- `theta`: estimated parameters θ. This argument is calculated by the function `FitSpatioTemporalFH`.

diagonalizematrix

Let A be a matrix with dimension $n \times m$ and $ntimes$ a number. This funcion generates a new matrix $Adiag = \underset{1,...,ntimes}{\text{diag}}(A)$ with dimension $(n \times ntimes) \times (m \times ntimes)$. The function is defined as

```
diagonalizematrix <- function(A,ntimes)
```

The arguments are:

- `A`: a matrix.
- `ntimes`: number of times.

The function returns:

- `Adiag`.

5.4 Examples of usage of R functions

This section shows how to use the R functions described in Section 5.3 to produce small area SEBA and SEBB estimators along with their corresponding estimated MSE, based on the basic Fay-Herriot model.

5.4.1 Example data set

Table 5.4.1.1. presents the data set used in the example. There are 12 areas, 3 time instants, 2 auxiliary variables $X1$ and $X2$ for each domain and time period. Dependent variable is labeled by Y and the error variances by Var. There are 36 observations. The data are sorted by area and time instants.

Area	*Time*	*Intercept*	$X1$	$X2$	Y	Var
2	1	1	0.502853	0.415588	0.303105	0.000728
2	2	1	0.510154	0.41998	0.250032	0.000593
2	3	1	0.499364	0.416099	0.261484	0.000819
3	1	1	0.50432	0.442839	0.179701	0.00016
3	2	1	0.50544	0.429477	0.177524	0.000163
3	3	1	0.501952	0.443046	0.175358	0.000186
8	1	1	0.511051	0.459058	0.108384	3.3e-05
8	2	1	0.509457	0.490652	0.10549	3.6e-05
8	3	1	0.508517	0.488867	0.09623	3.4e-05
12	1	1	0.50068	0.453259	0.175914	0.000736
12	2	1	0.485724	0.478878	0.247768	0.001015
12	3	1	0.510825	0.474373	0.12216	0.000716
13	1	1	0.513787	0.369786	0.309139	0.000686
13	2	1	0.508677	0.384735	0.341113	0.000677
13	3	1	0.506475	0.401539	0.294176	0.000672
16	1	1	0.498238	0.370316	0.400297	0.002473
16	2	1	0.507197	0.372423	0.342267	0.002168
16	3	1	0.500335	0.402757	0.412106	0.003134
17	1	1	0.492315	0.520161	0.113042	0.000335
17	2	1	0.490202	0.502947	0.120774	0.000407
17	3	1	0.491647	0.519456	0.057924	0.000213

Table 5.4.1.1. Data set *DataExample*.

Area	*Time*	*Intercept*	*X*1	*X*2	*Y*	*Var*
20	1	1	0.503211	0.465848	0.109374	0.00018
20	2	1	0.511708	0.480571	0.1076	0.00024
20	3	1	0.502282	0.47654	0.082388	0.000203
25	1	1	0.491693	0.45535	0.163808	0.00071
25	2	1	0.507129	0.459151	0.184013	0.000909
25	3	1	0.499759	0.472744	0.209146	0.001039
43	1	1	0.498541	0.466408	0.146345	0.000382
43	2	1	0.491464	0.474924	0.181178	0.000683
43	3	1	0.491474	0.495993	0.148671	0.000551
45	1	1	0.489951	0.405406	0.21923	0.000323
45	2	1	0.473074	0.447659	0.270293	0.000484
45	3	1	0.488254	0.457184	0.234361	0.000501
46	1	1	0.505742	0.450181	0.186366	0.000123
46	2	1	0.506344	0.47181	0.188873	0.000129
46	3	1	0.50103	0.477279	0.137869	0.000149

Table 5.4.1.1. Data set *DataExample*.

The proximity matrix for the 12 areas is presented on Table 5.4.1.2. This matrix is ordered according to the area codes.

0	0	0	0	0.5	0.5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0.333333	0	0.333333	0.333333	0	0
0	0	0	0	0	0	0	0	0	0	0	1
0.333333	0	0	0	0	0.333333	0	0	0	0	0.333333	0
0.2	0	0	0	0.2	0	0	0	0	0	0.2	0
0	0	0.5	0	0	0	0	0	0.5	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.333333	0	0	0	0.333333	0	0	0.333333	0	0
0	0	0.5	0	0	0	0	0	0.5	0	0	0
0	0	0	0	0.5	0.5	0	0	0	0	0	0
0	0.5	0	0.5	0	0	0	0	0	0	0	0

Table 5.4.1.2. Proximity matrix *WExample*.

5.4.2 Example of R code for running functions

We include an example of how to read the data sets with the model variables and the proximity matrix and how to run the functions `FitSpatioTemporalFH` and `BootMSE.SpatioTemporalFH`. We also give suitable comments explaining what is done by each code line.

```
#################################################################
### Example to calculate de SEBA estimator and the MSE
###
### Author:    Yolanda Marhuenda (y.marhuenda@umh.es)
### File name: Example_ModelA.R
### Updated:   January 20th, 2011
###
#################################################################
```

```
rm(list=ls(all.names=TRUE))                  # delete previous objects

setwd("C:/PROYECTOS/SAMPLE/04_softwareSPFH") # set path where data sets
                                             # and functions are

# Load library "MASS" and files where functions are
library("MASS")
source("Henderson.R")
source("MessageErrorFitting.R")
source("FitSpatioTemporalFH.R")
source("BootMseSpatioTemporalFH.R")

# Set the seed for mvrnorm function used in BootMSE.SpatioTemporalFH
# function. This step is not necessary
set.seed(301)

# Define data input parameters
model <- "A"                            # model: "A" or "B"
file_data <- "DataExample.txt"          # data input filename
file_W    <- "WExample.txt"             # proximity matrix filename

                                        # Set the column numbers or labels
                                        # of variables included in
                                        # file_data used in the model

c_dom       <- 1                        # area column number
c_period    <- 2                        # time instant column number
c_varaux    <- c("Intercept","X1","X2") # auxiliary variables labels
c_ydt       <- 6                        # direct estimator column number
c_sigma2dt  <- 7                        # direct estimator variance column
                                        # number

nT          <- 3                        # total number of time instants
intconfidence <- 0.95                   # confidence interval to calculate
                                        # theta, beta and pvalue
MAXITER     <- 200                      # maximum number of iterations of
                                        # Fisher-scoring algorithm
PRECISION   <- 0.0001                   # convergence tolerance limit for
                                        # the Fisher-scoring algorithm
nB          <- 100                      # number of bootstrap replications

# Read input data
datos    <- read.table(file=file_data, header=TRUE)
X        <- as.matrix(datos[,c_varaux]) # auxiliary variables
ydt      <- datos[,c_ydt]               # y
sigma2dt <- datos[,c_sigma2dt]          # variance
nD       <- nrow(datos)/nT              # number of areas
W        <- as.matrix(read.table(file=file_W,header=FALSE)) # proximity
```

```
                                                                   # matrix

#Obtain initial values of theta
seedsigma <- Henderson(X,ydt,sigma2dt)
if (seedsigma<0)
{
   cat("Henderson sigma:", seedsigma,"<0 \n")
   seedsigma<-min(sigma2dt)
   cat("Warning: Henderson sigma is set to the minimum variance.",seedsigma)
   #quit()
}
sigma21 <- sigma22 <- 0.5*seedsigma
rho1 <- 0.3
rho2 <- 0.3                            # model B

if (model=="A")
   theta0 <- rbind(sigma21, rho1, sigma22)
if (model=="B")
   theta0 <- rbind(sigma21, rho1, sigma22, rho2)

# Fit the spatio-temporal Fay Herriot A or B
result <- FitSpatioTemporalFH(model,X,ydt,nD,nT,sigma2dt,theta0,W,MAXITER,
                                                   PRECISION,intconfidence)

# Validate output
if (result$convergence==FALSE || result$validtheta==FALSE)
{
   MessageErrorFitting(model,0,result$convergence,result$iterations,
                                         result$validtheta,result$theta)
   quit();
}

# Calculate mean squared error
mse <- BootMSE.SpatioTemporalFH(model,nB,X,nD,nT,sigma2dt,
                                result$beta[,"coef"],
                                result$theta[,"estimate"],rho1,rho2,
                                W,MAXITER,PRECISION,intconfidence)
```

5.4.3 Outputs

In this section the results obtained for both models are included.

```
> # Print results for model A
> result$model
[1] "A"

> result$convergence
[1] TRUE
```

```
> result$iterations
[1] 50

> result$validtheta
[1] TRUE

> result$theta
            estimate    std.error
sigma21 0.0007727209 0.0011394158
rho1    0.4895125122 0.6899497254
sigma22 0.0004774741 0.0004818322

> result$beta
               coef std.error    tvalue       pvalue greater.alfa
Intercept  1.896265 0.9690140  1.956902 1.253310e-04        FALSE
X1        -2.140172 1.7091045 -1.252218 1.411579e-02        FALSE
X2        -1.416665 0.5443026 -2.602716 3.374536e-07        FALSE

> result$goodnessoffit
   loglike        AIC        BIC
  66.41347 -120.82695 -111.32583

> # Print direct estimates, variance and SEBA or SEBB estimates, mse and
> # residuals of the last period nT.
> dom    <- datos[,c_dom]
> period <- datos[,c_period]
> output <- data.frame(Area=dom, Time=period, Direct=ydt,
+                       Model=result$estimates, VarDirect=sigma2dt,
+                       MSEModel=mse, residuals=ydt-result$estimates)
> print (output[output[,"Time"]==nT,], row.names=FALSE)
 Area Time     Direct      Model    VarDirect     MSEModel     residuals

 Area Time   Direct      Model VarDirect     MSEModel     residuals
    2    3 0.261484 0.27249167  0.000819 4.064448e-04 -1.100767e-02
    3    3 0.175358 0.17697742  0.000186 1.793776e-04 -1.619418e-03
    8    3 0.096230 0.09620245  0.000034 3.354029e-05  2.754681e-05
   12    3 0.122160 0.13472920  0.000716 4.808737e-04 -1.256920e-02
   13    3 0.294176 0.28997895  0.000672 3.929303e-04  4.197053e-03
   16    3 0.412106 0.31947306  0.003134 8.274163e-04  9.263294e-02
   17    3 0.057924 0.06817393  0.000213 1.652186e-04 -1.024993e-02
   20    3 0.082388 0.09157257  0.000203 1.398547e-04 -9.184566e-03
   25    3 0.209146 0.17453431  0.001039 3.857674e-04  3.461169e-02
   43    3 0.148671 0.14269041  0.000551 3.859249e-04  5.980593e-03
   45    3 0.234361 0.22154094  0.000501 3.782584e-04  1.282006e-02
   46    3 0.137869 0.14280839  0.000149 1.178671e-04 -4.939394e-03

> # Print results for model B
> result$model
[1] "B"
```

```
> result$convergence
[1] TRUE

> result$iterations
[1] 44

> result$validtheta
[1] TRUE

> result$theta
             estimate    std.error
sigma21  0.0008838728 0.0012274716
rho1     0.5024407276 0.6367657213
sigma22  0.0004001407 0.0005881695
rho2    -0.1540283111 1.3214335088

> result$beta
               coef std.error    tvalue       pvalue greater.alfa
Intercept  1.819170 0.9756079  1.864653 2.575311e-04        FALSE
X1        -2.048671 1.7166565 -1.193408 1.933357e-02        FALSE
X2        -1.348500 0.5491056 -2.455811 1.484572e-06        FALSE

> result$goodnessoffit
   loglike        AIC        BIC
  66.36243 -118.72487 -107.64024

> # Print direct estimates, variance and SEBA or SEBB estimates, mse and
> # residuals of the last period nT.
> dom    <- datos[,c_dom]
> period <- datos[,c_period]
> output <- data.frame(Area=dom, Time=period, Direct=ydt,
+                       Model=result$estimates, VarDirect=sigma2dt,
+                       MSEModel=mse, residuals=ydt-result$estimates)
> print (output[output[,"Time"]==nT,], row.names=FALSE)
 Area Time   Direct      Model VarDirect     MSEModel     residuals
    2    3 0.261484 0.27400397  0.000819 3.499487e-04 -0.0125199696
    3    3 0.175358 0.17709457  0.000186 1.723063e-04 -0.0017365728
    8    3 0.096230 0.09606368  0.000034 2.872631e-05  0.0001663173
   12    3 0.122160 0.13575365  0.000716 4.436002e-04 -0.0135936534
   13    3 0.294176 0.29020881  0.000672 4.599797e-04  0.0039671914
   16    3 0.412106 0.32051839  0.003134 9.545559e-04  0.0915876076
   17    3 0.057924 0.06929316  0.000213 1.584498e-04 -0.0113691610
   20    3 0.082388 0.09145810  0.000203 1.550209e-04 -0.0090701028
   25    3 0.209146 0.17294244  0.001039 4.380790e-04  0.0362035600
   43    3 0.148671 0.14241817  0.000551 4.073340e-04  0.0062528350
   45    3 0.234361 0.21998893  0.000501 3.493211e-04  0.0143720737
   46    3 0.137869 0.14322305  0.000149 1.366389e-04 -0.0053540450
```

Chapter 6

Unit-level time models

6.1 Unit-level model with independent time effects

6.1.1 The methodology

Let us consider the model

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{W}^{-1/2}\mathbf{e}, \tag{6.1}$$

where $\mathbf{u}_1 = \mathbf{u}_{1,D\times 1} \sim N(0,\sigma_1^2\mathbf{I}_D)$, $\mathbf{u}_2 = \mathbf{u}_{2,M\times 1} \sim N(0,\sigma_2^2\mathbf{I}_M)$ and $\mathbf{e} = \mathbf{e}_{n\times 1} \sim N(0,\sigma_0^2\mathbf{I}_n)$ are independent, $\mathbf{y} = \mathbf{y}_{n\times 1}$, $\mathbf{X} = \mathbf{X}_{n\times p}$ with $\text{r}(\mathbf{X}) = p$, $\beta = \beta_{p\times 1}$, $\mathbf{Z}_1 = \underset{1\le d\le D}{\text{diag}}\,(\mathbf{1}_{n_d})_{n\times D}$, $\mathbf{Z}_2 = \underset{1\le d\le D}{\text{diag}}\,(\underset{1\le t\le m_d}{\text{diag}}\,(\mathbf{1}_{n_{dt}}))_{n\times M}$, $M = \sum_{d=1}^{D} m_d$, $n = \sum_{d=1}^{D} n_d$, $n_d = \sum_{t=1}^{m_d} n_{dt}$, $\mathbf{I}_a$ is the $a\times a$ identity matrix, $\mathbf{1}_a$ is the $a\times 1$ vector with all its elements equal to 1, $\mathbf{W} = \underset{1\le d\le D}{\text{diag}}\,(\mathbf{W}_d)$, $\mathbf{W}_d = \underset{1\le t\le m_d}{\text{diag}}\,(\mathbf{W}_{dt})$, $\mathbf{W}_{dt} = \underset{1\le j\le n_{dt}}{\text{diag}}\,(w_{dtj})_{n\times n}$ with known $w_{dtj} > 0$, $d = 1,\dots,D$, $t = 1,\dots,m_d$, $j = 1,\dots,n_{dt}$. Model (6.1) can alternatively be written in the form

$$y_{dtj} = \mathbf{x}_{dtj}\beta + u_{1,d} + u_{2,dt} + w_{dtj}^{-1/2}e_{dtj}, \quad d = 1,\dots,D,\, t = 1,\dots,m_d,\, j = 1,\dots,n_{dt}, \tag{6.2}$$

where y_{dtj} is the target variable for the sample unit j, time t and domain d, and $\mathbf{x}_{dtj}$ is the row (d,t,j) of matrix $\mathbf{X}$. In what follows we use the alternative parameters

$$\sigma^2 = \sigma_0^2, \quad \varphi_1 = \frac{\sigma_1^2}{\sigma_0^2}, \quad \varphi_2 = \frac{\sigma_2^2}{\sigma_0^2}.$$

Let $\sigma = (\sigma^2,\varphi_1,\varphi_2)$ be the vector of variance components, with $\sigma^2 > 0$, $\varphi_1 > 0$ and $\varphi_2 > 0$. Under the model (**??**), we have $\mathbf{V}_{u_1} = \text{var}(\mathbf{u}_1) = \sigma^2\varphi_1\mathbf{I}_D$, $\mathbf{V}_{u_2} = \text{var}(\mathbf{u}_2) = \sigma^2\varphi_2\mathbf{I}_M$, $\mathbf{V}_e = \text{var}(\mathbf{e}) = \sigma^2\mathbf{I}_n$ and

$$\mathbf{V} = \text{var}(\mathbf{y}) = \mathbf{Z}_1\text{var}(\mathbf{u}_1)\mathbf{Z}_1' + \mathbf{Z}_2\text{var}(\mathbf{u}_2)\mathbf{Z}_2' + \sigma^2\mathbf{W}^{-1} = \sigma^2\Sigma = \sigma^2\text{diag}(\Sigma_1,\dots,\Sigma_D),$$

where

$$\Sigma_d = \varphi_1\mathbf{1}_{n_d}\mathbf{1}_{n_d}' + \varphi_2\underset{1\le t\le m_d}{\text{diag}}\,(\mathbf{1}_{n_{dt}})\mathbf{I}_{m_d}\underset{1\le t\le m_d}{\text{diag}}\,(\mathbf{1}_{n_{dt}}') + \mathbf{W}_d^{-1},\ d = 1,\dots,D.$$

If σ is known, the BLUE of $\beta = (\beta_1,\dots,\beta_p)'$ and the BLUP of $\mathbf{u} = (\mathbf{u}_1',\mathbf{u}_2')'$ are

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y} \qquad \text{and} \qquad \widehat{\mathbf{u}} = \mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}\left(\mathbf{y} - \mathbf{X}\widehat{\beta}\right), \tag{6.3}$$

where $\mathbf{V}_u = \text{diag}(\mathbf{V}_{u_1},\mathbf{V}_{u_2})$.

The REML estimators are calculated by using the Fisher-scoring algorithm with the updating formula

$$\sigma^{k+1} = \sigma^k + \mathbf{F}^{-1}(\sigma^k)\mathbf{S}(\sigma^k).$$

The components of the vectors of scores $S(\sigma)$ are

$$\begin{aligned} S_{\sigma^2} &= -\frac{n-p}{2\sigma^2} + \frac{1}{2\sigma^4}\mathbf{y}'\mathbf{P}\mathbf{y}, \\ S_{\varphi_1} &= -\frac{1}{2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\} + \frac{1}{2\sigma^2}\mathbf{y}'\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{y}, \\ S_{\varphi_2} &= -\frac{1}{2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\} + \frac{1}{2\sigma^2}\mathbf{y}'\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\mathbf{P}\mathbf{y}, \end{aligned}$$

where $\mathbf{P} = \Sigma^{-1} - \Sigma^{-1}\mathbf{X}(\mathbf{X}'\Sigma^{-1}\mathbf{X})^{-1}\mathbf{X}'\Sigma^{-1}$. The the elements of the Fisher information matrix are

$$\begin{aligned} F_{\sigma^2\sigma^2} &= -\frac{n-p}{2\sigma^4} + \frac{1}{\sigma^4}\operatorname{tr}\{\mathbf{P}\Sigma\} = \frac{n-p}{2\sigma^4}, \quad F_{\sigma^2\varphi_1} = \frac{1}{2\sigma^2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\}, \\ F_{\sigma^2\varphi_2} &= \frac{1}{2\sigma^2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\}, \quad F_{\varphi_1\varphi_1} = \frac{1}{2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\}, \\ F_{\varphi_1\varphi_2} &= \frac{1}{2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\}, \quad F_{\varphi_2\varphi_2} = \frac{1}{2}\operatorname{tr}\{\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\mathbf{P}\mathbf{Z}_2\mathbf{Z}_2'\}. \end{aligned}$$

If $n_{dt} > 0$, the EBLUP of $\overline{Y}_{dt}$ is

$$\widehat{\overline{Y}}_{dt}^{eblup} = \overline{\mathbf{X}}_{dt}\widehat{\beta} + \overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1 + \overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2 + f_{dt}\left[\overline{\mathbf{y}}_{s,dt} - \overline{\mathbf{X}}_{s,dt}\widehat{\beta} - \overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1 - \overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2\right],$$

where $\overline{\mathbf{y}}_{s,dt} = \frac{1}{n_{dt}}\sum_{j=1}^{n_{dt}} y_{dtj}$, $\overline{\mathbf{X}}_{s,dt} = \frac{1}{n_{dt}}\sum_{j=1}^{n_{dt}} \mathbf{x}_{dtj}$, $f_{dt} = \frac{n_{dt}}{N_{dt}}$ and

$$\begin{aligned} \overline{\mathbf{Z}}_{1,dt} &= \frac{1}{N_{dt}} \underset{1\le\ell\le D}{\operatorname{col}'}\{\delta_{d\ell} \underset{1\le k\le m_\ell}{\operatorname{col}'}[\delta_{tk}\mathbf{1}'_{N_{\ell k}}]\} \underset{1\le\ell\le D}{\operatorname{diag}}(\mathbf{1}_{N_\ell}) = \underset{1\le\ell\le D}{\operatorname{col}'}\{\delta_{d\ell}\}, \\ \overline{\mathbf{Z}}_{2,dt} &= \frac{1}{N_{dt}} \underset{1\le\ell\le D}{\operatorname{col}'}\{\delta_{d\ell} \underset{1\le k\le m_\ell}{\operatorname{col}'}[\delta_{tk}\mathbf{1}'_{N_{\ell k}}]\} \underset{1\le\ell\le D}{\operatorname{diag}}(\underset{1\le k\le m_\ell}{\operatorname{diag}}(\mathbf{1}_{N_{\ell k}})) = \underset{1\le\ell\le D}{\operatorname{col}'}\{\underset{1\le k\le m_\ell}{\operatorname{col}'}\{\delta_{d\ell}\delta_{tk}\}\}. \end{aligned}$$

with $\delta_{ab} = 1$ if $a = b$ and $\delta_{ab} = 0$ otherwise. If $n_{dt} = 0$, the EBLUP of $\overline{Y}_{dt}$ is

$$\widehat{\overline{Y}}_{dt}^{eblup} = \overline{\mathbf{X}}_{dt}\widehat{\beta} + \overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1 + \overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2.$$

The mean squared error of the EBLUP of $\overline{Y}_{dt}$ is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup}) = g_1(\sigma) + g_2(\sigma) + g_3(\sigma) + g_4(\sigma),$$

where

$$\begin{aligned} g_1(\sigma) &= \mathbf{a}_r'\mathbf{Z}_r\mathbf{T}_s\mathbf{Z}_r'\mathbf{a}_r, \\ g_2(\sigma) &= [\mathbf{a}_r'\mathbf{X}_r - \mathbf{a}_r'\mathbf{Z}_r\mathbf{T}_s\mathbf{Z}_s'\mathbf{V}_{es}^{-1}\mathbf{X}_s]\mathbf{Q}_s[\mathbf{X}_r'\mathbf{a}_r - \mathbf{X}_s'\mathbf{V}_{es}^{-1}\mathbf{Z}_s\mathbf{T}_s\mathbf{Z}_r'\mathbf{a}_r], \\ g_3(\sigma) &\approx \operatorname{tr}\left\{(\nabla\mathbf{b}')\mathbf{V}_s(\nabla\mathbf{b}')'E\left[(\widehat{\sigma}-\sigma)(\widehat{\sigma}-\sigma)'\right]\right\}, \\ g_4(\sigma) &= \mathbf{a}_r'\mathbf{V}_{er}\mathbf{a}_r, \end{aligned}$$

and subindexes s and r are used to denote the sampled and non sampled population parts. The elements of formulas $g_1(\sigma) - g_4(\sigma)$ are

$$\begin{aligned}
\mathbf{a}_r' &= \frac{1}{N_{dt}} \underset{1\le\ell\le D}{\text{col}'} \left[\delta_{d\ell} \underset{1\le k\le m_\ell}{\text{col}'} [\delta_{tk} \mathbf{1}'_{N_{\ell k}-n_{\ell k}}] \right], \quad \mathbf{Z}_s = [\mathbf{Z}_{1s}\,\mathbf{Z}_{2s}], \\
\mathbf{Z}_r &= [\mathbf{Z}_{1r}\,\mathbf{Z}_{2r}], \quad \mathbf{T}_s = \mathbf{V}_u - \mathbf{V}_u\mathbf{Z}_s'\mathbf{V}_s^{-1}\mathbf{Z}_s\Sigma_u, \quad \mathbf{b}' = \mathbf{a}_r'\mathbf{Z}_r\mathbf{V}_u\mathbf{Z}_s'\mathbf{V}_s^{-1}, \\
\mathbf{V}_u &= \begin{pmatrix} \sigma_1^2\mathbf{I}_D & \mathbf{0} \\ \mathbf{0} & \sigma_2^2\mathbf{I}_M \end{pmatrix}, \quad \mathbf{V}_s^{-1} = \underset{1\le d\le D}{\text{diag}} \{\mathbf{V}_{ds}^{-1}\}, \quad \mathbf{V}_{es}^{-1} = \sigma^{-2}\mathbf{W}_s, \\
\mathbf{Q}_s &= (\mathbf{X}_s'\mathbf{V}^{-1}\mathbf{X}_s)^{-1}, \quad \mathbf{V}_{er} = \sigma^2\mathbf{W}_r^{-1},
\end{aligned}$$

and $E[(\widehat{\sigma}-\sigma)(\widehat{\sigma}-\sigma)']$ is approximated by the inverse of the REML Fisher information matrix.

6.1.2 The Software: description of R functions

This section describes the R functions that have been implemented for fitting the individual-level model with independent time effects (6.1). An example of how to use these functions is given in the next section and the related codes are listed in Appendix 20.1.

The developed R software contains a series of functions that return, as final output, the EBLUP estimates of interest. We recall that R functions are objects with the form

$$name \ <\!-\ function\ (arg_1, arg_2, ...)\ \{expression\}.$$

R functions allows to define a dependent variable *name* as output of a given procedure, when inputs variables are *arguments*. The *expression* within curly brackets contains the needed calculations to obtain *name* from *arguments*. The function codes appearing in *expression* are listed in Appendix 20.1.

A brief descriptions of programmed R functions is given in the next subsections. The functions can be used for calculating the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

REML.individual.indep

Function **REML.individual.indep** calculates the estimate of σ_u^2 and the Fisher amount of information F for the Restricted Maximum Likelihood (REML) method. The function is

$$REML.individual.indep \ <\!-\ function\ (X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, MAXITER = 500).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M \times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma0 sigma1 sigma2: Initial values of sigma. They are used as seed of the Fisher-scoring algorithm.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 500.

The function returns a list of five elements. First, second and third element **sigmaI** is the REML estimate of σ_u^2, fourth element **F3.inv** is the inverse estimated Fisher amount of information F and fifth element **R** is the inverse matrix appearing in the expression of $\hat{\beta}$.

BETA.U.individual.indep

Function **BETA.U.individual.indep** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.individual.indep <- function(X,Y,W,D,md,ndi,sigma0,sigma1,sigma2).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M \times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

ndi: vector containing the number of individuals in a instants m_d within each domain, with size D.

sigma0 sigma1 sigma2: estimated values of σ_u^2, calculated by the function **REML.individual.indep**.

The function returns a vector containing the estimated regression parameters $\widehat{\beta}$, with size p.

mse.individual.indep

Function **mse.individual.indep** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.individual.indep <- function(X,Y,W,D,md,ndi,MXm,NDI,MXp,sigma0,sigma1,sigma2,FInv).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M \times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

ndi: vector containing the number of individuals in a instants m_d within each domain. (Sampling data)

MXm: vector containing the average data in a instants m_d within each domain. (Sampling data)

NDI: vector containing the number of individuals in a instants m_d within each domain. (Population data)

MXp: vector containing the average data in a instants m_d within each domain. (Population data)

sigma0 sigma1 sigma2: estimated values of σ_u^2, calculated by the function **REML.individual.indep**.

FINV: is the inverse estimated Fisher of information F, calculated by the function **REML.individual.indep**.

The function returns a vector containing the MSE estimates $mse(\widehat{\overline{Y}}_{dt}^{eblup})$.

Interval.indep

Function **Interval.indep** calculates the asymptotic confidence intervals for σ_u^2 and β_i. The function is

$$Interval.indep \ <- \ function\,(fit,\ conf = 0.95).$$

The arguments are:

fit: returned object, obtained by applying the function **REM.individual.indep**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma0.std.err**, **sigma1.std.err**, **sigma2.std.err**, **rho.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ_u^2 and β_i respectively.

pvalue

Function **pvalue** calculates the asymptotic p-value of test statistics $\hat{\beta}_i$ for the null hypothesis $H_0 : \beta_i = 0$. The function is

$$pvalue \ <- \ function\,(beta0,\ fit).$$

The arguments are:

beta0: observed value of $\hat{\beta}_i$, calculated by the function **BETA.U.individual.indep**.

fit: returned object, obtained by applying the function **REML.individual.indep**.

This function returns the vector **pval** containing the asymptotic p-values for hypotheses $H_0 : \beta_i = 0$, $i = 1,\dots,p$, with size p.

6.1.3 Examples of usage of R functions

This section describes how to apply the described R routines to calculate EBLUP estimates and their corresponding mean squared errors.

Example data set

Table 6.1.3.1 presents the data sets used in the example.

Domain	*Unit*	*Time*	*One*	*Age*	*Y*
11	11	1	1	0.255	1.2550
11	21	1	1	0.000	1.0014
11	31	1	1	0.594	1.5940
11	41	1	1	0.501	1.5004
11	51	1	1	0.226	1.2245
11	11	2	1	0.000	1.0004
11	21	2	1	0.119	1.1192
11	31	2	1	0.530	1.5302
11	41	2	1	0.539	1.5387
11	51	2	1	0.120	1.1189
11	11	3	1	0.000	0.9996
11	21	3	1	0.182	1.1820
11	31	3	1	0.331	1.3310
11	41	3	1	0.401	1.4015
11	51	3	1	0.109	1.1093
12	12	1	1	0.513	1.5120
12	22	1	1	0.155	1.1546
12	32	1	1	0.335	1.3336
12	42	1	1	0.599	1.5987
12	52	1	1	0.282	1.2831
12	12	2	1	0.539	1.5383
12	22	2	1	0.179	1.1790
12	32	2	1	0.220	1.2205
12	42	2	1	0.637	1.6366
12	52	2	1	0.183	1.1830
12	12	3	1	0.662	1.6617
12	22	3	1	0.123	1.1226
12	32	3	1	0.198	1.1982
12	42	3	1	0.663	1.6637
12	52	3	1	0.358	1.3575

Table 6.1.3.1. Data set *dataExample*.

There are 2 domains, 5 sampling units (within each domain), 3 time periods and 1 independent variable *Age* for each domain and time period. Dependent variable is labeled by Y. There are are 30 observations. The file *dataExample.txt* contains the data. Data should be sorted by domains and time periods.

Example of R code

An R code for reading the data file and applying the above described functions is needed. The file *Example.R* contains this code and, for this example, is located in the folder *C:/IndividualtimemodelIndep*. It is important to take care on where to put this file and all the function *R* files. Note that under *Windows system*, the folder of the file is set by default installation. Otherwise the user can type the complete path. But under *Linux* the folder is the same than the one used to execute R.

The R file *Example.R* contains a program with the instructions for fitting the area level model to data in file *dataExample.txt*. First step is to open the *R* files containing all the above described R functions. Second step is to read the data file *dataExample.txt*. Third step is to run the application. The program creates several *txt* files in folder *C:/IndividualtimemodelIndep*. The new files contain the output of the program in what follows the code in *Example.R* is listed.

```
####################################################################
###
###                         Individual-level time models
###                              SAMPLE project
###
### Author: Laureano Santamaria Arana
### File name: Example.R
### Updated: June 25th, 2010
###
#####################################################################

### Establishing the folder where data and routine files are located.
setwd("C:/IndividualtimemodelIndep")
sink("Outputs.txt")

### Call functions

source("REML.R")
source("Estimacion BETA.R")
source("Estimacion MSE.R")
source("Varios.R")

### Reading data
data <- read.table(file = "dataExample.txt", header = T)

xdt <- as.matrix(data[,4:(ncol(data)-1)])
ydt <- data[,ncol(data)]
n <- nrow(data)
D <- length(unique(data[,1]))
md <- rep(3,D)
ndi <- c(5,5,5,5,5,5)
NDI <- c(148507,146226,149710,142580,148212,148096)

mdcum <- cumsum(md)
wdt <- rep(1,n)
W <- diag(wdt)
X <- as.matrix(xdt)
Y <- as.matrix(ydt)

Sem <- Calcular.Semilla(X, Y, D, md, ndi)

sigma0 <- as.numeric(Sem[[1]])
sigma1 <- as.numeric(Sem[[2]])
sigma2 <- as.numeric(Sem[[3]])

fit <- REML.area (X, Y, W, D, md, ndi, sigma0, sigma1,
       sigma2, MAXITER = 500)
```

```
for(i in 1:3) {
    if (fit[[i]]<0)
        fit[[i]] <- 0.001
}
sigma0.gorro <- fit[[1]]
sigma1.gorro <- fit[[2]]
sigma2.gorro <- fit[[3]]
FisherInv <- fit[[4]]
Iter <- fit[[5]]
Q <- fit[[6]]

B <- BETA.U.area(X, Y, W, D, md, ndi, sigma0.gorro,
    sigma1.gorro, sigma2.gorro)

beta.gorro <- B

fit0 <- list()
fit0[[1]] <- sigma0.gorro
fit0[[2]] <- FisherInv
fit0[[3]] <- Q

Int0 <- Interval.Indep (fit0, 0.90)
pvalue0 <- pvalue(beta.gorro, fit0)

### writing data
    cat("Number of Iter.\t",Iter,"\n")
    cat("\nbeta.gorro\n")
        beta.gorro
    v<-length(beta.gorro)
    for(d in 1:v) {
        cat("beta.gorro\t", beta.gorro[d],"\tInterval:
        (",beta.gorro[d]-Int0[[4]][d], beta.gorro[d]+Int0[[4]][d],")\n")
    }
    cat("\nSigma0.gorro\t",sigma0.gorro,"\tInterval:
    (",sigma0.gorro-Int0[[1]], sigma0.gorro+Int0[[1]],")\n")
    cat("Sigma1.gorro\t",sigma1.gorro,"\tInterval:
    (",sigma1.gorro-Int0[[2]], sigma1.gorro+Int0[[2]],")\n")
    cat("Sigma2.gorro\t",sigma2.gorro,"\tInterval:
    (",sigma2.gorro-Int0[[3]], sigma1.gorro+Int0[[3]],")\n")
    cat("\nPvalue\n")
        pvalue0
    cat("\nPvalue>0.1\n")
        pvalue0>0.1
### End writing datas


#### Calculate and read the population and samples means
Mxm <- Calcular.Media(X, D,  md, ndi)
Mxp <- read.table(file = "Medias.txt", header = F, dec="," )
```

```
### EBLUP of the population parameter for last time instant

Mxp <- as.matrix(Mxp)
Beta <- matrix(beta.gorro,nrow=ncol(X),ncol=1)

mudt.gorro <- Calcular.Yeblup.Indep (X, Y, W, D, md, ndi,
        Beta, sigma0.gorro, sigma1.gorro, sigma2.gorro, Mxp)
sqrt.mse <- sqrt(mse.area(X, Y, W, D, md, ndi, Mxm, NDI,
        Mxp, sigma0.gorro, sigma1.gorro, sigma2.gorro, FisherInv))

Ybarra <- Calcular.Media.Y(Y, D,  md, ndi)

residuals.gorro <- Ybarra-mudt.gorro

### Create .txt files in the folder that contains for
### the resulting output for last time instant
write.table(data.frame(Direct=Ybarra, EBLUP=mudt.gorro,
        Residuals=residuals.gorro, Sqrt.mse=sqrt.mse),
        file="EBLUP_Example.txt",row.names=FALSE, sep="\t")

sink()
```

Outputs

The resulting outputs appear in the files *EBLUP Example.txt, Outputs.txt* they are:

```
"EBLUP Example.txt" output:
"Direct"    "EBLUP"             "Residuals"             "Sqrt.mse"
1.31505574 1.31515569750924 -9.99575092435201e-05 0.0158186908694694
1.26148538 1.26158533713102 -9.99571310198455e-05 0.0158195371696525
1.20468282 1.20478276303545 -9.99430354511333e-05 0.0158205291621391
1.37641348 1.37651345096837 -9.99709683702221e-05 0.0237186804083336
1.3515003 1.35160024826939 -9.99482693868625e-05 0.0259175337125956
1.40074604 1.40084598259626 -9.9942596262892e-05 0.0230270294111105


"Outputs.txt" output:

Number of Iter.  1

beta.gorro
          [,1]
Ones  1.0000295
Age 0.9995325
beta.gorro  1.000029  Interval: ( -0.4585167 2.458576 )
beta.gorro  0.9995325  Interval: ( -0.5178786 2.516944 )

Sigma0.gorro  4.126837e-07  Interval: ( 3.461224e-07 4.79245e-07 )
```

```
Sigma1.gorro  0.001  Interval: ( -3.337813 3.339813 )
Sigma2.gorro  0.001  Interval: ( -1.513169 1.515169 )

Pvalue
           [,1]
Ones  0.2594178
Age 0.2785954

Pvalue>0.1
      [,1]
Ones  TRUE
Age TRUE
```

6.2 Unit-level model with correlated time effects

6.2.1 The methodology

Let us consider the model

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{W}^{-1/2}\mathbf{e}, \tag{6.4}$$

where $\mathbf{u}_1 = \mathbf{u}_{1,D\times 1} \sim N(0,\sigma_1^2\mathbf{I}_D)$, $\mathbf{u}_2 = \mathbf{u}_{2,M\times 1} \sim N(0,\sigma_2^2\Omega(\rho))$ and $\mathbf{e} = \mathbf{e}_{n\times 1} \sim N(0,\sigma_0^2\mathbf{I}_n)$ are independent, $\mathbf{y} = \mathbf{y}_{n\times 1}$, $\mathbf{X} = \mathbf{X}_{n\times p}$ with $\mathrm{r}(\mathbf{X}) = p$, $\beta = \beta_{p\times 1}$, $\mathbf{Z}_1 = \underset{1\le d\le D}{\mathrm{diag}}\,(\mathbf{1}_{n_d})_{n\times D}$, $\mathbf{Z}_2 = \underset{1\le d\le D}{\mathrm{diag}}\,(\underset{1\le t\le m_d}{\mathrm{diag}}\,(\mathbf{1}_{n_{dt}}))_{n\times M}$, $M = \sum_{d=1}^{D} m_d$, $n = \sum_{d=1}^{D} n_d$, $n_d = \sum_{t=1}^{m_d} n_{dt}$, $\mathbf{I}_a$ is the $a\times a$ identity matrix, $\mathbf{1}_a$ is the $a\times 1$ vector with all its elements equal to 1, $\mathbf{W} = \underset{1\le d\le D}{\mathrm{diag}}\,(\mathbf{W}_d)$, $\mathbf{W}_d = \underset{1\le t\le m_d}{\mathrm{diag}}\,(\mathbf{W}_{dt})$, $\mathbf{W}_{dt} = \underset{1\le j\le n_{dt}}{\mathrm{diag}}\,(w_{dtj})_{n\times n}$ with known $w_{dtj} > 0$, $d = 1,\dots,D$, $t = 1,\dots,m_d$, $j = 1,\dots,n_{dt}$, $\Omega(\rho) = \underset{1\le d\le D}{\mathrm{diag}}\,(\Omega_d)$ and

$$\Omega_d = \Omega_d(\rho) = \frac{1}{1-\rho^2}\begin{pmatrix} 1 & \rho & \dots & \rho^{m_d-2} & \rho^{m_d-1} \\ \rho & 1 & \ddots & & \rho^{m_d-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \rho^{m_d-2} & & \ddots & 1 & \rho \\ \rho^{m_d-1} & \rho^{m_d-2} & \dots & \rho & 1 \end{pmatrix}_{m_d\times m_d}.$$

Model (6.4) can alternatively be written in the form

$$y_{dtj} = \mathbf{x}_{dtj}\beta + u_{1,d} + u_{2,dt} + w_{dtj}^{-1/2}e_{dtj}, \quad d = 1,\dots,D, t = 1,\dots,m_d, j = 1,\dots,n_{dt}, \tag{6.5}$$

where y_{dtj} is the target variable for the sample unit j, time t and domain d, and $\mathbf{x}_{dtj}$ is the row (d,t,j) of matrix $\mathbf{X}$. The random vectors $(u_{2d1},\dots,u_{2dm_d})$, $d = 1,\dots,D$, are i.i.d. AR(1).

Under model (6.4), we have $\mathbf{V}_{u_1} = \mathrm{var}(\mathbf{u}_1) = \sigma^2\varphi_1\mathbf{I}_D$, $\mathbf{V}_{u_2} = \mathrm{var}(\mathbf{u}_2) = \sigma^2\varphi_2\Omega(\rho)$, $\mathbf{V}_e = \mathrm{var}(\mathbf{e}) = \sigma^2\mathbf{I}_n$ and

$$\mathbf{V} = \mathrm{var}(\mathbf{y}) = \mathbf{Z}_1\mathrm{var}(\mathbf{u}_1)\mathbf{Z}_1' + \mathbf{Z}_2\mathrm{var}(\mathbf{u}_2)\mathbf{Z}_2' + \sigma^2\mathbf{W}^{-1} = \sigma^2\Sigma = \sigma^2\mathrm{diag}(\Sigma_1,\dots,\Sigma_D),$$

where

$$\Sigma_d = \varphi_1\mathbf{1}_{n_d}\mathbf{1}_{n_d}' + \varphi_2\underset{1\le t\le m_d}{\mathrm{diag}}\,(\mathbf{1}_{n_{dt}})\,\Omega_d(\rho)\underset{1\le t\le m_d}{\mathrm{diag}}\,(\mathbf{1}_{n_{dt}}') + \mathbf{W}_d^{-1},\ d = 1,\dots,D.$$

In what follows we use the alternative parameters

$$\sigma^2=\sigma_0^2,\quad \varphi_1=\frac{\sigma_1^2}{\sigma_0^2},\quad \varphi_2=\frac{\sigma_2^2}{\sigma_0^2},\quad \rho=\rho.$$

Let $\sigma=(\sigma^2,\varphi_1,\varphi_2,\rho)$ be the vector of variance components, with $\sigma^2>0,\varphi_1>0,\varphi_2>0$ and $-1<\rho<1$. If σ is known, the BLUE of $\beta=(\beta_1,\dots,\beta_p)'$ and the BLUP of $\mathbf{u}=(\mathbf{u}_1',\mathbf{u}_2')'$ are

$$\widehat{\beta}=(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}\qquad\text{and}\qquad \widehat{\mathbf{u}}=\mathbf{V}_u\mathbf{Z}'\mathbf{V}^{-1}\left(\mathbf{y}-\mathbf{X}\widehat{\beta}\right). \tag{6.6}$$

where $\mathbf{V}_u=\text{diag}(\mathbf{V}_{u_1},\mathbf{V}_{u_2})$.

The REML estimators are calculated by using the Fisher-scoring algorithm with the updating formula

$$\sigma^{k+1}=\sigma^k+\mathbf{F}^{-1}(\sigma^k)\mathbf{S}(\sigma^k).$$

The components of the vectors of scores $S(\sigma)$ are

$$\begin{aligned}
S_{\sigma^2}&=-\frac{n-p}{2\sigma^2}+\frac{1}{2\sigma^4}\mathbf{y}'\mathbf{P}\mathbf{y},\\
S_{\varphi_1}&=-\frac{1}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\}+\frac{1}{2\sigma^2}\mathbf{y}'\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{y},\\
S_{\varphi_2}&=-\frac{1}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\}+\frac{1}{2\sigma^2}\mathbf{y}'\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\mathbf{P}\mathbf{y},\\
S_{\rho}&=-\frac{\varphi_2}{2}\text{tr}\left\{\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\right\}+\frac{\varphi_2}{2\sigma^2}\mathbf{y}'\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\mathbf{P}\mathbf{y},
\end{aligned}$$

The elements of the Fisher information matrix are

$$\begin{aligned}
F_{\sigma^2\sigma^2}&=-\frac{n-p}{2\sigma^4}+\frac{1}{\sigma^4}\text{tr}\{\mathbf{P}\Sigma\}=\frac{n-p}{2\sigma^4},\quad F_{\sigma^2\varphi_1}=\frac{1}{2\sigma^2}\text{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\},\\
F_{\sigma^2\varphi_2}&=\frac{1}{2\sigma^2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\},\quad F_{\sigma^2\rho}=\frac{\varphi_2}{2\sigma^2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\},\\
F_{\varphi_1\varphi_1}&=\frac{1}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\},\quad F_{\varphi_1\varphi_2}=\frac{1}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\}\\
F_{\varphi_1\rho}&=\frac{\varphi_2}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_1\mathbf{Z}_1'\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\},\quad F_{\varphi_2\varphi_2}=\frac{1}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\},\\
F_{\varphi_2\rho}&=\frac{\varphi_2}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\Omega(\rho)\mathbf{Z}_2'\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\},\quad F_{\rho\rho}=\frac{\varphi_2^2}{2}\text{tr}\{\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\mathbf{P}\mathbf{Z}_2\dot{\Omega}(\rho)\mathbf{Z}_2'\}.
\end{aligned}$$

If $n_{dt}>0$, the EBLUP of $\overline{Y}_{dt}$ is

$$\widehat{\overline{Y}}_{dt}^{eblup}=\overline{\mathbf{X}}_{dt}\widehat{\beta}+\overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1+\overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2+f_{dt}\left[\overline{\mathbf{y}}_{s,dt}-\overline{\mathbf{X}}_{s,dt}\widehat{\beta}-\overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1-\overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2\right],$$

where $\overline{\mathbf{y}}_{s,dt}=\frac{1}{n_{dt}}\sum_{j=1}^{n_{dt}}y_{dtj}$, $\overline{\mathbf{X}}_{s,dt}=\frac{1}{n_{dt}}\sum_{j=1}^{n_{dt}}\mathbf{x}_{dtj}$, $f_{dt}=\frac{n_{dt}}{N_{dt}}$ and

$$\begin{aligned}
\overline{\mathbf{Z}}_{1,dt}&=\frac{1}{N_{dt}}\underset{1\le\ell\le D}{\text{col}'}\{\delta_{d\ell}\underset{1\le k\le m_\ell}{\text{col}'}[\delta_{tk}\mathbf{1}_{N_{\ell k}}']\}\underset{1\le\ell\le D}{\text{diag}}(\mathbf{1}_{N_\ell})=\underset{1\le\ell\le D}{\text{col}'}\{\delta_{d\ell}\},\\
\overline{\mathbf{Z}}_{2,dt}&=\frac{1}{N_{dt}}\underset{1\le\ell\le D}{\text{col}'}\{\delta_{d\ell}\underset{1\le k\le m_\ell}{\text{col}'}[\delta_{tk}\mathbf{1}_{N_{\ell k}}']\}\underset{1\le\ell\le D}{\text{diag}}(\underset{1\le k\le m_\ell}{\text{diag}}(\mathbf{1}_{N_{\ell k}}))=\underset{1\le\ell\le D}{\text{col}'}\{\underset{1\le k\le m_\ell}{\text{col}'}\{\delta_{d\ell}\delta_{tk}\}\},
\end{aligned}$$

with $\delta_{ab}=1$ si $a=b$ and $\delta_{ab}=0$ si $a\neq b$.

If $n_{dt}=0$, the EBLUP of $\overline{Y}_{dt}$ is the synthetic part

$$\widehat{\overline{Y}}_{dt}^{eblup}=\overline{\mathbf{X}}_{dt}\widehat{\beta}+\overline{\mathbf{Z}}_{1,dt}\widehat{\mathbf{u}}_1+\overline{\mathbf{Z}}_{2,dt}\widehat{\mathbf{u}}_2.$$

A second order approximation to the mean squared error of the EBLUP is

$$MSE(\widehat{\overline{Y}}_{dt}^{eblup})=g_1(\sigma)+g_2(\sigma)+g_3(\sigma)+g_4(\sigma),$$

where

$$\begin{aligned}
g_1(\sigma) &= \mathbf{a}_r'\mathbf{Z}_r\mathbf{T}_s\mathbf{Z}_r'\mathbf{a}_r,\\
g_2(\sigma) &= [\mathbf{a}_r'\mathbf{X}_r-\mathbf{a}_r'\mathbf{Z}_r\mathbf{T}_s\mathbf{Z}_s'\mathbf{V}_{es}^{-1}\mathbf{X}_s]\mathbf{Q}_s[\mathbf{X}_r'\mathbf{a}_r-\mathbf{X}_s'\mathbf{V}_{es}^{-1}\mathbf{Z}_s\mathbf{T}_s\mathbf{Z}_r'\mathbf{a}_r],\\
g_3(\sigma) &\approx \operatorname{tr}\left\{(\nabla\mathbf{b}')\mathbf{V}_s(\nabla\mathbf{b}')'E\left[(\widehat{\sigma}-\sigma)(\widehat{\sigma}-\sigma)'\right]\right\},\\
g_4(\sigma) &= \mathbf{a}_r'\mathbf{V}_{er}\mathbf{a}_r.
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{a}_r' &= \frac{1}{N_{dt}}\underset{1\le\ell\le D}{\operatorname{col}'}\left[\delta_{d\ell}\underset{1\le k\le m_\ell}{\operatorname{col}'}[\delta_{tk}\mathbf{1}'_{N_{\ell k}-n_{\ell k}}]\right],\quad \mathbf{Z}_s=[\mathbf{Z}_{1s}\,\mathbf{Z}_{2s}],\\
\mathbf{Z}_r &= [\mathbf{Z}_{1r}\,\mathbf{Z}_{2r}],\quad \mathbf{T}_s=\mathbf{V}_u-\mathbf{V}_u\mathbf{Z}_s'\mathbf{V}_s^{-1}\mathbf{Z}_s\mathbf{V}_u,\quad \mathbf{Q}_s=(\mathbf{X}_s'\mathbf{V}_s^{-1}\mathbf{X}_s)^{-1},\\
\mathbf{V}_u &= \begin{pmatrix}\sigma_1^2\mathbf{I}_D & \mathbf{0}\\ \mathbf{0} & \sigma_2^2\Omega(\rho)\end{pmatrix},\quad \mathbf{V}_s^{-1}=\underset{1\le d\le D}{\operatorname{diag}}\{\mathbf{V}_{ds}^{-1}\},\quad \mathbf{V}_{es}^{-1}=\sigma^{-2}\mathbf{W}_s,\\
\mathbf{b}' &= \mathbf{a}_r'\mathbf{Z}_r\mathbf{V}_u\mathbf{Z}_s'\mathbf{V}_s^{-1},\quad \mathbf{V}_{er}=\sigma^2\mathbf{W}_r^{-1},
\end{aligned}$$

and $E\left[(\widehat{\sigma}-\sigma)(\widehat{\sigma}-\sigma)'\right]$ is approximated by the inverse of the REML Fisher information matrix.

6.2.2 The Software: description of R functions

This section describes the R functions that have been implemented for fitting the individual-level model with time correlated effects (6.4). A brief descriptions of programmed R functions is given in the next subsections and the related codes are listed in Appendix 20.2. The function can be used for calculating the REML variance estimates, the β estimate, the $\mathbf{u}$ predictor, the EBLUPs and the MSEs of EBLUPs.

REML.individual.autocorr

Function **REML.individual.autocorr** calculates the estimate of σ_u^2, the correlation coefficient ρ and the Fisher information matrix F for the Restricted Maximum Likelihood (REML) method. The function is

$$REML.individual.autocorr <- function\,(X,Y,W,D,md,ndi,sigma0,sigma1,sigma2,rho,MAXITER=500).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M\times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

sigma0, sigma1, sigma2: Initial values of sigma. They are used as seed of the Fisher-scoring algorithm.

rho: Auto-correlation parameter.

MAXITER: maximum number of iterations in the Fisher-scoring algorithm. Default value is 500.

The function returns a list of six elements. First, second and third element **sigmaI** is the REML estimate of σ_u^2, fourth element **rho** is the REML estimate of ρ, fifth element **F3.inv** is the inverse estimated Fisher amount of information F and sixth element **R** is the inverse matrix appearing in the expression of $\hat{\beta}$.

BETA.U.individual.autocorr

Function **BETA.U.individual.autocorr** calculates the estimator $\hat{\beta}$ and the predictor $\hat{\mathbf{u}}$. The function is

$$BETA.U.area.autocorr <- function\,(X,\,ydt,\,D,\,md,\,sigma2edt,\,sigmau,\,rho).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M \times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

ndi: vector containing the number of individuals in a instants m_d within each domain, with size D.

sigma0 sigma1 sigma2: estimated values of σ_u^2, calculated by the function **REML.individual.indep**.

rho: estimated value of ρ, calculated by the function **REML.individual.autocorr**.

The function returns a vector containing the estimated regression parameters $\widehat{\beta}$.

mse.individual.autocorr

Function **mse.individual.autocorr** calculates the estimator of the Mean Squared Error (MSE) of the EBLUP $\widehat{\mu}_{dt} = \mathbf{x}_{dt}\widehat{\beta} + \widehat{u}_{dt}$. The function is

$$mse.individual.autocorr <- function(X,Y,W,D,md,ndi,MXm,NDI,MXp,sigma0,sigma1,sigma2,rho,FInv).$$

The arguments are:

X: matrix containing values of p auxiliary variables, with dimension $M \times p$.

Y: vector containing the direct estimates of the dependent variable for area d and time instant t, with size M.

W: vector containing the weights of the dependent variable for area d and time instant t, with size M.

D: total number of domains.

md: vector containing the time instants totals m_d within each domain, with size D.

ndi: vector containing the number of individuals in a instants m_d within each domain. (Sampling data)

MXm: vector containing the average data in a instants m_d within each domain. (Sampling data)

NDI: vector containing the number of individuals in a instants m_d within each domain. (Population data)

MXp: vector containing the average data in a instants m_d within each domain. (Population data)

sigma0, sigma1, sigma2: estimated values of σ_u^2, calculated by the function **REML.individual.autocorr**.

rho: estimated value of ρ, calculated by the function **REML.individual.autocorr**.

FINV: is the inverse estimated Fisher amount of information F, calculated by the function **REML.individual.autocorr**.

The function returns a vector containing the MSE estimates $mse(\widehat{Y}_{dt}^{eblup})$.

Interval.autocorr

Function **Interval.autocorr** calculates the asymptotic confidence intervals for σ_u^2 and β_i. The function is

$$Interval.autocorr <- function\,(fit,\ conf = 0.95).$$

The arguments are:

fit: returned object, obtained by applying the function **REML.individual.autocorr**.

conf: interval confidence level $1-\alpha$. Default value is 0.95.

This function returns the semi-lengths **sigma0.std.err**, **sigma1.std.err**, **sigma2.std.err** and **beta.std.err** of the asymptotic confidence intervals for σ_u^2 and β_i respectively.

6.2.3 Examples of usage of R functions

This section demonstrates how the R routines described can be applied to produce EBLUP estimates with their corresponding mean squared errors.

Example data set

Table 6.2.3.1 presents the data sets used in the example. There are 2 domains, 5 sampling units (within each domain), 3 time periods and 1 independent variable *Age* for each domain and time period. Dependent variable is labeled by Y. There are are 30 observations. The file *dataExample.txt* contains the data. Data should be sorted by domains and time periods.

Domain	*Unit*	*Time*	*One*	*Age*	*Y*
11	11	1	1	0.255	1.2550
11	21	1	1	0.000	1.0014
11	31	1	1	0.594	1.5940
11	41	1	1	0.501	1.5004
11	51	1	1	0.226	1.2245
11	11	2	1	0.000	1.0004
11	21	2	1	0.119	1.1192
11	31	2	1	0.530	1.5302
11	41	2	1	0.539	1.5387
11	51	2	1	0.120	1.1189
11	11	3	1	0.000	0.9996
11	21	3	1	0.182	1.1820
11	31	3	1	0.331	1.3310
11	41	3	1	0.401	1.4015
11	51	3	1	0.109	1.1093

Table 6.2.3.1. Data set *dataExample*.

Domain	*Unit*	*Time*	*One*	*Age*	*Y*
12	12	1	1	0.513	1.5120
12	22	1	1	0.155	1.1546
12	32	1	1	0.335	1.3336
12	42	1	1	0.599	1.5987
12	52	1	1	0.282	1.2831
12	12	2	1	0.539	1.5383
12	22	2	1	0.179	1.1790
12	32	2	1	0.220	1.2205
12	42	2	1	0.637	1.6366
12	52	2	1	0.183	1.1830
12	12	3	1	0.662	1.6617
12	22	3	1	0.123	1.1226
12	32	3	1	0.198	1.1982
12	42	3	1	0.663	1.6637
12	52	3	1	0.358	1.3575

Table 6.2.3.1. Data set *dataExample*.

Example of R code

An R code for reading the data file and applying the above described functions is needed. The file *Example.R* contains this code and, for this example, is located in the folder *C:/IndividualtimemodelCorr*. It is important to take care on where to put this file and all the function *R* files. Note that under *Windows system*, the folder of the file is set by default installation. Otherwise the user can type the complete path. But under *Linux* the folder is the same than the one used to execute R.

The R file *Example.R* contains a program with the instructions for fitting the area level model to data in file *dataExample.txt*. First step is to open the *R* files containing all the above described R functions. Second step is to read the data file *dataExample.txt*. Third step is to run the application. The program creates several *txt* files in folder *C:/IndividualtimemodelCorr*. The new files contain the output of the program in what follows the code in *Example.R* is listed.

```
##################################################################
###
###                         Individual-level time models
###                              SAMPLE project
###
### Author: Laureano Santamaria Arana
### File name: Example.R
### Updated: June 25th, 2010
###
###################################################################

### Establishing the folder where data and routine files are located.
setwd("C:/IndividualtimemodelCorr/")
sink("Outputs.txt")

### Call functions

source("REML.R")
source("Estimacion BETA.R")
source("ICautocorr.R")
source("pvalue.R")
source("Estimacion MSE.R")


### Reading data
data <- read.table(file = "dataExample.txt", header = T)

xdt <- as.matrix(data[,4:(ncol(data)-1)])
ydt <- data[,ncol(data)]
n <- nrow(data)
D <- length(unique(data[,1]))
md <- rep(3,D)
ndi <- c(5,5,5,5,5,5)
NDI <- c(148507,146226,149710,142580,148212,148096)

mdcum <- cumsum(md)
wdt <- rep(1,n)
W <- diag(wdt)
X <- as.matrix(xdt)
Y <- as.matrix(ydt)

Sem <- Calcular.Semilla(X, Y, D, md, ndi)

sigma0 <- as.numeric(Sem[[1]])
sigma1 <- as.numeric(Sem[[2]])
sigma2 <- as.numeric(Sem[[3]])
rho <- as.numeric(Sem[[4]])
fit <- REML.area (X, Y, W, D, md, ndi, sigma0, sigma1,
                  sigma2, rho, MAXITER = 500)

for(i in 1:3) {
    if (fit[[i]]<0)
        fit[[i]] <- 0.001
}
sigma0.gorro <- fit[[1]]
sigma1.gorro <- fit[[2]]
sigma2.gorro <- fit[[3]]
rho.gorro <- fit[[4]]
FisherInv <- fit[[5]]
Iter <- fit[[6]]
Q <- fit[[7]]

B <- BETA.U.area(X, Y, W, D, md, ndi, sigma0.gorro,
                 sigma1.gorro, sigma2.gorro, rho.gorro)

beta.gorro <- B

fit0 <- list()
```

```
fit0[[1]] <- sigma0.gorro
fit0[[2]] <- FisherInv
fit0[[3]] <- Q

Int0 <- Interval.Indep (fit0, 0.90)
pvalue0 <- pvalue(beta.gorro, fit0)

### writing data
    cat("Number of Iter.\t",Iter,"\n")
    cat("\nbeta.gorro\n")
        beta.gorro
    v<-length(beta.gorro)
    for(d in 1:v) {
      cat("beta.gorro\t", beta.gorro[d],"\tInterval:
      (",beta.gorro[d]-Int0[[4]][d], beta.gorro[d]+Int0[[4]][d],")\n")
    }
    cat("\nSigma0.gorro\t",sigma0.gorro,"\tInterval:
    (",sigma0.gorro-Int0[[1]], sigma0.gorro+Int0[[1]],")\n")
    cat("Sigma1.gorro\t",sigma1.gorro,"\tInterval:
    (",sigma1.gorro-Int0[[2]], sigma1.gorro+Int0[[2]],")\n")
    cat("Sigma2.gorro\t",sigma2.gorro,"\tInterval:
    (",sigma2.gorro-Int0[[3]], sigma2.gorro+Int0[[3]],")\n")
    cat("rho.gorro\t",rho.gorro,"\tInterval:
    (",rho.gorro-Int0[[4]], rho.gorro+Int0[[4]],")\n")
    cat("\nPvalue\n")
        pvalue0
    cat("\nPvalue>0.1\n")
        pvalue0>0.1
### End writing datas

#### Calculate and read the population and samples means
Mxm <- Calcular.Media(X, D,  md, ndi)
Mxp <- read.table(file = "Medias.txt", header = F, dec="," )

### EBLUP of the population parameter for last time instant

Mxp <- as.matrix(Mxp)
Beta <- matrix(beta.gorro,nrow=ncol(X),ncol=1)

mudt.gorro <- Calcular.Yeblup (X, Y, W, D, md, ndi, Beta,
   sigma0.gorro, sigma1.gorro, sigma2.gorro, rho.gorro, Mxp)

sqrt.mse <- sqrt(mse.area(X, Y, W, D, md, ndi, Mxm, NDI, Mxp,
   sigma0.gorro, sigma1.gorro, sigma2.gorro, rho.gorro, FisherInv))

Ybarra <- Calcular.Media.Y(Y, D,  md, ndi)

residuals.gorro <- Ybarra-mudt.gorro
```

```
### Create .txt files in the folder containing the outputs
#write.table(data.frame(Direct=Ybarra, EBLUP=mudt.gorro,
Residuals=residuals.gorro, Sqrt.mse=sqrt.mse),
file="EBLUP_Reales.txt",row.names=FALSE, sep="\t")

sink()
```

Outputs

The resulting outputs appear in the files *EBLUP Example.txt, Outputs.txt* they are:

```
"EBLUP Example.txt" output:
"Direct"    "EBLUP"             "Residuals"             "Sqrt.mse"
1.31505574 1.31515569750924 -9.99575092435201e-05 0.0158186908694694
1.26148538 1.26158533713102 -9.99571310198455e-05 0.0158195371696525
1.20468282 1.20478276303545 -9.99430354511333e-05 0.0158205291621391
1.37641348 1.37651345096837 -9.99709683702221e-05 0.0237186804083336
1.3515003 1.35160024826939 -9.99482693868625e-05 0.0259175337125956
1.40074604 1.40084598259626 -9.9942596262892e-05 0.0230270294111105

"Outputs.txt" output:

Number of Iter.  1

beta.gorro
           [,1]
Ones  1.0000295
Age 0.9995325
beta.gorro  1.000029  Interval: ( -0.4585167 2.458576 )
beta.gorro  0.9995325  Interval: ( -0.5178786 2.516944 )

Sigma0.gorro  4.126837e-07  Interval: ( 3.461224e-07 4.79245e-07 )
Sigma1.gorro  0.001  Interval: ( -3.337813 3.339813 )
Sigma2.gorro  0.001  Interval: ( -1.513169 1.515169 )

Pvalue
           [,1]
Ones  0.2594178
Age 0.2785954

Pvalue>0.1
      [,1]
Ones  TRUE
Age TRUE
```

Chapter 7

M-quantile small area estimators of the mean

7.1 Methodology

A recently proposed approach to small area estimation is based on the use of M-quantile models (Chambers and Tzavidis, 2006). M-quantile regression provides a "quantile-like" generalization of regression based on influence functions (Breckling and Chambers, 1988). M-quantile models do not depend on strong distributional assumptions nor on a predefined hierarchical structure, and outlier robust inference is automatically performed when these models are fitted. The M-quantile of order q for the conditional density of y given $\mathbf{X}$ is defined as the solution $Q_q(x;\psi)$ of the estimating equation $\int \psi_q(y-Q)f(y|\mathbf{X})dy = 0$, where ψ denotes an influence function associated with the M-quantile. In a linear M-quantile regression model the q-th M-quantile $Q_q(x,\psi)$ of the conditional distribution of y given $\mathbf{X}$ is such that

$$Q_q(x;\psi) = \mathbf{X}\beta_\psi(q) \tag{7.1}$$

where $\psi_q(r_{iq\psi}) = 2\psi\{s^{-1}r_{iq\psi}\}\ \{qI(r_{jq\psi} > 0) + (1-q)I(r_{jq\psi} \leq 0)\}$ and s is a suitable robust estimate of scale, e.g. the MAD estimate $s = median\left|r_{jq\psi}\right|/0.6745$. A popular choice for the influence function is the Huber Proposal 2, $\psi(u) = uI(-c \leq u \leq c) + c\,\mathtt{sgn}(u)$. However, other influence functions are also possible. For specified q and continuous ψ, an estimate $\hat{\beta}_\psi(q)$ of $\beta_\psi(q)$ is obtained via iterative weighted least squares. Note that there is a different set of regression parameters for each q.

Let $\Omega_d = \{1,\ldots,N_d\}$ be the population of area d. Let $\mathbf{y}_d = (y_1,\ldots,y_{N_d})'$ denote the variable values for the N_d small area population elements. We consider a sample $s_d \subset \Omega_d$, of $n_d \leq N_d$ units, and we denote with $r_d = \Omega_d - s_d$ the set of non sampled units. For each population unit j, let $\mathbf{x}_j = (x_{1j},\ldots,x_{pj})$ denote a vector of p known auxiliary variables. The small area specific empirical distribution function of y for area d is

$$F_d = N_d^{-1}\Big[\sum_{j\in s_d} \mathrm{I}(y_j \leqslant t) + \sum_{j\in r_d} \mathrm{I}(y_j \leqslant t)\Big]. \tag{7.2}$$

The problem of estimating $F_d(t)$ given the sample data essentially reduces to predicting the values y_j for the non-sampled units in small area d. One straightforward way of achieving this is to simply replace the unknown non-sample values of y (7.2) by their predicted values $\hat{y}_j$ under an appropriate model, leading to a plug-in estimator of (7.2) of the form

$$\hat{F}_d = N_d^{-1}\Big[\sum_{j\in s_d} \mathrm{I}(y_j \leqslant t) + \sum_{j\in r_d} \mathrm{I}(\hat{y}_j \leqslant t)\Big]. \tag{7.3}$$

An estimator of the mean $\overline{Y}_d$ of y in area d is then defined by the value of the mean functional defined by (7.3). This leads to the usual plug-in estimator of the mean,

$$\hat{\overline{Y}}_d = \int_{-\infty}^{\infty} td\hat{F}_d(t) = N_d^{-1}\left(\sum_{j\in s_d} y_j + \sum_{j\in r_d} \hat{y}_j\right).$$

The predicted value of a non-sample unit j in area d corresponds to an estimate $\hat{\mu}_j$ of its expected value given that it is located in area d.

When the conditional M-quantiles are assumed to follow a linear model, with $\beta_\psi(q)$ a sufficiently smooth function of q, this suggests an estimator of the distribution function

$$\hat{F}_d^{MQ}(t) = N_d^{-1}\left\{\sum_{j\in s_d} I(y_j \le t) + \sum_{j\in r_d} I(\mathbf{x}_j\hat{\beta}_\psi(\hat{\theta}_d) \le t)\right\} \tag{7.4}$$

where $\mathbf{x}_j\beta_\psi(\theta_d)$ is used to predict the unobserved value y_j for population unit $j \in r_d$. When there are no sampled observations in area d then $\hat{\theta}_d = 0.5$.

Using the empirical distribution function and the linear M-quantile small area models one can defined an estimators of the small area mean

$$\hat{\overline{Y}}_d^{MQ}(t) = \int_{-\infty}^{\infty} td\hat{F}_d^{MQ}(t) = N_d^{-1}\left\{\sum_{j\in s_d} y_j + \sum_{j\in r_d} \mathbf{x}_j\hat{\beta}_\psi(\hat{\theta}_d)\right\}. \tag{7.5}$$

Chambers and Tzavidis (2006) observed that the naive M-quantile mean estimator (7.5) can be biased. The distribution function estimator (7.3) underlying (7.4) is not consistent in general. Thus, when the non-sample predicted values in (7.3) are estimated expectations that converge in probability to the actual expected values, we see that

$$\sum_{j\in r_d} \mathrm{I}(\hat{y}_j \leqslant t) = \sum_{j\in r_d} \mathrm{I}(y_j - (y_j - \hat{y}_j) \leqslant t) = \sum_{j\in r_d} \mathrm{I}(y_j \leqslant t + \varepsilon_j) \neq \sum_{j\in r_d} \mathrm{I}(y_j \leqslant t),$$

where ε_j are the actual regression errors. If these errors are independently and identically distributed symmetrically about zero we expect that the summation on the left hand side above will closely approximate the summation on the right for values of t near the median of the non-sampled area d values of y but not anywhere else. More generally, for heteroskedastic and/or asymmetric errors this correspondence will typically occur elsewhere in the support of y, although one would expect that in most reasonable situations it will be "close" to the median of y. In other words, it is not advisable to use (7.3) to predict a quantile of the area d distribution of y other than the median.

By combining a smearing argument (Duan, 1983) with a model for the finite population distribution of y, Chambers and Dunstan (1986, hereafter referred to as CD) developed a model-consistent estimator for a finite population distribution function. In the context of the small area distribution function (7.2), and assuming that the residuals are homoskedastic within the small area of interest, this is of the form

$$\hat{F}_d^{CD}(t) = N_d^{-1}\left\{\sum_{j\in s_d} I(y_j \le t) + \sum_{k\in r_d} n_d^{-1} \sum_{j\in s_d} I(\hat{y}_k + (y_j - \hat{y}_j) \le t)\right\}. \tag{7.6}$$

It can be shown that under the CD estimator of the small area distribution function the mean functional defined by (7.6) takes the value

$$\hat{\overline{Y}}_d^{CD} = \int_{-\infty}^{\infty} td\hat{F}_d^{CD}(t) = N_d^{-1}\left\{\sum_{j\in s_d} y_j + \sum_{j\in r_d} \hat{y}_j + (f_d^{-1} - 1)\sum_{j\in s_d} (y_j - \hat{y}_j)\right\} \tag{7.7}$$

where $f_d = n_d N_d^{-1}$ is the sampling fraction in area d, $\hat{y}_j = \mathbf{x}_j\hat{\beta}_\psi(\hat{\theta}_d)$, where $\hat{y}_j$ can be obtained either under the linear or the nonparametric M-quantile small area models. We refer to (7.7) as the bias adjusted M-quantile mean predictor. Due to the bias correction in (7.7), this predictor will have higher variability than (7.5) and so it should only be used when (7.4) are expected to have substantial bias, e.g. when there are large outlying data points.

7.2 The Software: description of R functions

In this document we present the function mq.sae that is designed for producing small area mean estimates under the M-quantile small area model proposed by Chambers and Tzavidis (2006) and Tzavidis, Marchetti and Chambers (2010).The computation of small area estimates and of the corresponding Mean Squared Error (MSE) using the R software is illustrated with a simulated data set. Full R codes are included in Appendix 7.

7.2.1 Required R packages

Before using mq.sae install the following packages

- MASS

7.2.2 mq.sae

```
>mq.sae(y,x,regioncode.s,m,p,x.outs,regioncode.r,tol.value,
 maxit.value,k.value)
```

- *x*: a $n \times p$ matrix of auxiliary variables which also has include a vector of ones for the intercept term
- *y*: the (numeric) response vector for sampled units
- *regioncode.s*: area code for sampled units
- *m*: the number of small areas
- *p*: size of x +1 (including the intercept)
- *x*.outs: covariate information for out of sample units
- *regioncode.r*: area code for out of sample units
- *tol.value*: convergence tolerance limit for the M-quantile model. Default to 0.0001
- *maxit.value*: maximum number of iterations for the iterative weighted least squares. Default to 100
- *k.value*: tuning constant used with the Huber proposal 2 scale estimation. Default to 1.345

The function returns small area estimates of the mean under the M-quantile model as well as the corresponding MSE estimates.

- *mq.cd*: Estimates of small area means using the M-quantile Chambers-Dunstan estimator (Tzavidis et al. 2010)
- *mq.naive*: Estimates of small area means using the M-quantile naive estimator (Chambers and Tzavidis 2006)
- *mse.cd*: MSE estimates for the M-quantile CD small area means
- *mse.naive*: MSE estimates for the M-quantile naive small area means
- *code.area*: the codes of the small areas

7.3 Examples of usage of R functions

7.3.1 Data generation

For illustrating the use of the mq.sae function, data are generated under the following location-shift model

$$y_{ij} = 100 + 2x_{ij} + g_j + e_{ij}, i = 1, ..., 5, j = 1, ..., 40,$$

where the values of $x \sim LogNormal(\mathrm{N}, log(4.5) - 0.5, 0.5)$ and the error terms are generated from $g_j \sim N(0,9)$ and $e_{ij} \sim N(0,36)$.

```
> # MQ-EBLUP
> source("c:\\MQ_sae.R")
> library(pps)
> sigmasq.u=3
> sigmasq=6
> NoSim<-1
> m=40
> ni=rep(5,m)
> Ni=rep(100,m)
> N=sum(Ni)
> n=sum(ni)
> set.seed(1973)
> u=rnorm(m,0,sqrt(sigmasq.u))
> u=rep(u,each=100)
> e <- rnorm(N, 0, sqrt(sigmasq))
> gr=rep(1:40,each=100)
> ar=unique(gr)
> uno=matrix(c(rlnorm(N,log(4.5)-0.5,0.5)),nrow=N,ncol=1
> y=100+5*uno+u+e
> pop.matrix<-cbind(y,uno,gr)
> pop<-as.data.frame(pop.matrix)
> names(pop)<-c("y","x","area")
> # Drawing a sample
> s<-stratsrs(pop$area,ni)
> x.lme=pop[s,]$x
> y.lme=pop[s,]$y
> regioncode.lme=pop[s,]$area
> pop.r<-pop[-s,]
```

7.3.2 Example of R code for running function mq.sae

```
tmp<-mq.sae(y=y.lme,x=x.lme,regioncode.s=regioncode.lme,m=40,
p=2,x.outs=pop.r[,2], regioncode.r=pop.r[,3],tol.value=0.0001,
maxit.value=100,k.value=1.345)
```

7.3.3 Output of function mq.sae

```
> tmp

mq.cd
```

```
[1] 115.7275 117.9384 115.3374 115.5339 116.3331 ...

mq.naive
 [1] 115.9003 117.6583 115.0192 115.6947 116.0871 ...

mse.cd
 [1] 0.55237498 0.80473242 1.54140859 0.75538562 2.13604316 ...

mse.naive
[1] 0.09710564 0.02790977 0.16425263 0.05226719 0.12559878...

code.area

[1]  1  2  3  4  5...
```

Chapter 8

Nonparametric M-quantile small area estimators of the mean

8.1 Methodology

M-quantile models do not depend on strong distributional assumptions, but they assume that the quantiles of the distribution are some known parametric function of the covariates. When the functional form of the relationship between the q-th M-quantile and the covariates deviates from the assumed one, the traditional M-quantile regression can lead to biased estimates of the β coefficients. Pratesi et al. (2008) and Salvati et al. (2010) have extended this approach to the M-quantile method for the estimation of the small area parameters using a nonparametric specification of the conditional M-quantile of the response variable given the covariates. When the functional form of the relationship between the q-th M-quantile and the covariates deviates from the assumed one, the traditional M-quantile regression can lead to biased estimators of the small area parameters. Using p-splines for M-quantile regression, beyond having the properties of M-quantile models, allows for dealing with an undefined functional relationship that can be estimated from the data. When the relationship between the q-th M-quantile and the covariates is not linear, a p-splines M-quantile regression model may have significant advantages compared to the linear M-quantile model.

Let us consider only smoothing with one covariate x_1, a nonparametric model for the qth quantile can be written as $Q_q(x_1,\psi) = \tilde{m}_{\psi,q}(x_1)$, where the function $\tilde{m}_{\psi,q}(\cdot)$ is unknown and, in the smoothing context, usually assumed to be continuous and differentiable. Here, we will assume that it can be approximated sufficiently well by the following function

$$m_{\psi,q}[x_1;\beta_\psi(q),\gamma_\psi(q)] = \beta_{0\psi}(q) + \beta_{1\psi}(q)x_1 + \ldots + \beta_{p\psi}(q)x_1^p + \sum_{k=1}^{K} \gamma_{k\psi}(q)(x_1-\kappa_k)_+^p, \tag{8.1}$$

where p is the degree of the spline, $(t)_+^p = t^p$ if $t > 0$ and 0 otherwise, κ_k for $k = 1,\ldots,K$ is a set of fixed knots, $\beta_\psi(q) = (\beta_{0\psi}(q),\beta_{1\psi}(q),\ldots,\beta_{p\psi}(q))^t$ is the coefficient vector of the parametric portion of the model and $\beta\gamma_\psi(q) = (\gamma_{1\psi}(q),\ldots,\gamma_{K\psi}(q))^t$ is the coefficient vector for the spline one. The latter portion of the model allows for handling nonlinearities in the structure of the relationship. The spline model (11.1) uses a truncated polynomial spline basis to approximate the function $\tilde{m}_{\psi,q}(\cdot)$. Other bases can be used; in particular radial basis functions can be used to handle bivariate smoothing. More details on bases and knots choice can be found in Ruppert et al. (2003).

Salvati et al. (2010) have applied the P-splines M-quantile regression to the estimation of a small area mean as follows. The first step is to estimate the M-quantile coefficients q_{jd} as illustrated in the Deliverable D12 and D16 for the linear case treated in Chambers and Tzavidis (2006). Recall that the M-quantile coefficient q_{jd} of unit j in area d is the value q_{jd} such that $Q_{q_{jd}}(x_{1jd},\psi) = y_{jd}$. The unit level coefficients are estimated by defining a fine grid of values on the interval $(0,1)$ and using the sample data to fit the p-splines M-quantile regression functions at each value q on this grid. If a data point lies exactly on the q-th fitted curve, then the coefficient of the corresponding sample unit is equal to q. Otherwise, to obtain q_{jd}, a linear interpolation over the grid is used. An estimate of the mean quantile for area d is obtained by taking the corresponding average value of the sample M-quantile coefficient of each unit in area d. The small area estimator of the mean may be taken as:

$$\hat{\bar{Y}}_d = \frac{1}{N_d}\Big\{\sum_{j\in s_d} y_{jd} + \sum_{j\in r_d} \hat{y}_{jd}\Big\}, \tag{8.2}$$

where the unobserved value for population unit $j \in r_d$ is predicted using

$$\hat{y}_{jd} = \mathbf{x}_{jd}\hat{\beta}_\psi(\hat{\theta}_d) + \mathbf{z}_{jd}\hat{\gamma}_\psi(\hat{\theta}_d),$$

where $\hat{\beta}_\psi(\hat{\theta}_d)$ and $\hat{\gamma}_\psi(\hat{\theta}_d)$ are the coefficient vectors of the parametric and spline portion, respectively, of the fitted p-splines M-quantile regression function at $\hat{\theta}_d$.

However, the estimator of the small area mean can be biased for small areas containing outliers. This has already been noted in Chambers and Tzavids (2006) for the estimator under the a linear M-quantile regression model. They propose an adjustment for bias based on the Chambers and Dusntan (1986) estimator of the small area distribution function. This adjustment can be used also in case of p-splines M-quantile regression models. The bias-adjusted estimator for the mean is given by

$$\hat{\bar{Y}}_d^{NPMQ} = \frac{1}{N_d}\Big\{\sum_{j\in U_d} \hat{y}_{jd} + \frac{N_d}{n_d}\sum_{j\in s_d}(y_{jd} - \hat{y}_{jd})\Big\}, \tag{8.3}$$

where $\hat{y}_{jd}$ denotes the predicted values for the population units in s_d and in U_d.

Due to the bias correction in (8.3), this predictor will have higher variability and so should only be used when the estimator (8.2) is expected to have substantial bias, e.g. when there are large outlying data points. An alternative approach to dealing with the bias-variance trade off in (8.3) in such a situation is to limit the variability of the bias correction term in (8.3) by using robust (huberized) residuals instead of raw residuals. In particular,

$$\hat{\bar{Y}}_d = \frac{1}{N_d}\Big\{\sum_{j\in s_d} y_{jd} + \sum_{j\in r_d} \hat{y}_{jd} + \frac{N_d - n_d}{n_d}\sum_{j\in s_d}\nu_d\psi\Big(\frac{y_{jd} - \hat{y}_{jd}}{\nu_d}\Big)\Big\} \tag{8.4}$$

where ν_d is a robust estimate of scale for area d (Tzavidis et al., 2010).

8.1.1 Mean squared error estimation

Salvati et al. (2010) propose also an estimator of the MSE of the small area mean. For fixed q and λ, the $\hat{\bar{Y}}_j$ in (8.4) can be written as the following linear combination of the observed y_{jd} plus an additional part due to the huberized residuals. In particular,

$$\hat{\bar{Y}}_d = \frac{1}{N_d}\sum_{j\in s} w_{jd} y_{jd}, \tag{8.5}$$

where the weights $\mathbf{w}_d = (w_{1d}, \dots, w_{nd})^T$ are given by

$$\mathbf{w}_d = \Big\{1 + \frac{N_d - n_d}{n_d} b_{jd}\Big\}\mathbf{1}_{s_d} + \tag{8.6}$$

$$+\mathbf{W}(\hat{\theta}_d)[\mathbf{X},\mathbf{Z}]\Big([\mathbf{X},\mathbf{Z}]\text{trace}\mathbf{W}(\hat{\theta}_d)[\mathbf{X},\mathbf{Z}] + \lambda\mathbf{G}\Big)^{-1}\Big(\mathbf{T}_{r_d} - \frac{N_d - n_d}{n_d}\mathbf{T}_{s_d}\Big) \tag{8.7}$$

with $b_{jd} = \psi\left(\frac{y_{jd}-\hat{y}_{jd}}{\nu_d}\right)/\left(\frac{y_{jd}-\hat{y}_{jd}}{\nu_d}\right)$, $\mathbf{1}_{s_d}$ the n-vector with j^{th} component equal to one whenever the corresponding sample unit is in area j and to zero otherwise, $\mathbf{W}(\hat{\theta}_d)$ a diagonal $n \times n$ matrix that contains the final set of weights produced by the iteratively reweighted penalized least squares algorithm used to estimate the regression coefficients, $\mathbf{G} = \text{diag}\{\mathbf{0}_{1+p}, \mathbf{1}_K\}$ with $1+p$ the number of columns of $\mathbf{X}$ and K the number of columns of $\mathbf{Z}$, and with $\mathbf{T}_{r_d}$ and $\mathbf{T}_{s_d}$ the totals of the covariates for the non-sampled and the sampled units in area d, respectively. Note that $\mathbf{T}_{s_d} = \sum_{j \in s_d} [\mathbf{x}_{jd}\ \mathbf{z}_{jd}]^T b_{jd}$.

The weights derived from (8.7) are treated as fixed and a "plug in" estimator of the mean squared error of estimator (8.5) can be proposed by using standard methods for robust estimation of the variance of unbiased weighted linear estimators (Royall and Cumberland, 1978) and by following the results due to Chambers and Tzavidis (2006). The prediction variance of (8.5) can be approximated by

$$\text{var}(\hat{\bar{Y}}_d - \bar{Y}_d) \approx \frac{1}{N_d^2}\Big[\sum_{j \in s_d}\Big\{d_{jd}^2 + \frac{N_d - n_d}{n_d - 1}\Big\}\text{var}(y_{jd}) + \sum_{j \in s \setminus s_d} b_{jd}^2 \text{var}(y_{jd})\Big] \qquad (8.8)$$

with $b_{jd} = w_{jd} - 1$ if $j \in s_d$ and $b_{jd} = w_{jd}$ otherwise, and $s \setminus s_d$ the set of sampled units outside area d. Following the area level residual approach Chambers and Tzavidis (2006), we can interpret $\text{var}(y_{jd})$ conditionally to the specific area d from which y_d is drawn and hence replace $\text{var}(y_{jd})$ in (8.8) by $(y_{jd} - \hat{y}_{jd})^2$. Salvati et al. (2010) develop a robust estimator of the mean squared error of (8.5) that is given by

$$\widehat{\text{var}}(\hat{\bar{Y}}_d) = \frac{1}{N_d^2}\left[\sum_{j \in s_d}\left\{b_{jd}^2 + \frac{N_d - n_d}{n_d - 1}\right\}(y_{jd} - \hat{y}_{jd})^2 + \sum_{j \in s \setminus s_d} b_{jd}^2 (y_{jd} - \hat{y}_{jd})^2\right]. \qquad (8.9)$$

Since the bias-adjusted nonparametric M-quantile estimator is an approximately unbiased estimator of the small area mean, the squared bias will not impact significantly the mean squared error estimator. The main limitation of the MSE estimator is that it does not account for the variability introduced in estimating the area specific q's and λ. We note also that we can obtain an estimate only for areas where there are at least two sampled units.

8.2 The Software: description of R functions

In this document we present the function *sae.npmq* that is designed for producing estimates of the mean of the small area distribution function using the nonparametric M-quantile small area model. The function produces point estimates of small area means as well as estimates of Mean Squared Error (MSE) using the methodology in Salvati et al. 2010. The computation of the small area estimates and of the corresponding MSEs using the **R** software is illustrated with a simulated data set.

8.3 Required R packages

Before using sae.npmq install the following packages

- MASS
- spline

8.4 Usage

```
sae.npmq(y,x,z.spline,z.spline.r,regioncode.s,m,p,x.outs,regioncode.r,
tol.value=0.0001,maxit.value=100,k.value=1.345)
```

8.5 Arguments

- *y*: the response variable for sampled units
- *x*: the sample design matrix to be used for fitting the M-quantile model (without intercept)
- *zspline*: the sample design matrix of the nonlinear variable to be used for splines
- *zspline.r*: the design matrix of the nonlinear variable of the entire population
- *regioncode.s*: vector of area codes
- *m*: number of small areas
- *p*: number of auxiliary variable (with intercept)
- *x.outs*: the design matrix for the non sampled units
- *regioncode.r*: vector of area codes for the population data
- *tol.value*: convergence tolerance limit for the M-quantile model. Default to 0.0001
- *maxit.value*: maximum number of iterations for the iterative weighted least squares. Default to 100
- *k.value*: tuning constant used with the Huber proposal 2 scale estimation. Default to 1.345

8.6 Value

The function returns small area estimates of the mean under the M-quantile model as well as the corresponding MSE estimates.

- *npmq.cd*: Estimates of small area means using the M-quantile Chambers-Dunstan estimator (Salvati et al. 2010)
- *npmq.naive*: Estimates of small area means using the M-quantile naive estimator (Salvati et al. 2010)
- *mse.cd*: MSE estimates for the M-quantile CD small area means
- *mse.naive*: MSE estimates for the M-quantile naive small area means
- *code.area*: the codes of the small areas

8.7 Examples of usage of R functions

8.7.1 Data generation

```
source("npmq.sae.R")

#GENERATE DATA
m=30
set.seed(1975)
ar<-seq(1,m)
pop.size=c(525, 538, 510, 468, 526, 484, 516, 458, 529, 518, 502, 524, 509,
484, 487, 459, 542, 498, 512,500, 497, 492, 443, 506, 513, 536, 506, 495,
463, 460)

ni=rep(10,m)
```

```
Ni=pop.size
n<-sum(ni)
N<-sum(Ni)


X<-rchisq(sum(pop.size),20)
e=rnorm(sum(pop.size),0,4)
regioncode<-rep(1:m,pop.size)            # Population region codes
d1<-cbind(X,e,regioncode)
gamma0=rnorm(30,0,2)

beta0=500; beta1=2
Y1=beta0+sin(beta1*pi*(d1[,1][d1[,3]==1]))+(gamma0[1])+(d1[,2][d1[,3]==1])
Y2=beta0+sin(beta1*pi*(d1[,1][d1[,3]==2]))+(gamma0[2])+(d1[,2][d1[,3]==2])
Y3=beta0+sin(beta1*pi*(d1[,1][d1[,3]==3]))+(gamma0[3])+(d1[,2][d1[,3]==3])
Y4=beta0+sin(beta1*pi*(d1[,1][d1[,3]==4]))+(gamma0[4])+(d1[,2][d1[,3]==4])
Y5=beta0+sin(beta1*pi*(d1[,1][d1[,3]==5]))+(gamma0[5])+(d1[,2][d1[,3]==5])
Y6=beta0+sin(beta1*pi*(d1[,1][d1[,3]==6]))+(gamma0[6])+(d1[,2][d1[,3]==6])
Y7=beta0+sin(beta1*pi*(d1[,1][d1[,3]==7]))+(gamma0[7])+(d1[,2][d1[,3]==7])
Y8=beta0+sin(beta1*pi*(d1[,1][d1[,3]==8]))+(gamma0[8])+(d1[,2][d1[,3]==8])
Y9=beta0+sin(beta1*pi*(d1[,1][d1[,3]==9]))+(gamma0[9])+(d1[,2][d1[,3]==9])
Y10=beta0+sin(beta1*pi*(d1[,1][d1[,3]==10]))+(gamma0[10])+(d1[,2][d1[,3]==10])
Y11=beta0+sin(beta1*pi*(d1[,1][d1[,3]==11]))+(gamma0[11])+(d1[,2][d1[,3]==11])
Y12=beta0+sin(beta1*pi*(d1[,1][d1[,3]==12]))+(gamma0[12])+(d1[,2][d1[,3]==12])
Y13=beta0+sin(beta1*pi*(d1[,1][d1[,3]==13]))+(gamma0[13])+(d1[,2][d1[,3]==13])
Y14=beta0+sin(beta1*pi*(d1[,1][d1[,3]==14]))+(gamma0[14])+(d1[,2][d1[,3]==14])
Y15=beta0+sin(beta1*pi*(d1[,1][d1[,3]==15]))+(gamma0[15])+(d1[,2][d1[,3]==15])
Y16=beta0+sin(beta1*pi*(d1[,1][d1[,3]==16]))+(gamma0[16])+(d1[,2][d1[,3]==16])
Y17=beta0+sin(beta1*pi*(d1[,1][d1[,3]==17]))+(gamma0[17])+(d1[,2][d1[,3]==17])
Y18=beta0+sin(beta1*pi*(d1[,1][d1[,3]==18]))+(gamma0[18])+(d1[,2][d1[,3]==18])
Y19=beta0+sin(beta1*pi*(d1[,1][d1[,3]==19]))+(gamma0[19])+(d1[,2][d1[,3]==19])
Y20=beta0+sin(beta1*pi*(d1[,1][d1[,3]==20]))+(gamma0[20])+(d1[,2][d1[,3]==20])
Y21=beta0+sin(beta1*pi*(d1[,1][d1[,3]==21]))+(gamma0[21])+(d1[,2][d1[,3]==21])
Y22=beta0+sin(beta1*pi*(d1[,1][d1[,3]==22]))+(gamma0[22])+(d1[,2][d1[,3]==22])
Y23=beta0+sin(beta1*pi*(d1[,1][d1[,3]==23]))+(gamma0[23])+(d1[,2][d1[,3]==23])
Y24=beta0+sin(beta1*pi*(d1[,1][d1[,3]==24]))+(gamma0[24])+(d1[,2][d1[,3]==24])
Y25=beta0+sin(beta1*pi*(d1[,1][d1[,3]==25]))+(gamma0[25])+(d1[,2][d1[,3]==25])
Y26=beta0+sin(beta1*pi*(d1[,1][d1[,3]==26]))+(gamma0[26])+(d1[,2][d1[,3]==26])
Y27=beta0+sin(beta1*pi*(d1[,1][d1[,3]==27]))+(gamma0[27])+(d1[,2][d1[,3]==27])
Y28=beta0+sin(beta1*pi*(d1[,1][d1[,3]==28]))+(gamma0[28])+(d1[,2][d1[,3]==28])
Y29=beta0+sin(beta1*pi*(d1[,1][d1[,3]==29]))+(gamma0[29])+(d1[,2][d1[,3]==29])
Y30=beta0+sin(beta1*pi*(d1[,1][d1[,3]==30]))+(gamma0[30])+(d1[,2][d1[,3]==30])

Y<-c(Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10,Y11,Y12,Y13,Y14,Y15,Y16,Y17,Y18,Y19,Y20,
Y21,Y22,Y23,Y24,Y25,Y26,Y27,Y28,Y29,Y30)

id<-seq(1:sum(pop.size))
pop<-cbind(id,Y,X,e,regioncode);dim(pop)
```

```
Z.pop=matrix(0,N,m)
t=0
        for (j in 1:m){for(i in 1:Ni[j]){
        t=t+1
        Z.pop[t,j]=1}}

XP<- cbind(rep(1,N),pop[,3]) ;dim(XP) # Population level Design Matrix

knots<-default.knots(pop[,3])
z.spline<-outer(pop[,3],knots,"-")
z.spline<-z.spline*(z.spline>0)

# Drawing a sample
s1<-sample(pop[,1][pop[,5]==ar[1]],ni[1])
s2<-sample(pop[,1][pop[,5]==ar[2]],ni[2])
s3<-sample(pop[,1][pop[,5]==ar[3]],ni[3])
s4<-sample(pop[,1][pop[,5]==ar[4]],ni[4])
s5<-sample(pop[,1][pop[,5]==ar[5]],ni[5])
s6<-sample(pop[,1][pop[,5]==ar[6]],ni[6])
s7<-sample(pop[,1][pop[,5]==ar[7]],ni[7])
s8<-sample(pop[,1][pop[,5]==ar[8]],ni[8])
s9<-sample(pop[,1][pop[,5]==ar[9]],ni[9])
s10<-sample(pop[,1][pop[,5]==ar[10]],ni[10])
s11<-sample(pop[,1][pop[,5]==ar[11]],ni[11])
s12<-sample(pop[,1][pop[,5]==ar[12]],ni[12])
s13<-sample(pop[,1][pop[,5]==ar[13]],ni[13])
s14<-sample(pop[,1][pop[,5]==ar[14]],ni[14])
s15<-sample(pop[,1][pop[,5]==ar[15]],ni[15])
s16<-sample(pop[,1][pop[,5]==ar[16]],ni[16])
s17<-sample(pop[,1][pop[,5]==ar[17]],ni[17])
s18<-sample(pop[,1][pop[,5]==ar[18]],ni[18])
s19<-sample(pop[,1][pop[,5]==ar[19]],ni[19])
s20<-sample(pop[,1][pop[,5]==ar[20]],ni[20])
s21<-sample(pop[,1][pop[,5]==ar[21]],ni[21])
s22<-sample(pop[,1][pop[,5]==ar[22]],ni[22])
s23<-sample(pop[,1][pop[,5]==ar[23]],ni[23])
s24<-sample(pop[,1][pop[,5]==ar[24]],ni[24])
s25<-sample(pop[,1][pop[,5]==ar[25]],ni[25])
s26<-sample(pop[,1][pop[,5]==ar[26]],ni[26])
s27<-sample(pop[,1][pop[,5]==ar[27]],ni[27])
s28<-sample(pop[,1][pop[,5]==ar[28]],ni[28])
s29<-sample(pop[,1][pop[,5]==ar[29]],ni[29])
s30<-sample(pop[,1][pop[,5]==ar[30]],ni[30])
s<-c(s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,
s21,s22,s23,s24,s25,s26,s27,s28,s29,s30)

y<-pop[s,2]
x<-pop[s,3]
```

```
z.spline.s=z.spline[s,]
z.spline.r=z.spline[-s,]

regioncode.s<-pop[s,5]
regioncode.r<-pop[-s,5]
```

8.7.2 Example of R code for running function npmq.sae

```
tmp<-sae.npmq(y,x,z.spline=z.spline[s,],z.spline.r=z.spline[-s,],
regioncode.s=pop[s,5],m=30,p=2,x.outs=XP[-s,2],regioncode.r=pop[-s,5],
tol.value=0.0001,maxit.value=100,k.value=1.345)
```

8.7.3 Output of function npmq.sae

```
#See the output of the function
tmp

$npmq.cd
 [1] 499.6882 499.2256 500.3049 504.9718 497.5279 ...

$npmq.naive
 [1] 500.1488 499.7740 500.4263 504.1665 498.7667 ...

$mse.cd
 [1] 1.4434034 0.8887345 0.9638141 0.5280205 1.5045435 ...

$mse.naive
 [1]  0.22021365  0.56203408  0.08885575  7.57940161  2.64952267  ...

$code.area
 [1]  1  2  3  4  5  ...
```

Chapter 9

M-quantile Geographically Weighted Regression

9.1 Methodology

Typically, random effects models assume independence of the random area effects. This independence assumption is also implicit in M-quantile small area models. In economic applications, however, observations that are spatially close may be more related than observations that are further apart. This spatial correlation can be accounted for by extending the random effects model to allow for spatially correlated area effects using, for example, a Simultaneous Autoregressive (SAR) model (Petrucci and Salvati, 2006; Pratesi and Salvati, 2008; Pratesi and Salvati, 2009). An alternative approach to incorporate the spatial information in the regression model is by assuming that the regression coefficients vary spatially across the geography of interest. Geographically Weighted Regression (GWR) (Brundson et al. 1996) extends the traditional regression model by allowing local rather than global parameters to be estimated. In a recent paper Salvati et al. (2008) proposed an M-quantile GWR small area model. The authors proposed an extension to the GWR model, the M-quantile GWR model, i.e. a locally robust model for the M-quantiles of the conditional distribution of the outcome variable given the covariates. Here we report a brief description of the M-quantile GWR model.

The GWR model is a model for the conditional expectation of $\mathbf{y}$ given $\mathbf{X}$ at location u. This is easily generalised to a model for the M-quantile of order q of the conditional distribution of $\mathbf{y}$ given $\mathbf{X}$ at u. That is, we write

$$Q_q(\mathbf{X};\psi,u) = \mathbf{X}\beta_\psi(u;q) \tag{9.1}$$

where $\beta_\psi(u;q)$ varies with u as well as with q. That is, model (9.1) allows the entire conditional distribution (not just the mean) of y given $\mathbf{X}$ to vary from location to location. The parameter $\beta_\psi(u;q)$ in (9.1) can be estimated by solving normal equations by an iteratively re-weighted least squares algorithm that combines the iteratively re-weighted least squares algorithm used to fit a "spatially stationary" M-quantile model and the weighted least squares algorithm used to fit a GWR model.

The model (9.1) was then used to define a predictor of the small area characteristic of interest that accounts for spatial structure of the data. The M-quantile GWR small area model integrates the concepts of robust small area estimation and borrowing strength over space within a unified modeling framework. Extending further the M-quantile GWR for poverty measures will enable the comparison of alternative robust models for borrowing strength over space in small area estimation and will significantly improve the collection of small area estimation tools.

9.2 The Software: description of R functions

In this document we explain how to produce Small Area estimates using MQGWR and we present an example. The model and the estimator are described in Salvati, Tzavidis, Pratesi and Chambers (2008) and the computation of small area estimates using the R software is illustrated with a simulated data set. Full R codes are included in Appendix 9.

9.2.1 mqgwr.sae

```
> mqgwr.sae(x,y,m,area,lon,lat,x.r,area.r,lon.r,lat.r,method,
  k.value=1.345,mqgwrweight)
```

- *x*: a $n \times p$ matrix of auxiliary variables with the intercept term for sampled units
- *y*: the (numeric) response vector for sampled units
- *m*: the number of small areas
- *area*: a vector of small area codes for sampled units
- *lon* is a vector of longitude of points representing the spatial positions of the sampled observations
- *lat*: a vector of latitude of points representing the spatial positions of the sampled observations
- *x.r*: a $(N-n) \times p$ matrix of auxiliary variables with the intercept term for out of sample units
- *area.r*: a $N-n$ vector of small area codes for out of sample units
- *lon.r*: a $N-n$ vector of longitude of points representing the spatial positions of the out of sample observations
- *lat.r*: a $N-n$ vector of latitude of points representing the spatial positions of the out of sample observations
- *k.value*: tuning constant used for Huber proposal 2 scale estimation. Default to 1.345
- *method*: a character string. If 'mqgwr' the M-quantile GWR model is used to fit the M-quantile surface. If 'mqgwr-li' the MQGWR-LI (Local Intercepts) is used. Defaults to 'mqgwr'
- *mqgwrweight*: geographical weighting function: gwr.gauss() if it is TRUE or gwr.bisquare() if it is FALSE. Defaults to TRUE

9.3 Examples of usage of R functions

9.3.1 Example data

The data used in this example is a part of the synthetic population generated by Salvati et al. (2008). This pseudo-population was obtained by using the data from the U.S. Environmental Protection Agency's Environmental Monitoring and Assessment Program (EMAP) Northeast lakes survey. Between 1991 and 1995, researchers from the U.S. Environmental Protection Agency (EPA) conducted an environmental health study of the lakes in the north-eastern states of the U.S.A. For this study, a sample of 334 lakes (or more accurately, lake locations) was selected from the population of 21,026 lakes in these states using a random systematic design. The lakes making up this population are grouped into 113 8-digit Hydrologic Unit Codes (HUCs). We defined HUCs as the small areas of interest, with lakes grouped within HUCs. The variable of interest was Acid Neutralizing Capacity (ANC), an indicator of the acidification risk of water bodies. In addition to ANC values and associated survey weights for the sampled locations, the EMAP data set also contained the elevation and geographical coordinates of the centroid of each lake in the target area. Given the 21,026 lake locations, a synthetic population of ANC individual values were non parametrically simulated using a nearest-neighbour imputation algorithm that retained the spatial structure of the observed ANC values in the EMAP sample data. Details on the data generation are in Salvati et al. (2008). In this example we consider the population for 10 HUCs, in which are grouped 1737 lakes. Moreover we don't draw any units for the last HUC. In other words we have one out of sample area.

```
>source("mqgwr.R")
>library(sampling)
>data.lake=read.table("PopSynthIn.txt",header=TRUE,dec=",")
>s=strata(data.lake,"HUC",size=c(rep(5,9),1))
>s=s[-46,]
>sample=getdata(data.lake,s)
>lake.r=data.lake[-(s$ID_unit),]
```

A total of 45 lakes are drawn from the population. The data frame *sample* contains the observed lakes, whereas the data frame *lake.r* represents the out of sample units.

9.3.2 Example of R code for running function mqgwr.sae

```
>SaeEst=mqgwr.sae(x=sample$x,y=sample$y,m=10,area=sample$HUC,
lon=sample$lon,
lat=sample$lat,x.r=lake.r$x,area.r=lake.r$HUC,lon.r=lake.r$lon,
lat.r=lake.r$lat,k.value=1.345, method="mqgwr", mqgwrweight=TRUE)
```

9.3.3 Output of function mqgwr.sae

```
mqgwr-SAE estimates

data:  EMAP
SaeEst
$Area.code.in
[1] 1010001 1010002 1010003 1010004 1010005 1020001 1020002 1020003
1020004

$Area.code.out
[1] 1020005

$Est.Mean.in
[1] 345.1899 509.2659 387.1552 398.6101 548.1446 253.3449 433.2629
278.2129
356.8936

$Est.Mean.out
[1] 280.9551

$Est.mse.in
[1]  5856.302  2546.586  3523.715  3389.980  3953.433  6490.745
3697.981 9095.215 36428.013
```

Return the small area estimates of the mean and of its MSE:

- $Area.code.in$: the codes of the sampled areas
- $Area.code.out$: the codes of the out of sample areas
- $Est.Mean.in$: the estimates of the small area mean for sampled areas
- $Est.Mean.out$: the estimates of the small area mean for out of sample areas
- $Est.mse.in$: the estimates of the MSE for sampled areas

Chapter 10

M-quantile CD estimators of the CDF

10.1 Methodology

Estimating the quantiles of a distribution function in addition to conventional outliers sensitive measures of central tendency, such as averages, provides a more complete picture of the study variable. This is the case particularly when handling highly skewed variables, such as income or consumption where we expect the median to be different from the mean. In this paper we focus on estimating the quantiles of the small area distribution function using the M-quantile model proposed by Chambers and Tzavidis (2006) and the estimator of the finite population distribution function proposed by Chambers and Dunstan (1986).

In what follows we assume that a vector of p auxiliary variable $\mathbf{x}_i$ is known for each population unit i in small area j and that values of the variable of interest y are available from a random sample, s, that include units from all the small areas of interest. We denote the population size, sample size, sampled part of the population and non sampled part of the population in area j respectively by N_j, n_j, s_j and r_j. We assume that the sum over the areas of N_j and n_j is equal to N and n respectively.

Under the Chambers and Tzavidis (2006) approach the M-quantile small area model for unit i in area j is defined as follows

$$y_{ij} = \mathbf{x}'_{ij}\beta_{\psi}(\theta_j) + \varepsilon_{ij}, \tag{10.1}$$

where ε_{ij} denotes the regression error for the unit i in area j,and $\hat{\theta}_j$ denotes the area specific M-quantile coefficients that describe the intra-area variation. Using the Chambers-Dunstan estimator of the distribution function and the linear M-quantile small area model, an estimator of the small area distribution function is

$$\hat{F}_j^{CD}(t) = N_j^{-1}\left[\sum_{i\in s_j} I(y_{ij} \leq t) + n_j^{-1}\sum_{k\in r_j}\sum_{i\in s_j} I(\mathbf{x}'_{kj}\hat{\beta}_{\psi}(\hat{\theta}_j) + \hat{e}_{ij} \leq t)\right], \tag{10.2}$$

where $\hat{\beta}_{\psi}$ and $\hat{\theta}_j$ are obtained following Chambers and Tzavidis (2006) and $\hat{e}_{ij}$ are the residuals under model (12.1). A point estimate of the small area quantile τ is then obtained by numerically solving

$$\int_{-\infty}^{\hat{q}(j;\tau)} d\hat{F}_j^{CD}(t) = \tau. \tag{10.3}$$

10.1.1 Bootstrap MSE Estimation for Small Area Quantiles

Analytic estimation of the MSE of the estimates of small area quantiles is complex. In this section we describe a semi-parametric bootstrap approach for estimating the MSE of small area quantiles. Our approach is based on an extension of the bootstrap procedure proposed by Lombarda et al. (2003).

Having estimated model (12.1), we compute the estimated M-quantile small area model residuals,

$$\hat{e}_{ij} = y_{ij} - \mathbf{x}'_{ij}\hat{\beta}_{\psi}(\hat{\theta}_j).$$

A bootstrap population $\Omega^* = \{y^*_{ij}, \mathbf{x}_{ij}\}$, $i \in \Omega$, $j = 1, \ldots, d$ can be generated with

$$y^*_{ij} = \mathbf{x}'_{ij}\hat{\beta}_{\psi}(\hat{\theta}_j) + e^*_{ij},$$

where the bootstrap residuals e^*_{ij} are obtained by sampling from an estimator of the distribution function $\hat{G}(t)$ of the model residuals $\hat{e}_{ij}$. For defining $\hat{G}(t)$ we consider two approaches: (1) sampling from the empirical distribution function of the model residuals and (2) sampling from a smoothed distribution function of these residuals. For each abovementioned case sampling of the residuals can be done in two ways, (1) by sampling from the distribution of all residuals without conditioning on the small area. We refer to this as the unconditional approach; and (2) by sampling from the conditional distribution of residuals within small area j. We refer to this as the conditional approach. Hence, in total there are four possible ways of defining e^*_{ij}.

The steps of our bootstrap procedure are as follows. Starting from sample s, selected from a finite population Ω without replacement, we generate B bootstrap populations, Ω^{*b}, using one of the four above mentioned methods for estimating the distribution of the residuals. From each bootstrap population, Ω^{*b}, we select L samples using simple random sampling within the small areas and without replacement in a way such that $n^*_j = n_j$. Bootstrap estimators of the bias and variance of our predictor of the distribution function in area j are defined respectively by

$$\widehat{\text{Bias}}_j = B^{-1}L^{-1}\sum_{b=1}^{B}\sum_{l=1}^{L}\left(\hat{F}_j^{*bl,CD}(t) - F^{*b}_{N,j}(t)\right),$$

$$\widehat{\text{Var}}_j = B^{-1}L^{-1}\sum_{b=1}^{B}\sum_{l=1}^{L}\left(\hat{F}_j^{*bl,CD}(t) - \bar{\hat{F}}_j^{*bl,CD}(t)\right)^2,$$

where $F^{*b}_{N,j}(t)$ is the distribution function of the bth bootstrap population, $\hat{F}_j^{*bl,CD}(t)$ is the Chambers-Dunstan estimator of $F^{*b}_{N,j}(t)$ computed from the lth sample of the bth bootstrap population and $\bar{\hat{F}}_j^{*bl,CD}(t) = L^{-1}\sum_{l=1}^{L}\hat{F}_j^{*bl,CD}(t)$. The bootstrap Mean Squared Error estimator of the estimated small area quantile is then defined as

$$\widehat{MSE}\left(\hat{F}_j^{CD}(t)\right) = \widehat{\text{Var}}_j + \widehat{\text{Bias}}_j^2.$$

Finally, using a normal approximation we can further compute approximate confidence intervals for the estimated small area quantiles.

More details are given in Marchetti, Tzavidis and Pratesi (2011) and in Tzavidis, Marchetti and Chambers (2010).

10.2 The Software: description of R functions

In this document we present the function *MQ.SAE.quant* that is designed for producing estimates of the quantiles of the small area distribution function using the M-quantile small area model and the Chambers-Dunstan estimator of the cumulative distribution function. The function produces point estimates of small area quantiles as well as non-parametric bootstrap estimates of Mean Squared Error (MSE) using the methodology in Marchetti, Tzavidis and Pratesi (2011). The computation of the small area estimates and of the corresponding MSEs using the **R** software is illustrated with a simulated data set.

10.3 Required R packages

Before using MQ.SAE.quant install the following packages

- MASS
- np

10.4 Usage

```
MQ.SAE.quant(q,y,x.design,X.pop,regioncode,regioncodepop,adjseed,MSE,B,R,
method,maxit)
```

10.5 Arguments

- *q*: a vector of quantiles order to be estimated for each small area
- *y*: the response variable for sampled units
- *x.design*: the sample design matrix to be used for fitting the M-quantile model
- *X.pop*: the design matrix for the entire population
- *regioncode*: vector of area codes
- *regioncodepop*: vector of area codes for the population data
- *adjseed*: parameter for the starting point for the numerical solution of the quantile integral. Default to $\max(0.15, mean(y)/500)$
- *MSE*: If TRUE the function computes bootstrap MSE of estimates of small area quantiles. If FALSE only point estimates of small area quantiles are provided
- *B*: number of bootstrap populations to be generated. Default to 1
- *R*: number of bootstrap samples selected from each bootstrap population. Default to 400
- *method*: method defines the type of residuals used for generating the bootstrap population: 'su' (smooth unconditional),'eu' (emprirical unconditional),'sc' (smooth conditional),'ec' (empirical uncoditional). Default is set to 'su'
- *maxit*: number of maximum iteration allowed in the integration algorithm. A warning is provided if algorithm does not converge in maxit iteration
- Remark: data must be ordered by area

10.6 Value

The function returns small area estimates of the quantiles of interest as well as the corresponding estimates of the root MSE.

- *quantiles*: estimates of small area quantiles for each small area. The rows represent the quantiles and the columns the small areas
- *rmse*: estimates of root mean squared error for each small area quantile estimate in each small area. The rows represent the quantiles and the columns the small areas
- *Area.Code*: the area code for which estimates are provided

10.7 Examples of usage of R functions

10.7.1 Data generation

```
source("MQ.SAE.quant.R")

#GENERATE DATA
areas<-10
n.area<-30
n.area.pop<-300
X.pop<-rnorm(n.area.pop*areas,10,2)
X.pop<-cbind(rep(1,n.area.pop*areas),X.pop)
x<-rnorm(n.area*areas,10,2)
x.design<-cbind(rep(1,n.area*areas),x)
y<-5+2*x+rep(rnorm(areas,1,2),n.area)+rnorm(n.area*areas,0,4)

####VARIABLES DESCRIPTION
#areas <- the number of small area
#n.area <- the sample size of the areas
#y <- the target variable
#x <- auxiliary variable
#X.pop <- auxiliary variable for the population

####SET VARIABLES FOR THE ESTIMATION PROCEDURE
qgrid<-c(0.1,0.25,0.50,0.75,0.9) #Set of quantiles to be estimated
regioncode<-rep(1:10,each=30)
regioncodepop<-rep(1:10,each=300)
```

10.7.2 Example of R code for running function mq.sae.quant

```
cdf.cd.est<-MQ.SAE.quant(qgrid,y,x.design,X.pop,regioncode,
regioncodepop, myMSE=TRUE,B=1,R=40,method="eu")
```

10.7.3 Output of function mq.sae.quant

```
#See the output of the function
cdf.cd.est

#Example of the output
$quantiles
            1        2        3        4        5        ...
0.1  18.02338 19.28048 18.15346 18.27154 16.98312 ...
0.25 21.62927 21.91104 21.84364 21.38364 19.85068 ...
0.5  25.88229 25.70668 25.06824 25.30973 24.51608 ...
0.75 32.70681 29.84444 28.61270 29.51154 27.21317 ...
0.9  35.11521 33.07180 31.74385 33.03619 30.07705 ...

$rmse
             1         2         3         4         5          ...
```

```
0.1  1.1595381 1.3712362 1.1664052 1.2772312 1.0793872 ...
0.25 1.0568906 0.8799666 1.0041534 1.1280502 1.0285473 ...
0.5  0.8476319 0.6893870 1.2079049 1.0794124 0.8962513 ...
0.75 1.1068735 0.8294108 0.7490893 0.7651702 1.0633727 ...
0.9  0.9891367 1.0894626 0.9488074 1.1451405 1.1892031 ...

#Get quantile orders, namely "probs" in the R quantile function
as.numeric(row.names(cdf.cd.est$quantiles))
```

Chapter 11

Nonparametric M-quantile CD estimators of the CDF

11.1 Methodology

M-quantile models do not depend on strong distributional assumptions, but they assume that the quantiles of the distribution are some known parametric function of the covariates. When the functional form of the relationship between the q-th M-quantile and the covariates deviates from the assumed one, the traditional M-quantile regression can lead to biased estimates of the β coefficients. Pratesi et al. (2008) and Salvati et al. (2010) have extended this approach to the M-quantile method for the estimation of the small area parameters using a nonparametric specification of the conditional M-quantile of the response variable given the covariates. When the functional form of the relationship between the q-th M-quantile and the covariates deviates from the assumed one, the traditional M-quantile regression can lead to biased estimators of the small area parameters. Using p-splines for M-quantile regression, beyond having the properties of M-quantile models, allows for dealing with an undefined functional relationship that can be estimated from the data. When the relationship between the q-th M-quantile and the covariates is not linear, a p-splines M-quantile regression model may have significant advantages compared to the linear M-quantile model.

Let us consider only smoothing with one covariate x_1, a nonparametric model for the qth quantile can be written as $Q_q(x_1,\psi) = \tilde{m}_{\psi,q}(x_1)$, where the function $\tilde{m}_{\psi,q}(\cdot)$ is unknown and, in the smoothing context, usually assumed to be continuous and differentiable. Here, we will assume that it can be approximated sufficiently well by the following function

$$m_{\psi,q}[x_1;\beta_\psi(q),\gamma_\psi(q)] = \beta_{0\psi}(q) + \beta_{1\psi}(q)x_1 + \ldots + \beta_{p\psi}(q)x_1^p + \sum_{k=1}^{K} \gamma_{k\psi}(q)(x_1 - \kappa_k)_+^p, \tag{11.1}$$

where p is the degree of the spline, $(t)_+^p = t^p$ if $t > 0$ and 0 otherwise, κ_k for $k = 1,\ldots,K$ is a set of fixed knots, $\beta_\psi(q) = (\beta_{0\psi}(q), \beta_{1\psi}(q),\ldots,\beta_{p\psi}(q))^t$ is the coefficient vector of the parametric portion of the model and $\beta\gamma_\psi(q) = (\gamma_{1\psi}(q),\ldots,\gamma_{K\psi}(q))^t$ is the coefficient vector for the spline one. The latter portion of the model allows for handling nonlinearities in the structure of the relationship. The spline model (11.1) uses a truncated polynomial spline basis to approximate the function $\tilde{m}_{\psi,q}(\cdot)$. Other bases can be used; in particular radial basis functions can be used to handle bivariate smoothing. More details on bases and knots choice can be found in Ruppert et al. (2003).

Salvati et al. (2010) have applied the P-splines M-quantile regression to the estimation of a small area mean as follows. The first step is to estimate the M-quantile coefficients q_{jd} as illustrated in the Deliverable D12 and D16 for the linear case treated in Chambers and Tzavidis (2006). Recall that the M-quantile coefficient q_{jd} of unit j in area d is the value q_{jd} such that $Q_{q_{jd}}(x_{1jd},\psi) = y_{jd}$. The unit level coefficients are estimated by defining a fine grid of values on the interval $(0,1)$ and using the sample data to fit the p-splines M-quantile regression functions at each value q on this grid. If a data point lies exactly on the q-th fitted curve, then the coefficient of the corresponding sample unit is equal to q. Otherwise, to obtain q_{jd}, a linear interpolation over the grid is used.

Using the nonparametric M-quantile predictor for the non sampled units we can define a model unbiased estimator of the small area distribution function:

$$\hat{F}_d^{NPQM}(t) = N_d^{-1}\left\{\sum_{j\in s_d} I(y_j \le t) + \sum_{k\in r_d} n_d^{-1}\sum_{j\in s_d} I(\mathbf{x}_{kd}\hat{\beta}_\psi(\hat{\theta}_d) - \mathbf{z}_{jd}\hat{\gamma}_\psi(\hat{\theta}_d) + (y_j - \mathbf{x}_{jd}\hat{\beta}_\psi(\hat{\theta}_d) + \mathbf{z}_{jd}\hat{\gamma}_\psi(\hat{\theta}_d)) \le t)\right\}. \tag{11.2}$$

Similarly to M-quantile small area models, the qth quantile $\hat{\mu}_{qd}$ of the distribution of y in area d is straightforwardly estimated by the solution to

$$\int_{-\infty}^{\hat{\mu}_{qd}} d\hat{F}_d^{NPQM}(t) = q. \tag{11.3}$$

11.2 The Software: description of R functions

In this document we present the function *NPMQ.SAE.quant* that is designed for producing estimates of the quantiles of the small area distribution function using the nonparametric M-quantile small area model. The function produces point estimates of small area quantiles. The computation of the small area estimates using the **R** software is illustrated with a simulated data set.

11.3 Required R packages

Before using NPMQ.SAE.quant install the following packages

- MASS
- SemiPar
- splines

11.4 Usage

```
NPMQ.SAE.quant(q,y,x.design,X.pop,zspline,Zspline.design.pop,regioncode,
regioncodepop,adjseed)
```

11.5 Arguments

- *q*: a vector of quantiles order to be estimated for each small area
- *y*: the response variable for sampled units
- *x.design*: the sample design matrix to be used for fitting the M-quantile model
- *X.pop*: the design matrix for the entire population
- *zspline.design*: the sample design matrix of the nonlinear variable to be used for splines
- *Zspline.pop*: the design matrix of the nonlinear variable of the entire population

- *regioncode*: vector of area codes
- *regioncodepop*: vector of area codes for the population data
- *adjseed*: parameter for the starting point for the numerical solution of the quantile integral. Default to $\max(0.15, mean(y)/500)$
- Remark: data must be ordered by area

11.6 Value

The function returns small area estimates of the quantiles of interest as well as the corresponding estimates of the root MSE.

- *quantiles*: estimates of small area quantiles for each small area. The rows represent the quantiles and the columns the small areas
- *Area.Code*: the area code for which estimates are provided

11.7 Examples of usage of R functions

11.7.1 Data generation

```
source("NPMQ.SAE.quant.R")

#GENERATE DATA
areas<-10
n.area<-30
n.area.pop<-300
Zspline.pop<-(rep(1:10,each=300)/3+rnorm(n.area.pop*areas,0,0.01))
Zspline.pop<-as.matrix(Zspline.pop)
X.pop<-cbind(rep(1,n.area.pop*areas))
x.design<-cbind(rep(1,n.area*areas))
zspline.design<-(rep(1:10,each=30)/3+rnorm(n.area*areas,0,0.01))
zspline.design<-as.matrix(zspline.design)
y<-5+exp(zspline.design)+rep(rnorm(areas,1,2),n.area)+rnorm(n.area*areas,0,4)

####VARIABLES DESCRIPTION
#areas <- the number of small area
#n.area <- the sample size of the areas
#y <- the target variable
#x.design <- auxiliary variable
#zspline.design <- auxiliary nonlinear variable
#X.pop <- auxiliary variable for the population
#Zspline.pop <- auxiliary nonlinear variable for the population

####SET VARIABLES FOR THE ESTIMATION PROCEDURE
qgrid<-c(0.1,0.25,0.50,0.75,0.9) #Set of quantiles to be estimated
regioncode<-rep(1:10,each=30)
regioncodepop<-rep(1:10,each=300)
```

11.7.2 Example of R code for running function mq.sae.quant

```
cdf.cd.est<-NPMQ.SAE.quant(qgrid,y,x.design,X.pop,
zspline.design,Zspline.pop,regioncode,regioncodepop)
```

11.7.3 Output of function npmq.sae.quant

```
#See the output of the function
cdf.cd.est

#Example of the output
$quantiles
           [,1]      [,2]       [,3]      [,4]      [,5]     ...
[1,]  0.9241095  2.264210  0.3753754  6.887994  7.678133 ...
[2,]  4.4202968  3.810345  5.5170922  8.539407  9.044254 ...
[3,]  6.5008797  7.759887  9.5768338 10.421209 12.950599 ...
[4,] 10.4625977  9.840318 11.6160119 14.294551 14.846934 ...
[5,] 12.9032715 13.328124 12.9452193 16.364936 16.272127 ...

$Area.Code
 [1]  1  2  3  4  5  6  7  8  9 10
```

Chapter 12

M-quantile poverty indicators estimators

12.1 Methodology

Relying only on averages may not provide a very informative picture about the distribution of wealth in a small area. In economic applications for example, estimates of average income may not provide an accurate picture of the area wealth due to the high within area inequality. Here we focus exclusively on the estimation of two poverty indicators i.e. the incidence of poverty or *Head Count Ratio* F_0 and the *Poverty Gap* F_1 (**?**).

In what follows we assume that a vector of p auxiliary variable $\mathbf{x}_i$ is known for each population unit i in small area j and that values of the variable of interest y are available from a random sample, s, that include units from all the small areas of interest. We denote the population size, sample size, sampled part of the population and non sampled part of the population in area j respectively by N_j, n_j, s_j and r_j. We assume that the sum over the areas of N_j and n_j is equal to N and n respectively.

Denoting by t the poverty line, the FGT poverty measures for a small area d are defined as

$$F_{\alpha,d} = N_d^{-1} \sum_{j=1}^{n_d} (\frac{t - y_{jd}}{t})^{\alpha} \mathrm{I}(y_{jd} \leqslant t).$$

Setting $\alpha = 0$ defines the *Head Count Ratio* whereas setting $\alpha = 1$ defines the *Poverty Gap*.

Under the Chambers and Tzavidis (2006) approach the M-quantile small area model for unit i in area j is defined as follows

$$y_{ij} = \mathbf{x}'_{ij} \beta_{\psi}(\theta_j) + \varepsilon_{ij}, \tag{12.1}$$

where ε_{ij} denotes the regression error for the unit i in area j,and θ_j denotes the (true and unknown) area specific M-quantile coefficients that describe the intra-area variation. Using the Chambers-Dunstan estimator of the distribution function and the linear M-quantile small area model, an estimator of the small area FGT poverty measures is

$$F_{\alpha d} = N_d^{-1} \Big[\sum_{j \in s_d} (\frac{t - y_{jd}}{t})^{\alpha} \mathrm{I}(y_{jd} \leqslant t) + \sum_{j \in r_d} (\frac{t - y_{jd}}{t})^{\alpha} \mathrm{I}(y_{jd} \leqslant t) \Big],$$

The out of sample component in the expression is estimated under the M-quantile small area model. To estimate $F_{\alpha d}$ we use a smearing-type estimator of the distribution function such as the Chambers-Dunstan estimator. In this case, an estimator $\hat{F}_{\alpha d}^{MQ}$ of $F_{\alpha d}^{MQ}$ is

$$\hat{F}_{ad} = N_d^{-1} \left\{ \sum_{j \in s_d} I(y_j \leq t) + \sum_{k \in r_d} n_d^{-1} \sum_{j \in s_d} I(\hat{y}_k + (y_j - \hat{y}_j) \leq t) \right\}$$

The above can be evaluated using the following procedure.

1 Fit the M-quantile small area model (1.1) using the raw $\mathbf{y}_s$ sample values and obtain estimates of β and q_d;

2 draw an out of sample vector using

$$y^*_{jdr} = \mathbf{x}_{jdr}\hat{\beta}(\hat{\theta}_d) + e^*_{jdr},$$

where e^*_{jdr} is a vector of size $N_d - n_d$ drawn from the Empirical Distribution Function (EDF) of the estimated M-quantile regression residuals or from a smooth version of this distribution and $\hat{\beta}$, $\hat{\theta}_d$ are obtained from the previous step;

3 repeat the process H times. Each time combine the sample data and out of sample data for estimating the target using

$$\hat{F}^{MQ}_{ad} = N_d^{-1}\Big[\sum_{j\in s_d} \mathrm{I}(y_j \leqslant t) + \sum_{j\in r_d} \mathrm{I}(y^*_j \leqslant t)\Big];$$

4 average the results over H simulations.

12.1.1 Bootstrap MSE Estimation for Small Area Poverty Indicators

Analytic estimation of the MSE of the estimates of small area poverty indicators is complex. We describe a semi-parametric bootstrap approach for estimating the MSE of small area poverty indicators. Our approach is based on an extension of the bootstrap procedure proposed by Lombarda et al. (2003).

Having estimated model (12.1), we compute the estimated M-quantile small area model residuals,

$$\hat{e}_{ij} = y_{ij} - \mathbf{x}'_{ij}\hat{\beta}_{\psi}(\hat{\theta}_j).$$

A bootstrap population $\Omega^* = \{y^*_{ij}, \mathbf{x}_{ij}\}$, $i \in \Omega$, $j = 1,\dots,d$ can be generated with

$$y^*_{ij} = \mathbf{x}'_{ij}\hat{\beta}_{\psi}(\hat{\theta}_j) + e^*_{ij},$$

where the bootstrap residuals e^*_{ij} are obtained by sampling from an estimator of the distribution function $\hat{G}(t)$ of the model residuals $\hat{e}_{ij}$. For defining $\hat{G}(t)$ we consider two approaches: (1) sampling from the empirical distribution function of the model residuals and (2) sampling from a smoothed distribution function of these residuals. For each abovementioned case sampling of the residuals can be done in two ways, (1) by sampling from the distribution of all residuals without conditioning on the small area. We refer to this as the unconditional approach; and (2) by sampling from the conditional distribution of residuals within small area j. We refer to this as the conditional approach. Hence, in total there are four possible ways of defining e^*_{ij}.

The steps of our bootstrap procedure are as follows. Starting from sample s, selected from a finite population Ω without replacement, we generate B bootstrap populations, Ω^{*b}, using one of the four above mentioned methods for estimating the distribution of the residuals. From each bootstrap population, Ω^{*b}, we select L samples using simple random sampling within the small areas and without replacement in a way such that $n^*_j = n_j$. Bootstrap estimators of the bias and variance of our predictor of our targets in area j are defined respectively by

$$\widehat{\mathrm{Bias}}_j = B^{-1}L^{-1}\sum_{b=1}^{B}\sum_{l=1}^{L}\left(\hat{F}^{*bl,MQ}_{ad} - F^{*b}_{ad}\right),$$

$$\widehat{\mathrm{Var}}_j = B^{-1}L^{-1}\sum_{b=1}^{B}\sum_{l=1}^{L}\left(\hat{F}^{*bl,MQ}_{ad} - \bar{\hat{F}}^{*bl,MQ}_{ad}\right)^2,$$

where F^{*b}_{ad} is the FGT poverty indicators ($a = 0,1$) of the bth bootstrap population, $\hat{F}^{*bl,MQ}_{ad}$ is the estimator of F^{*b}_{ad} computed from the lth sample of the bth bootstrap population and $\bar{\hat{F}}^{*bl,MQ}_{ad} = L^{-1}\sum_{l=1}^{L}\hat{F}^{*bl,MQ}_{ad}$. The bootstrap Mean Squared Error estimator of the estimated small area quantile is then defined as

$$\widehat{MSE}\left(\hat{F}^{MQ}_{ad}\right) = \widehat{\mathrm{Var}}_j + \widehat{\mathrm{Bias}}^2_j.$$

Finally, using a normal approximation we can further compute approximate confidence intervals for the estimated small area quantiles.

More details are given in Marchetti, Tzavidis and Pratesi (2011).

12.2 The Software: description of R functions

In this document we present the function *MQ.SAE.poverty* that is designed for producing estimates of the small area poverty indicators, namely the head count ratio and the poverty gap, using the M-quantile small area model. The function produces point estimates of small area poverty indicators as well as semiparametric bootstrap estimates of Mean Squared Error (MSE) using the methodology in Marchetti, Tzavidis and Pratesi (2011). The computation of the small area estimates and of the corresponding MSEs using the **R** software is illustrated with a simulated data set.

12.3 Required R packages

Before using MQ.SAE.poverty install the following packages

- MASS
- np

12.4 Usage

```
MQ.SAE.poverty(y,x.design,X.pop,regioncode,regioncodepop,L,MSE,B,R,method,
pov.l)
```

12.5 Arguments

- *y*: the response variable for sampled units
- *x.design*: the sample design matrix to be used for fitting the M-quantile model
- *X.pop*: the design matrix for the entire population
- *regioncode*: vector of area codes
- *regioncodepop*: vector of area codes for the population data
- *L*: number of Monte Carlo runs in the estimation process. Default to 50
- *MSE*: If TRUE the function computes bootstrap MSE of estimates of small area quantiles. If FALSE only point estimates of small area quantiles are provided
- *B*: number of bootstrap populations to be generated. Default to 1
- *R*: number of bootstrap samples selected from each bootstrap population. Default to 400
- *method*: method defines the type of residuals used for generating the bootstrap population: 'su' (smooth unconditional),'eu' (emprirical unconditional),'sc' (smooth conditional),'ec' (empirical uncoditional). Default is set to 'su'
- *pov.l*: the poverty line value. If it is set to NULL the poverty line is computed as $0.6 \times median(y)$
- Remark: data must be ordered by area

12.6 Value

The function returns small area estimates of the head count ratio and poverty gap as well as the corresponding estimates of the root MSE.

- *HCR.MQ*: estimates of small area head count ratio for each small area
- *PG.MQ*: estimates of small area poverty gap for each small area
- *RMSE.HCR.MQ*: estimates of root mean squared error for each small area head count ratio estimate in each small area
- *RMSE.PG.MQ*: estimates of root mean squared error for each small area poverty gap estimate in each small area
- *Area.Code*: the area code for which estimates are provided
- *Pov.Line*: the value of the poverty line used to compute poverty indicators

12.7 Examples of usage of R functions

12.7.1 Data generation

```
source("MQ.SAE.poverty.R")

#GENERATE DATA
areas<-10
n.area<-30
n.area.pop<-300
X.pop<-rnorm(n.area.pop*areas,10,2)
X.pop<-cbind(rep(1,n.area.pop*areas),X.pop)
x<-rnorm(n.area*areas,10,2)
x.design<-cbind(rep(1,n.area*areas),x)
y<-5+2*x+rep(rnorm(areas,1,2),n.area)+rnorm(n.area*areas,0,4)

####VARIABLES DESCRIPTION
#areas <- the number of small area
#n.area <- the sample size of the areas
#y <- the target variable
#x <- auxiliary variable
#X.pop <- auxiliary variable for the population

####SET VARIABLES FOR THE ESTIMATION PROCEDURE
regioncode<-rep(1:10,each=30)
regioncodepop<-rep(1:10,each=300)
```

12.7.2 Example of R code for running function mq.sae.quant

```
poverty.est<-MQ.SAE.poverty(y,x.design,X.pop,regioncode,regioncodepop,L=50,
myMSE=TRUE,myB=1,myR=40,method="eu",pov.l=NULL)
```

12.7.3 Output of function mq.sae.quant

```
#See the output of the function
poverty.est

#Example of the output
$HCR.MQ
 [1] 0.03933333 0.04046667 0.05180000 0.04986667 0.05373333 0.03646667
 [7] 0.03320000 0.04173333 0.04013333 0.03860000

$PG.MQ
 [1] 0.006854624 0.006397652 0.008898864 0.008859422 0.008384273 0.00594783
 [7] 0.005147244 0.006599265 0.006881494 0.006370927

$RMSE.HCR.MQ
 [1] 0.01588061 0.02599965 0.01186003 0.01666897 0.01759610 0.01180885
 [7] 0.01227573 0.01341351 0.01555744 0.01128206

$RMSE.PG.MQ
 [1] 0.002970214 0.002544353 0.003524704 0.005841524 0.003550814 0.002756377
 [7] 0.002695069 0.002796721 0.003396079 0.002700411

$Area.Code
 [1]  1  2  3  4  5  6  7  8  9 10

$Pov.Line
[1] 15.45099
```

Chapter 13

EB prediction of poverty measures with unit level models

13.1 Methodology

13.1.1 EB method for poverty estimation

Consider a population U partitioned in D domains or areas $U_1,\dots,U_D$ of sizes $N_1,\dots,N_D$. A sample $s_d \subset U_d$ of size n_d has been drawn from each domain d, $d=1,\dots,D$. Let E_{dj} be the value of a quantitative welfare measure for j-th individual within d-th domain and z a poverty line defined for the population. Our target parameters are the FGT poverty measures for domain d, given by

$$F_{\alpha d} = \frac{1}{N_d}\sum_{j=1}^{N_d}\left(\frac{z-E_{dj}}{z}\right)^{\alpha} I(E_{dj}<z), \quad \alpha=0,1,2. \tag{13.1}$$

Calculation of the BP of $F_{\alpha d}$ requires to express $F_{\alpha d}$ in terms of a domain vector $\mathbf{y}_d$, for which the conditional distribution of the out-of-sample vector $\mathbf{y}_{dr}$ given sample data $\mathbf{y}_{ds}$ is known. The distribution of the welfare variables E_{dj} is seldom Normal due to the typical right-skewness of these kind of economical variables. However, here we suppose that there is a one-to-one transformation $Y_{dj}=T(E_{dj})$ following a Normal distribution. In particular, we will assume that Y_{dj} follows the nested error model

$$\begin{array}{l} Y_{dj} = \mathbf{x}_{dj}\beta + u_d + e_{dj}, \qquad j=1,\dots,N_d, \quad d=1,\dots D, \\ u_d \sim \text{ iid } N(0,\sigma_u^2), \quad e_{dj} \sim \text{ iid } N(0,\sigma_e^2) \end{array} \tag{13.2}$$

where $\mathbf{x}_{dj}$ is a row vector with the values of p explanatory variables, u_d is a random area-specific effect and e_{dj} are residual errors. Let $\mathbf{y}_d = (\mathbf{y}'_{ds},\mathbf{y}'_{dr})'$ be the vector containing the values of the transformed variables Y_{dj} for the sample and out-of-sample units within domain d. Then $F_{\alpha d}$ is function of $\mathbf{y}_d$, that is

$$F_{\alpha d} = \frac{1}{N_d}\sum_{j=1}^{N_d}\left(\frac{z-T^{-1}(Y_{dj})}{z}\right)^{\alpha} I(T^{-1}(Y_{dj})<z) =: h_\alpha(\mathbf{y}_d), \quad \alpha=0,1,2.$$

Thus, the FGT poverty measure of order α is a non-linear function $h_\alpha(\mathbf{y}_d)$ of $\mathbf{y}_d$. Then the BP of $F_{\alpha d}$ is given by

$$\hat{F}^B_{dj} = E_{\mathbf{y}_{dr}}\left[h_\alpha(\mathbf{y}_d)|\mathbf{y}_{ds}\right] = \int_{\mathbb{R}} h_\alpha(\mathbf{y}_d) f(\mathbf{y}_{dr}|\mathbf{y}_{ds})\, d\mathbf{y}_{dr}, \tag{13.3}$$

where $f(\mathbf{y}_{dr}|\mathbf{y}_{ds})$ is the joint density of $\mathbf{y}_{dr}$ given the observed data vector $\mathbf{y}_{ds}$. Due to the complexity of the function $h_\alpha(\cdot)$, there is not explicit expression for the expectation in (13.3), but this expectation can be approximated by

Monte Carlo. For this, generate L replicates $\{\mathbf{y}_d^{(\ell)}; \ell = 1,\ldots,L\}$ of $\mathbf{y}_d$ from the distribution of $\mathbf{y}_{dr}|\mathbf{y}_{ds}$. In practice, this is done by generating univariate values $Y_{dj}^{(\ell)}$ from the model

$$Y_{dj}^{(\ell)} = \mathbf{x}_{dj}\hat{\beta} + \hat{u}_d + v_d + \varepsilon_{di},\ v_d \sim N(0, \hat{\sigma}_u^2(1-\hat{\gamma}_d)),\ \varepsilon_{dj} \sim N(0, \hat{\sigma}_e^2),\ j \in U_d - s_d,\ d = 1,\ldots,D, \tag{13.4}$$

where $\gamma_d = \sigma_u^2(\sigma_u^2 + \sigma_e^2/n_d)^{-1}$ and n_d is the sample size in domain d. Then, an approximation to the best predictor of $F_{\alpha d}$ is

$$\hat{F}_{\alpha d}^{B} \approx \frac{1}{L}\sum_{\ell=1}^{L} h_\alpha(\mathbf{y}_d^{(\ell)}).$$

Typically, the mean vector μ_d and the covariance matrix $\mathbf{V}_d$ depend on an unknown vector of parameters θ. Then the conditional density $f(\mathbf{y}_{dr}|\mathbf{y}_{ds})$ depends on θ, and we make this explicit by writing $f(\mathbf{y}_{dr}|\mathbf{y}_{ds},\theta)$. We take an estimator $\hat{\theta}$ of θ such as the maximum likelihood (ML) or restricted ML (REML) estimator. Then the expectation can be approximated by generating values from the estimated density $f(\mathbf{y}_{dr}|\mathbf{y}_{ds},\hat{\theta})$. The result is the EBP, denoted $\hat{F}_{\alpha d}^{EB}$.

13.1.2 Parametric bootstrap for MSE estimation

The parametric bootstrap of González-Manteiga et al. (2008) has been used to derive an estimator of the MSE of the EB estimator of the poverty measures. This method works as follows.

(1) Fit the nested-error model (13.2) by ML, REML or Henderson method III, deriving model parameter estimates $\hat{\beta}$, $\hat{\sigma}_u^2$ and $\hat{\sigma}_e^2$.

(2) Generate bootstrap domain effects from:

$$u_d^* \overset{iid}{\sim} N(0, \hat{\sigma}_u^2), \quad d = 1,\ldots,D.$$

(3) Generate, independently of $u_1^*,\ldots,u_D^*$, disturbances:

$$e_{dj}^* \overset{iid}{\sim} N(0, \hat{\sigma}_e^2), \quad j = 1,\ldots,N_d,\ d = 1,\ldots,D$$

(4) Generate a bootstrap population from the model

$$y_{dj}^* = \mathbf{x}_{dj}^T\hat{\beta} + u_d^* + e_{dj}^*, \quad j = 1,\ldots,N_d,\ d = 1,\ldots,D.$$

(5) Let us define the vector $\mathbf{y}_d^* = (y_{d1}^*,\ldots,y_{dN_d}^*)^t$. Calculate target quantities for the bootstrap population

$$F_{\alpha d}^* = h_\alpha(\mathbf{y}_d^*), \quad d = 1,\ldots,D.$$

(6) Let $\mathbf{y}_s^*$ be the vector containing the bootstrap observations y_{dj}^* whose indices are contained in the sample $s = s_1 \cup \cdots \cup s_D$. Fit again the nested-error model (13.2) to bootstrap sample data $\mathbf{y}_s^*$ and obtain bootstrap estimators $\hat{\sigma}_\upsilon^{2*}$, $\hat{\sigma}_e^{2*}$ and $\hat{\beta}^*$.

(7) Using the bootstrap sample data $\mathbf{y}_s^*$, obtain the bootstrap EBP $\hat{F}_{\alpha d}^{EB*}$ through the Monte Carlo approximation.

(8) Repeat (2)–(7) B times. Let $F_{\alpha d}^*(b)$ be true value and $\hat{F}_{\alpha b}^{EB*}(b)$ the EBP for bootstrap sample b, $b = 1,\ldots,B$.

(9) The bootstrap estimator of the MSE is given by

$$\text{mse}_B(\hat{F}_{\alpha d}^{EB}) = B^{-1}\sum_{b=1}^{B}\left\{\hat{F}_{\alpha d}^{EB*}(b) - F_{\alpha d}^*(b)\right\}^2.$$

13.2 The Software: description of R functions

This section describes the implemented R function that obtains EB estimators of domain FGT poverty measures with $\alpha = 0$ (poverty incidence) and $\alpha = 1$ (poverty gap), when the values of the auxiliary variables for out-of-sample data are available. It describes also the R function to obtain the estimated MSEs of the EB estimators. An extra function that obtains approximate EB estimators in the case that only sample data is available is included. Examples of how to implement these functions are provided in Section 13.3 and full codes are included in Appendix 27.

13.2.1 FGTpovertyEB

R function `FGTpovertyEB()` fits the unit level model of Battese, Harter and Fuller (1988) to the log of the welfare variable (adding previously a quantity to make it always positive) and computes empirical EB estimates of domain FGT poverty measures of orders $\alpha = 1$ (poverty incidence) and $\alpha = 2$ (poverty gap). This function is defined as

```
FGTpovertyEB<-function(dom,seldomain=unique(dom),Xrdtot,welfare,Xs,weight,
z=0.6*median(welfare),L=50,seed=Sys.time())
```

The arguments of this function are:

`dom`: Domain indicator. It must contain numbers identifying the domain to which each sample unit belongs.

`seldomain`: A selection of domains in which we want to obtain EB estimators. Default is the set of all (unique) domains.

`Xrdtot`: A list containing a number of matrices equal to the length of seldomain. Matrix i must contain in each column the values of each of p auxiliary variables (including the constant in first column) for the out-of-sample units in i-th selected domain.

`welfare`: welfare variable used to quantify the level of richness of each individual or unit.

`Xs`: $n \times p$ matrix containing the values of p auxiliary variables for the n sample units. The elements in the first column should be all equal to 1 if the model includes and intercept.

`weight`: Sampling weight of the unit.

`z`: Poverty line. Default value is 0.6 times the median of the welfare values for the sample units.

`L`: Number of Monte Carlo replicates for the empirical approximation of the EB estimator. Default value is $L = 50$.

`seed`: Seed to be used in the random number generation of the Monte Carlo replication process. Default is the system time.

The function returns a list with the following objects:

`EstimatedPoverty`: Data.frame with number of rows equal to number of domains, containing in its columns domain indicator (`Domain`), EB estimators of the poverty incidence (`PovInc`) and EB estimators of the poverty gap (`PovGap`).

`ComputTime`: Computation time (min.) of the Monte Carlo approximation of the EB method.

`Resultsfit`: A list containing the following objects: summary of the unit level model fitting (`Summary`), vector with the estimated values of the fixed effects (`FixedEffects`), vector with the predicted random effects (`RandomEffects`), residual variance (`ResVar`), estimated variance of the random effects (`RandomEffVar`), log-likelihood (`Loglike`) and vector of raw residuals (`RawResiduals`).

13.2.2 PBMSE.EB

R function `PBMSE.EB()` obtains estimators of the mean squared errors of EB estimators of FGT poverty measures of order 1 (poverty incidence) and order 2 (poverty gap) by the parametric bootstrap method described in Section 13.1.2. Population values of auxiliary variables are required. This function is defined as

```
PBMSE.EB<-function(dom,seldomain=unique(dom),Xrdtot,welfare,Xs,weight,
z=0.6*median(welfare),B=50,LB=50,seed=Sys.time())
```

The arguments of this function are:

`dom`: Domain indicator. It must contain numbers identifying the domain to which each sample unit belongs.

`seldomain`: A selection of domains in which we want to obtain EB estimators. Default is the set of all (unique) domains.

`Xrdtot`: A list containing a number of matrices equal to the length of seldomain. Matrix i must contain in each column the values of each of p auxiliary variables (including the constant in first column) for the out-of-sample units in i-th selected domain.

`welfare`: welfare variable used to quantify the level of richness of each individual or unit.

`Xs`: $n \times p$ matrix containing the values of p auxiliary variables for the n sample units. The elements in the first column should be all equal to 1 if the model includes and intercept.

`weight`: Sampling weight of the unit.

`z`: Poverty line. Default value is 0.6 times the median of the welfare values for the sample units.

`B`: Number of bootstrap replicates. Default value is $B = 50$.

`LB`: Number of Monte Carlo replicates for the empirical approximation of the EB estimator for each bootstrap sample. Default value is $LB = 50$.

`seed`: Seed to be used in the random number generation of the Monte Carlo replication process. Default is the system time.

The function returns a list with the following objects:

`EstimatedPoverty`: Data.frame with number of rows equal to number of domains, containing in its columns domain indicator (`Domain`), EB estimators of the poverty incidence (`PovInc`) and EB estimators of the poverty gap (`PovGap`).

`ComputTime`: Computation time (min.) of the Monte Carlo approximation of the EB method.

`Resultsfit`: A list containing the following objects: summary of the unit level model fitting (`Summary`), vector with the estimated values of the fixed effects (`FixedEffects`), vector with the predicted random effects (`RandomEffects`), residual variance (`ResVar`), estimated variance of the random effects (`RandomEffVar`), log-likelihood (`Loglike`) and vector of raw residuals (`RawResiduals`).

13.2.3 FGTpovertyEBsample

R function `FGTpovertyEBsample()` fits the unit level model of Battese, Harter and Fuller (1988) to the log of the welfare variable (adding previously a quantity to make it always positive) and computes approximate EB estimates of domain FGT poverty measures with $\alpha = 1$ (poverty incidence) and $\alpha = 2$ (poverty gap). This function assumes that only sample data on the auxiliary variables are available, and then only an approximation of the EB estimates is obtained. Theoretically, the EB method requires the population values of the auxiliary variables. When the auxiliary variables are indicators as in the two example data sets, only the true totals of the auxiliary variables in the covariate classes within domains are required. When these true totals are not available, they can be approximated by the direct estimators obtained with sample data. This function is doing this approximation. The function is defined as

```
FGTpovertyEBsample<-function(dom,seldomain=unique(dom),welfare,Xs,weight,
z=0.6*median(welfare),L=50,seed=Sys.time())
```

Arguments of this function are:

`dom`: Domain indicator. It must contain numbers identifying the domain to which each sample unit belongs.

`seldomain`: A selection of domains in which we want to obtain EB estimators. Default is the set of all (unique) domains.

`welfare`: welfare variable used to quantify the level of richness of each individual or unit.

`Xs`: $n \times p$ matrix containing the values of p auxiliary variables for the n sample units. The elements in the first column should be all equal to 1 if the model includes and intercept.

`weight`: Sampling weight of the unit.

`z`: Poverty line. Default value is 0.6 times the median of the welfare values for the sample units.

`L`: Number of Monte Carlo replicates for the empirical approximation of the EB estimator. Default value is $L = 50$.

`seed`: Seed to be used in the random number generation of the Monte Carlo replication process. Default is the system time.

The function returns a list with the following objects:

`EstimatedPoverty`: Data.frame with number of rows equal to number of domains, containing in its columns domain indicator (`Domain`), sample size (`SampSize`), EB estimators of the poverty incidence (`EBPovInc`) and EB estimators of the poverty gap (`EBPovGap`).

`ComputTime`: Computation time (min.) of the Monte Carlo approximation of the EB method.

`Resultsfit`: A list containing the following objects: summary of the unit level model fitting (`Summary`), vector with the estimated values of the fixed effects (`FixedEffects`), vector with the predicted random effects (`RandomEffects`), residual variance (`ResVar`), estimated variance of the random effects (`RandomEffVar`), log-likelihood (`Loglike`) and vector of raw residuals (`RawResiduals`).

13.3 Examples of usage of R functions

This section describes how to execute the R functions described in Section 13.2 with data sets to produce EB estimates of domain poverty incidences and poverty gaps, along with their estimated MSEs.

13.3.1 Example data set 1

We consider a data set obtained by applying a resampling method to the real data set from the Spanish Survey on Income and Living Conditions from first quarter of year 2006. We consider as domains the Spanish provinces. The welfare variable is the equivalized income (`normin c`) and the auxiliary variables are a constant, indicators of the four age groups 16-24, 25-49, 49-64 and +65 (`age2`, `age3`, `age4` and `age5` respectively), indicator of having Spanish nationality (`nat1`), indicators of education level equal to primary school and third level studies (`educ1` and `educ3` respectively) and indicators of employed and unemployed labor states (`sitemp1` and `sitemp2` respectively). The first 10 rows from this data set look like this:

```
   provl prov ac gen age nat educ sitemp age2 age3 age4 age5 educ1
1  Alava    1 16   2   5   1    3      3    0    0    0    1     0
2  Alava    1 16   2   5   1    1      3    0    0    0    1     1
3  Alava    1 16   1   5   1    2      3    0    0    0    1     0
4  Alava    1 16   1   4   1    2      3    0    0    1    0     0
5  Alava    1 16   2   4   1    1      3    0    0    1    0     1
6  Alava    1 16   1   4   1    2      1    0    0    1    0     0
7  Alava    1 16   2   3   1    2      3    0    1    0    0     0
8  Alava    1 16   1   3   1    2      1    0    1    0    0     0
9  Alava    1 16   1   2   1    2      1    1    0    0    0     0
10 Alava    1 16   2   4   1    2      1    0    0    1    0     0
   educ2 educ3 nat1 sitemp1 sitemp2 sitemp3   norminc    weight
1      0     1    1       0       0       1 10357.931 2133.7582
2      0     0    1       0       0       1 13117.838 2580.5675
3      1     0    1       0       0       1  5450.227 2580.5675
4      1     0    1       0       0       1  7525.012  850.0648
5      0     0    1       0       0       1 23189.200  608.9678
6      1     0    1       1       0       0 25130.136 1596.9997
7      1     0    1       0       0       1  5150.399 1596.9997
8      1     0    1       1       0       0 19713.600 1596.9997
9      1     0    1       1       0       0  1972.133 1596.9997
10     1     0    1       1       0       0 13469.470 1135.6591
...
```

13.3.2 Example data set 2

We consider a simulated data set with $D = 80$ domains with $n_d = 50$ observations in each domain d, for $d = 1,\dots,D$, with two dummy variables as auxiliary variables (`X1` and `X2`) together with the constant (`Constant`), a welfare variable (`Welfare`), the domain indicator (`Domain`), and the sampling weights obtained from a simple random sampling without replacement of 1 out of 5 within each domain (`SampWeight`). This means that the domain sizes are $N_d = 250, d = 1,\dots,D$. The first 10 observations of the data set look like this:

```
     Welfare Constant X1 X2 Domain SampWeight
1  20.362377        1  1  1      1          5
2  12.228778        1  1  0      1          5
3  16.827734        1  0  1      1          5
4   9.864449        1  1  1      1          5
5  12.839637        1  0  0      1          5
6   9.566156        1  1  0      1          5
7  26.127734        1  0  0      1          5
8  10.833009        1  0  1      1          5
```

```
9  27.285174        1  0  0      1          5
10 13.346398        1  0  0      1          5
...
```

13.3.3 Example of R code for running function FGTpovertyEB

This section includes the code required to execute the function FGTpovertyEB and obtain approximate EB estimates of domain poverty incidences and gaps using Example data set 1.

```
# Read the data file silc0106_SAMPLE.txt

data<-read.table("silc0106_SAMPLE.txt",header=TRUE)
attach(data)

# We select for EB estimation the 10 provinces with largest CVs of direct
# estimators of poverty incidence (those in which we could improve more).

povinc15<-c(42,5,34,44,40,21,25,19,16,2)

# These selected domains are, in order, the following Spanish provinces:
# Soria, Avila, Palencia, Teruel, Segovia, Huelva, Lerida, Guadalajara,
# Cuenca and Albacete.

# Create matrix of auxiliary variables X for sample elements

n<-dim(data)[1]
X<-as.matrix(cbind(constant=rep(1,n),age2,age3,age4,age5,nat1,educ1,educ3,
sitemp1,sitemp2))

# We will apply the EB method when out-of-sample X values are available
# Read non-sample values of auxiliary variables from files for selected
# provinces

Xrd1<-as.matrix(read.table("X_Soria.txt",header=TRUE))
Xrd2<-as.matrix(read.table("X_Avila.txt",header=TRUE))
Xrd3<-as.matrix(read.table("X_Palencia.txt",header=TRUE))
Xrd4<-as.matrix(read.table("X_Teruel.txt",header=TRUE))
Xrd5<-as.matrix(read.table("X_Segovia.txt",header=TRUE))
Xrd6<-as.matrix(read.table("X_Huelva.txt",header=TRUE))
Xrd7<-as.matrix(read.table("X_Lerida.txt",header=TRUE))
Xrd8<-as.matrix(read.table("X_Guadalajara.txt",header=TRUE))
Xrd9<-as.matrix(read.table("X_Cuenca.txt",header=TRUE))
Xrd10<-as.matrix(read.table("X_Albacete.txt",header=TRUE))

# Construct design matrix with out-of-sample values

Xrdtot<-list(Xrd1,Xrd2,Xrd3,Xrd4,Xrd5,Xrd6,Xrd7,Xrd8,Xrd9,Xrd10)

# Load file where function is located
```

```
source("FGTpovertyEB.R")

# Execute function to compute EB predictors of poverty measures

resultsEB<-FGTpovertyEB(prov,povinc15,Xrdtot,norminc,X,weight,L=100,seed=1111)
```

13.3.4 Output of function FGTpovertyEB

This section lists the R commands required to print each of the output objects of R function FGTpovertyEB() followed by the obtained R results.

```
print(results$EstimatedPoverty)

--------------------------------------------------
   Domain   PovInc    PovGap
1      42 23.17550  7.263587
2       5 20.37840  5.984184
3      34 22.38464  6.880210
4      44 30.86688 10.243227
5      40 26.70518  8.567364
6      21 18.14058  5.298647
7      25 15.78190  4.430357
8      19 28.30154  9.265694
9      16 24.20650  7.513111
10      2 21.25588  6.349842
--------------------------------------------------

# Print computation time (in min.)
print(results$ComputTime)

--------------------------------------------------
Time difference of 0.7116833 mins
--------------------------------------------------

# Print a summary of the unit level model fit
print(results$Resultsfit$Summary)

--------------------------------------------------
Linear mixed-effects model fit by REML
 Data: NULL
       AIC      BIC    logLik
  49102.49 49203.83 -24539.25

Random effects:
 Formula: ~1 | as.factor(dom)
        (Intercept)  Residual
StdDev:   0.1121727 0.4922459

Fixed effects: ys ~ -1 + Xs
               Value   Std.Error    DF  t-value p-value
```

```
Xsconstant  9.058560 0.021478405 34328 421.7520  0.0000
Xsage2     -0.039202 0.010882325 34328  -3.6023  0.0003
Xsage3     -0.026805 0.010012532 34328  -2.6771  0.0074
Xsage4      0.085182 0.010787907 34328   7.8961  0.0000
Xsage5      0.041878 0.011102710 34328   3.7719  0.0002
Xsnat1      0.296140 0.013678530 34328  21.6500  0.0000
Xseduc1    -0.195997 0.007616290 34328 -25.7340  0.0000
Xseduc3     0.316883 0.008952407 34328  35.3964  0.0000
Xssitemp1   0.185018 0.007468557 34328  24.7730  0.0000
Xssitemp2  -0.066906 0.014897775 34328  -4.4910  0.0000
 Correlation:
          Xscnst Xsage2 Xsage3 Xsage4 Xsage5 Xsnat1 Xsedc1 Xsedc3
Xsage2    -0.170
Xsage3    -0.197  0.581
Xsage4    -0.153  0.506  0.685
Xsage5    -0.142  0.418  0.523  0.601
Xsnat1    -0.608 -0.015  0.002 -0.048 -0.060
Xseduc1   -0.017 -0.115 -0.239 -0.423 -0.564  0.022
Xseduc3    0.012 -0.058 -0.170 -0.161 -0.189 -0.016  0.274
Xssitemp1 -0.009 -0.279 -0.557 -0.353 -0.047  0.016  0.076 -0.118
Xssitemp2 -0.016 -0.200 -0.251 -0.141 -0.006  0.027 -0.001 -0.012
          Xsstm1
Xsage2
Xsage3
Xsage4
Xsage5
Xsnat1
Xseduc1
Xseduc3
Xssitemp1
Xssitemp2  0.303

Standardized Within-Group Residuals:
         Min           Q1          Med           Q3          Max
-19.50339536  -0.61427053   0.05152173   0.68394085   3.32160694

Number of Observations: 34389
Number of Groups: 52
-----------------------------------------------------

# Print the estimated coefficients of covariates
print(resultsEB$Resultsfit$FixedEffects)

-----------------------------------------------------
 Xsconstant      Xsage2      Xsage3      Xsage4      Xsage5
 9.05856019 -0.03920177 -0.02680484  0.08518231  0.04187844
     Xsnat1     Xseduc1     Xseduc3   Xssitemp1   Xssitemp2
 0.29614008 -0.19599729  0.31688257  0.18501826 -0.06690599
-----------------------------------------------------
```

```
# Print the predicted domain random effects
print(results$Resultsfit$RandomEffects[,1])

----------------------------------------------------
 [1] -0.213313405  0.087552888 -0.001434107  0.037389922  0.081442345
 [6]  0.078697984  0.178172026 -0.127733526  0.052459550 -0.142463139
[11]  0.198610343 -0.074441560  0.090803379 -0.084094650 -0.030615355
[16]  0.011658401 -0.074691949 -0.136413547 -0.050338426 -0.060050918
[21]  0.130646937 -0.128257402 -0.051556780 -0.026778165  0.146311096
[26] -0.057990073 -0.105415297  0.028450273 -0.038878506  0.108169754
[31]  0.076302480 -0.045798254 -0.102560779  0.023617850  0.176019784
[36]  0.145902823  0.093859637 -0.022816735 -0.209439577 -0.049812219
[41] -0.024708085  0.037549961 -0.260640462 -0.098005832  0.164347345
[46] -0.036359270  0.095121560 -0.052826579 -0.039266715  0.187297411
[51]  0.077288096  0.039029468
----------------------------------------------------

# Print the residual variance of the unit level model fit
print(resultsEB$Resultsfit$ResVar)

-----------------------------------------------------
[1] 4.922459e-01
-----------------------------------------------------

# Print the estimated variance of the domain random effects
print(resultsEB$Resultsfit$RandomEffVar)

-----------------------------------------------------
[1] 1.258271e-02
-----------------------------------------------------

# Print the loglikelihood of the unit level model fit
print(resultsEB$Resultsfit$Loglike)

-----------------------------------------------------
[1] -2.453925e+04
-----------------------------------------------------

# Print raw residuals obtained from the unit level model fit
print(results$Resultsfit$RawResiduals)

-----------------------------------------------------
 [1] -0.311065481  0.407009344 -0.513399885 -0.303785484  0.880689517
 [6]  0.574593368 -0.487192985  0.461640055 -1.275929913  0.006050411
[11] -1.232357460 -0.326392234 -0.154397762  0.218288638  0.006755633
[16] -0.799496063  0.544477945 -1.137501244  0.006548324 -0.313700347
[21] -1.172306850  0.416721684 -1.074617448  0.283465858 -0.209258314
[26] -0.556619880 -0.157540917 -0.196951329 -0.356776776 -0.535499466
```

```
[31]  0.303750102 -0.903428203  1.061927641 -0.493474640  0.338891321
[36] -0.383703627 -0.599010472  0.355547287 -0.871554792 -0.031145534
[41]  0.345295921 -0.710517777  0.109911066 -1.413883307 -0.031580446
[46] -0.230772218  0.358927038  0.468438602 -0.582843686  0.049334166
-----------------------------------------------------
```

13.3.5 Example of R code for running function PBMSE.EB

This section includes the code required to execute the function PBMSE.EB() that gives bootstrap estimates of the mean squared errors of EB estimators of domain poverty incidences and gaps using the Example data set 1.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read the data file silc0106_SAMPLE.txt

data<-read.table("silc0106_SAMPLE.txt",header=TRUE)
attach(data)

# Select provinces with the largest CVs of direct estimators
# of poverty incidence (those in which we could improve more).

povinc15<-c(42,5,34,44,40,21,25,19,16,2)

# These selected domains are, in order: Soria, Avila, Palencia,
# Teruel, Segovia, Huelva, Lerida, Guadalajara, Cuenca and Albacete.

# Create matrix of auxiliary variables X for sample elements

n<-dim(data)[1]
X<-as.matrix(cbind(constant=rep(1,n),age2,age3,age4,age5,nat1,educ1,educ3,
sitemp1,sitemp2))

# We will apply the EB method when out-of-sample X values are available
# Read non-sample values of auxiliary variables from files for selected
# provinces.

Xrd1<-as.matrix(read.table("X_Soria.txt",header=TRUE))
Xrd2<-as.matrix(read.table("X_Avila.txt",header=TRUE))
Xrd3<-as.matrix(read.table("X_Palencia.txt",header=TRUE))
Xrd4<-as.matrix(read.table("X_Teruel.txt",header=TRUE))
Xrd5<-as.matrix(read.table("X_Segovia.txt",header=TRUE))
Xrd6<-as.matrix(read.table("X_Huelva.txt",header=TRUE))
Xrd7<-as.matrix(read.table("X_Lerida.txt",header=TRUE))
Xrd8<-as.matrix(read.table("X_Guadalajara.txt",header=TRUE))
Xrd9<-as.matrix(read.table("X_Cuenca.txt",header=TRUE))
Xrd10<-as.matrix(read.table("X_Albacete.txt",header=TRUE))

# Construct design matrix with non-sample values
```

```
Xrdtot<-list(Xrd1,Xrd2,Xrd3,Xrd4,Xrd5,Xrd6,Xrd7,Xrd8,Xrd9,Xrd10)

# Load file where function is located

source("PBMSE_EB.R")

# Compute parametric boostrap MSE estimators of the EB predictors

ResultsPBMSE<-PBMSE.EB(prov,povinc15,Xrdtot,norminc,X,weight,B=50,LB=50,
seed=2222)
```

13.3.6 Output of function PBMSE.EB

This section shows the R command required to print the output of R function PBMSE.EB() followed by the obtained R results.

```
print(ResultsPBMSE)

------------------------------------------------------
  Domain SampSize PBMSEpovinc PBMSEpovgap
1      42       41   13.930360   2.4449993
2       5      116    7.547312   1.2733205
3      34      143    4.789817   0.8214452
4      44      144    4.826459   0.8704941
5      40      115    6.298437   1.0425301
6      21      244    4.144693   0.7767003
7      25      260    4.126503   0.6720109
8      19      178    6.535062   1.1849335
9      16      183    3.396893   0.5019584
10      2      346    2.887742   0.5002812
------------------------------------------------------
```

13.3.7 Example of R code for running function FGTpovertyEBsample

This section includes the code required to execute the function FGTpovertyEBsample() that gives empirical EB estimates of domain poverty incidences and gaps using the Example data set 2.

```
# Set the Path or folder where data set and functions are.
setwd("Path")

# Read data set
data<-read.table("SimulDataPoverty.txt",header=TRUE)
attach(data)

# Create matrix of auxiliary variables

X<-as.matrix(data[,2:4])

# Load file, in which function is located
```

```
source("FGTpovertyEBsample.R")

# Call function FGTpovertyEBsample

results<-FGTpovertyEBsample(dom=Domain,seldomain=unique(Domain),
welfare=Welfare,Xs=X,weight=SampWeight,L=100,seed=1111)
```

13.3.8 Output of function FGTpovertyEBsample

This section lists the R commands required to print each of the output objects of R function FGTpovertyEBsample() followed by the obtained R results.

```
print(results$EstimatedPoverty)
```

```
-----------------------------------------------------
   Domain SampSz PovInc    PovGap
1       1     50 18.188 4.4489962
2       2     50 18.008 4.4822918
3       3     50  8.580 1.9828690
4       4     50 15.056 3.6341423
5       5     50 22.068 5.6574396
6       6     50 18.056 4.5715071
7       7     50 31.072 8.9835929
8       8     50 16.336 3.7289055
9       9     50 10.184 2.2203065
10     10     50 13.252 3.0668479
-----------------------------------------------------
```

Printing other output such as computation time and results from model fit is done exactly the same as with function FGTpovertyEB().

Chapter 14

Fast EB method for estimation of fuzzy poverty measures

14.1 Methodology

14.1.1 Fuzzy monetary and supplementary indicators

Let $U = \{1, \ldots, N\}$ be a finite population of size N, where E_i is the value of a welfare variable (e.g. equivalised income) for individual i. Let us consider the empirical distribution function of $\{E_1, \ldots, E_N\}$, defined as

$$F_E(x) = \frac{1}{N}\sum_{j=1}^{N} I\{E_j \leq x\}, \quad x \in \mathbf{R},$$

where $I\left\{E_j \leq x\right\} = 1$ if $E_j \leq x$ and 0 otherwise. Consider also the (empirical) Lorenz curve, given by

$$L_E(x) = \frac{\sum_{j=1}^{N} E_j I\left\{E_j \leq x\right\}}{\sum_{j=1}^{N} E_j}, \quad x \in \mathbf{R}.$$

Following the Integrated Fuzzy and Relative (IFR) approach of Betti et al. (2006), the Fuzzy Monetary Index (FMI) for individual i is defined as

$$\begin{aligned} FM_i &= \left\{\frac{N}{N-1}(1-F_E(E_i))\right\}^{\alpha-1}\{1-L_E(E_i)\} \\ &= \left\{\frac{1}{N-1}\sum_{j=1}^{N} I\left\{E_j > E_i\right\}\right\}^{\alpha-1}\left\{\frac{\sum_{j=1}^{N} E_j I\left\{E_j > E_i\right\}}{\sum_{j=1}^{N} E_j}\right\}, \quad i \in U. \end{aligned}$$

Here, $1 - F_E(E_i)$ is the proportion of individuals that are less poor than individual i. This gives a degree of poverty of individual i and it was proposed by Cheli e Lemmi (1995) as a poverty indicator. Observe that $N(1-F_E(E_i))/(N-1)$ is equal to 1 when individual i is the poorest. Moreover, $1 - L_E(E_i)$ is the share of the total welfare of all individuals that are less poor than this individual, indicator that was proposed by Betti and Verma

(1999). The average FMI for the population is given by

$$FM = \frac{1}{N}\sum_{i=1}^{N} FM_i \tag{14.1}$$

Observe that the FMI for individual i depends on the whole population of welfare values, $\{E_1,\dots,E_N\}$.

Consider now a score variable S_i for i-th individual defined using the IFR approach, instead of a welfare variable E_i. These scores S_i are obtained by applying a multidimensional approach that takes into account a variety of non-monetary indicators of deprivation. Then the Fuzzy Supplementary Index (FSI) for individual i is defined analogously to the FMI, but in terms of the scores $\{S_1,\dots,S_N\}$, as

$$\begin{aligned} FS_i &= \left\{\frac{N}{N-1}(1-F_S(S_i))\right\}^{\alpha-1}\{1-L_S(S_i)\} \\ &= \left\{\frac{1}{N-1}\sum_{j=1}^{N} I\{S_j > S_i\}\right\}^{\alpha-1}\left\{\frac{\sum_{j=1}^{N} S_j I\{S_j > S_i\}}{\sum_{j=1}^{N} S_j}\right\}, \quad i \in U. \end{aligned}$$

Here, $F_S(x)$ is the empirical distribution function and $L_S(x)$ the Lorenz curve of the score variables $\{S_1,\dots,S_N\}$. Similarly, $1-F_S(S_i)$ is the proportion of individuals who are less deprived than individual i and $1-L_S(S_i)$ is the share of the total lack of deprivation score assigned to all individuals less deprived than individual i. The average FSI for the population is given by

$$FS = \frac{1}{N}\sum_{i=1}^{N} FS_i \tag{14.2}$$

Now consider that the population U is partitioned into D domains or areas $U_1,\dots,U_D$ of sizes $N_1,\dots,N_D$. Let E_{dj} be the welfare for individual j within domain d. The average fuzzy monetary index for domain d is

$$FM_d = \frac{1}{N_d}\sum_{j=1}^{N_d} FM_{dj}, \quad d = 1,\dots,D, \tag{14.3}$$

where FM_{dj} is the FMI for j-th individual from d-th domain.

A random sample $s \subseteq U$ of size $n \le N$ is drawn from the population. Let s_d be the subsample from domain d, $d = 1,\dots,D$. A design-based estimator of the average FMI for domain d, FM_d, is

$$\widehat{FM}_d^{DB} = \frac{\sum_{j\in s_d} w_{dj}\widehat{FM}_{dj}^{DB}}{\sum_{j\in s_d} w_{dj}}, \quad d = 1,\dots,D, \tag{14.4}$$

where w_{dj} is the sampling weight for individual j within domain d and

$$\widehat{FM}_{dj}^{DB} = \left\{\frac{\sum_{\ell=1}^{D}\sum_{i\in s_d} w_{\ell i} I\{E_{\ell i} > E_{dj}\}}{\sum_{\ell=1}^{D}\sum_{i\in s_d} w_{\ell i}}\right\}^{\alpha-1}\left\{\frac{\sum_{\ell=1}^{D}\sum_{i\in s_d} w_{\ell i}E_{\ell i} I\{E_{\ell i} > E_{dj}\}}{\sum_{\ell=1}^{D}\sum_{i\in s_d} w_{\ell i}E_{\ell i}}\right\}. \tag{14.5}$$

Observe that $\widehat{FM}_{dj}^{DB}$ is not a direct estimator because it uses the sample data from the whole population and not only from domain d. The average FSI for domain d is given by

$$FS_d = \frac{1}{N_d}\sum_{j=1}^{N_d} FS_{dj}, \quad d = 1,\dots,D. \tag{14.6}$$

Finally, a design-based estimator of FS_d would be

$$\widehat{FS}_d^{DB} = \frac{\sum_{j \in s_d} w_{dj} \widehat{FS}_{dj}^{DB}}{\sum_{j \in s_d} w_{dj}}, \quad d = 1, \ldots, D. \tag{14.7}$$

where

$$\widehat{FS}_{dj}^{DB} = \left\{ \frac{\sum_{\ell=1}^{D} \sum_{i \in s_d} w_{\ell i} I\{S_{\ell i} > S_{dj}\}}{\sum_{\ell=1}^{D} \sum_{i \in s_d} w_{\ell i}} \right\}^{\alpha-1} \left\{ \frac{\sum_{\ell=1}^{D} \sum_{i \in s_d} w_{\ell i} S_{\ell i} I\{S_{\ell i} > S_{dj}\}}{\sum_{\ell=1}^{D} \sum_{i \in s_d} w_{\ell i} S_{\ell i}} \right\}. \tag{14.8}$$

In these poverty indicators, the parameter α can be fixed to the value such that the FM and FS indicators coincide with the head count ratio or poverty incidence computed for the official poverty line (60% of the median).

14.1.2 Fast EB method for estimation of fuzzy poverty measures

In order to apply the EB method of Molina and Rao (2010) to estimate the domain average FMI, FM_d, we need to express this indicator in terms of a population vector $\mathbf{y} = (\mathbf{y}_s', \mathbf{y}_r')'$, for which the conditional distribution of the non-sampled part $\mathbf{y}_r$ given the sample data $\mathbf{y}_s$ is known. The distribution of the welfare variable E_{dj} is seldom Normal. However, many times it is possible to find a transformation whose distribution is approximately Normal. Suppose that there exists a one-to-one transformation $Y_{dj} = T(E_{dj})$ of the welfare variable E_{dj}, which follows a Normal distribution. Concretely, we assume that Y_{dj} follows the nested error linear regression model of Battese, Harter and Fuller (1988), defined as

$$\begin{array}{l} Y_{dj} = \mathbf{x}_{dj}\beta + u_d + e_{dj}, \qquad j = 1, \ldots, N_d, \quad d = 1, \ldots D, \\ u_d \sim \text{ iid } N(0, \sigma_u^2), \quad e_{dj} \sim \text{ iid } N(0, \sigma_e^2) \end{array} \tag{14.9}$$

where $\mathbf{x}_{dj}$ is a row vector with the values of p explanatory variables, u_d is a random area-specific effect and e_{dj} are residual errors. Let $\mathbf{y}_d = (Y_{d1}, \ldots, Y_{dN_D})'$ be vector of responses for domain d and $\mathbf{y} = (\mathbf{y}_1', \ldots, \mathbf{y}_D')'$ be the full population vector. Then, observe that the individual FMIs can be expressed as

$$\begin{aligned} FM_{dj} &= \left\{ \frac{1}{N-1} \sum_{\ell=1}^{D} \sum_{i=1}^{N_\ell} I\left\{T^{-1}(Y_{\ell i}) > T^{-1}(Y_{dj})\right\} \right\}^{\alpha-1} \\ &\times \left\{ \frac{\sum_{\ell=1}^{D} \sum_{i=1}^{N_\ell} T^{-1}(Y_{\ell i}) I\left\{T^{-1}(Y_{\ell i}) > T^{-1}(Y_{dj})\right\}}{\sum_{\ell=1}^{D} \sum_{i=1}^{N_\ell} T^{-1}(Y_{\ell i})} \right\}, \quad j = 1, \ldots, N_d, \ d = 1, \ldots, D. \end{aligned}$$

This means that the average FMI for domain d is a non-linear function of the population vector $\mathbf{y}$, that is,

$$FM_d = \frac{1}{N_d} \sum_{j=1}^{N_d} FM_{dj} = h_d(\mathbf{y}), \quad d = 1, \ldots, D.$$

Let us separate the population vector of responses $\mathbf{y}$ in the sample and non-sample parts, that is, $\mathbf{y} = (\mathbf{y}_s', \mathbf{y}_r')'$, where $\mathbf{y}_s$ corresponds to the sample and $\mathbf{y}_r$ to the non-sample. Then the BP of FM_d is

$$\widehat{FM}_d^B = E_{\mathbf{y}_r}(FM_d | \mathbf{y}_s) = E_{\mathbf{y}_r}(h_d(\mathbf{y}) | \mathbf{y}_s). \tag{14.10}$$

This expectation can be empirically approximated by Monte Carlo simulation. For this, first fit the nested-error model (14.9) to the sample data $\mathbf{y}_s$, to obtain estimates $\hat{\beta}$, $\hat{\sigma}_u^2$ and $\hat{\sigma}_e^2$ of the model parameters β, σ_u^2 and σ_e^2 respectively. Obtain also the EB predictor $\hat{u}_d$ of u_d, given by $E(u_d|\mathbf{y}_s)$ with unknown parameters replaced by estimated values. Then, using those estimates, generate a large number L of vectors $\mathbf{y}_r$ from the estimated conditional distribution $\mathbf{y}_r|\mathbf{y}_s$. Let $\mathbf{y}_r^{(l)}$ be the vector generated in l-th generation. We attach this vector to the sample vector to obtain the full population vector $\mathbf{y}^{(l)} = (\mathbf{y}_s', (\mathbf{y}_r^{(l)})')'$. Using the elements of $\mathbf{y}^{(l)}$, we calculate the domain parameter of interest $FM_d^{(l)} = h_d(\mathbf{y}^{(l)})$, $d = 1, \ldots, D$. Then, a Monte Carlo approximation to the EB predictor of FM_d is given by

$$\widehat{FM}_d^{EB} \approx \frac{1}{L}\sum_{l=1}^{L} FM_d^{(l)}, \quad d = 1, \ldots, D. \tag{14.11}$$

Observe that for each population $l = 1, \ldots, L$, instead of generating a multivariate normal vector of size $N - n$, we just need to generate univariate values $Y_{dj}^{(\ell)}$ from

$$Y_{dj}^{(\ell)} = \mathbf{x}_{dj}\hat{\beta} + \hat{u}_d + v_d + \varepsilon_{di},\ v_d \sim N(0, \hat{\sigma}_u^2(1-\hat{\gamma}_d)),\ \varepsilon_{dj} \sim N(0, \hat{\sigma}_e^2),\ j \in U_d - s_d,\ d = 1, \ldots, D, \tag{14.12}$$

where $\gamma_d = \sigma_u^2(\sigma_u^2 + \sigma_e^2/n_d)^{-1}$ and n_d is the sample size in domain d. Still, for large populations and/or complex indicators, the EB method can be unfeasible. FMIs require sorting of all population elements, and this needs to be repeated for $l = 1, \ldots, L$. This is too time consuming for large N and large L. Here we propose a faster version of the EB estimator that is based on replacing the true value of the domain average FMI in population l, $FM_d^{(l)}$, by the design-based estimator given in (14.4). Since the design-based estimator is obtained from a sample drawn from l-th population, this avoids the task of generation of the full population of responses (we need to generate only the responses for the sample elements) and the sorting of all the population elements. Concretely, for each Monte Carlo replication l, we take a sample $s(l) \subseteq U$ using the same sampling scheme and the same sample size allocation as in the original sample s. We take the values of the auxiliary variables corresponding to the units in $s(l)$, that is, we take $\mathbf{x}_{dj}$, $j \in s_d(l)$, where $s_d(l)$ is the subsample from d-th domain. Then we generate the corresponding responses $Y_{dj}^{(\ell)}$, $j \in s_d(l)$, for $d = 1, \ldots, D$, as in (14.12). Let us denote the vector containing those values as $\mathbf{y}_{s(l)}$. With $\mathbf{y}_{s(l)}$, calculate the design-based estimator as in (14.4) and (14.5), that is, obtain

$$\widehat{FM}_d^{DB}(l) = \frac{\sum_{j \in s_d(l)} w_{dj}\widehat{FM}_{dj}^{DB}(l)}{\sum_{j \in s_d(l)} w_{dj}}, \quad d = 1, \ldots, D, \tag{14.13}$$

where

$$\widehat{FM}_{dj}^{DB}(l) = \left\{\frac{\sum_{\ell=1}^{D}\sum_{i \in s_d(l)} w_{\ell i} I\{T^{-1}(Y_{\ell i}^{(\ell)}) > T^{-1}(Y_{dj}^{(\ell)})\}}{\sum_{\ell=1}^{D}\sum_{i \in s_d(l)} w_{\ell i}}\right\}^{\alpha-1} \left\{\frac{\sum_{\ell=1}^{D}\sum_{i \in s_d(l)} w_{\ell i} T^{-1}(Y_{\ell i}^{(\ell)}) I\{T^{-1}(Y_{\ell i}^{(\ell)}) > T^{-1}(Y_{dj}^{(\ell)})\}}{\sum_{\ell=1}^{D}\sum_{i \in s_d(l)} w_{\ell i} T^{-1}(Y_{\ell i}^{(\ell)})}\right\}.$$

Finally, the fast EB estimator of FM_d is given by

$$\widehat{FM}_d^{FEB} = \frac{1}{L}\sum_{l=1}^{L}\widehat{FM}_{dj}^{DB}(l), \quad d = 1, \ldots, D.$$

14.2 The Software: description of R functions

This section describes the implemented R function that obtains FAST-EB estimators of domain fuzzy poverty measures, when the values of the auxiliary variables for out-of-sample data are available. Examples of how to implement this function are provided in Section 14.3.1 and full codes are included in Appendix 28.

14.2.1 FUZZYpovertyFAST.EB

R function `FUZZYpovertyFAST.EB()` fits the unit level model of Battese, Harter and Fuller (1988) to the log of the welfare variable and the clog-log transformation of the score variable and computes empirical FAST-EB estimates of domain fuzzy poverty measures (respectively FM fuzzy monetary indicator and FS fuzzy non-monetary indicator). This function is defined as:

```
FUZZYpovertyFAST.EB<function(dom,welfare,score,Xs,weight,
z=0.6*median(welfare),alpha=2,L=50)
```

The arguments of this function are:

`dom`: Domain indicator. It must contain numbers identifying the domain to which each sample unit belongs.

`welfare`: welfare variable used to quantify the level of richness of each individual or unit.

`score`: variable constructed by aggregating over a group of items used to quantify the level of non-monetary richness of each individual or unit.

`Xs`: *nxp* matrix containing the values of *p* auxiliary variables for the n sample units. The elements in the first column should be all equal to 1 if the model includes the intercept.

`weight`: Sampling weight of the unit.

`z`: Poverty line. Default value is 0.6 times the median of the welfare values for the sample units.

`alpha`: Parameter of the fuzzy monetary measures. Default value is 2.

`L`: Number of Monte Carlo replicates for the empirical approximation of the FASTEB estimator. Default value is L=50.

The function returns a list with the following objects:

`EstimatedPoverty`: Data.frame with number of rows equal to number of domains, containing in its columns domain indicator (Domain), population size (PopnSize), sample size (SampSize), FASTEB estimators of HCR (FASTEBPovinc), FM (FASTEBFM) and FS indicators (FASTEBFS).

`Resultsfit`: A list containing the following objects: summary of the unit level model fitting of welfare variable (Summary), vector with the estimated values of the fixed effects (FixedEffects), vector with the predicted random effects (RandomEffects), residual variance (ResVar), estimated variance of the random effects (RandomEffVar), log-likelihood (Loglike) and vector of raw residuals (RawResiduals).

`Resultsfits`: A list containing the following objects: summary of the unit level model fitting of non-monetary-variable (Summary), vector with the estimated values of the fixed effects (FixedEffects), vector with the predicted random effects (RandomEffects), residual variance (ResVar), estimated variance of the random effects (RandomEffVar), log-likelihood (Loglike) and vector of raw residuals (RawResiduals).

14.3 Examples of usage of R functions

This section describes how to execute the R function described in Section 14.2 with data sets to produce FASTEB estimates of fuzzy poverty measures.

14.3.1 Example data set

We used Tuscany data from 2004 EU-SILC survey. We consider as domains the 10 Tuscany Provinces. The welfare variable is the equivalised annual net income and the auxiliary variables are a constant, indicators of the 5 quinquennial groupings of variable age, indicator of having Italian nationality, indicators of 3 education levels of variable educational level and 3 categories of the variable employment. The first 10 rows from this data set look like this:

```
      Weight Domain Constant age2 age3 age4 age5 edu1 edu3 nat1 empl1 empl2
1   973.5813      1        1    0    1    0    0    0    0    1     1     0
2   973.5813      1        1    1    0    0    0    0    0    1     0     0
3  1237.5130      1        1    0    1    0    0    0    0    1     1     0
4   817.6650      1        1    0    1    0    0    0    1    1     1     0
5   817.6650      1        1    0    1    0    0    0    0    1     0     0
6   817.6650      1        1    0    0    0    0    0    0    1     0     0
7   817.6650      1        1    0    0    0    0    0    0    1     0     0
8   837.3646      1        1    0    0    1    0    0    0    1     1     0
9   837.3646      1        1    0    1    0    0    0    0    1     0     0
10  837.3646      1        1    1    0    0    0    0    0    1     0     1

    Welfare     Score
1  14348.00 0.9946286
2  14348.00 0.9946286
3  13980.00 0.9920049
4  26172.86 0.9904591
5  26172.86 0.9904591
6  26172.86 0.9904591
7  26172.86 0.9904591
8   9873.00 0.9725277
9   9873.00 0.9725277
10  9873.00 0.9725277
```

14.3.2 Example of R code for running function FUZZYpovertyFASTEB

This section includes the code required to execute the function FUZZYpovertyFAST.EB and obtain approximate FAST-EB estimates of domain fuzzy poverty measures using Tuscany data described in section 14.3.1.

```
# Read data set

data<-read.table("DataExample.txt",header=TRUE)
attach(data)

# Create matrix of auxiliary variables X

X<-as.matrix(data[,3:12])

# Load file where function is located

source("FUZZYpovertyFAST_EB.R")

# Execute function to compute FAST-EB predictors of poverty measures
```

```
results<FUZZYpovertyFAST.EB(welfare=Welfare,score=Score,Xs=X,
dom=Domain,weight=Weight,alpha=2,L=50)
```

14.3.3 Output of function FUZZYpovertyFASTEB

This section lists the R commands required to print each of the output objects of R function FUZZYpoverty-FAST.EB() followed by the obtained R results.

```
# Print function output
print(results$EstimatedPoverty)

-----------------------------------------------------
  Domain PopnSize SampSize FASTEBPovinc  FASTEBFM  FASTEBFS
1       1   251471      301    0.2327575 0.4674269 0.3265127
2       2   265293      315    0.1948571 0.4280292 0.3581722
3       3   267076      344    0.1855233 0.4143100 0.3796176
4       4  1119377     1403    0.1545830 0.3814060 0.3676503
5       5   290122      339    0.1631858 0.3928746 0.3729869
6       6   335777      399    0.1964411 0.4292763 0.3445257
7       7   304121      416    0.1351442 0.3577515 0.2922240
8       8   278495      338    0.1614201 0.3907552 0.3227387
9       9   149082      155    0.1581935 0.3887456 0.2030940
10     10   319320      416    0.1743750 0.4037955 0.3496956
-----------------------------------------------------

Linear mixed-effects model fit by REML
 Data: NULL
       AIC      BIC    logLik
  6352.651 6429.367 -3164.326


Random effects:
 Formula: ~1 | as.factor(dom)
        (Intercept)  Residual
StdDev:  0.05864517 0.4910157

Fixed effects: ys ~ -1 + Xs
               Value  Std.Error   DF   t-value p-value
XsConstant  9.434639 0.04440711 4407 212.45785  0.0000
Xsage2     -0.016709 0.03500726 4407  -0.47730  0.6332
Xsage3      0.005400 0.03072540 4407   0.17573  0.8605
Xsage4      0.158332 0.03048972 4407   5.19298  0.0000
Xsage5      0.055813 0.03194252 4407   1.74731  0.0807
Xsedu1     -0.211209 0.02208564 4407  -9.56320  0.0000
Xsedu3      0.245325 0.02510362 4407   9.77250  0.0000
Xsnat1      0.177515 0.03603744 4407   4.92584  0.0000
Xsempl1     0.107823 0.02081862 4407   5.17915  0.0000
Xsempl2    -0.115263 0.04317264 4407  -2.66981  0.0076
 Correlation:
        XsCnst Xsage2 Xsage3 Xsage4 Xsage5 Xsedu1 Xsedu3 Xsnat1 Xsmpl1
Xsage2  -0.299
Xsage3  -0.359  0.588
Xsage4  -0.321  0.547  0.736
Xsage5  -0.281  0.475  0.592  0.689
Xsedu1  -0.036 -0.060 -0.156 -0.339 -0.514
Xsedu3   0.024 -0.012 -0.129 -0.113 -0.125  0.180
Xsnat1  -0.755 -0.023  0.001 -0.054 -0.083  0.044 -0.031
Xsempl1  0.008 -0.221 -0.537 -0.329 -0.096  0.160 -0.073 -0.010
Xsempl2 -0.025 -0.147 -0.260 -0.127 -0.029  0.041 -0.023  0.032  0.327

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-4.66823861 -0.54233384  0.01043265  0.57353177  5.67467198

Number of Observations: 4426
Number of Groups: 10

-----------------------------------------------------

print(results$Resultsfit$FixedEffects)

-----------------------------------------------------
 XsConstant       Xsage2       Xsage3       Xsage4       Xsage5
 9.434638959 -0.016709012  0.005399513  0.158332415  0.055813419
 Xsedu1       Xsedu3       Xsnat1      Xsempl1      Xsempl2
 -0.211209341  0.245325220  0.177514758  0.107822671 -0.115262728
-----------------------------------------------------

print(results$Resultsfit$RandomEffects[,1])

-----------------------------------------------------
[1] -0.115224907 -0.038681662 -0.006406057  0.035579916  0.010238082
[6] -0.042017887  0.071844772  0.030157514  0.037516377  0.016993852
------------------------------------------------------

# Print the residual variance of the unit level model fit

print(results$Resultsfit$ResVar)

-----------------------------------------------------
[1] 0.4910157
-----------------------------------------------------


# Print the estimated variance of the domain random effects

print(results$Resultsfit$RandomEffVar)
```

```
-----------------------------------------------------
[1] 0.003439256
-----------------------------------------------------

# Print the loglikelihood of the unit level model fit

print(results$Resultsfit$Loglike)

-----------------------------------------------------
[1] -3164.326
-----------------------------------------------------

print(results$Resultsfit$RawResiduals[1:50])

-----------------------------------------------------
 [1] -0.154010062 -0.024078867 -0.179992885  0.201777045  0.554924936
 [6]  0.560324449  0.560324449 -0.680749765 -0.419994192 -0.282622939
[11] -0.515299162  0.499986124  0.060135470 -0.042383526  0.512474970
[16]  0.655281194  0.229954644 -1.064221208 -1.332839314 -0.206627212
[21]  0.054128361 -0.182915956 -0.005401198 -0.377827270  0.537554682
[26]  0.537554682  0.233953023  0.371455572 -0.449847219 -0.342024548
[31] -0.319916023 -0.319916023 -0.049973831 -0.049973831  0.401583041
[36] -0.228691851 -0.120869181 -0.344085876  0.568080480  0.390565722
[41]  0.704294059 -0.152756318  0.619632092 -0.604167681 -0.451234779
[46] -0.321303584 -0.473474601  0.060748162 -0.242853497 -0.242853497
------------------------------------------------------

print(results$Resultsfits$Summary)

-----------------------------------------------------
Linear mixed-effects model fit by REML
 Data: NULL
       AIC      BIC    logLik
  3908.576 3985.291 -1942.288

Random effects:
 Formula: ~1 | as.factor(dom)
        (Intercept)  Residual
StdDev:  0.07663074 0.3719493

Fixed effects: yss ~ -1 + Xs
                Value  Std.Error   DF   t-value p-value
XsConstant  0.9264946 0.03902919 4407 23.738502  0.0000
Xsage2     -0.1198104 0.02652597 4407 -4.516720  0.0000
Xsage3     -0.0566792 0.02327718 4407 -2.434967  0.0149
Xsage4     -0.0500512 0.02310541 4407 -2.166210  0.0303
Xsage5      0.0458473 0.02420242 4407  1.894329  0.0582
Xsedu1     -0.1353946 0.01673728 4407 -8.089404  0.0000
Xsedu3      0.0859442 0.01901830 4407  4.519026  0.0000
```

```
Xsnat1      0.1289693 0.02730920 4407  4.722557  0.0000
Xsempl1     0.0390022 0.01577347 4407  2.472644  0.0134
Xsempl2    -0.1712626 0.03270924 4407 -5.235908  0.0000
 Correlation:
        XsCnst Xsage2 Xsage3 Xsage4 Xsage5 Xsedu1 Xsedu3 Xsnat1 Xsmpl1
Xsage2  -0.258
Xsage3  -0.310  0.588
Xsage4  -0.277  0.547  0.736
Xsage5  -0.243  0.475  0.592  0.689
Xsedu1  -0.031 -0.060 -0.157 -0.339 -0.514
Xsedu3   0.021 -0.013 -0.129 -0.113 -0.125  0.180
Xsnat1  -0.651 -0.022  0.001 -0.054 -0.083  0.045 -0.031
Xsempl1  0.007 -0.221 -0.537 -0.329 -0.096  0.161 -0.073 -0.010
Xsempl2 -0.022 -0.147 -0.260 -0.126 -0.029  0.041 -0.023  0.032  0.327

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-4.10191706 -0.66550366 -0.01227919  0.68237675  3.61160133
Number of Observations: 4426
Number of Groups: 10

-----------------------------------------------------

print(results$Resultsfits$FixedEffects)

-----------------------------------------------------
 XsConstant      Xsage2      Xsage3      Xsage4      Xsage5      Xsedu1
 0.92649460 -0.11981038 -0.05667919 -0.05005117  0.04584735 -0.13539464
 Xsedu3      Xsnat1     Xsempl1     Xsempl2
 0.08594418  0.12896928  0.03900219 -0.17126256
-----------------------------------------------------

print(results$Resultsfits$RandomEffects[,1])

-----------------------------------------------------
[1]  0.013054328 -0.045014363 -0.069028120 -0.056634929 -0.057649764
[6] -0.024310964  0.047476612  0.015368337  0.181148065 -0.004409203
------------------------------------------------------

# Print the residual variance of the unit level model fit

print(results$Resultsfits$ResVar)

-----------------------------------------------------
[1] 0.3719493
-----------------------------------------------------

# Print the estimated variance of the domain random effects
```

```
print(results$Resultsfits$RandomEffVar)

-----------------------------------------------------
[1] 0.00587227
-----------------------------------------------------

# Print the loglikelihood of the unit level model fit

print(results$Resultsfits$Loglike)

-----------------------------------------------------
[1] -3164.326
-----------------------------------------------------

print(results$Resultsfits$RawResiduals[1:50])

-----------------------------------------------------
[1]   0.61598625  0.71811963  0.53683851  0.41360278  0.53854915  0.48186997
[7]   0.48186997  0.23501143  0.28064164  0.51503539  0.07009848  0.20579661
[13]  0.17557345  0.27147197 -0.08955661  0.09228609  0.04329534  0.09190668
[19]  0.02003638 -0.02007078  0.02555943 -0.08356510  0.04540418 -0.06698275
[25]  0.06301459  0.06301459 -0.15169620 -0.10475421 -0.17956769 -0.14056550
[31] -0.07743431 -0.07743431 -0.17687887 -0.17687887 -0.11172525 -0.29517794
[37] -0.25617574 -0.27898873 -0.18219118 -0.31116045 -0.27166433 -0.34986104
[43] -0.37663483 -0.50258912 -0.49596110 -0.39382772 -0.37164053 -0.38594891
[49] -0.60065971 -0.60065971
------------------------------------------------------
```

Chapter 15

Appendix 1: R code for the Fay-Herriot model

15.1 R code of fitFH

The R code of the function `fitFH` is listed bellow.

```
#################################################################
###
### This function fits a Fay-Herriot model.
### Fitting method can be chosen between REML and FH methods.
###
### Work for European project SAMPLE
###
### Author: Isabel Molina Peralta
### File name: Fitting_FHModel.R
### Updated: March 15th, 2010
###
#################################################################

fitFH<-function(X,y,Dvec,method="REML",MAXITER=500) {

  m<-length(y) # Sample size or number of areas
  p<-dim(X)[2] # Num. of X columns of num. of auxiliary variables
  Xt<-t(X)

  # Fisher-scoring algorithm for REML estimator of variance A starts

  if (method=="REML") {

    # Initial value of variance A is fixed to the median of Dvec
    Aest.REML<-0
    Aest.REML[1]<-median(Dvec)

    k<-0
    diff<-1
    while ((diff>0.0001)&(k<MAXITER))
    {
      k<-k+1
      Vi<-1/(Aest.REML[k]+Dvec)
      XtVi<-t(Vi*X)
      Q<-solve(XtVi%*%X)
      P<-diag(Vi)-t(XtVi)%*%Q%*%XtVi
      Py<-P%*%y
      # Score function obtained from restricted log-likelihood
      s<-(-0.5)*sum(diag(P))+0.5*(t(Py)%*%Py)
      # Fisher information obtained from restricted log-likelihood
      F<-0.5*sum(diag(P%*%P))
      # Updating equation
      Aest.REML[k+1]<-Aest.REML[k]+s/F
      # Relative difference of estimators in 2 iterations
      # for stopping condition
      diff<-abs((Aest.REML[k+1]-Aest.REML[k])/Aest.REML[k])
    } # End of while

    # Final estimator of variance A
    A.REML<-max(Aest.REML[k+1],0)
    print(Aest.REML)

    # Indicator of convergence

    if(k<MAXITER) {conv<-TRUE} else {conv<-FALSE}

    # Computation of the coefficients'estimator beta

    Vi<-1/(A.REML+Dvec)
    XtVi<-t(Vi*X)
    Q<-solve(XtVi%*%X)
    beta.REML<-Q%*%XtVi%*%y

    # Significance of the regression coefficients

    varA<-1/F

    std.errorbeta<-sqrt(diag(Q))
    tvalue<-beta.REML/std.errorbeta
    pvalue<-2*pnorm(abs(tvalue),lower.tail=FALSE)

   # Goodness of fit measures: loglikelihood, AIC, BIC

   Xbeta.REML<-X%*%beta.REML
   resid<-y-Xbeta.REML

   loglike<-(-0.5)*(sum(log(2*pi*(A.REML+Dvec))
           +(resid^2)/(A.REML+Dvec)))
   AIC<-(-2)*loglike+2*(p+1)
   BIC<-(-2)*loglike+(p+1)*log(m)

   goodness<-c(loglike=loglike,AIC=AIC,BIC=BIC)

   # Computation of the empirical best (EB) predictor

   thetaEB.REML<-Xbeta.REML+A.REML*Vi*resid
   coef<-data.frame(beta.REML,std.errorbeta,tvalue,pvalue)

   return(list(convergence=conv,modelcoefficients=coef,
   variance=A.REML,goodnessoffit=goodness,EBpredictor=thetaEB.REML))

   # Fisher-scoring algorithm for REML estimator of variance A starts

  } else if (method=="FH") {

    # Initial value of variance A is fixed to the median of Dvec
    Aest.FH<-NULL
    Aest.FH[1]<-median(Dvec)

    k<-0
    diff<-1
    while ((diff>0.0001)&(k<MAXITER)){
      k<-k+1
      Vi<-1/(Aest.FH[k]+Dvec)
      XtVi<-t(Vi*X)
      Q<-solve(XtVi%*%X)
      betaaux<-Q%*%XtVi%*%y
      resaux<-y-X%*%betaaux
      # Left-hand side of equation for FH estimator
      s<-sum((resaux^2)*Vi)-(m-p)
      # Expectation of negative derivative of s
      F<-sum(Vi)
      # Updating equation
      Aest.FH[k+1]<-Aest.FH[k]+s/F
      # Relative difference of estimators in 2 iterations
      # for stopping condition
      diff<-abs((Aest.FH[k+1]-Aest.FH[k])/Aest.FH[k])

    } # End of while

    A.FH<-max(Aest.FH[k+1],0)
    print(Aest.FH)

    # Indicator of convergence

    if(k<MAXITER) {conv<-TRUE} else {conv<-FALSE}
```

```
    # Computation of the coefficients'estimator beta

    Vi<-1/(A.FH+Dvec)
    XtVi<-t(Vi*X)
    Q<-solve(XtVi%*%X)
    beta.FH<-Q%*%XtVi%*%y

    # Significance of the regression coefficients

    varA<-1/F
    varbeta<-diag(Q)
    std.errorbeta<-sqrt(varbeta)
    zvalue<-beta.FH/std.errorbeta
    pvalue<-2*pnorm(abs(zvalue),lower.tail=FALSE)

    # Goodness of fit measures: loglikelihood, AIC, BIC

    Xbeta.FH<-X%*%beta.FH
    resid<-y-Xbeta.FH

    loglike<-(-0.5)*(sum(log(2*pi*(A.FH+Dvec))+(resid^2)/(A.FH+Dvec)))
    AIC<-(-2)*loglike+2*(p+1)
    BIC<-(-2)*loglike+(p+1)*log(m)

    goodness<-c(loglike=loglike,AIC=AIC,BIC=BIC)

    # Computation of the empirical best (EB) predictor

    thetaEB.FH<-Xbeta.FH+A.FH*Vi*resid
    coef<-data.frame(beta.FH,std.errorbeta,zvalue,pvalue)

    return(list(convergence=conv,modelcoefficients=coef,variance=A.FH,
        goodnessoffit=goodness,EBpredictor=thetaEB.FH))

    # Error printing when method is different from REML of FH.
  }  else { print("Error: Unknown method") }

}
```

15.2 R code of MSE.FHmodel

The R code of the function `MSE.FHmodel` is listed bellow.

```
##################################################################
###
### This function gives the MSE estimator of the EB estimator under
### a FH model. The EB estimator is obtained either by REML or by
### FH fitting methods.
```

```
###
### Work for European project SAMPLE
###
### Author: Isabel Molina Peralta
### File name: MSE_FHModel.R
### Updated: March 15th, 2010
###
####################################################################

MSE.FHmodel<-function(X,Dvec,A,method="REML"){

  m<-dim(X)[1] # Sample size or number of areas
  p<-dim(X)[2] # Num. of X columns of num. of auxiliary variables

  # Initialize vectors containing the values of g1-g3 and mse
  # for each area
  g1d<-rep(0,m)
  g2d<-rep(0,m)
  g3d<-rep(0,m)
  mse2d<-rep(0,m)

  # Elements of the inverse covariance matrix in a vector
  Vi<-1/(A+Dvec)
  # Auxiliary calculations
  Bd<-Dvec/(A+Dvec)
  SumAD2<-sum(Vi^2)
  XtVi<-t(Vi*X)
  Q<-solve(XtVi%*%X)

  # Calculation of g1-g3 and final MSE when fitting method is REML

  if (method=="REML"){

    # Asymptotic variance of REML estimator of variance A
    VarA<-2/SumAD2

    for (d in 1:m){
      g1d[d]<-Dvec[d]*(1-Bd[d])
      xd<-matrix(X[d,],nr=1,nc=p)
      g2d[d]<-(Bd[d]^2)*xd%*%Q%*%t(xd)
      g3d[d]<-(Bd[d]^2)*VarA/(A+Dvec[d])
      mse2d[d]<-g1d[d]+g2d[d]+2*g3d[d]
    }

  return(mse=mse2d)

  # Calculation of g1-g3 and final MSE when fitting method is FH

  } else if (method=="FH") {
    SumAD<-sum(Vi)
    # Asymptotic variance of FH estimator of variance A
    VarA<-2*m/(SumAD^2)

    # Asymptotic bias of FH estimator of A
    b<-2*(m*SumAD2-SumAD^2)/(SumAD^3)

    for (d in 1:m){
      g1d[d]<-Dvec[d]*(1-Bd[d])
      xd<-matrix(X[d,],nr=1,nc=p)
      g2d[d]<-(Bd[d]^2)*xd%*%Q%*%t(xd)
      g3d[d]<-(Bd[d]^2)*VarA/(A+Dvec[d])
      mse2d[d]<-g1d[d]+g2d[d]+2*g3d[d]-b*(Bd[d]^2)
    }

  return(mse=mse2d)

  # Error printing when fitting method is different from REML of FH.
  }  else { print("Error: Unknown method") }

}
```

Chapter 16

Appendix 2: R code for the area-level spatial model

16.1 R code of fitSpatialFH

The R code of the function `fitSpatialFH` is listed bellow.

```
####################################################################
###
### This function fits a spatial Fay-Herriot model, in which random
### effects follow a Simultaneously Autorregressive (SAR) process
### Fitting method can be chosen between REML and ML methods.
###
### Work for European project SAMPLE
###
### Authors: Nicola Salvati and Isabel Molina Peralta
### File name: Fitting_SpatialFHModel.R
### Updated: March 15th, 2010
###
#####################################################################


fitSpatialFH<-function(X,y,Dvec,W,method="REML",MAXITER=500) {

  m<-length(y)  # Sample size or number of areas
  p<-dim(X)[2]  # Num. of X columns of num. of auxiliary variables
  Xt<-t(X)
  yt<-t(y)
  Wt<-t(W)
  I<-diag(1,m)

  # Initialize vectors containing estimators of variance and
  # spatial correlation
  par.stim<-matrix(0,2,1)
  stime.fin<-matrix(0,2,1)
```

```
# Initialize scores vector and Fisher information matrix
s<-matrix(0,2,1)
Idev<-matrix(0,2,2)

# Initial value of variance set to the mean of sampling variances Dvec
# Initial value of spatial correlation set to 0.5
sigma2.u.stim.S<-0
rho.stim.S<-0

sigma2.u.stim.S[1]<-median(Dvec)
rho.stim.S[1]<-0.5

# Fisher-scoring algorithm for REML estimators start

if (method=="REML"){

  k<-0
  diff.S<-1

  while ((diff.S>0.0001)&(k<MAXITER)){

    k<-k+1
    # Derivative of covariance matrix V with respect to variance
    derSigma<-solve((I-rho.stim.S[k]*Wt)%*%(I-rho.stim.S[k]*W))
    # Derivative of covariance matrix V with respect to
    # spatial autocorrelation
    derRho<-2*rho.stim.S[k]*Wt%*%W-W-Wt
    derVRho<-(-1)*sigma2.u.stim.S[k]*(derSigma%*%derRho%*%derSigma)
    # Covariance matrix and inverse covariance matrix
    V<-sigma2.u.stim.S[k]*derSigma+I*Dvec
    Vi<-solve(V)
    # Matrix P and coefficients'estimator beta
    XtVi<-Xt%*%Vi
    Q<-solve(XtVi%*%X)
    P<-Vi-t(XtVi)%*%Q%*%XtVi
    b.s<-Q%*%XtVi%*%y
    # Terms involved in scores vector and Fisher information matrix
    PD<-P%*%derSigma
    PR<-P%*%derVRho
    Pdir<-P%*%y
    # Scores vector
    s[1,1]<-(-0.5)*sum(diag(PD))+(0.5)*(yt%*%PD%*%Pdir)
    s[2,1]<-(-0.5)*sum(diag(PR))+(0.5)*(yt%*%PR%*%Pdir)
    # Fisher information matrix
    Idev[1,1]<-(0.5)*sum(diag(PD%*%PD))
    Idev[1,2]<-(0.5)*sum(diag(PD%*%PR))
    Idev[2,1]<-Idev[1,2]
    Idev[2,2]<-(0.5)*sum(diag(PR%*%PR))
    # Updating equation
    par.stim[1,1]<-sigma2.u.stim.S[k]
    par.stim[2,1]<-rho.stim.S[k]

    stime.fin<-par.stim+solve(Idev)%*%s
    print(stime.fin)

    # Restrict spatial correlation to (-0.999,0.999) and variance
    # to (0.0001,infty)
    if ((stime.fin[2,1]<=0.999)&(stime.fin[2,1]>=-0.999)&
      (stime.fin[1,1]>0.0001)){
      sigma2.u.stim.S[k+1]<-stime.fin[1,1]
      rho.stim.S[k+1]<-stime.fin[2,1]
      diff.S<-max(abs(stime.fin-par.stim)/par.stim)
    }else{ diff.S<-0.00001 }

  } # End of while

# Fisher-scoring algorithm for REML estimators start

}else if (method=="ML"){

  k<-0
  diff.S<-1

  while ((diff.S>0.0001)&(k<MAXITER)){

    k<-k+1
    # Derivative of covariance matrix V with respect to variance
    derSigma<-solve((I-rho.stim.S[k]*Wt)%*%(I-rho.stim.S[k]*W))
    # Derivative of covariance matrix V with respect to
    # spatial autocorrelation
    derRho<-2*rho.stim.S[k]*Wt%*%W-W-Wt
    derVRho<-(-1)*sigma2.u.stim.S[k]*(derSigma%*%derRho%*%derSigma)
    # Covariance matrix and inverse covariance matrix
    V<-sigma2.u.stim.S[k]*derSigma+I*Dvec
    Vi<-solve(V)
    # Coefficients'estimator beta and matrix P
    XtVi<-Xt%*%Vi
    Q<-solve(XtVi%*%X)
    P<-Vi-t(XtVi)%*%Q%*%XtVi
    b.s<-Q%*%XtVi%*%y
    # Terms involved in scores vector and Fisher information matrix
    PD<-P%*%derSigma
    PR<-P%*%derVRho
    Pdir<-P%*%y
    ViD<-Vi%*%derSigma
    ViR<-Vi%*%derVRho
    # Scores vector
    s[1,1]<-(-0.5)*sum(diag(ViD))+(0.5)*(yt%*%PD%*%Pdir)
    s[2,1]<-(-0.5)*sum(diag(ViR))+(0.5)*(yt%*%PR%*%Pdir)
    # Fisher information matrix
    Idev[1,1]<-(0.5)*sum(diag(ViD%*%ViD))
    Idev[1,2]<-(0.5)*sum(diag(ViD%*%ViR))
    Idev[2,1]<-Idev[1,2]
    Idev[2,2]<-(0.5)*sum(diag(ViR%*%ViR))

    # Updating equation
    par.stim[1,1]<-sigma2.u.stim.S[k]
    par.stim[2,1]<-rho.stim.S[k]

    stime.fin<-par.stim+solve(Idev)%*%s

    # Restrict spatial correlation to (-0.999,0.999) and variance
    # to (0.0001,infty)
    if ((stime.fin[2,1]<=0.999)&(stime.fin[2,1]>=-0.999)&
      (stime.fin[1,1]>0.0001)){
      sigma2.u.stim.S[k+1]<-stime.fin[1,1]
      rho.stim.S[k+1]<-stime.fin[2,1]
      diff.S<-max(abs(stime.fin-par.stim)/par.stim)
    }else{ diff.S<-0.00001 }

  } # End of while
# Error message if method is different from REML or ML
} else { print("Error: Unknown method") }

# Final values of estimators
rho<-rho.stim.S[k+1]
sigma2u<-max(sigma2.u.stim.S[k+1],0)
#print(c(sigma2u,rho))

# Indicator of convergence

if(k<MAXITER) {conv<-TRUE} else {conv<-FALSE}

# Computation of the coefficients'estimator (Bstim)

A<-solve((I-rho*Wt)%*%(I-rho*W))
G<-sigma2u*A
V<-G+I*Dvec
Vi<-solve(V)
XtVi<-Xt%*%Vi
Q<-solve(XtVi%*%X)
Bstim<-Q%*%XtVi%*%y

# Significance of the regression coefficients
```

```
  std.errorbeta<-sqrt(diag(Q))
  tvalue<-Bstim/std.errorbeta
  pvalue<-2*pnorm(abs(tvalue),lower.tail=FALSE)

  coef<-data.frame(beta=Bstim,std.errorbeta,tvalue,pvalue)

  # Goodness of fit measures: loglikelihood, AIC, BIC

  Xbeta<-X%*%Bstim
  resid<-y-Xbeta
  loglike<-(-0.5)*(m*log(2*pi)+determinant(V,logarithm=T)$modulus+
    t(resid)%*%Vi%*%resid)
  AIC<-(-2)*loglike+2*(p+2)
  BIC<-(-2)*loglike+(p+2)*log(m)

  goodness<-c(loglike=loglike,AIC=AIC,BIC=BIC)

  # Computation of the Spatial EBLUP

  res<-y-X%*%Bstim
  thetaSpat<-X%*%Bstim+G%*%Vi%*%res

  return(list(convergence=conv,modelcoefficients=coef,variance=sigma2u,
    spatialcorr=rho,goodnessoffit=goodness,EBpredictor=thetaSpat))

}
```

16.2 R code of MSE.SpatialFH

The R code of the function `MSE.SpatialFH` is listed bellow.

```
###################################################################
###
### This function gives the analytical MSE estimator of the EB
### estimator under a Spatial FH model, in which random effects
### follow a Simultaneously Autorregressive (SAR) process.
### The EB estimator is obtained by REML fitting method.
###
### Work for European project SAMPLE
###
### Authors: Nicola Salvati and Isabel Molina Peralta
### File name: MSE_SpatialFHModel.R
### Updated: March 15th, 2010
###
####################################################################

MSE.SpatialFHmodel<-function(X,Dvec,A,rho,W,method="REML"){

  m<-dim(X)[1]
```

```
p<-dim(X)[2]

g1d<-rep(0,m)
g2d<-rep(0,m)
g3d<-rep(0,m)
g5d<-rep(0,m)
mse2d.aux<-rep(0,m)
mse2d<-rep(0,m)

I<-diag(1,m)
Xt<-t(X)
Wt<-t(W)
Ci<-solve((I-rho*Wt)%*%(I-rho*W))
G<-A*Ci
V<-G+I*Dvec
Vi<-solve(V)
XtVi<-Xt%*%Vi
Q<-solve(XtVi%*%X)

Ga<-G-G%*%Vi%*%G

# g1 contains the diagonal elements of Ga
for (i in 1:m) {g1d[i]<-Ga[i,i]}

Gb<-G%*%Vi%*%X
Xa<-matrix(0,1,p)

for (i in 1:m) {
  Xa[1,]<-X[i,]-Gb[i,]
  g2d[i]<-Xa%*%Q%*%t(Xa)
}

derRho<-2*rho*Wt%*%W-W-Wt
Amat<-(-1)*A*(Ci%*%derRho%*%Ci)
P<-Vi-t(XtVi)%*%Q%*%XtVi
PCi<-P%*%Ci
PAmat<-P%*%Amat

Idev<-matrix(0,2,2)
Idev[1,1]<-(0.5)*sum(diag((PCi%*%PCi)))
Idev[1,2]<-(0.5)*sum(diag((PCi%*%PAmat)))
Idev[2,1]<-Idev[1,2]
Idev[2,2]<-(0.5)*sum(diag((PAmat%*%PAmat)))
Idevi<-solve(Idev)

ViCi<-Vi%*%Ci
ViAmat<-Vi%*%Amat

l1<-ViCi-A*ViCi%*%ViCi
```

```
  l1t<-t(l1)
  l2<-ViAmat-A*ViAmat%*%ViCi
  l2t<-t(l2)
  L<-matrix(0,2,m)
  for (i in 1:m)
  {
    L[1,]<-l1t[i,]
    L[2,]<-l2t[i,]
    g3d[i]<-sum(diag(L%*%V%*%t(L)%*%Idevi))
  }

  mse2d.aux<-g1d+g2d+2*g3d

  # Bias correction of Singh et al

  psi<-diag(Dvec,m)
  D12aux<-(-1)*(Ci%*%derRho%*%Ci)
  D22aux<-2*A*Ci%*%derRho%*%Ci%*%derRho%*%Ci-2*A*Ci%*%Wt%*%W%*%Ci
  D<-psi%*%Vi%*%D12aux%*%Vi%*%psi*Idevi[1,2]
    +psi%*%Vi%*%D12aux%*%Vi%*%psi*Idevi[2,1]+
    psi%*%Vi%*%D22aux%*%Vi%*%psi*Idevi[2,2]
  for (i in 1:m) {g5d[i]<-(0.5)*D[i,i]}

  # Computation of analytical estimated of Singh et al

  mse2d<-mse2d.aux-g5d

  return(mse=mse2d)
}
```

16.3 R code of PBMSE.SpatialFH

The R code of the function `PBMSE.SpatialFH` is listed bellow.

```
##################################################################
###
### This function gives a parametric bootstrap MSE estimator of the
### EB estimator under a Spatial FH model, in which random effects
### follow a Simultaneously Autorregressive (SAR) process. The EB
### estimator is obtained either by REML or by FH fitting methods.
###
### Work for European project SAMPLE
###
### Authors: Nicola Salvati and Isabel Molina Peralta
### File name: PBMSE_SpatialFHModel.R
### Updated: March 15th, 2010
###
###################################################################
```

```
source("Fitting_SpatialFHModel.R")

PBMSE.SpatialFHmodel<-(X,Dvec,beta,A,rho,W,n.boot,method="REML",
seed=Sys.time()){

  m<-dim(X)[1]  # Sample size or number of areas
  p<-dim(X)[2]  # Num. of X columns of num. of auxiliary variables

  # Initial estimators of model coefficients, variance and spatial
  # correlation actong as true values in the bootstrap procedure.

  Bstim.boot<-beta
  rho.boot<-rho
  sigma2.boot<-A

  I<-diag(1,m)
  Xt<-t(X)
  Wt<-t(W)

  # Analytical estimators of g1 and g2, used for
  # the   bias-corrected PB MSE estimator

  g1sp<-rep(0,m)
  g2sp<-rep(0,m)

  Amat.sblup<-solve((I-rho.boot*Wt)%*%(I-rho.boot*W))
  G.sblup<-sigma2.boot*Amat.sblup
  V.sblup<-G.sblup+I*Dvec
  V.sblupi<-solve(V.sblup)
  XtV.sblupi<-Xt%*%V.sblupi
  Q.sblup<-solve(XtV.sblupi%*%X)

  # Calculate g1

  Ga.sblup<-G.sblup-G.sblup%*%V.sblupi%*%G.sblup
  for (i in 1:m) {g1sp[i]<-Ga.sblup[i,i]}

  # Calculate g2

  Gb.sblup<-G.sblup%*%t(XtV.sblupi)
  Xa.sblup<-matrix(0,1,p)

  for (i in 1:m){
    Xa.sblup[1,]<-X[i,]-Gb.sblup[i,]
    g2sp[i]<-Xa.sblup%*%Q.sblup%*%t(Xa.sblup)
  }

  # Initialize vectors adding g1, g2, g3 and naive PB MSE estimators
  summse.pb<-rep(0,m)
```

```
sumg1sp.pb<-rep(0,m)
sumg2sp.pb<-rep(0,m)
sumg3sp.pb<-rep(0,m)
g1sp.aux<-rep(0,m)
g2sp.aux<-rep(0,m)

# Bootstrap cycle starts

for (boot in 1:n.boot) {

  cat(date(),"Bootstrap iteration",boot,"\n",fill=T)

  # Generate a bootstrap sample
  u.boot<-rnorm(m,0,sqrt(sigma2.boot))
  v.boot<-solve(I-rho.boot*W)%*%u.boot
  theta.boot<-X%*%Bstim.boot+v.boot
  e.boot<-rnorm(m,0,sqrt(Dvec))
  direct.boot<-theta.boot+e.boot

  # Fit of the model to bootstrap data

  resultsSp<-fitSpatialFH(X,direct.boot,Dvec,W,method="REML")

  # While the estimators are not satisfactory
  # we generate a new sample
  conv<-resultsSp$convergence
  v<-resultsSp$variance
  r<-resultsSp$spatialcorr
  print(conv)
  print(c(v,r))

  if (conv==FALSE) {v<-0}
  if (is.na(v)==TRUE) {v<-0}
  if (is.na(r)==TRUE) {r<-1}

  while ((v<0.0001)|(r<(-0.999))|(r>0.999)){
    print("New sample")
    u.boot<-rnorm(m,0,sqrt(sigma2.boot))
    v.boot<-solve(I-rho.boot*W)%*%u.boot
    theta.boot<-X%*%Bstim.boot+v.boot
    e.boot<-rnorm(m,0,sqrt(Dvec))
    direct.boot<-theta.boot+e.boot
    resultsSp<-fitSpatialFH(X,direct.boot,Dvec,W,method="REML")
    conv<-resultsSp$convergence
    v<-resultsSp$variance
    r<-resultsSp$spatialcorr
    if (conv==FALSE) {v<-0}
    if (is.na(v)==TRUE) {v<-0}
    if (is.na(r)==TRUE) {r<-1}
  }

  # Fit of the model to bootstrap data

  sigma2.simula.ML<-resultsSp$variance
  rho.simula.ML<-resultsSp$spatialcorr
  beta.ML<-resultsSp$modelcoefficients$beta

  # Calculation of the bootstrap Spatial EBLUP

  Amat<-solve((I-rho.simula.ML*Wt)%*%(I-rho.simula.ML*W))
  G<-sigma2.simula.ML*Amat
  V<-G+I*Dvec
  Vi<-solve(V)
  Xbeta<-X%*%beta.ML
  thetaEBLUPSpat.boot<-Xbeta+G%*%Vi%*%(direct.boot-Xbeta)

  # Naive parametric bootstrap MSE

  summse.pb<-summse.pb+(thetaEBLUPSpat.boot-theta.boot)^2

  # Bias-corrected parametric bootstrap:
  # For de bias of g1 and g2, calculate g1sp and g2sp for
  # each bootstrap sample

  XtVi<-Xt%*%Vi
  Q<-solve(XtVi%*%X)

  Ga<-G-G%*%Vi%*%G
  for (i in 1:m) {g1sp.aux[i]<-Ga[i,i]}
  # g1sp contains the diagonal elements of Ga

  Gb<-G%*%Vi%*%X
  Xa<-matrix(0,1,p)

  for (i in 1:m){
    Xa[1,]<-X[i,]-Gb[i,]
    g2sp.aux[i]<-Xa%*%Q%*%t(Xa)
  }

  # Bootstrap spatial BLUP
  Bstim.sblup<-solve(XtV.sblupi%*%X)%*%XtV.sblupi%*%direct.boot
  Xbeta.sblup<-X%*%Bstim.sblup
  thetaEBLUPSpat.sblup.boot<-Xbeta.sblup
  +G.sblup%*%V.sblupi%*%(direct.boot-Xbeta.sblup)

  # Parametric bootstrap estimator of g3
  sumg3sp.pb<-sumg3sp.pb
  +(thetaEBLUPSpat.boot-thetaEBLUPSpat.sblup.boot)^2
```

```
    # Expectation of estimated g1 and g2
    sumg1sp.pb<-sumg1sp.pb+g1sp.aux
    sumg2sp.pb<-sumg2sp.pb+g2sp.aux

  } # End of bootstrap cycle

  # Final naive parametric bootstrap MSE estimator
  mse.pb<-summse.pb/n.boot
  # Final bias-corrected bootstrap MSE estimator
  g1sp.pb<-sumg1sp.pb/n.boot
  g2sp.pb<-sumg2sp.pb/n.boot
  g3sp.pb<-sumg3sp.pb/n.boot
  mse.pb2<-2*(g1sp+g2sp)-g1sp.pb-g2sp.pb+g3sp.pb

  # Return naive and bias-corrected parametric bootstrap
  return(data.frame(PBmse=mse.pb,bcPBmse=mse.pb2))

}
```

16.4 R code of NPBMSE.SpatialFH

The R code of the function `NPBMSE.SpatialFH` is listed bellow.

```
#####################################################################
###
### This function gives a nonparametric bootstrap MSE estimator of
### the EB estimator under a Spatial FH model, in which random
### effects follow a Simultaneously Autorregressive (SAR) process.
### The EB estimator is obtained either by REML or by FH fitting
### methods.
###
### Work for European project SAMPLE
###
### Author: Nicola Salvati and Isabel Molina
### File name: NPBMSE_SpatialFHModel.R
### Updated: February 8th, 2011
###
#####################################################################

source("Fitting_SpatialFHModel.R")

NPBMSE.SpatialFHmodel<-function(X,y,Dvec,W,n.boot,method="REML",
seed=Sys.time()){

# Set the seed for random number generation, required for the
# bootstrap method.

set.seed(seed)
```

```
m<-dim(X)[1]  # Sample size or number of areas
p<-dim(X)[2]  # Num. of auxiliary variables (including intercept)

# Fit the model to initial sample data using the given method

results.SpFH<-try(fitSpatialFH(X,y,Dvec,W,method))

# Initial estimators of model coefficients, variance and spatial
# correlation which will act as true values in the bootstrap
# procedure.

Bstim.boot<-results.SpFH$modelcoefficients[,1]
rho.boot<-results.SpFH$spatialcorr
sigma2.boot<-results.SpFH$variance

# Auxiliary calculations

I<-diag(1,m)
Wt<-t(W)
Xt<-t(X)

IrhoW<-I-rho.boot*W
IrhoWt<-t(IrhoW)
Ar<-solve(IrhoWt%*%IrhoW)
Gr<-sigma2.boot*Ar
Vr<-Gr+I*Dvec
Vri<-solve(Vr)
Qr<-solve(Xt%*%Vri%*%X)

# Analytical estimators of g1 and g2, used for the bias-corrected
# PB MSE estimator.

g1sp<-rep(0,m)
g2sp<-rep(0,m)

XtVri<-Xt%*%Vri
Qr<-solve(XtVri%*%X)

# Calculate g1 and g2

Ga<-Gr-Gr%*%Vri%*%Gr
Gb<-Gr%*%t(XtVri)
Xa<-matrix(0,1,p)

for (i in 1:m) {
  g1sp[i]<-Ga[i,i]
  Xa[1,]<-X[i,]-Gb[i,]
  g2sp[i]<-Xa%*%Qr%*%t(Xa)
}

# Residual vectors

res<-y-X%*%Bstim.boot
vstim<-Gr%*%Vri%*%res

# Calculate covariance matrices of residual vectors

VG<-Vr-Gr
P<-Vri-Vri%*%X%*%Qr%*%Xt%*%Vri

Ve<-VG%*%P%*%VG
Vu<-IrhoW%*%Gr%*%P%*%Gr%*%IrhoWt

# Square roots of covariance matrices

VecVe0<-eigen(Ve)$vectors
VecVe<-VecVe0[,1:(m-p)]
ValVe0<-eigen(Ve)$values
Valve<-diag(sqrt(1/ValVe0[1:(m-p)]))
Vei05<-VecVe%*%Valve%*%t(VecVe)

VecVu0<-eigen(Vu)$vectors
VecVu<-VecVu0[,1:(m-p)]
ValVu0<-1/(eigen(Vu)$values)
ValVu<-diag(sqrt(ValVu0[1:(m-p)]))
Vui05<-VecVu%*%ValVu%*%t(VecVu)

# Standardize residual vectors

ustim<-as.real(Vui05%*%((IrhoW)%*%vstim))
estim<-as.real(Vei05%*%(res-vstim))
sdu<-sqrt(sigma2.boot)

u.std<-rep(0,m)
e.std<-rep(0,m)

for (i in 1:m){
  u.std[i]<-(sdu*(ustim[i]-mean(ustim)))/
  sqrt(mean((ustim-mean(ustim))^2))
  e.std[i]<-(estim[i]-mean(estim))/
  sqrt(mean((estim-mean(estim))^2))
}

# Bootstrap algorithm starts

difmse.npb<-matrix(0,m,1)
difg3Spat.npb<-matrix(0,m,1)
```

```
g1sp.aux<-matrix(0,m,1)
g2sp.aux<-matrix(0,m,1)
difg1sp.npb<-matrix(0,m,1)
difg2sp.npb<-matrix(0,m,1)

countdetB<-0

for (boot in 1:n.boot) {

  cat("Bootstrap iteration",boot,"\n")

  # Generate boostrap data

  u.boot<-sample(u.std,m,replace=TRUE)
  e.samp<-sample(e.std,m,replace=TRUE)
  e.boot<-sqrt(Dvec)*e.samp
  v.boot<-solve(IrhoW)%*%u.boot
  theta.boot<-X%*%Bstim.boot+v.boot
  direct.boot<-theta.boot+e.boot

  # Fit the model to bootstrap data

  results.SpFH.boot<-fitSpatialFH(X,direct.boot[,1],Dvec,W,method)

  # While the estimators are not satisfactory we generate a new sample

  conv<-results.SpFH.boot$convergence
  v<-results.SpFH.boot$variance
  r<-results.SpFH.boot$spatialcorr
  print(conv)
  print(c(v,r))

  if (conv==FALSE) {v<-0}
  if (is.na(v)==TRUE) {v<-0}
  if (is.na(r)==TRUE) {r<-1}

  while ((v<0.0001)|(r<(-0.999))|(r>0.999)){
    print("New sample")
    u.boot<-sample(u.std,m,replace=TRUE)
    e.samp<-sample(e.std,m,replace=TRUE)
    v.boot<-solve(IrhoW)%*%u.boot
    theta.boot<-X%*%Bstim.boot+v.boot
    direct.boot<-theta.boot+e.boot
    results.SpFH.boot<-fitSpatialFH(X,direct.boot[,1],Dvec,W,method)
    conv<-results.SpFH.boot$convergence
    v<-results.SpFH.boot$variance
    r<-results.SpFH.boot$spatialcorr
    if (conv==FALSE) {v<-0}
    if (is.na(v)==TRUE) {v<-0}
    if (is.na(r)==TRUE) {r<-1}
  }

  Bstim.ML.boot<-results.SpFH.boot$modelcoefficients[,1]
  rho.ML.boot<-results.SpFH.boot$spatialcorr
  sigma2.ML.boot<-results.SpFH.boot$variance
  thetaEB.SpFH.boot<-results.SpFH.boot$EBpredictor

  # Nonparametric bootstrap estimator of g3

  Bstim.sblup<-Qr%*%XtVri%*%direct.boot[,1]
  thetaEB.SpFH.sblup.boot<-X%*%Bstim.sblup+Gr%*%Vri%*%
  (direct.boot[,1]-X%*%Bstim.sblup)

  difg3Spat.npb[,1]<-difg3Spat.npb[,1]+
  (thetaEB.SpFH.boot-thetaEB.SpFH.sblup.boot)^2

  # Naive nonparametric bootstrap MSE

  difmse.npb[,1]<-difmse.npb[,1]+(thetaEB.SpFH.boot[,1]-theta.boot)^2

  # g1 and g2 for each bootstrap sample

  A<-solve((I-rho.ML.boot*Wt)%*%(I-rho.ML.boot*W))
  G<-sigma2.ML.boot*A
  V<-G+I*Dvec
  Vi<-solve(V)
  XtVi<-Xt%*%Vi
  Q<-solve(XtVi%*%X)

  Ga<-G-G%*%Vi%*%G
  Gb<-G%*%Vi%*%X
  Xa<-matrix(0,1,p)

  for (i in 1:m){
    g1sp.aux[i]<-Ga[i,i]
    Xa[1,]<-X[i,]-Gb[i,]
    g2sp.aux[i]<-Xa%*%Q%*%t(Xa)
  }

  difg1sp.npb<-difg1sp.npb+g1sp.aux
  difg2sp.npb<-difg2sp.npb+g2sp.aux

} # End of bootstrap cycle

# Final naive nonparametric bootstrap MSE estimator
mse.npb<-difmse.npb[,1]/n.boot

# Final bias-corrected nonparametric bootstrap MSE estimator
```

```
g3Spat.npb<-difg3Spat.npb/n.boot
g1sp.npb<-difg1sp.npb/n.boot
g2sp.npb<-difg2sp.npb/n.boot
mse.npb2<-2*(g1sp+g2sp)-difg1sp.npb[,1]/n.boot-difg2sp.npb[,1]/n.boot+
difg3Spat.npb[,1]/n.boot

return (data.frame(NPBmse=mse.npb,bcNPBmse=mse.npb2))

}
```

Chapter 17

Appendix 3: R code for area-level time models

17.1 R code for the area-level models with independent time effects

17.1.1 R code of H3area

The R code of the function **H3area** is listed bellow.

```
####################################################################
###
###      Area level model with independent time effects
###                  SAMPLE project
###
### Author: Agustin Perez Martin
### File name: H3.R
### Updated: November 25th, 2009
###
####################################################################

H3area <- function(X, ydt, D, md, sigma2edt) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    M <- sum(md)

    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[a[[d]]]
        Xd[[d]] <- X[a[[d]],]
    }
    Vd.inv <- VinvXd <- list()
    Q2.inv <- XV2X <- matrix(0, nrow=p, ncol=p)
    yVX <- 0
    for(d in 1:D) {

        ### Elements of the variance matrix
        vd <- sigma2edt[a[[d]]]

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- diag(1/vd)

        ### Product between V^-1_ed  and  X_d for all d submatrices
        VinvXd[[d]] <- Vd.inv[[d]]%*%Xd[[d]]

        ### Inverse of Q2. Next we calculate Q2
        Q2.inv <- Q2.inv + t(Xd[[d]])%*%VinvXd[[d]]

        ### Sum in d of the product with  y^t_d  and  V^-1_ed  and  X_d
        yVX <- yVX + yd[[d]]%*%VinvXd[[d]]
    }
    Q2 <- solve(Q2.inv)

    tr.XV2XQ2 <- 0
    for(d in 1:D)
        tr.XV2XQ2 <- tr.XV2XQ2
                     + sum(diag( t(VinvXd[[d]])%*%VinvXd[[d]]%*%Q2))
    tr.P2 <- sum(1/sigma2edt) - tr.XV2XQ2
    yP2y <- sum(ydt^2/sigma2edt) - yVX%*%Q2%*%t(yVX)
    sigma.u <- (yP2y - (M-p))/tr.P2

    return(as.vector(sigma.u))
}
```

17.1.2 R code of REMLarea.indep

The R code of the function **REMLarea.indep** is listed bellow.

```
#####################################################################
###
###      Area level model with independent time effects
###                 SAMPLE project
###
### Author: Agustin Perez Martin
### File name: REMLindep.R
### Updated: November 25th, 2009
###
#####################################################################
```

```
REMLarea.indep <- function(X,ydt,D,md,sigma2edt,sigma.0,MAXITER=500){

    sigma.f <- sigma.0
    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]


    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[a[[d]]]
        Xd[[d]] <- X[a[[d]],]
    }

    for(ITER in 1:MAXITER){
        Vd.inv <- Vinvyd <- VinvXd <- list()
        Q.inv <- XV2X <- XV3X <- matrix(0, nrow=p, ncol=p)
        tr.V.inv <- tr.V2.inv <- yV2y <- yVX <- XV2y <- 0
        for(d in 1:D) {

          ### Elements of the variance matrix
          vd <- (sigma.f + sigma2edt)[a[[d]]]

          ### Inverse matrix of the variance and submatrices
          Vd.inv[[d]] <- diag(1/vd)

          ### Product between V^-1_ed  and  y_d for all d submatrices
          Vinvyd[[d]] <- Vd.inv[[d]]%*%yd[[d]]

          ### Product between V^-1_ed  and  X_d for all d submatrices
          VinvXd[[d]] <- Vd.inv[[d]]%*%Xd[[d]]

          ### Inverse of Q. Next we calculate Q
          Q.inv <- Q.inv + t(Xd[[d]])%*%VinvXd[[d]]

          ### Sum traces of V^-1_d
          tr.V.inv <- tr.V.inv + sum(1/vd)

          ### Sum traces of V^-2_d
          tr.V2.inv <- tr.V2.inv + sum(1/vd^2)

          ### Sum on d of the product between  X^t_d, V^-2_d and X_d
          XV2X <- XV2X + t(VinvXd[[d]])%*%VinvXd[[d]]

          ### Sum on d of the product between  X^t_d, V^-3_d and X_d
          XV3X <- XV3X + t(VinvXd[[d]])%*%Vd.inv[[d]]%*%VinvXd[[d]]
          ### Sum on d of the product between  y^t_d, V^-2_d and y_d
          yV2y <- yV2y + t(Vinvyd[[d]])%*%Vinvyd[[d]]

          ### Sum on d of the product between  y^t_d, V^-1_d and X_d
          yVX <- yVX + yd[[d]]%*%VinvXd[[d]]

          ### Sum on d of the product between  X^t_d, V^-2_d and y_d
          XV2y <- XV2y + t(VinvXd[[d]])%*%Vinvyd[[d]]
        }
        Q <- solve(Q.inv)

        tr.XV2XQ <- 0
        for(d in 1:D)
            tr.XV2XQ <- tr.XV2XQ
                        + sum(diag( t(VinvXd[[d]])%*%VinvXd[[d]]%*%Q))
        tr.P <- tr.V.inv - tr.XV2XQ
        tr.P2 <- tr.V2.inv - 2*sum(diag(XV3X%*%Q))
                 + sum(diag(XV2X%*%Q%*%XV2X%*%Q))
        yP2y <- yV2y - 2*yVX%*%Q%*%XV2y + yVX%*%Q%*%XV2X%*%Q%*%t(yVX)

        ### Scores and Fisher information matrix
        Ssig <- -0.5*tr.P + 0.5*yP2y
        Fsig <- 0.5*tr.P2

        ### Fisher-Scoring Algorithm
        dif <- Ssig/Fsig
        sigma.f <- sigma.f + dif

        ### Stopping criterion
        if(abs(dif)<0.000001)
            break
    }

    return(list(as.vector(sigma.f), Fsig, Q))
}
```

17.1.3 R code of BETA.U.area.indep

The R code of the function **BETA.U.area.indep** is listed bellow.

```
##################################################################
###
###      Area level model with independent time effects
###                SAMPLE project
###
### Author: Agustin Perez Martin
### File name: EstimationBETAindep.R
### Updated: November 25th, 2009
###
##################################################################
```

```
BETA.U.area.indep <- function(X, ydt, D, md, sigma2edt, sigmau) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- Vd <- Vd.inv <- list()
    Q.inv <- matrix(0, nrow=p, ncol=p)
    XVy <- 0
    for(d in 1:D) {
      yd[[d]] <- ydt[a[[d]]]
      Xd[[d]] <- X[a[[d]],]

      ### Elements of the variance matrix
      vd <- (sigmau + sigma2edt)[a[[d]]]

      ### Inverse matrix of the variance and d submatrices
      Vd.inv[[d]] <- diag(1/vd)

      ### Inverse of Q. Next we calculate Q
      Q.inv <- Q.inv + t(Xd[[d]])%*%Vd.inv[[d]]%*%Xd[[d]]

      ### Product between X^t_d,  V^-1_d and y_d for all d submatrices
      XVy <- XVy + t(Xd[[d]])%*%Vd.inv[[d]]%*%yd[[d]]
    }
    Q <- solve(Q.inv)

    beta <- Q%*%XVy

    u <- list()
    for(d in 1:D)
        u[[d]] <- sigmau*Vd.inv[[d]]%*%(yd[[d]]-Xd[[d]]%*%beta)
    u <- as.matrix(unlist(u))

    return(rbind(beta,u))
}
```

17.1.4 R code of mse.area.indep

The R code of the function **mse.area.indep** is listed bellow.

```
##################################################################
###
###    Area level model with independent time effects
###                SAMPLE project
###
### Author: Agustin Perez Martin
### File name: EstimationMSEindep.R
### Updated: November 25th, 2009
###
##################################################################

mse.area.indep <- function(X, D, md, sigma2edt, sigmau, Fsig) {

p <- ncol(X)
a <- list(1:md[1])
mdcum <- cumsum(md)
for(d in 2:D)
a[[d]] <- (mdcum[d-1]+1):mdcum[d]

Xd <- Vd.inv <- Sed.inv <- SinvXd <- VinvSinvXd <- g2.a <- list()
Q.inv <- matrix(0, nrow=p, ncol=p)
XVy <- 0
for(d in 1:D) {
  Xd[[d]] <- X[a[[d]],]

  ### Elements of the variance matrix
  vd <- (sigmau + sigma2edt)[a[[d]]]

  ### Inverse matrix of the variance and d submatrices
  Vd.inv[[d]] <- diag(1/vd)

  ### Elements of the variance matrix Sigma_e
  sd <- sigma2edt[a[[d]]]

  ### Inverse matrix of Sigma_ed in all d submatrices
  Sed.inv[[d]] <- diag(1/sd)

  ### Product between Sigma^-1_ed and X_d for all d submatrices
  SinvXd[[d]] <- Sed.inv[[d]]%*%Xd[[d]]

  ### Product between V^-1_d, Sigma^-1_ed and X_d for all d submatrices
  VinvSinvXd[[d]] <- Vd.inv[[d]]%*%SinvXd[[d]]

  ### First part of g2 (the second is its transpose)
  g2.a[[d]] <- Xd[[d]] - sigmau*SinvXd[[d]] + sigmau^2*VinvSinvXd[[d]]
  ### Inverse of Q. Next we calculate Q
  Q.inv <- Q.inv + t(Xd[[d]])%*%Vd.inv[[d]]%*%Xd[[d]]
 }
    Q <- solve(Q.inv)

    ### Elements of the variance matrix
    vd <- sigmau + sigma2edt
    q <- 1/vd - 2*(sigmau/vd^2) + (sigmau^2/vd^3)

    ### Calculation of g
    g1 <- (sigmau*sigma2edt)/vd
    g2 <- list()
    for(d in 1:D)
        g2[[d]] <- diag(g2.a[[d]]%*%Q%*%t(g2.a[[d]]))
    g2 <- unlist(g2)
    g3 <- q/Fsig

    return(g1+g2+2*g3)
}
```

17.1.5 R code of Interval.indep

The R code of the function **Interval.indep** is listed bellow.

```
#################################################################
###
###    Area level model with independent time effects
###                  SAMPLE project
###
### Author: Agustin Perez Martin
### File name: ICindep.R
### Updated: November 25th, 2009
###
#################################################################

Interval.indep <- function(fit, conf=0.95) {
    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- solve(fit[[2]])

    sigma.std.err <- z*sqrt(Finv[1,1])
    beta.std.err <- z*sqrt(as.vector(diag(fit[[3]])))

    return( list(sigma.std.err, beta.std.err) )
}
```

17.1.6 R code of pvalue

The R code of the function **pvalue** is listed bellow.

```
####################################################################
####
###    Area level model with independent time effects
###        and with time correlated effects
###                 SAMPLE project
###
### Author: Agustin Perez Martin
### File name: pvalue.R
### Updated: November 25th, 2009
###
####################################################################

pvalue <- function(beta0, fit) {

    z <- abs(beta0)/sqrt(as.vector(diag(fit[[3]])))
    pval <- 2*pnorm(z, lower.tail=F)

    return( pval )
}
```

17.2 R code for the area-level models with correlated time effects

17.2.1 R code of REMLarea.autocorr

The R code of the function **REMLarea.autocorr** is listed bellow.

```
#######################################################################
###
###        Area level model with time correlated effects
###                       SAMPLE project
###
### Author: Maria Dolores Esteban Lefler
### File name: REMLautocorr.R
### Updated: November 25th, 2009
###
#######################################################################

REMLarea.autocorr<-function(X,ydt,D,md,sigma2edt,sigma.0,MAXITER=500){

rho.f <- 0
sigma.f <- sigma.0
theta.f  <- c(sigma.f,rho.f)
p <- ncol(X)
a <- list(1:md[1])
mdcum <- cumsum(md)
for(d in 2:D)
```

```
   a[[d]] <- (mdcum[d-1]+1):mdcum[d]

   yd <- Xd <- list()
    for(d in 1:D) {
       yd[[d]] <- ydt[a[[d]]]
       Xd[[d]] <- X[a[[d]],]
    }

for(ITER in 1:MAXITER){
    Vd.inv <- Vad  <- Vbd <- VinvVad <- VinvVbd  <-
    Vinvyd <- VinvXd <- XtVinvVadVinvX  <- XtVinvVbdVinvX <-
    VinvVadVinvVad <-  VinvVadVinvVad <- VinvVadVinvVad <-
    XtVinvVadVinvVadVinvX <- VinvVbdVinvVbd <- VinvVadVinvVbd <-
    XtVinvVbdVinvVbdVinvX <- XtVinvVadVinvVbdVinvX <- list()

    Q.inv <- matrix(0, nrow=p, ncol=p)

    tr.VinvVad  <- tr.VinvVbd   <- tr.VinvVadVinvVad  <-
    tr.VinvVbdVinvVbd <- tr.VinvVadVinvVbd  <- ytVinvX <-
    ytVinvVadVinvy  <- SumXtVinvVadVinvX <- ytVinvVadVinvX <-
    ytVinvVbdVinvy  <- ytVinvVbdVinvX <- SumXtVinvVbdVinvX  <- 0

### Matrix Omegad and its derivative

for(d in 1:D) {
    Omegad<-matrix(0,nrow=md[d],ncol=md[d])
    Omegad[lower.tri(Omegad)]<-rho.f^sequence((md[d]-1):1)
    Omegad<-Omegad+t(Omegad)
    diag(Omegad)<-1
    Omegad <- (1/(1-rho.f^2))*Omegad
    Vad[[d]] <- Omegad

### Derivative

    OmegadFirst<-matrix(0,nrow=md[d],ncol=md[d])
    OmegadFirst[lower.tri(OmegadFirst)]<-sequence((md[d]-1):1)*
                                     rho.f^(sequence((md[d]-1):1)-1)
    OmegadFirst<-OmegadFirst+t(OmegadFirst)
    OmegadFirst<- (1/(1-rho.f^2))*OmegadFirst
    OmegadFirst <- OmegadFirst + (2*rho.f/(1-rho.f^2))*Omegad
    Vbd[[d]] <- sigma.f*OmegadFirst

### Matrix Calculation for Scores and F

### Elements of the variance matrix
    Vd <- (sigma.f * Omegad + diag(sigma2edt[a[[d]]]))

### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- solve(Vd)
```

```
### Product between V^-1_ed  and  y_d for all d submatrices
    Vinvyd[[d]] <- Vd.inv[[d]]%*%yd[[d]]

### Product between V^-1_ed  and  X_d for all d submatrices
    VinvXd[[d]] <- Vd.inv[[d]]%*%Xd[[d]]

### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]])%*%VinvXd[[d]]

### Sa
    VinvVad[[d]] <- Vd.inv[[d]]%*%Vad[[d]]
    tr.VinvVad <-  tr.VinvVad + sum(diag(VinvVad[[d]]))

    XtVinvVadVinvX[[d]]<- t(VinvXd[[d]])%*%Vad[[d]]%*%VinvXd[[d]]

    ytVinvX <- ytVinvX + t(yd[[d]])%*%VinvXd[[d]]
    ytVinvVadVinvy <- ytVinvVadVinvy
                    + t(Vinvyd[[d]])%*%Vad[[d]]%*%Vinvyd[[d]]
    ytVinvVadVinvX <- ytVinvVadVinvX
                    + t(Vinvyd[[d]])%*%Vad[[d]]%*%VinvXd[[d]]
    SumXtVinvVadVinvX <- SumXtVinvVadVinvX + XtVinvVadVinvX[[d]]

### Sb
    VinvVbd[[d]] <- Vd.inv[[d]]%*%Vbd[[d]]
    tr.VinvVbd <-  tr.VinvVbd + sum(diag(VinvVbd[[d]]))

    XtVinvVbdVinvX[[d]] <- t(VinvXd[[d]])%*%Vbd[[d]]%*%VinvXd[[d]]

    ytVinvVbdVinvy <- ytVinvVbdVinvy
                    + t(Vinvyd[[d]])%*%Vbd[[d]]%*%Vinvyd[[d]]
    ytVinvVbdVinvX <- ytVinvVbdVinvX
                    + t(Vinvyd[[d]])%*%Vbd[[d]]%*%VinvXd[[d]]
    SumXtVinvVbdVinvX <- SumXtVinvVbdVinvX + XtVinvVbdVinvX[[d]]

### Faa
    VinvVadVinvVad[[d]] <- Vd.inv[[d]]%*%Vad[[d]]%*%
                           Vd.inv[[d]]%*%Vad[[d]]
    tr.VinvVadVinvVad <- tr.VinvVadVinvVad
                         + sum(diag(VinvVadVinvVad[[d]]))

    XtVinvVadVinvVadVinvX[[d]] <- t(VinvXd[[d]])%*%Vad[[d]]%*%
                           Vd.inv[[d]]%*%Vad[[d]]%*%VinvXd[[d]]

### Fbb
    VinvVbdVinvVbd[[d]] <- Vd.inv[[d]]%*%Vbd[[d]]%*%
                           Vd.inv[[d]]%*%Vbd[[d]]
    tr.VinvVbdVinvVbd <- tr.VinvVbdVinvVbd
                         + sum(diag(VinvVbdVinvVbd[[d]]))
    XtVinvVbdVinvVbdVinvX[[d]] <- t(VinvXd[[d]])%*%Vbd[[d]]%*%
                          Vd.inv[[d]]%*%Vbd[[d]]%*%VinvXd[[d]]

### Fab
    VinvVadVinvVbd[[d]]  <- Vd.inv[[d]]%*%
                            Vad[[d]]%*%Vd.inv[[d]]%*%Vbd[[d]]
    tr.VinvVadVinvVbd <- tr.VinvVadVinvVbd
                         + sum(diag(VinvVadVinvVbd[[d]]))

    XtVinvVadVinvVbdVinvX[[d]] <- t(VinvXd[[d]])%*%Vad[[d]]%*%
                          Vd.inv[[d]]%*%Vbd[[d]]%*%VinvXd[[d]]
        }

Q <- solve(Q.inv)


tr.XtVinvVadVinvXQ <- tr.XtVinvVbdVinvXQ  <-
tr.XtVinvVadVinvVadVinvXQ <- tr.XtVinvVbdVinvVbdVinvXQ <-
tr.XtVinvVadVinvVbdVinvXQ <- tr.XtVinvVbdVinvVbdVinvXQ <-
XtVinvVadVinvXQ <- XtVinvVbdVinvXQ <- 0


for(d in 1:D){
  tr.XtVinvVadVinvXQ <- tr.XtVinvVadVinvXQ
                        + sum(diag(XtVinvVadVinvX[[d]]%*%Q))
  tr.XtVinvVbdVinvXQ <- tr.XtVinvVbdVinvXQ
                        + sum(diag(XtVinvVbdVinvX[[d]]%*%Q))
  tr.XtVinvVadVinvVadVinvXQ <- tr.XtVinvVadVinvVadVinvXQ
                        + sum(diag(XtVinvVadVinvVadVinvX[[d]]%*%Q))
  XtVinvVadVinvXQ  <- XtVinvVadVinvXQ
                       + XtVinvVadVinvX[[d]]%*%Q
   tr.XtVinvVbdVinvVbdVinvXQ <- tr.XtVinvVbdVinvVbdVinvXQ
                         + sum(diag(XtVinvVbdVinvVbdVinvX[[d]]%*%Q))
   XtVinvVbdVinvXQ  <- XtVinvVbdVinvXQ + XtVinvVbdVinvX[[d]]%*%Q
   tr.XtVinvVadVinvVbdVinvXQ <- tr.XtVinvVadVinvVbdVinvXQ
                         + sum(diag(XtVinvVadVinvVbdVinvX[[d]]%*%Q))
      }

   tr.XtVinvVadVinvXQXtVinvVadVinvXQ <- sum(diag(XtVinvVadVinvXQ%*%
                                        XtVinvVadVinvXQ))
   tr.XtVinvVbdVinvXQXtVinvVbdVinvXQ <- sum(diag(XtVinvVbdVinvXQ%*%
                                        XtVinvVbdVinvXQ))
   tr.XtVinvVadVinvXQXtVinvVbdVinvXQ <- sum(diag(XtVinvVadVinvXQ%*%
                                        XtVinvVbdVinvXQ))

   tr.PVa <- tr.VinvVad - tr.XtVinvVadVinvXQ
   tr.PVb <- tr.VinvVbd - tr.XtVinvVbdVinvXQ
   tr.PVaPVa  <- tr.VinvVadVinvVad - 2*tr.XtVinvVadVinvVadVinvXQ
```

```
                + tr.XtVinvVadVinvXQXtVinvVadVinvXQ
tr.PVbPVb  <- tr.VinvVbdVinvVbd - 2*tr.XtVinvVbdVinvVbdVinvXQ
                + tr.XtVinvVbdVinvXQXtVinvVbdVinvXQ
tr.PVaPVb  <- tr.VinvVadVinvVbd - 2*tr.XtVinvVadVinvVbdVinvXQ
                + tr.XtVinvVadVinvXQXtVinvVbdVinvXQ

ytPVaPy  <- ytVinvVadVinvy - ytVinvVadVinvX%*%Q%*%t(ytVinvX)
              - ytVinvX%*%Q%*%t(ytVinvVadVinvX) + ytVinvX%*%Q%*%
              SumXtVinvVadVinvX%*%Q%*%t(ytVinvX)

ytPVbPy  <- ytVinvVbdVinvy - ytVinvVbdVinvX%*%Q%*%t(ytVinvX)
              - ytVinvX%*%Q%*%t(ytVinvVbdVinvX) + ytVinvX%*%Q%*%
              SumXtVinvVbdVinvX%*%Q%*%t(ytVinvX)


### Scores and Fisher information matrix

Sa <- -0.5*tr.PVa + 0.5*ytPVaPy
Sb <- -0.5*tr.PVb + 0.5*ytPVbPy
Faa  <- 0.5*tr.PVaPVa
Fbb  <- 0.5*tr.PVbPVb
Fab  <- 0.5*tr.PVaPVb

Ssig <- c(Sa,Sb)
Fsig <- matrix(c(Faa,Fab,Fab,Fbb),ncol=2)

### Fisher-Scoring Algorithm

Fsig.inv <- solve(Fsig)
dif <- Fsig.inv%*%Ssig

theta.f <- theta.f + dif

###print(rho.f)
rho.f <- theta.f[2,1]

###print(sigma.f)
sigma.f <- theta.f[1,1]


### output3 <- data.frame(ITER, Sa, Sb, Faa, Fbb, Fab)
### output3 <- as.data.frame(t(output3))
### write.table(output3, file="Output3.txt", sep="\t")

### Stopping criterion
if(abs(dif[1,1])<0.000001 && abs(dif[2,1])<0.000001)
   break
 }
```

```
   return(list(as.vector(theta.f), Fsig, Q))
}
```

17.2.2 R code of BETA.U.area.autocorr

The R code of the function **BETA.U.area.autocorr** is listed bellow.

```
#####################################################################
###
###          Area level model with time correlated effects
###                          SAMPLE project
###
### Author: Maria Dolores Esteban Lefler
### File name: EstimationBETAautocorr.R
### Updated: November 25th, 2009
###
#####################################################################

BETA.U.area.autocorr <- function(X,ydt,D,md,sigma2edt,sigmau,rho) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- Vd <- Vd.inv <- list()
    Q.inv <- matrix(0, nrow=p, ncol=p)
    XVy <- 0
    for(d in 1:D) {
        yd[[d]] <- ydt[a[[d]]]
        Xd[[d]] <- X[a[[d]],]
        Omegad<-matrix(0,nrow=md[d],ncol=md[d])
              Omegad[lower.tri(Omegad)]<-rho^sequence((md[d]-1):1)
              Omegad<-Omegad+t(Omegad)
              diag(Omegad)<-1
              Omegad<- (1/(1-rho^2))*Omegad

      ### Elements of the variance matrix
        Vd <- (sigmau * Omegad + diag(sigma2edt[a[[d]]]))

      ### Inverse matrix of the variance and d submatrices
        Vd.inv[[d]] <- solve(Vd)

      ### Inverse of Q. Next we calculate Q
        Q.inv <- Q.inv + t(Xd[[d]])%*%Vd.inv[[d]]%*%Xd[[d]]

      ### Product between X^t_d, V^-1_d and y_d for all d submatrices
        XVy <- XVy + t(Xd[[d]])%*%Vd.inv[[d]]%*%yd[[d]]
    }
    Q <- solve(Q.inv)

    beta <- Q%*%XVy

    u <- list()
    for(d in 1:D)
        u[[d]] <- sigmau*(Omegad%*%Vd.inv[[d]]%*%(yd[[d]]
                  - Xd[[d]]%*%beta))
    u <- as.matrix(unlist(u))

    return(rbind(beta,u))
}
```

17.2.3 R code of mse.area.autocorr

The R code of the function **mse.area.autocorr** is listed bellow.

```
##################################################################
###
###       Area level model with time correlated effects
###                    SAMPLE project
###
### Author: Maria Dolores Esteban Lefler
### File name: EstimationMSEautocorr.R
### Updated: November 25th, 2009
###
##################################################################


mse.area.autocorr <- function(X,D,md,sigma2edt,sigmau,rho,Fsig) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    Xd <- Vd.inv <- Sed.inv <- Omegad  <- OmegadFirst <- VinvOmega <-
    OmegaVinvOmega <- VinvOmegaFirst  <- OmegaVinvOmegaFirst <-
    OmegaFirstVinvOmegaFirst <- SinvXd <- OmegaVinvOmegadSinvXd <-
    g1.a  <- g2.a  <- q11  <- q12 <- q22 <- list()

    Q.inv <- matrix(0, nrow=p, ncol=p)

    for(d in 1:D) {

### Matrix Omegad and its derivative

    Omegad[[d]]<-matrix(0,nrow=md[d],ncol=md[d])
    Omegad[[d]][lower.tri(Omegad[[d]])]<-rho^sequence((md[d]-1):1)
    Omegad[[d]]<-Omegad[[d]]+t(Omegad[[d]])
    diag(Omegad[[d]])<-1
    Omegad[[d]] <- (1/(1-rho^2))*Omegad[[d]]

### Derivative

    OmegadFirst[[d]]<-matrix(0,nrow=md[d],ncol=md[d])
    OmegadFirst[[d]][lower.tri(OmegadFirst[[d]])] <-
    sequence((md[d]-1):1)*rho^(sequence((md[d]-1):1)-1)
    OmegadFirst[[d]]<-OmegadFirst[[d]]+t(OmegadFirst[[d]])
    OmegadFirst[[d]]<- (1/(1-rho^2))*OmegadFirst[[d]]
    OmegadFirst[[d]] <- OmegadFirst[[d]]
                        + (2*rho/(1-rho^2))*Omegad[[d]]

    Xd[[d]] <- X[a[[d]],]

### Matrix Sigma_e
    Sed <- diag(sigma2edt[a[[d]]])

### Matrix of variance
     Vd <- (sigmau * Omegad[[d]] + Sed)

### Inverse matrix of the variance and d submatrices
    Vd.inv[[d]] <- solve(Vd)

### Inverse matrix of Sigma_ed in all d submatrices
    Sed.inv[[d]] <- solve(Sed)

### Product between V^-1_d and Omega
    VinvOmega[[d]] <-  Vd.inv[[d]]%*%  Omegad[[d]]
    VinvOmegaFirst[[d]] <-  Vd.inv[[d]]%*% OmegadFirst[[d]]

### Product between Omega, V^-1_d and Omega
       OmegaVinvOmega[[d]] <- t(VinvOmega[[d]])%*%Omegad[[d]]

### Product between Omega, V^-1_d and OmegadFirst
    OmegaVinvOmegaFirst[[d]] <- Omegad[[d]] %*% Vd.inv[[d]] %*%
                                OmegadFirst[[d]]

### Product between OmegadFirst, V^-1_d and OmegadFirst
    OmegaFirstVinvOmegaFirst[[d]] <- OmegadFirst[[d]]%*%
                                     Vd.inv[[d]]%*%OmegadFirst[[d]]

### Product between Sigma^-1_ed and X_d for all d submatrices
     SinvXd[[d]] <- Sed.inv[[d]]%*%Xd[[d]]

### Product between Omegad, V^-1_d, Omegad, Sigma^-1_ed
### and X_d for d submatrices
    OmegaVinvOmegadSinvXd[[d]] <- OmegaVinvOmega[[d]]%*%SinvXd[[d]]
```

```
### First part of g1 (the second is its transpose)
    g1.a[[d]] <- sigmau * Omegad[[d]] - sigmau^2 *OmegaVinvOmega[[d]]

### First part of g2 (the second is its transpose)
g2.a[[d]] <- Xd[[d]] - sigmau*Omegad[[d]]%*%SinvXd[[d]]
           + sigmau^2* OmegaVinvOmegadSinvXd[[d]]

q11[[d]] <- OmegaVinvOmega[[d]] - 2*sigmau*OmegaVinvOmega[[d]]%*%
          VinvOmega[[d]] + sigmau^2*OmegaVinvOmega[[d]] %*%
          Vd.inv[[d]]%*%OmegaVinvOmega[[d]]

q12[[d]] <- sigmau*OmegaVinvOmegaFirst[[d]] - sigmau^2*
          OmegaVinvOmegaFirst[[d]]%*%VinvOmega[[d]]
          - sigmau^2*OmegaVinvOmega[[d]]%*%VinvOmegaFirst[[d]]
          + sigmau^3*OmegaVinvOmega[[d]]%*%VinvOmegaFirst[[d]]%*%
          VinvOmega[[d]]
q22[[d]] <- sigmau^2*OmegaFirstVinvOmegaFirst[[d]]
          - 2*sigmau^3*OmegaVinvOmegaFirst[[d]]%*%
           VinvOmegaFirst[[d]] + sigmau^4* OmegaVinvOmegaFirst[[d]]%*%
           Vd.inv[[d]]%*%t(OmegaVinvOmegaFirst[[d]])

### Inverse of Q. Next we calculate Q
 Q.inv <- Q.inv + t(Xd[[d]])%*%Vd.inv[[d]]%*%Xd[[d]]
 }

   Q <- solve(Q.inv)

   ### Calculation of g

   g1 <- g2 <- g3 <- list()
   for(d in 1:D){
       g1[[d]] <- diag(g1.a[[d]])
       g2[[d]] <- diag(g2.a[[d]]%*%Q%*%t(g2.a[[d]]))
       q11[[d]] <- diag(q11[[d]])
       q12[[d]] <- diag(q12[[d]])
       q22[[d]] <- diag(q22[[d]])
       }

   for(d in 1:D){
       g3[[d]] <- vector()
          for(t in 1:md[d]){
              g3[[d]][t] <- sum(diag(matrix(c(q11[[d]][t],
                             rep(q12[[d]][t],2),q22[[d]][t]),
                             nrow=2)%*%solve(Fsig)))
                }
    }
   g1 <- unlist(g1)
   g2 <- unlist(g2)
   g3 <- unlist(g3)

   return(g1+g2+2*g3)
}
```

17.2.4 R code of Interval.autocorr

The R code of the function **Interval.autocorr** is listed bellow.

```
################################################################
###
###       Area level model with time correlated effects
###                    SAMPLE project
###
### Author: Maria Dolores Esteban Lefler
### File name: ICautocorr.R
### Updated: November 25th, 2009
###
################################################################

Interval.autocorr <- function(fit, conf=0.95) {
    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- solve(fit[[2]])

    sigma.std.err <- z*sqrt(Finv[1,1])
    rho.std.err <- z*sqrt(Finv[2,2])
    beta.std.err <- z*sqrt(as.vector(diag(fit[[3]])))

    return( list(sigma.std.err, rho.std.err, beta.std.err) )
}
```

Chapter 18

Appendix 4: R code for the area-level partitioned time models

18.1 R code for the partitioned Fay-Herriot model 1

18.1.1 R code of H3area

The R code of the function **H3area** is listed bellow.

```
#######################################################################
###   Area level Partitioned F-H model with independent time effects
###                          Pagliarella model 1
### Author: Agustin Perez Martin
### File name: H3.R
### Updated: November 25th, 2009
###
#######################################################################

H3area <- function(X, ydt, D, md, sigma2edt) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    M <- sum(md)

    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[a[[d]]]
        Xd[[d]] <- X[a[[d]],]
    }

    Vd.inv <- VinvXd <- list()
    Q2.inv <- XV2X <- matrix(0, nrow=p, ncol=p)
    yVX <- 0
    for(d in 1:D) {

        ### Elements of the variance matrix
        vd <- sigma2edt[a[[d]]]

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- diag(1/vd)

        ### Product between V^-1_ed  and  X_d for all d submatrices
        VinvXd[[d]] <- Vd.inv[[d]]%*%Xd[[d]]

        ### Inverse of Q2. Next we calculate Q2
        Q2.inv <- Q2.inv + t(Xd[[d]])%*%VinvXd[[d]]

        ### Sum in d of the product with  y^t_d  and  V^-1_ed  and  X_d
        yVX <- yVX + yd[[d]]%*%VinvXd[[d]]
    }
    Q2 <- solve(Q2.inv)

    tr.XV2XQ2 <- 0
    for(d in 1:D)
        tr.XV2XQ2 <- tr.XV2XQ2
                     + sum(diag( t(VinvXd[[d]])%*%VinvXd[[d]]%*%Q2))

    tr.P2 <- sum(1/sigma2edt) - tr.XV2XQ2
    yP2y <- sum(ydt^2/sigma2edt) - yVX%*%Q2%*%t(yVX)

    sigma.u <- (yP2y - (M-p))/tr.P2

    return(as.vector(sigma.u))
}
```

18.1.2 R code of REMLarea

The R code of the function **REMLarea** is listed bellow.

```
#######################################################################
###   Area level Partitioned F-H model with independent time effects
###                         Pagliarella model 1
### Author: Maria Chiara Pagliarella
### File name: REML.R
### Updated: February 2010
###
#######################################################################


REMLarea <- function(X, ydt, D, Da, Db, md, sigma2edt, sigma.0
= sigma.0, MAXITER = 100) {
```

```
sigma.fa <- sigma.0
sigma.fb <- sigma.0
p <- ncol(X)
i <- list(1:md[1])
mdcum <- cumsum(md)
Db <- D-Da

for(d in 2:D){
    i[[d]] <- (mdcum[d-1]+1):mdcum[d]
    if (d<= Da) ia <- i[1:Da]
    else ib <- i[(Da+1):D]
}

yda <- Xda <- list()
for(d in 1:Da) {
    yda[[d]] <- ydt[ia[[d]]]
    Xda[[d]] <- X[ia[[d]],]
}
ydb <- Xdb <- list()
for(d in 1:Db) {
    ydb[[d]] <- ydt[ib[[d]]]
    Xdb[[d]] <- X[ib[[d]],]
}
yd <- Xd <- list()
for(d in 1:D) {
    yd[[d]] <- ydt[i[[d]]]
    Xd[[d]] <- X[i[[d]],]
}

Bad <- Flag <- 0

for(ITER in 1:MAXITER){
    Vda.inv <- VinvXa <- Vinvya <- list()
    XaV2Xa <- XaV3Xa <- matrix(0, nrow=p, ncol=p)
    tr.Vinva <- tr.V2inva <- XaV2ya <- yaV2ya <- yaV2Xa <- 0
    for(d in 1:Da) {
        ### Elements of the variance matrix
        vda <- (sigma.fa + sigma2edt)[ia[[d]]]

        if (abs(det(vda))<0.000000001 || abs(det(vda))>10000000000){
            Flag <- 1
            Bad <- Bad+1
            break }


        ### Inverse matrix of the variance and submatrices
        Vda.inv[[d]] <- diag(1/vda)
        ### Product between V^-1_da and X_da
        ### for all d submatrices
        VinvXa[[d]] <- Vda.inv[[d]] %*% Xda[[d]]

        ### Product between V^-1_da  and  y_da
        ### for all d submatrices
        Vinvya[[d]] <- Vda.inv[[d]] %*% yda[[d]]

        ### Sum traces of V^-1_da
        tr.Vinva <- tr.Vinva + sum(1/vda)

        ### Sum traces of V^-2_da
        tr.V2inva <- tr.V2inva + sum(1/vda^2)

        ### Sum on d of the product between  X^t_da, V^-2_da and X_da
        XaV2Xa <- XaV2Xa + t(VinvXa[[d]]) %*% VinvXa[[d]]

        ### Sum on d of the product between  X^t_da, V^-3_da and X_da
        XaV3Xa <- XaV3Xa + t(VinvXa[[d]]) %*% Vda.inv[[d]]%*%VinvXa[[d]]

        ### Sum on d of the product between  y^t_da, V^-2_da and X_da
        XaV2ya <- XaV2ya + t(VinvXa[[d]]) %*% Vinvya[[d]]

        ### Sum on d of the product between  y^t_da, V^-2_da and y_da
        yaV2ya <- yaV2ya + t(Vinvya[[d]]) %*% Vinvya[[d]]

        ### Sum on d of the product between  y^t_da, V^-2_da and X_da
        yaV2Xa <- yaV2Xa + t(Vinvya[[d]]) %*% VinvXa[[d]]
    }

    if (Flag==1) {
        ITER <- MAXITER
        Flag <- 0
        break }


    Vdb.inv <- VinvXb <- Vinvyb <- list()
    XbV2Xb <- XbV3Xb <- matrix(0, nrow=p, ncol=p)
    tr.Vinvb <- tr.V2invb <- XbV2yb <- ybV2yb <- ybV2Xb <- 0
    for(d in 1:Db) {
        ### Elements of the variance matrix
        vdb <- (sigma.fb + sigma2edt)[ib[[d]]]

        if  (abs(det(vdb))<0.000000001 || abs(det(vdb))>10000000000){
            Flag <- 1
            Bad <- Bad + 1
            break }
        ### Inverse matrix of the variance and submatrices
        Vdb.inv[[d]] <- diag(1/vdb)

        ### Product between V^-1_db and X_db
        ### for all d submatrices
        VinvXb[[d]] <- Vdb.inv[[d]] %*% Xdb[[d]]

        ### Product between V^-1_db  and  y_db
        ### for all d submatrices
        Vinvyb[[d]] <- Vdb.inv[[d]] %*% ydb[[d]]

        ### Sum traces of V^-1_db
        tr.Vinvb <- tr.Vinvb + sum(1/vdb)

        ### Sum traces of V^-2_db
        tr.V2invb <- tr.V2invb + sum(1/vdb^2)

        ### Sum on d of the product between  X^t_db, V^-2_db and X_db
        XbV2Xb <- XbV2Xb + t(VinvXb[[d]]) %*% VinvXb[[d]]

        ### Sum on d of the product between  X^t_db, V^-3_db and X_db
        XbV3Xb <- XbV3Xb + t(VinvXb[[d]]) %*% Vdb.inv[[d]]%*%VinvXb[[d]]

        ### Sum on d of the product between  y^t_db, V^-2_db and X_db
        XbV2yb <- XbV2yb + t(VinvXb[[d]]) %*% Vinvyb[[d]]

        ### Sum on d of the product between  y^t_db, V^-2_db and y_db
        ybV2yb <- ybV2yb + t(Vinvyb[[d]]) %*% Vinvyb[[d]]

        ### Sum on d of the product between  y^t_db, V^-2_db and X_db
        ybV2Xb <- ybV2Xb + t(Vinvyb[[d]]) %*% VinvXb[[d]]
    }

    if (Flag==1) {
        ITER <- MAXITER
        Flag <- 0
        break }


    Vd.inv <- Vinvy <- VinvX <- list()
    Q.inv <- matrix(0, nrow=p, ncol=p)
    yVX <- 0
        for(d in 1:D) {
        ### Elements of the variance matrix
        if (d <= Da)
            vd <-(sigma.fa + sigma2edt)[ia[[d]]]
        else
            vd <-(sigma.fb + sigma2edt)[i[[d]]]
            if  (abs(det(vd))<0.000000001 || abs(det(vd))>10000000000) {
                Flag <- 1
                Bad <- Bad + 1
                break }


            ### Inverse matrix of the variance and submatrices
            Vd.inv[[d]] <- diag(1/vd)

            ### Product between V^-1_ed  and  y_d
            ### for all d submatrices
            Vinvy[[d]] <- Vd.inv[[d]] %*% yd[[d]]

            ### Product between V^-1_ed  and  X_d
            ### for all d submatrices
            VinvX[[d]] <- Vd.inv[[d]] %*% Xd[[d]]

            ### Inverse of Q. Next we calculate Q
            Q.inv <- Q.inv + t(Xd[[d]]) %*% VinvX[[d]]

            ### Sum on d of the product between  y^t_d, V^-1_d and X_d
            yVX <- yVX + yd[[d]] %*% VinvX[[d]]
        }

        if (Flag==1) {
            ITER <- MAXITER
            Flag <- 0
            break }

    ### Calculation of Q
    Q <- solve(Q.inv)


    tr.XaV2XaQ <- 0
    tr.XaV2XaQ <- tr.XaV2XaQ + sum(diag( XaV2Xa %*% Q ))

    tr.PV1 <- tr.Vinva - tr.XaV2XaQ
    tr.PV1PV1 <- tr.V2inva - 2*sum(diag(XaV3Xa %*% Q))
                + sum(diag(XaV2Xa %*% Q %*% XaV2Xa %*% Q))
    yPV1Py <- yaV2ya - (yaV2Xa %*% Q %*% t(yVX)) - ( yVX %*% Q %*% XaV2ya)
                    + ( yVX %*% Q %*% XaV2Xa %*% Q %*% t(yVX))


    tr.XbV2XbQ <- 0
    tr.XbV2XbQ <- tr.XbV2XbQ + sum(diag( XbV2Xb %*% Q ))

    tr.PV2 <- tr.Vinvb - tr.XbV2XbQ
    tr.PV2PV2 <- tr.V2invb - 2*sum(diag(XbV3Xb %*% Q))
                + sum(diag(XbV2Xb %*% Q %*% XbV2Xb %*% Q))
    yPV2Py <- ybV2yb - (ybV2Xb %*% Q %*% t(yVX)) - ( yVX %*% Q %*% XbV2yb)
                + ( yVX %*% Q %*% XbV2Xb %*% Q %*% t(yVX))

        tr.PV1PV2 <- sum(diag(XaV2Xa %*% Q %*% XbV2Xb %*% Q))
        tr.PV2PV1 <- sum(diag(XbV2Xb %*% Q %*% XaV2Xa %*% Q))

        ### Scores and Fisher information matrix
        S1 <- -0.5*tr.PV1 + 0.5*yPV1Py
        S2 <- -0.5*tr.PV2 + 0.5*yPV2Py

        F11 <- 0.5*tr.PV1PV1
        F12 <- 0.5*tr.PV1PV2
        F21 <- 0.5*tr.PV2PV1
        F22 <- 0.5*tr.PV2PV2

        # FINAL Fisher information Matrix
        Fsig <- matrix(c(F11, F12, F21, F22), ncol=p, nrow=p)
        SumFsig <- abs(sum(Fsig))
        # print(det(Fsig))
        if (abs(det(Fsig))<0.000000001 || SumFsig > 10000000) {
            # print(ITER)
            ITER <- MAXITER
            Bad <- Bad + 1
            break }

        ### Fisher-Scoring Algorithm
        difa <- S1/F11
        difb <- S2/F22

        sigma.fa <- sigma.fa + difa
        sigma.fb <- sigma.fb + difb

        ITER <- ITER

        ### Stopping criterion
        if((abs(difa)<0.000001) && (abs(difb)<0.000001)) break

        #x<-"ITER greater than 6"
        #z<-"ITER smaller than 6"
        #if (ITER>6) print(x) else print(z)

  }
  if (sigma.fa < 0 || sigma.fb < 0) {
      ITER <- MAXITER
      Bad <- Bad + 1 }

  return(list(as.vector(sigma.fa), as.vector(sigma.fb), F, ITER, Q, Bad))
}
```

18.1.3 R code of BETA.U.area

The R code of the function **BETA.U.area** is listed bellow.

```
#######################################################################
###   Area level Partitioned F-H model with independent time effects
###                          Pagliarella model 1
### Author: Maria Chiara Pagliarella
### File name: EstimationBETA.R
### Updated: November 2009
###
#######################################################################

BETA.U.area <- function(X, ydt, D, Da, Db, md, sigma2edt,
sigmaua, sigmaub) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
    Db <- D-Da

    for(d in 2:D)
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]

    ia <- i[1:Da]
    ib <- i[(Da+1):D]


    yda <- Xda <- list()
    for(d in 1:Da) {
        yda[[d]] <- ydt[ia[[d]]]
        Xda[[d]] <- X[ia[[d]],]
    }
    ydb <- Xdb <- list()
    for(d in 1:Db) {
        ydb[[d]] <- ydt[ib[[d]]]
        Xdb[[d]] <- X[ib[[d]],]
    }
    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[i[[d]]]
        Xd[[d]] <- X[i[[d]],]
    }


    Vd.inv <- list()
    Q.inv <- matrix(0, nrow=p, ncol=p)
    XVy <-0
    for(d in 1:D) {
      ### Elements of the variance matrix
      if (d <= Da)
         vd<-(sigmaua + sigma2edt)[ia[[d]]]
      else
         vd<-(sigmaub + sigma2edt)[i[[d]]]

    ### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- diag(1/vd)

    ### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]])%*%Vd.inv[[d]]%*%Xd[[d]]

    ### Product between X^t_d,  V^-1_d and y_d for all d submatrices
    XVy <- XVy + t(Xd[[d]])%*%Vd.inv[[d]]%*%yd[[d]]
 }
 Q <- solve(Q.inv)

    beta <- Q%*%XVy


    ua <- ub <- list()
    for(d in 1:Da){
        ua[[d]] <- sigmaua*Vd.inv[[d]]%*%(yda[[d]]-Xda[[d]]%*%beta)
    }
    for(d in 1:Db){
        ub[[d]] <- sigmaub*Vd.inv[[d]]%*%(ydb[[d]]-Xdb[[d]]%*%beta)
    }
    ua<-as.matrix(unlist(ua))
    ub<-as.matrix(unlist(ub))
    u <- c(ua,ub)

    return(c(beta,u))
}
```

18.1.4 R code of mse.area

The R code of the function **mse.area** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with independent time effects
###                         Pagliarella model 1
### Author: Maria Chiara Pagliarella
### File name: EstimationMSE.R
### Updated: Dicember 2009
###
########################################################################

mse.area <- function(X, D, Da, Db, md, sigma2edt, sigmaua,
sigmaub, F11, F22) {

    p <- ncol(X)
```

```
   i <- list(1:md[1])
   mdcum <- cumsum(md)
   Db <- D-Da

   for(d in 2:D){
       i[[d]] <- (mdcum[d-1]+1):mdcum[d]
       if (d<= Da) ia <- i[1:Da]
       else ib <- i[(Da+1):D]
   }


   Xda <- list()
   for(d in 1:Da) {
       Xda[[d]] <- X[ia[[d]],]
   }
   Xdb <- list()
   for(d in 1:Db) {
       Xdb[[d]] <- X[ib[[d]],]
   }
   Xd <- list()
   for(d in 1:D) {
       Xd[[d]] <- X[i[[d]],]
   }

   Vd.inv <- list()
   Q.inv <- matrix(0, nrow=p, ncol=p)
   for(d in 1:D) {
       ### Elements of the variance matrix
       if (d <= Da)
           vd<-(sigmaua + sigma2edt)[ia[[d]]]
       else
           vd<-(sigmaub + sigma2edt)[i[[d]]]

       ### Inverse matrix of the variance and submatrices
       Vd.inv[[d]] <- diag(1/vd)

       ### Inverse of Q. Next we calculate Q
       Q.inv <- Q.inv + t(Xd[[d]]) %*% Vd.inv[[d]] %*% Xd[[d]]

   }
   Q <- solve(Q.inv)

   ##############################
   ### Calculation of MSE_A   ###
   ##############################

Vda.inv <- Sinv.a <- SinvXda <- VinvSinvXda <- g2.1a <- list()
for(d in 1:Da) {
   ### Elements of the variance matrix
```

```
  vda<- (sigmaua + sigma2edt)[ia[[d]]]

  ### Inverse matrix of the variance and submatrices
  Vda.inv[[d]] <- diag(1/vda)

  ### Elements of the variance matrix Sigma_ed
  sed.a <- sigma2edt[ia[[d]]]

  ### Inverse matrix of Sigma_ed for all d submatrices
  Sinv.a[[d]] <- diag(1/sed.a)

  ### Product between Sigma_a^-1_ed and X_da for all d submatrices
  SinvXda[[d]] <- Sinv.a[[d]]%*%Xda[[d]]

  ### Product between V^-1_da, Sigma^-1_ed and X_da for all d submatrices
  VinvSinvXda[[d]] <- Vda.inv[[d]]%*%SinvXda[[d]]

      ### First part of g2_a (the second is its transpose)
      g2.1a[[d]] <- Xda[[d]] - sigmaua*SinvXda[[d]]
                  + (sigmaua^2) * VinvSinvXda[[d]]
  }

  g2a <- mse.a <- list()
  for (d in 1:Da){
      vda <- (sigmaua + sigma2edt)[ia[[d]]]
      q11 <- sigmaua^2 / vda^3

      ### Calculation of g_a
      g1a <- (sigmaua * sigma2edt[ia[[d]]]) / vda

      g2a[[d]] <- diag(g2.1a[[d]] %*% Q %*% t(g2.1a[[d]]))
      g3a <- q11/F11

      g2a[[d]] <- diag(g2.1a[[d]] %*% Q %*% t(g2.1a[[d]]))

      mse.a[[d]] <- g1a + g2a[[d]] + 2 * g3a
  }

  mse.a <- unlist(mse.a)



  ##############################
  ### Calculation of MSE_B   ###
  ##############################


Vdb.inv <- Sinv.b <- SinvXdb <- VinvSinvXdb <- g2.1b <- list()
for(d in 1:Db) {
    ### Elements of the variance matrix
    vdb<- (sigmaub + sigma2edt)[ib[[d]]]

    ### Inverse matrix of the variance and submatrices
    Vdb.inv[[d]] <- diag(1/vdb)

    ### Elements of the variance matrix Sigma_ed
    sed.b <- sigma2edt[ib[[d]]]

    ### Inverse matrix of Sigma_ed for all d submatrices
    Sinv.b[[d]] <- diag(1/sed.b)

    ### Product between Sigma_b^-1_ed and X_db for all d submatrices
    SinvXdb[[d]] <- Sinv.b[[d]]%*%Xdb[[d]]

    ### Product between V^-1_db, Sigma^-1_ed and X_db for all d submatrices
    VinvSinvXdb[[d]] <- Vdb.inv[[d]]%*%SinvXdb[[d]]

    ### First part of g2_b (the second is its transpose)
    g2.1b[[d]] <- Xdb[[d]] - sigmaub*SinvXdb[[d]]
                + sigmaub^2*VinvSinvXdb[[d]]
  }


    g2b <- mse.b <- list()
    for (d in 1:Db){
        vdb <- (sigmaub + sigma2edt)[ib[[d]]]
        q22 <- sigmaub^2/vdb^3

        ### Calculation of g_b
        g1b <- (sigmaub * sigma2edt[ib[[d]]]) / vdb

        g2b[[d]] <- diag(g2.1b[[d]] %*% Q %*% t(g2.1b[[d]]))
        g3b <- q22/F22

        g2b[[d]] <- diag(g2.1b[[d]] %*% Q %*% t(g2.1b[[d]]))

        mse.b[[d]] <- g1b + g2b[[d]] + 2 * g3b
    }

    mse.b <- unlist(mse.b)

    mse <- c(mse.a, mse.b)

    return(mse)
}
```

18.1.5 R code of Interval

The R code of the function **Interval** for the confidence intervals and *p*-values is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with independent time effects
###                          Pagliarella model 1
### Author: Maria Chiara Pagliarella
### File name: IC.R
### Updated: February 2010
###
########################################################################

Interval <- function(Fisher, conf=0.95) {

    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- solve(Fisher[[3]])

    sigma.a.std.err <- z*sqrt(Finv[1,1])
    sigma.b.std.err <- z*sqrt(Finv[2,2])
    sigma.ab.std.err <- z*sqrt(Finv[1,1]+Finv[2,2]-2*Finv[1,2])

    beta.std.err <- z*sqrt(as.vector(diag(Fisher[[5]])))

    infbeta <- beta0.hat-beta.std.err
    supbeta <- beta0.hat+beta.std.err
    testbeta <- beta0.hat-beta.std.err<0 & beta0.hat+beta.std.err>0

    infsigmaua <- sigmaua.hat-sigma.a.std.err
    supsigmaua <- sigmaua.hat+sigma.a.std.err
    testsigmaua <- sigmaua.hat-sigma.a.std.err<0
                   & sigmaua.hat+sigma.a.std.err>0

    infsigmaub <- sigmaub.hat-sigma.b.std.err
    supsigmaub <- sigmaub.hat+sigma.b.std.err
    testsigmaub <- sigmaub.hat-sigma.b.std.err<0
                   & sigmaub.hat+sigma.b.std.err>0

    infdif <- (sigmaua.hat-sigmaub.hat)-sigma.ab.std.err
    supdif <- (sigmaua.hat-sigmaub.hat)+sigma.ab.std.err
    testdif <- (sigmaua.hat-sigmaub.hat)-sigma.ab.std.err<0
               & (sigmaua.hat-sigmaub.hat)+sigma.ab.std.err>0


    return( list(sigma.a.std.err, sigma.b.std.err,
    sigma.ab.std.err, beta.std.err,
    infbeta, supbeta, testbeta,
    infsigmaua, supsigmaua, testsigmaua,
    infsigmaub, supsigmaub, testsigmaub,
    infdif, supdif, testdif) )
}



pvalueBeta <- function(beta0.hat, Fisher) {

    z <- abs(beta0.hat)/sqrt(as.vector(diag(Fisher[[5]])))
    p.beta <- pnorm(z, lower.tail=F)

    return( 2*p.beta )
}
```

18.2 R code for the partitioned Fay-Herriot model 2

18.2.1 R code of H3area

The R code of the function **H3area** is listed bellow.

```
#######################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 2
### Author: Agustin Perez Martin
### File name: H3.R
### Updated: November 25th, 2009
###
#######################################################################

H3area <- function(X, ydt, D, md, sigma2edt) {

    p <- ncol(X)
    a <- list(1:md[1])
    mdcum <- cumsum(md)
    M <- sum(md)

    for(d in 2:D)
        a[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[a[[d]]]
        Xd[[d]] <- X[a[[d]],]
    }

    Vd.inv <- VinvXd <- list()
    Q2.inv <- XV2X <- matrix(0, nrow=p, ncol=p)
    yVX <- 0
    for(d in 1:D) {

        ### Elements of the variance matrix
        vd <- sigma2edt[a[[d]]]

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- diag(1/vd)

        ### Product between V^-1_ed  and  X_d for all d submatrices
        VinvXd[[d]] <- Vd.inv[[d]]%*%Xd[[d]]

        ### Inverse of Q2. Next we calculate Q2
        Q2.inv <- Q2.inv + t(Xd[[d]])%*%VinvXd[[d]]

        ### Sum in d of the product with  y^t_d  and  V^-1_ed  and  X_d
        yVX <- yVX + yd[[d]]%*%VinvXd[[d]]
    }
    Q2 <- solve(Q2.inv)

    tr.XV2XQ2 <- 0
    for(d in 1:D)
        tr.XV2XQ2 <- tr.XV2XQ2
                    + sum(diag( t(VinvXd[[d]])%*%VinvXd[[d]]%*%Q2))

    tr.P2 <- sum(1/sigma2edt) - tr.XV2XQ2
    yP2y <- sum(ydt^2/sigma2edt) - yVX%*%Q2%*%t(yVX)

    sigma.u <- (yP2y - (M-p))/tr.P2

    return(as.vector(sigma.u))
}
```

18.2.2 R code of REMLarea.corr

The R code of the function **REMLarea.corr** is listed bellow.

```
#######################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 2
### Author: Maria Chiara Pagliarella
### File name: REMLcorr.R
### Updated: June 2010
###
#######################################################################

REMLarea.corr <- function(X, ydt, D, Da, Db, md, mda, mdb, sigma2edt,
                 sigma.0 = sigma.0, MAXITER = 100) {

    sigma.fa <- sigma.0
    sigma.fb <- 0.8 # sigma.0
```

```
rho <- 0.5
theta.f  <- as.vector(c(sigma.fa, sigma.fb, rho))

p <- ncol(X)
i <- list(1:md[1])
mdcum <- cumsum(md)
Db <- D-Da

for(d in 2:D){
    i[[d]] <- (mdcum[d-1]+1):mdcum[d]
    if (d<= Da) ia <- i[1:Da]
    else ib <- i[(Da+1):D]
}

yda <- Xda <- list()
for(d in 1:Da) {
    yda[[d]] <- ydt[ia[[d]]]
    Xda[[d]] <- X[ia[[d]],]
}
ydb <- Xdb <- list()
for(d in 1:Db) {
    ydb[[d]] <- ydt[ib[[d]]]
    Xdb[[d]] <- X[ib[[d]],]
}
yd <- Xd <- list()
for(d in 1:D) {
    yd[[d]] <- ydt[i[[d]]]
    Xd[[d]] <- X[i[[d]],]
}

Bad <- Flag <- 0

for(ITER in 1:MAXITER) {

    V1 <- Vda.inv <- VinvXa <- Vinvya <- VinvV1 <- XVinvV1VinvX
    <- VinvV1VinvV1 <- XVinvV1VinvV1VinvX <- list()
    tr.VinvV1 <- yVinvXa <- yVinvV1Vinvy <- yVinvV1VinvX
    <- SumXVinvV1VinvX <- tr.VinvV1VinvV1 <- 0

    V3.a <- VinvV1VinvV3.a <- XVinvV1VinvV3VinvX.a <- list()
    tr.VinvV1VinvV3.a <- 0

    for(d in 1:Da) {

        ### Matrix Omega and its derivatives ------------- A
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <- rho^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
```

```
Omega.a <- (1/(1-rho^2)) * Omega.a
V1[[d]] <- Omega.a

#print(V1)

OmegaFirst.a <- matrix(0,nrow=mda[d],ncol=mda[d])
OmegaFirst.a[lower.tri(OmegaFirst.a)]
<- sequence((mda[d]-1):1)*rho^(sequence((mda[d]-1):1)-1)
OmegaFirst.a <- OmegaFirst.a + t(OmegaFirst.a)
OmegaFirst.a <- (1/(1-rho^2)) * OmegaFirst.a
OmegaFirst.a <- OmegaFirst.a + (2*rho / (1-rho^2)) * Omega.a
V3.a[[d]] <- sigma.fa * OmegaFirst.a


### Elements of the variance matrix
Vda <- (sigma.fa * Omega.a + diag(sigma2edt[ia[[d]]]))

if (abs(det(Vda))<0.000000001 || abs(det(Vda))>10000000000) {
    Flag <- 1
    Bad <- Bad+1
    break }


### Inverse of variance matrix
Vda.inv[[d]] <- solve(Vda)

### Product between V^-1_da and X_da
### for all d submatrices
VinvXa[[d]] <- Vda.inv[[d]] %*% Xda[[d]]

### Product between V^-1_da  and  y_da
### for all d submatrices
Vinvya[[d]] <- Vda.inv[[d]] %*% yda[[d]]


# calculation of the elements function of V1
# derivatives of V with respect to sigma^2_A

# S1

VinvV1[[d]] <- Vda.inv[[d]] %*% V1[[d]]
tr.VinvV1 <- tr.VinvV1 + sum(diag(VinvV1[[d]]))

XVinvV1VinvX[[d]] <- t(VinvXa[[d]]) %*% V1[[d]]
                     %*% VinvXa[[d]]
SumXVinvV1VinvX <- SumXVinvV1VinvX + XVinvV1VinvX[[d]]

yVinvXa <- yVinvXa + t(yda[[d]]) %*% VinvXa[[d]]
yVinvV1Vinvy <- yVinvV1Vinvy + t(Vinvya[[d]])
```

```
                    %*% V1[[d]] %*% Vinvya[[d]]
    yVinvV1VinvX <- yVinvV1VinvX + t(Vinvya[[d]])
                    %*% V1[[d]] %*% VinvXa[[d]]

    # F11

    VinvV1VinvV1[[d]] <- Vda.inv[[d]] %*% V1[[d]]
             %*% Vda.inv[[d]] %*% V1[[d]]
    tr.VinvV1VinvV1 <- tr.VinvV1VinvV1
             + sum(diag(VinvV1VinvV1[[d]]))

    XVinvV1VinvV1VinvX[[d]] <- t(VinvXa[[d]])
             %*% V1[[d]] %*% Vda.inv[[d]]
             %*% V1[[d]] %*% VinvXa[[d]]


    # calculation of the elements function of V3_A
    # derivatives of V_a with respect to rho

    # F13.a

    VinvV1VinvV3.a[[d]] <- Vda.inv[[d]] %*% V1[[d]]
             %*% Vda.inv[[d]] %*% V3.a[[d]]
    tr.VinvV1VinvV3.a <- tr.VinvV1VinvV3.a
             + sum(diag(VinvV1VinvV3.a[[d]]))

    XVinvV1VinvV3VinvX.a[[d]] <- t(VinvXa[[d]]) %*% V1[[d]]
             %*% Vda.inv[[d]] %*% V3.a[[d]] %*% VinvXa[[d]]

}

if (Flag==1) {
    ITER <- MAXITER
    Flag <- 0
    break }


V2 <- Vdb.inv <- VinvXb <- Vinvyb <- VinvV2 <- XVinvV2VinvX
<- VinvV2VinvV2 <- XVinvV2VinvV2VinvX <- list()
tr.VinvV2 <- yVinvXb <- yVinvV2Vinvy <- yVinvV2VinvX
<- SumXVinvV2VinvX <- tr.VinvV2VinvV2 <- 0

V3.b <- VinvV2VinvV3.b <- XVinvV2VinvV3VinvX.b <- list()
tr.VinvV2VinvV3.b <- 0

for(d in 1:Db) {

    ### Matrix Omega and its derivatives ------------- B
    Omega.b <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    Omega.b[lower.tri(Omega.b)] <- rho^sequence((mdb[d]-1):1)
    Omega.b <- Omega.b + t(Omega.b)
    diag(Omega.b) <- 1
    Omega.b <- (1/(1-rho^2)) * Omega.b
    V2[[d]] <- Omega.b

    OmegaFirst.b <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    OmegaFirst.b[lower.tri(OmegaFirst.b)]
    <- sequence((mdb[d]-1):1)*rho^(sequence((mdb[d]-1):1)-1)
    OmegaFirst.b <- OmegaFirst.b + t(OmegaFirst.b)
    OmegaFirst.b <- (1/(1-rho^2)) * OmegaFirst.b
    OmegaFirst.b <- OmegaFirst.b + (2*rho / (1-rho^2)) * Omega.b
    V3.b[[d]] <- sigma.fb * OmegaFirst.b


    ### Elements of the variance matrix
    Vdb <- (sigma.fb * Omega.b + diag(sigma2edt[ib[[d]]]))

    #print(Vdb)
    #print(sigma2edt[ib[[d]]])
    #print(sigma.fb)
    #print(Omega.b)

    if (abs(det(Vdb))<0.000000001 || abs(det(Vdb))>10000000000) {
        Flag <- 1
        Bad <- Bad + 1
        break }

    ### Inverse of variance matrix
    Vdb.inv[[d]] <- solve(Vdb)

    ### Product between V^-1_db and X_db
    ### for all d submatrices
    VinvXb[[d]] <- Vdb.inv[[d]] %*% Xdb[[d]]

    ### Product between V^-1_db  and  y_db
    ### for all d submatrices
    Vinvyb[[d]] <- Vdb.inv[[d]] %*% ydb[[d]]

    # calculation of the elements function of V2
    # derivatives of V with respect to sigma^2_B

    # S2

    VinvV2[[d]] <- Vdb.inv[[d]] %*% V2[[d]]
    tr.VinvV2 <- tr.VinvV2 + sum(diag(VinvV2[[d]]))

    XVinvV2VinvX[[d]] <- t(VinvXb[[d]]) %*% V2[[d]]
                         %*% VinvXb[[d]]
```

```
        SumXVinvV2VinvX <- SumXVinvV2VinvX + XVinvV2VinvX[[d]]

        yVinvXb <- yVinvXb + t(ydb[[d]]) %*% VinvXb[[d]]
        yVinvV2Vinvy <- yVinvV2Vinvy + t(Vinvyb[[d]])
                        %*% V2[[d]] %*% Vinvyb[[d]]
        yVinvV2VinvX <- yVinvV2VinvX + t(Vinvyb[[d]])
                        %*% V2[[d]] %*% VinvXb[[d]]

        # F22

        VinvV2VinvV2[[d]] <- Vdb.inv[[d]] %*% V2[[d]]
                             %*% Vdb.inv[[d]] %*% V2[[d]]
        tr.VinvV2VinvV2 <- tr.VinvV2VinvV2
                           + sum(diag(VinvV2VinvV2[[d]]))

        XVinvV2VinvV2VinvX[[d]] <- t(VinvXb[[d]]) %*% V2[[d]]
                 %*% Vdb.inv[[d]] %*% V2[[d]] %*% VinvXb[[d]]

        # calculation of the elements function of V3_B
        # derivatives of V_B with respect to rho

        # F23.b

        VinvV2VinvV3.b[[d]] <- Vdb.inv[[d]]
                 %*% V2[[d]] %*% Vdb.inv[[d]] %*% V3.b[[d]]
        tr.VinvV2VinvV3.b <- tr.VinvV2VinvV3.b
                 + sum(diag(VinvV2VinvV3.b[[d]]))

        XVinvV2VinvV3VinvX.b[[d]] <- t(VinvXb[[d]])
                 %*% V2[[d]] %*% Vdb.inv[[d]]
                 %*% V3.b[[d]] %*% VinvXb[[d]]

    }

    if (Flag==1) {
        ITER <- MAXITER
        Flag <- 0
        break }


    V3 <- Vd.inv <- VinvX <- Vinvy <- VinvV3 <- XVinvV3VinvX
    <- VinvV3VinvV3 <- XVinvV3VinvV3VinvX <- list()
    tr.VinvV3 <- SumXVinvV3VinvX <- yVinvX <- yVinvV3Vinvy
    <- yVinvV3VinvX <- tr.VinvV3VinvV3 <- 0
    Q.inv <- matrix(0, nrow=p, ncol=p)
    for(d in 1:D) {

        if (d <= Da){
            ### Matrix Omega and its derivatives ------------- A
            OmegaFirst.a <- matrix(0,nrow=mda[d],ncol=mda[d])
            OmegaFirst.a[lower.tri(OmegaFirst.a)]
            <- sequence((mda[d]-1):1)*rho^(sequence((mda[d]-1):1)-1)
            OmegaFirst.a <- OmegaFirst.a + t(OmegaFirst.a)
            OmegaFirst.a <- (1/(1-rho^2)) * OmegaFirst.a
            OmegaFirst.a <- OmegaFirst.a
                            + (2*rho / (1-rho^2)) * Omega.a
            V3[[d]] <- sigma.fa * OmegaFirst.a}
        if (d > Da){
            ### Matrix Omega and its derivatives ------------- B
            OmegaFirst.b <- matrix(0,nrow=md[d],ncol=md[d])
            OmegaFirst.b[lower.tri(OmegaFirst.b)]
            <- sequence((md[d]-1):1)*rho^(sequence((md[d]-1):1)-1)
            OmegaFirst.b <- OmegaFirst.b + t(OmegaFirst.b)
            OmegaFirst.b <- (1/(1-rho^2)) * OmegaFirst.b
            OmegaFirst.b <- OmegaFirst.b
                            + (2*rho / (1-rho^2)) * Omega.b
            V3[[d]] <- sigma.fb * OmegaFirst.b}

        if (d <= Da)
            vd <-(sigma.fa * Omega.a + diag(sigma2edt[ia[[d]]]))
        else
            vd <-(sigma.fb * Omega.b + diag(sigma2edt[i[[d]]]))


        if (abs(det(vd))<0.000000001 || abs(det(vd))>10000000000) {
            Flag <- 1
            Bad <- Bad + 1
            break }

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- solve(vd)


        ### Product between V^-1_d and X_d
        ### for all d submatrices
        VinvX[[d]] <- Vd.inv[[d]] %*% Xd[[d]]

        ### Product between V^-1_d and  y_d
        ### for all d submatrices
        Vinvy[[d]] <- Vd.inv[[d]] %*% yd[[d]]

        ### Inverse of Q. Next we calculate Q
        Q.inv <- Q.inv + t(Xd[[d]]) %*% VinvX[[d]]


        # calculation of the elements function of V3
        # derivatives of V (total) with respect to rho
        # S3

        VinvV3[[d]] <- Vd.inv[[d]] %*% V3[[d]]
        tr.VinvV3 <- tr.VinvV3 + sum(diag(VinvV3[[d]]))

        XVinvV3VinvX[[d]] <- t(VinvX[[d]]) %*% V3[[d]]
                             %*% VinvX[[d]]
        SumXVinvV3VinvX <- SumXVinvV3VinvX + XVinvV3VinvX[[d]]

        yVinvX <- yVinvX + t(yd[[d]]) %*% VinvX[[d]]
        yVinvV3Vinvy <- yVinvV3Vinvy + t(Vinvy[[d]])
                        %*% V3[[d]] %*% Vinvy[[d]]
        yVinvV3VinvX <- yVinvV3VinvX + t(Vinvy[[d]])
                        %*% V3[[d]] %*% VinvX[[d]]

        # F33

        VinvV3VinvV3[[d]] <- Vd.inv[[d]] %*% V3[[d]]
                             %*% Vd.inv[[d]] %*% V3[[d]]
        tr.VinvV3VinvV3 <- tr.VinvV3VinvV3
                           + sum(diag(VinvV3VinvV3[[d]]))

        XVinvV3VinvV3VinvX[[d]] <- t(VinvX[[d]]) %*% V3[[d]]
                 %*% Vd.inv[[d]] %*% V3[[d]] %*% VinvX[[d]]

    }

    if (Flag==1) {
        ITER <- MAXITER
        Flag <- 0
        break }

    ### Calculation of Q
    Q <- solve(Q.inv)


    tr.XVinvV1VinvXQ <- tr.XVinvV1VinvV1VinvXQ <- XVinvV1VinvXQ
    <- tr.XVinvV1VinvV3VinvX.aQ <- 0
    for(d in 1:Da){
    tr.XVinvV1VinvXQ <- tr.XVinvV1VinvXQ
                        + sum(diag(XVinvV1VinvX[[d]] %*% Q))
    tr.XVinvV1VinvV1VinvXQ <- tr.XVinvV1VinvV1VinvXQ
                 + sum(diag(XVinvV1VinvV1VinvX[[d]] %*% Q))
    XVinvV1VinvXQ <- XVinvV1VinvXQ + XVinvV1VinvX[[d]] %*% Q
    tr.XVinvV1VinvV3VinvX.aQ <- tr.XVinvV1VinvV3VinvX.aQ
                 + sum(diag(XVinvV1VinvV3VinvX.a[[d]] %*% Q))
    }

    tr.XVinvV2VinvXQ <- tr.XVinvV2VinvV2VinvXQ <- XVinvV2VinvXQ
```

```
<- tr.XVinvV2VinvV3VinvX.bQ <- 0
for(d in 1:Db){
tr.XVinvV2VinvXQ <- tr.XVinvV2VinvXQ
             + sum(diag(XVinvV2VinvX[[d]] %*% Q))
tr.XVinvV2VinvV2VinvXQ <- tr.XVinvV2VinvV2VinvXQ
             + sum(diag(XVinvV2VinvV2VinvX[[d]] %*% Q))
XVinvV2VinvXQ <- XVinvV2VinvXQ + XVinvV2VinvX[[d]] %*% Q
tr.XVinvV2VinvV3VinvX.bQ <- tr.XVinvV2VinvV3VinvX.bQ
             + sum(diag(XVinvV2VinvV3VinvX.b[[d]] %*% Q))
}

tr.XVinvV3VinvXQ <- tr.XVinvV3VinvV3VinvXQ
<- XVinvV3VinvXQ <- 0
for(d in 1:D){
    tr.XVinvV3VinvXQ <- tr.XVinvV3VinvXQ
             + sum(diag(XVinvV3VinvX[[d]] %*% Q))
    tr.XVinvV3VinvV3VinvXQ <- tr.XVinvV3VinvV3VinvXQ
             + sum(diag(XVinvV3VinvV3VinvX[[d]] %*% Q))
    XVinvV3VinvXQ <- XVinvV3VinvXQ + XVinvV3VinvX[[d]] %*% Q
}


# Calculation of PV1, and yPV1Py     --------- A

tr.XVinvV1VinvXQXVinvV1VinvXQ <- sum(diag(XVinvV1VinvXQ
                                 %*% XVinvV1VinvXQ))
tr.XVinvV1VinvXQXVinvV2VinvXQ <- sum(diag(XVinvV1VinvXQ
                                 %*% XVinvV2VinvXQ))
tr.XVinvV1VinvXQXVinvV3VinvXQ <- sum(diag(XVinvV1VinvXQ
                                 %*% XVinvV3VinvXQ))


tr.PV1 <- tr.VinvV1 - tr.XVinvV1VinvXQ
tr.PV1PV1 <- tr.VinvV1VinvV1 - 2 * tr.XVinvV1VinvV1VinvXQ
             + tr.XVinvV1VinvXQXVinvV1VinvXQ
tr.PV1PV2 <- tr.XVinvV1VinvXQXVinvV2VinvXQ
tr.PV1PV3 <- tr.VinvV1VinvV3.a - 2 * tr.XVinvV1VinvV3VinvX.aQ
             + tr.XVinvV1VinvXQXVinvV3VinvXQ

yPV1Py <- yVinvV1Vinvy - yVinvV1VinvX %*% Q %*% t(yVinvXa)
          - yVinvXa %*% Q %*% t(yVinvV1VinvX) + yVinvXa
          %*% Q %*% SumXVinvV1VinvX %*% Q %*% t(yVinvXa)


# Calculation of PV2 and yPV2Py      --------- B

tr.XVinvV2VinvXQXVinvV2VinvXQ <- sum(diag(XVinvV2VinvXQ
                                 %*% XVinvV2VinvXQ))
tr.XVinvV2VinvXQXVinvV1VinvXQ <- sum(diag(XVinvV2VinvXQ
                                  %*% XVinvV1VinvXQ))
tr.XVinvV2VinvXQXVinvV3VinvXQ <- sum(diag(XVinvV2VinvXQ
                                  %*% XVinvV3VinvXQ))

tr.PV2 <- tr.VinvV2 - tr.XVinvV2VinvXQ
tr.PV2PV2 <- tr.VinvV2VinvV2 - 2 * tr.XVinvV2VinvV2VinvXQ
             + tr.XVinvV2VinvXQXVinvV2VinvXQ
tr.PV2PV1 <- tr.XVinvV2VinvXQXVinvV1VinvXQ
tr.PV2PV3 <- tr.VinvV2VinvV3.b - 2 * tr.XVinvV2VinvV3VinvX.bQ
             + tr.XVinvV2VinvXQXVinvV3VinvXQ

yPV2Py <- yVinvV2Vinvy - yVinvV2VinvX %*% Q %*% t(yVinvXb)
          - yVinvXb %*% Q %*% t(yVinvV2VinvX) + yVinvXb
          %*% Q %*% SumXVinvV2VinvX %*% Q %*% t(yVinvXb)


# Calculation of PV3 and yPV3Py      --------- TOTAL A and B

tr.XVinvV3VinvXQXVinvV3VinvXQ <- sum(diag(XVinvV3VinvXQ
                                  %*% XVinvV3VinvXQ))

tr.PV3 <- tr.VinvV3 - tr.XVinvV3VinvXQ
tr.PV3PV3 <- tr.VinvV3VinvV3 - 2 * tr.XVinvV3VinvV3VinvXQ
             + tr.XVinvV3VinvXQXVinvV3VinvXQ

yPV3Py <- yVinvV3Vinvy - yVinvV3VinvX %*% Q %*% t(yVinvX)
          - yVinvX %*% Q %*% t(yVinvV3VinvX) + yVinvX
          %*% Q %*% SumXVinvV3VinvX %*% Q %*% t(yVinvX)



# Scores and elements of Fisher information Matrix
# F11, F22, F12, F33, F44, F34  for A / B

S1 <- -0.5 * tr.PV1 + 0.5 * yPV1Py
S2 <- -0.5 * tr.PV2 + 0.5 * yPV2Py
S3 <- -0.5 * tr.PV3 + 0.5 * yPV3Py

Ssig <- c(S1, S2, S3)


F11  <- 0.5 * tr.PV1PV1
F22  <- 0.5 * tr.PV2PV2
F12  <- 0.5 * tr.PV1PV2

F13  <- 0.5 * tr.PV1PV3
F23  <- 0.5 * tr.PV2PV3
F33  <- 0.5 * tr.PV3PV3
```

```
        # FINAL Fisher information Matrix
        Fsig <- matrix(c(F11, F12, F13, F12, F22,
                         F23, F13, F23, F33),ncol=3)
        SumFsig <- abs(sum(Fsig))
        # print(det(Fsig))
        if (abs(det(Fsig))<0.000000001 || SumFsig > 10000000) {
            # print(ITER)
            ITER <- MAXITER
            Bad <- Bad + 1
            break }

        # print(ITER)

        Fsig.inv <- solve(Fsig)

        # Fisher-Scoring Algorithm
        dif <- Fsig.inv %*% Ssig

        #################################################################

        theta.f <- theta.f + dif

        sigma.fa <- theta.f[1,1]
        sigma.fb <- theta.f[2,1]
        rho <- theta.f[3,1]

        # Stopping criterion
        # print(theta.f)

        if(abs(dif[1,1])<0.00001 && abs(dif[2,1])<0.00001
           && abs(dif[3,1])<0.00001)
        break

        # print(Q)
        # print(sigma.fa)
        # print(sigma.fb)
        # print(rho)
        # results3 <- data.frame(theta.f, Fsig)
        # results3 <- as.data.frame(t(results3))
        # write.table(results3, file="REML.txt", sep="\t")

        # x<-"ITER greater than 49"
        # z<-"ITER smaller than 49"
        # if (ITER>49) print(x) else print(z)

    }

    if (sigma.fa < 0 || sigma.fb < 0 || rho < -1 || rho > 1 ) {
        ITER <- MAXITER
        Bad <- Bad + 1 }

    return(list(as.vector(theta.f), Fsig, ITER, Bad))
}
```

18.2.3 R code of BETA.U.area.corr

The R code of the function **BETA.U.area.corr** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 2
### Author: Maria Chiara Pagliarella
### File name: EstimationBETAcorr.R
### Updated: July 2010
###
########################################################################

BETA.U.area.corr <- function(X, ydt, D, Da, Db, md, mda, mdb,
sigma2edt, sigmaua, sigmaub, rho) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
    Db <- D-Da

    for(d in 2:D)
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]

    ia <- i[1:Da]
    ib <- i[(Da+1):D]


    yda <- Xda <- list()
    for(d in 1:Da) {
        yda[[d]] <- ydt[ia[[d]]]
        Xda[[d]] <- X[ia[[d]],]
    }
    ydb <- Xdb <- list()
    for(d in 1:Db) {
        ydb[[d]] <- ydt[ib[[d]]]
        Xdb[[d]] <- X[ib[[d]],]
    }
    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[i[[d]]]
        Xd[[d]] <- X[i[[d]],]
    }
```

```
Vd.inv <- list()
Q.inv <- matrix(0, nrow=p, ncol=p)
XVy <-0

for (d in 1:D) {
    ## Elements of the variance matrix TOTAL (A & B)
    if (d <= Da) {
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <- rho^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
        Omega.a <- (1/(1-rho^2)) * Omega.a
        vd <-(sigmaua * Omega.a + diag(sigma2edt[ia[[d]]]))}
    else {
        Omega.b <- matrix(0,nrow=md[d],ncol=md[d])
        Omega.b[lower.tri(Omega.b)] <- rho^sequence((md[d]-1):1)
        Omega.b <- Omega.b + t(Omega.b)
        diag(Omega.b) <- 1
        Omega.b <- (1/(1-rho^2)) * Omega.b
        vd <-(sigmaub * Omega.b + diag(sigma2edt[i[[d]]]))}

    ### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- solve(vd)

    ### Product between X^t_d,  V^-1_d and y_d for all d submatrices
    XVy <- XVy + t(Xd[[d]]) %*% Vd.inv[[d]] %*% yd[[d]]

    ### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]]) %*% Vd.inv[[d]] %*% Xd[[d]]
}
# print(Omega.a)
# print(Omega.b)
# print(vd)

Q <- solve(Q.inv)

beta <- Q %*% XVy


ua <- ub <- list()
for(d in 1:Da){
    ua[[d]] <- sigmaua * Omega.a %*% Vd.inv[[d]]
                  %*% (yda[[d]] - Xda[[d]] %*% beta)
}
for(d in 1:Db){
    ub[[d]] <- sigmaub * Omega.b %*% Vd.inv[[d]]
                  %*% (ydb[[d]] - Xdb[[d]] %*% beta)
}
```

```
    ua<-as.matrix(unlist(ua))
    ub<-as.matrix(unlist(ub))
    u <- c(ua,ub)

    return(c(beta,u))
}
```

18.2.4 R code of mse.area.corr

The R code of the function **mse.area.corr** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 2
### Author: Maria Chiara Pagliarella
### File name: EstimationMSEcorr.R
### Updated: July 2010
###
########################################################################

mse.area.corr <- function(X, D, Da, Db, md, mda, mdb, sigma2edt,
            sigmaua, sigmaub, rho, F11, F22, F33, F13, F23) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
    Db <- D-Da

    for(d in 2:D){
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]
        if (d<= Da) ia <- i[1:Da]
        else ib <- i[(Da+1):D]
    }


    Xda <- list()
    for(d in 1:Da) {
        Xda[[d]] <- X[ia[[d]],]
    }
    Xdb <- list()
    for(d in 1:Db) {
        Xdb[[d]] <- X[ib[[d]],]
    }
    Xd <- list()
    for(d in 1:D) {
        Xd[[d]] <- X[i[[d]],]
    }
```

```
########################################################
###       Calculation of elements of g1, g2 and g3 FOR A

Omega.a <- OmegaFirst.a <- Vda.inv <- VinvOmega.a <-
VinvOmegaFirst.a <- OmegaVinvOmega.a <- OmegaVinvOmegaFirst.a <-
OmegaFirstVinvOmegaFirst.a <- g1.a <- Sinv.a <- SinvXda <-
OmegaVinvOmegaSinvXda <- g2.1a <- q11 <- q13 <- q33a <- list()
for(d in 1:Da) {

    ### Elements of the variance matrix
    Omega.a[[d]] <- matrix(0,nrow=mda[d],ncol=mda[d])
    Omega.a[[d]][lower.tri(Omega.a[[d]])] <- rho^sequence((mda[d]-1):1)
    Omega.a[[d]] <- Omega.a[[d]] + t(Omega.a[[d]])
    diag(Omega.a[[d]]) <- 1
    Omega.a[[d]] <- (1/(1-rho^2)) * Omega.a[[d]]

    vda <-(sigmaua * Omega.a[[d]] + diag(sigma2edt[ia[[d]]]))

    OmegaFirst.a[[d]] <- matrix(0,nrow=mda[d],ncol=mda[d])
    OmegaFirst.a[[d]][lower.tri(OmegaFirst.a[[d]])] <-
    sequence((mda[d]-1):1)*rho^(sequence((mda[d]-1):1)-1)
    OmegaFirst.a[[d]] <- OmegaFirst.a[[d]] + t(OmegaFirst.a[[d]])
    OmegaFirst.a[[d]] <- (1/(1-rho^2)) * OmegaFirst.a[[d]]
    OmegaFirst.a[[d]] <- OmegaFirst.a[[d]] +
                         (2*rho / (1-rho^2)) * Omega.a[[d]]

    ### Inverse matrix of the variance and submatrices
    Vda.inv[[d]] <- solve(vda)

    ### Product between V^-1_da and Omega_a
    VinvOmega.a[[d]] <- Vda.inv[[d]] %*% Omega.a[[d]]
    ### Product between V^-1_da and OmegaFirst_a
    VinvOmegaFirst.a[[d]] <- Vda.inv[[d]] %*% OmegaFirst.a[[d]]
    ### Product between Omega.a, V^-1_da and Omega.a
    OmegaVinvOmega.a[[d]] <- t(VinvOmega.a[[d]]) %*% Omega.a[[d]]
    ### Product of Omega_a with V^-1_da and OmegaFirst_a
    OmegaVinvOmegaFirst.a[[d]] <- Omega.a[[d]] %*% Vda.inv[[d]]
                                  %*% OmegaFirst.a [[d]]
    ### Product of OmegaFirst_a with V^-1_da and OmegaFirst_a
    OmegaFirstVinvOmegaFirst.a[[d]] <- OmegaFirst.a[[d]]
                 %*% Vda.inv[[d]] %*% OmegaFirst.a[[d]]


    g1.a[[d]] <- (sigmaua * Omega.a[[d]]) - ((sigmaua)^2 *
                 OmegaVinvOmega.a[[d]])     ### Calculation of g1_a

    ### Elements of the variance matrix Sigma_ed_a
    sed.a <- sigma2edt[ia[[d]]]
    ### Inverse matrix of Sigma_ed for all d submatrices_a
    Sinv.a[[d]] <- diag(1/sed.a)
    ### Product between Sigma_a^-1_ed and X_da for all d submatrices_a
    SinvXda[[d]] <- Sinv.a[[d]] %*% Xda[[d]]
    ### Product Omega.a with V^-1_da for Omega.a
    ### and Sigma^-1_ed for X_da for all submatrices
    OmegaVinvOmegaSinvXda[[d]] <- OmegaVinvOmega.a[[d]]
                                  %*% SinvXda[[d]]

    ### First part of g2_a (the second is its transpose)
    g2.1a[[d]] <- Xda[[d]] - sigmaua * Omega.a[[d]]
                  %*% SinvXda[[d]] + (sigmaua^2) *
                  OmegaVinvOmegaSinvXda[[d]]

    ### Elements q11, q22 and q12 for calcultion of g3_a
    q11[[d]] <- OmegaVinvOmega.a[[d]] - 2 * sigmaua *
                OmegaVinvOmega.a[[d]] %*% VinvOmega.a[[d]] +
                (sigmaua)^2 * OmegaVinvOmega.a[[d]] %*%
                Vda.inv[[d]] %*% OmegaVinvOmega.a[[d]]
    q13[[d]] <- sigmaua * OmegaVinvOmegaFirst.a[[d]] - (sigmaua)^2 *
                OmegaVinvOmegaFirst.a[[d]] %*% VinvOmega.a[[d]] -
                (sigmaua)^2 * OmegaVinvOmega.a[[d]] %*%
                VinvOmegaFirst.a[[d]] + (sigmaua)^3 *
                OmegaVinvOmega.a[[d]] %*% VinvOmegaFirst.a[[d]]
                %*% VinvOmega.a[[d]]
    q33a[[d]] <- (sigmaua)^2 * OmegaFirstVinvOmegaFirst.a[[d]] -
                 2 * (sigmaua)^3 * OmegaVinvOmegaFirst.a[[d]] %*%
                 VinvOmegaFirst.a[[d]] + (sigmaua)^4 *
                 OmegaVinvOmegaFirst.a[[d]] %*% Vda.inv[[d]] %*%
                 t(OmegaVinvOmegaFirst.a[[d]])
}


########################################################
###       Calculation of elements of g1, g2 and g3 FOR B


Omega.b <- OmegaFirst.b <- Vdb.inv <- VinvOmega.b <-
VinvOmegaFirst.b <- OmegaVinvOmega.b <- OmegaVinvOmegaFirst.b <-
OmegaFirstVinvOmegaFirst.b <- g1.b <- Sinv.b <- SinvXdb <-
OmegaVinvOmegaSinvXdb <- g2.1b <- q22 <- q23 <- q33b <- list()
for(d in 1:Db) {

    ### Elements of the variance matrix
    Omega.b[[d]] <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    Omega.b[[d]][lower.tri(Omega.b[[d]])] <- rho^sequence((mdb[d]-1):1)
    Omega.b[[d]] <- Omega.b[[d]] + t(Omega.b[[d]])
    diag(Omega.b[[d]]) <- 1
    Omega.b[[d]] <- (1/(1-rho^2)) * Omega.b[[d]]
```

```
vdb <-(sigmaub * Omega.b[[d]] + diag(sigma2edt[ib[[d]]]))

OmegaFirst.b[[d]] <- matrix(0,nrow=mdb[d],ncol=mdb[d])
OmegaFirst.b[[d]][lower.tri(OmegaFirst.b[[d]])] <-
             sequence((mdb[d]-1):1)*rho^(sequence((mdb[d]-1):1)-1)
OmegaFirst.b[[d]] <- OmegaFirst.b[[d]] + t(OmegaFirst.b[[d]])
OmegaFirst.b[[d]] <- (1/(1-rho^2)) * OmegaFirst.b[[d]]
OmegaFirst.b[[d]] <- OmegaFirst.b[[d]] +
                     (2*rho / (1-rho^2)) * Omega.b[[d]]


### Inverse matrix of the variance and submatrices
Vdb.inv[[d]] <- solve(vdb)

### Product between V^-1_db and Omega_b
VinvOmega.b[[d]] <- Vdb.inv[[d]] %*% Omega.b[[d]]
### Product between V^-1_db and OmegaFirst_b
VinvOmegaFirst.b[[d]] <- Vdb.inv[[d]] %*% OmegaFirst.b[[d]]
### Product between Omega.b, V^-1_db and Omega.b
OmegaVinvOmega.b[[d]] <- t(VinvOmega.b[[d]]) %*% Omega.b[[d]]
### Product of Omega_b with V^-1_db and OmegaFirst_b
OmegaVinvOmegaFirst.b[[d]] <- Omega.b[[d]] %*%
             Vdb.inv[[d]] %*% OmegaFirst.b [[d]]
### Product of OmegaFirst_b with V^-1_db and OmegaFirst_b
OmegaFirstVinvOmegaFirst.b[[d]] <- OmegaFirst.b[[d]] %*%
             Vdb.inv[[d]] %*% OmegaFirst.b[[d]]

### Calculation of g1_b
g1.b[[d]] <- (sigmaub * Omega.b[[d]]) - ((sigmaub)^2 *
             OmegaVinvOmega.b[[d]])

### Elements of the variance matrix Sigma_ed_b
sed.b <- diag(sigma2edt[ib[[d]]])
### Inverse matrix of Sigma_ed for all d submatrices_b
Sinv.b[[d]] <- solve(sed.b)
### Product between Sigma_b^-1_ed and X_db for all d submatrices_b
SinvXdb[[d]] <- Sinv.b[[d]] %*% Xdb[[d]]
### Product Omega.b with V^-1_db for Omega.b and Sigma^-1_ed
### for X_db for all submatrices
OmegaVinvOmegaSinvXdb[[d]] <- OmegaVinvOmega.b[[d]] %*% SinvXdb[[d]]

### First part of g2_b (the second is its transpose)
g2.1b[[d]] <- Xdb[[d]] - sigmaub * Omega.b[[d]] %*%
              SinvXdb[[d]] + (sigmaub^2) * OmegaVinvOmegaSinvXdb[[d]]

### Elements q33, q44 and q34 for calcultion of g3_b
q22[[d]] <- OmegaVinvOmega.b[[d]] - 2 * sigmaub *
            OmegaVinvOmega.b[[d]] %*% VinvOmega.b[[d]] +
            (sigmaub)^2 * OmegaVinvOmega.b[[d]] %*% Vdb.inv[[d]] %*%
```

```
                 OmegaVinvOmega.b[[d]]
    q23[[d]] <- sigmaub * OmegaVinvOmegaFirst.b[[d]] - (sigmaub)^2 *
                 OmegaVinvOmegaFirst.b[[d]] %*% VinvOmega.b[[d]] -
                (sigmaub)^2 * OmegaVinvOmega.b[[d]] %*%
                 VinvOmegaFirst.b[[d]] + (sigmaub)^3 *
                 OmegaVinvOmega.b[[d]] %*% VinvOmegaFirst.b[[d]] %*%
                 VinvOmega.b[[d]]
    q33b[[d]] <- (sigmaub)^2 * OmegaFirstVinvOmegaFirst.b[[d]] -
                 2 * (sigmaub)^3 * OmegaVinvOmegaFirst.b[[d]] %*%
                 VinvOmegaFirst.b[[d]] + (sigmaub)^4 *
                 OmegaVinvOmegaFirst.b[[d]] %*% Vdb.inv[[d]] %*%
                 t(OmegaVinvOmegaFirst.b[[d]])
}


# CALCULATION of Q

Vd.inv <- VinvX <- list()
Q.inv <- matrix(0, nrow=p, ncol=p)

for(d in 1:D) {

    ## Elements of the variance matrix TOTAL (A & B)
    if (d <= Da) {
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <- rho^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
        Omega.a <- (1/(1-rho^2)) * Omega.a
        vd <-(sigmaua * Omega.a + diag(sigma2edt[ia[[d]]]))}
    else {
        Omega.b <- matrix(0,nrow=md[d],ncol=md[d])
        Omega.b[lower.tri(Omega.b)] <- rho^sequence((md[d]-1):1)
        Omega.b <- Omega.b + t(Omega.b)
        diag(Omega.b) <- 1
        Omega.b <- (1/(1-rho^2)) * Omega.b
        vd <-(sigmaub * Omega.b + diag(sigma2edt[i[[d]]]))}

    ### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- solve(vd)

    ### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]]) %*% Vd.inv[[d]] %*% Xd[[d]]
}

Q <- solve(Q.inv)


# Calculation of MSE_A
```

```
Fsig.a <- matrix(c(F11, F13, F13, F33),ncol=2)
g1a <- g2a <- g3a <- list()
for(d in 1:Da){
    g1a[[d]] <- diag(g1.a[[d]])
    g2a[[d]] <- diag(g2.1a[[d]] %*% Q %*% t(g2.1a[[d]]))
    q11[[d]] <- diag(q11[[d]])
    q13[[d]] <- diag(q13[[d]])
    q33a[[d]] <- diag(q33a[[d]])
}
for(d in 1:Da){
g3a[[d]] <- vector()
    for(i in 1:mda[d]){
        g3a[[d]][i] <- sum(diag(matrix(c(q11[[d]][i],
                       rep(q13[[d]][i],2),q33a[[d]][i]),
                       nrow=2) %*% solve(Fsig.a)))
    }
}



# Calculation of MSE_B

Fsig.b <- matrix(c(F22, F23, F23, F33),ncol=2)
g1b <- g2b <- g3b <- list()
for(d in 1:Db){
    g1b[[d]] <- diag(g1.b[[d]])
    g2b[[d]] <- diag(g2.1b[[d]] %*% Q %*% t(g2.1b[[d]]))
    q22[[d]] <- diag(q22[[d]])
    q23[[d]] <- diag(q23[[d]])
    q33b[[d]] <- diag(q33b[[d]])
}
for(d in 1:Db){
g3b[[d]] <- vector()
    for(i in 1:mdb[d]){
        g3b[[d]][i] <- sum(diag(matrix(c(q22[[d]][i],
                       rep(q23[[d]][i],2),q33b[[d]][i]),
                       nrow=2) %*% solve(Fsig.b)))
    }
}


g1a <- unlist(g1a)
g1b <- unlist(g1b)
g2a <- unlist(g2a)
g2b <- unlist(g2b)
g3a <- unlist(g3a)
g3b <- unlist(g3b)
```

```
    # Calculation of MSE

    mse.a <- g1a + g2a + 2 * g3a
    mse.b <- g1b + g2b + 2 * g3b
    mse <- c(mse.a, mse.b)

    return(mse)
}
```

18.2.5 R code of Interval.corr

The R code of the function **Interval.corr** for the confidence intervals and p-values is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 2
### Author: Maria Chiara Pagliarella
### File name: IC.corr.R
### Updated: August 2010
###
########################################################################

Interval <- function(Fisher, conf=0.95) {

    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- solve(Fisher[[2]])

    sigma.a.std.err <- z*sqrt(Finv[1,1])
    sigma.b.std.err <- z*sqrt(Finv[2,2])
    sigma.ab.std.err <- z*sqrt(Finv[1,1] + Finv[2,2] - 2*Finv[1,2])

    rho.std.err <- z*sqrt(Finv[3,3])

    beta.std.err <- z*sqrt(as.vector(diag(Fisher[[5]])))

    infbeta <- beta0.hat - beta.std.err
    supbeta <- beta0.hat + beta.std.err
    testbeta <- beta0.hat - beta.std.err < 0
                & beta0.hat + beta.std.err > 0

    infsigmaua <- sigmaua.hat - sigma.a.std.err
    supsigmaua <- sigmaua.hat + sigma.a.std.err
    testsigmaua <- sigmaua.hat - sigma.a.std.err < 0
                   & sigmaua.hat + sigma.a.std.err > 0

    infsigmaub <- sigmaub.hat - sigma.b.std.err
    supsigmaub <- sigmaub.hat + sigma.b.std.err
    testsigmaub <- sigmaub.hat - sigma.b.std.err < 0
                   & sigmaub.hat + sigma.b.std.err > 0

    infdif.sigma <- (sigmaua.hat - sigmaub.hat)
                    - sigma.ab.std.err
    supdif.sigma <- (sigmaua.hat - sigmaub.hat)
                    + sigma.ab.std.err
    testdif.sigma <- (sigmaua.hat - sigmaub.hat)
                     - sigma.ab.std.err < 0
                     & (sigmaua.hat - sigmaub.hat)
                     + sigma.ab.std.err > 0

    infrho <- rho.hat - rho.std.err
    suprho <- rho.hat + rho.std.err
    testrho <- rho.hat - rho.std.err < 0
               & rho.hat + rho.std.err > 0

    return(list(
    sigma.a.std.err, sigma.b.std.err, sigma.ab.std.err,
    rho.std.err, beta.std.err,
    infbeta, supbeta, testbeta,
    infsigmaua, supsigmaua, testsigmaua,
    infsigmaub, supsigmaub, testsigmaub,
    infdif.sigma, supdif.sigma, testdif.sigma,
    infrho, suprho, testrho))
}

pvalueBeta.corr <- function(beta0.hat, Fisher) {
    z <- abs(beta0.hat) / sqrt(as.vector(diag(Fisher[[5]])))
    p.beta <- pnorm(z, lower.tail=F)

    return( 2*p.beta )
}
```

18.3 R code for the partitioned Fay-Herriot model 3

18.3.1 R code of H3area

The R code of the function **H3area** is listed bellow.

```
######################################################################
###   Area level Partitioned F-H model with correlated time effects
###                      Pagliarella model 3
### Author: Maria Chiara Pagliarella
### File name: H3.R
### Updated: May 2010
###
######################################################################
```

```
H3area <- function(X, ydt, D, md, sigma2edt) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)

    for(d in 2:D)
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]

    yd <- Xd <- list()

    for(d in 1:D) {
        yd[[d]] <- ydt[i[[d]]]
        Xd[[d]] <- X[i[[d]],]
    }

    Vd.inv <-  VinvX <- list()
    Q2.inv <- XV2X <- matrix(0, nrow=p, ncol=p)
    yVX <- 0

    for(d in 1:D) {

        ### Elements of the variance matrix
        vd <- sigma2edt[i[[d]]]

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- diag(1/vd)

        ### Product between V^-1_ed  and  X_d
        ### for all d submatrices
        VinvX[[d]] <- Vd.inv[[d]] %*% Xd[[d]]

        ### Inverse of Q2. Next we calculate Q2
        Q2.inv <- Q2.inv + t(Xd[[d]]) %*% VinvX[[d]]

        ### Sum in d of the product between  y^t_d, V^-1_d and X_d
        yVX <- yVX + yd[[d]] %*% VinvX[[d]]
    }

  Q2 <- solve(Q2.inv)
  tr.XV2XQ2 <- 0
  for(d in 1:D)
  tr.XV2XQ2 <- tr.XV2XQ2 + sum(diag( t(VinvX[[d]]) %*% VinvX[[d]] %*% Q2))
  tr.P2 <- sum(1/sigma2edt) - tr.XV2XQ2
  yP2y <- sum(ydt^2/sigma2edt) - yVX %*% Q2 %*% t(yVX)
  sigma.u <- (yP2y - (M-p))/tr.P2

  return(as.vector(sigma.u))
}
```

18.3.2 R code of REMLarea.2corr

The R code of the function **REMLarea.2corr** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                       Pagliarella model 3
### Author: Maria Chiara Pagliarella
### File name: REML2corr.R
### Updated: July 2010
###
########################################################################

REMLarea.2corr <- function(X, ydt, D, Da, Db, md, mda, mdb,
sigma2edt, sigma.0 = sigma.0, MAXITER = 100) {

    sigma.fa <- sigma.0
    sigma.fb <- 0.8 # sigma.0
    rho.fa <- 0.75
    rho.fb <- 0.25
    theta.f  <- as.vector(c(sigma.fa, rho.fa, sigma.fb, rho.fb))

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
    Db <- D-Da

    for(d in 2:D){
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]
        if (d<= Da) ia <- i[1:Da]
        else ib <- i[(Da+1):D]
    }

    yda <- Xda <- list()
    for(d in 1:Da) {
        yda[[d]] <- ydt[ia[[d]]]
        Xda[[d]] <- X[ia[[d]],]
    }
    ydb <- Xdb <- list()
    for(d in 1:Db) {
        ydb[[d]] <- ydt[ib[[d]]]
        Xdb[[d]] <- X[ib[[d]],]
    }
    yd <- Xd <- list()
    for(d in 1:D) {
        yd[[d]] <- ydt[i[[d]]]
        Xd[[d]] <- X[i[[d]],]
    }

    Bad <- Flag <- 0
```

```
for(ITER in 1:MAXITER) {

    V1 <- V2 <- Vda.inv <- VinvXa <- Vinvya <- VinvV1 <-
    XVinvV1VinvX <- VinvV2 <- XVinvV2VinvX <- VinvV1VinvV1 <-
    XVinvV1VinvV1VinvX <- VinvV2VinvV2 <- XVinvV2VinvV2VinvX <-
    VinvV1VinvV2 <- XVinvV1VinvV2VinvX <- list()
    tr.VinvV1 <- yVinvXa <- yVinvV1Vinvy <- yVinvV1VinvX <-
    SumXVinvV1VinvX <- tr.VinvV2 <- yVinvV2Vinvy <- yVinvV2VinvX <-
    SumXVinvV2VinvX <- tr.VinvV1VinvV1 <- tr.VinvV2VinvV2 <-
    tr.VinvV1VinvV2 <- 0

    for(d in 1:Da) {
        ### Matrix Omega and its derivatives ------------- A
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <- rho.fa^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
        Omega.a <- (1/(1-rho.fa^2)) * Omega.a
        V1[[d]] <- Omega.a

        ### Derivatives
        OmegaFirst.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        OmegaFirst.a[lower.tri(OmegaFirst.a)] <-
                 sequence((mda[d]-1):1)*rho.fa^(sequence((mda[d]-1):1)-1)
        OmegaFirst.a <- OmegaFirst.a + t(OmegaFirst.a)
        OmegaFirst.a <- (1/(1-rho.fa^2)) * OmegaFirst.a
        OmegaFirst.a <- OmegaFirst.a +
                        (2*rho.fa / (1-rho.fa^2)) * Omega.a
        V2[[d]] <- sigma.fa * OmegaFirst.a

        ### Elements of the variance matrix
        Vda <- (sigma.fa * Omega.a + diag(sigma2edt[ia[[d]]]))

        if (abs(det(Vda))<0.000000001 || abs(det(Vda))>10000000000) {
            Flag <- 1
            Bad <- Bad+1
            break }


        ### Inverse of variance matrix
        Vda.inv[[d]] <- solve(Vda)

        ### Product between V^-1_da and X_da
        ### for all d submatrices
        VinvXa[[d]] <- Vda.inv[[d]] %*% Xda[[d]]

        ### Product between V^-1_da  and  y_da
        ### for all d submatrices
```

```
Vinvya[[d]] <- Vda.inv[[d]] %*% yda[[d]]


# calculation of the elements function of V1
# derivatives of V with respect to sigma^2_A

# S1

VinvV1[[d]] <- Vda.inv[[d]] %*% V1[[d]]
tr.VinvV1 <- tr.VinvV1 + sum(diag(VinvV1[[d]]))

XVinvV1VinvX[[d]] <- t(VinvXa[[d]]) %*% V1[[d]]
                     %*% VinvXa[[d]]
SumXVinvV1VinvX <- SumXVinvV1VinvX + XVinvV1VinvX[[d]]

yVinvXa <- yVinvXa + t(yda[[d]]) %*% VinvXa[[d]]
yVinvV1Vinvy <- yVinvV1Vinvy + t(Vinvya[[d]]) %*%
                V1[[d]] %*% Vinvya[[d]]
yVinvV1VinvX <- yVinvV1VinvX + t(Vinvya[[d]]) %*%
                V1[[d]] %*% VinvXa[[d]]


# calculation of the elements function of V2
# derivatives of V with respect to rho_A

# S2

VinvV2[[d]] <- Vda.inv[[d]] %*% V2[[d]]
tr.VinvV2 <- tr.VinvV2 + sum(diag(VinvV2[[d]]))

XVinvV2VinvX[[d]] <- t(VinvXa[[d]]) %*% V2[[d]] %*% VinvXa[[d]]
SumXVinvV2VinvX <- SumXVinvV2VinvX + XVinvV2VinvX[[d]]

yVinvV2Vinvy <- yVinvV2Vinvy + t(Vinvya[[d]]) %*%
                V2[[d]] %*% Vinvya[[d]]
yVinvV2VinvX <- yVinvV2VinvX + t(Vinvya[[d]]) %*%
                V2[[d]] %*% VinvXa[[d]]


# F11

VinvV1VinvV1[[d]] <- Vda.inv[[d]] %*% V1[[d]] %*%
                     Vda.inv[[d]] %*% V1[[d]]
tr.VinvV1VinvV1 <- tr.VinvV1VinvV1 +
                   sum(diag(VinvV1VinvV1[[d]]))

XVinvV1VinvV1VinvX[[d]] <- t(VinvXa[[d]]) %*%
         V1[[d]] %*% Vda.inv[[d]] %*% V1[[d]] %*% VinvXa[[d]]
```

```
    # F22

    VinvV2VinvV2[[d]] <- Vda.inv[[d]] %*% V2[[d]] %*%
                           Vda.inv[[d]] %*% V2[[d]]
    tr.VinvV2VinvV2 <- tr.VinvV2VinvV2 +
                       sum(diag(VinvV2VinvV2[[d]]))

    XVinvV2VinvV2VinvX[[d]] <- t(VinvXa[[d]]) %*%
             V2[[d]] %*% Vda.inv[[d]] %*% V2[[d]] %*%
             VinvXa[[d]]

    # F12

    VinvV1VinvV2[[d]]  <- Vda.inv[[d]] %*% V1[[d]] %*%
                            Vda.inv[[d]] %*% V2[[d]]
    tr.VinvV1VinvV2 <- tr.VinvV1VinvV2 +
                       sum(diag(VinvV1VinvV2[[d]]))

    XVinvV1VinvV2VinvX[[d]] <- t(VinvXa[[d]]) %*% V1[[d]]
             %*% Vda.inv[[d]] %*% V2[[d]] %*% VinvXa[[d]]
}

if (Flag==1) {
    ITER <- MAXITER
    Flag <- 0
    break }

V3 <- V4 <- Vdb.inv <- VinvXb <- Vinvyb <- VinvV3 <-
XVinvV3VinvX <- VinvV4 <- XVinvV4VinvX <- VinvV3VinvV3 <-
XVinvV3VinvV3VinvX <- VinvV4VinvV4 <- XVinvV4VinvV4VinvX <-
VinvV3VinvV4 <- XVinvV3VinvV4VinvX <- list()
tr.VinvV3 <- yVinvXb <- yVinvV3Vinvy <- yVinvV3VinvX <-
SumXVinvV3VinvX <- tr.VinvV4 <- yVinvV4Vinvy <- yVinvV4VinvX <-
SumXVinvV4VinvX <- tr.VinvV3VinvV3 <- tr.VinvV4VinvV4 <-
tr.VinvV3VinvV4 <- 0

for(d in 1:Db) {
    ### Matrix Omega and its derivatives ------------- B
    Omega.b <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    Omega.b[lower.tri(Omega.b)] <- rho.fb^sequence((mdb[d]-1):1)
    Omega.b <- Omega.b + t(Omega.b)
    diag(Omega.b) <- 1
    Omega.b <- (1/(1-rho.fb^2)) * Omega.b
    V3[[d]] <- Omega.b

 ### Derivatives
 OmegaFirst.b <- matrix(0,nrow=mdb[d],ncol=mdb[d])
 OmegaFirst.b[lower.tri(OmegaFirst.b)] <-
          sequence((mdb[d]-1):1)*rho.fb^(sequence((mdb[d]-1):1)-1)
```

```
OmegaFirst.b <- OmegaFirst.b + t(OmegaFirst.b)
OmegaFirst.b <- (1/(1-rho.fb^2)) * OmegaFirst.b
OmegaFirst.b <- OmegaFirst.b + (2*rho.fb / (1-rho.fb^2))*Omega.b
V4[[d]] <- sigma.fb * OmegaFirst.b

   ### Elements of the variance matrix
   Vdb <- (sigma.fb*Omega.b + diag(sigma2edt[ib[[d]]]))
   #print(Vdb)
   #print(sigma2edt[ib[[d]]])
   #print(sigma.fb)
   #print(Omega.b)

   if (abs(det(Vdb))<0.000000001 || abs(det(Vdb))>10000000000) {
       Flag <- 1
       Bad <- Bad + 1
       break }

   ### Inverse of variance matrix
   Vdb.inv[[d]] <- solve(Vdb)

   ### Product between V^-1_db and X_db
   ### for all d submatrices
   VinvXb[[d]] <- Vdb.inv[[d]] %*% Xdb[[d]]

   ### Product between V^-1_db  and  y_db
   ### for all d submatrices
   Vinvyb[[d]] <- Vdb.inv[[d]] %*% ydb[[d]]

   # calculation of the elements function of V3
   # derivatives of V with respect to sigma^2_B

   # S3

   VinvV3[[d]] <- Vdb.inv[[d]] %*% V3[[d]]
   tr.VinvV3 <- tr.VinvV3 + sum(diag(VinvV3[[d]]))

   XVinvV3VinvX[[d]] <- t(VinvXb[[d]]) %*% V3[[d]]
                        %*% VinvXb[[d]]
   SumXVinvV3VinvX <- SumXVinvV3VinvX + XVinvV3VinvX[[d]]

   yVinvXb <- yVinvXb + t(ydb[[d]]) %*% VinvXb[[d]]
   yVinvV3Vinvy <- yVinvV3Vinvy + t(Vinvyb[[d]]) %*% V3[[d]]
                   %*% Vinvyb[[d]]
   yVinvV3VinvX <- yVinvV3VinvX + t(Vinvyb[[d]]) %*% V3[[d]]
                   %*% VinvXb[[d]]

   # calculation of the elements function of V4
   # derivatives of V with respect to rho_B
```

```
    # S4

    VinvV4[[d]] <- Vdb.inv[[d]] %*% V4[[d]]
    tr.VinvV4 <- tr.VinvV4 + sum(diag(VinvV4[[d]]))

    XVinvV4VinvX[[d]] <- t(VinvXb[[d]]) %*% V4[[d]]
                         %*% VinvXb[[d]]
    SumXVinvV4VinvX <- SumXVinvV4VinvX + XVinvV4VinvX[[d]]

    yVinvV4Vinvy <- yVinvV4Vinvy + t(Vinvyb[[d]]) %*% V4[[d]]
                    %*% Vinvyb[[d]]
    yVinvV4VinvX <- yVinvV4VinvX + t(Vinvyb[[d]]) %*% V4[[d]]
                    %*% VinvXb[[d]]

  # F33

  VinvV3VinvV3[[d]] <- Vdb.inv[[d]] %*% V3[[d]] %*%
                         Vdb.inv[[d]] %*% V3[[d]]
  tr.VinvV3VinvV3 <- tr.VinvV3VinvV3 + sum(diag(VinvV3VinvV3[[d]]))

  XVinvV3VinvV3VinvX[[d]] <- t(VinvXb[[d]]) %*% V3[[d]]
           %*% Vdb.inv[[d]] %*% V3[[d]] %*% VinvXb[[d]]

    # F44

    VinvV4VinvV4[[d]] <- Vdb.inv[[d]] %*% V4[[d]]
             %*% Vdb.inv[[d]] %*% V4[[d]]
    tr.VinvV4VinvV4 <- tr.VinvV4VinvV4 +
             sum(diag(VinvV4VinvV4[[d]]))

    XVinvV4VinvV4VinvX[[d]] <- t(VinvXb[[d]]) %*% V4[[d]] %*%
             Vdb.inv[[d]] %*% V4[[d]] %*% VinvXb[[d]]

    # F34

    VinvV3VinvV4[[d]] <- Vdb.inv[[d]] %*% V3[[d]]
                         %*% Vdb.inv[[d]] %*% V4[[d]]
    tr.VinvV3VinvV4 <- tr.VinvV3VinvV4 +
                       sum(diag(VinvV3VinvV4[[d]]))

    XVinvV3VinvV4VinvX[[d]] <- t(VinvXb[[d]]) %*% V3[[d]] %*%
              Vdb.inv[[d]] %*% V4[[d]] %*% VinvXb[[d]]
}

if (Flag==1) {
    ITER <- MAXITER
    Flag <- 0
    break }
```

```
# CALCULATION of Q
Vd.inv <- VinvX <- list()
Q.inv <- matrix(0, nrow=p, ncol=p)

for(d in 1:D) {
## Elements of the variance matrix TOTAL (A & B)
  if (d <= Da)
        vd <-(sigma.fa * Omega.a + diag(sigma2edt[ia[[d]]]))
  else
        vd <-(sigma.fb * Omega.b + diag(sigma2edt[i[[d]]]))

  if (abs(det(vd))<0.000000001 || abs(det(vd))>10000000000) {
        Flag <- 1
        Bad <- Bad + 1
        break }

    ### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- solve(vd)

    ### Product between V^-1_ed  and  X_d
    ### for all d submatrices
    VinvX[[d]] <- Vd.inv[[d]] %*% Xd[[d]]

    ### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]]) %*% VinvX[[d]]
}

if (Flag==1) {
    ITER <- MAXITER
    Flag <- 0
    break }

Q <- solve(Q.inv)

tr.XVinvV1VinvXQ <- tr.XVinvV2VinvXQ  <-
tr.XVinvV1VinvV1VinvXQ <- XVinvV1VinvXQ <-
tr.XVinvV2VinvV2VinvXQ <- XVinvV2VinvXQ <-
tr.XVinvV1VinvV2VinvXQ <- 0

for(d in 1:Da){
    tr.XVinvV1VinvXQ <- tr.XVinvV1VinvXQ +
             sum(diag(XVinvV1VinvX[[d]] %*% Q))
    tr.XVinvV2VinvXQ <- tr.XVinvV2VinvXQ +
             sum(diag(XVinvV2VinvX[[d]] %*% Q))
    tr.XVinvV1VinvV1VinvXQ <- tr.XVinvV1VinvV1VinvXQ +
             sum(diag(XVinvV1VinvV1VinvX[[d]] %*% Q))
    XVinvV1VinvXQ <- XVinvV1VinvXQ + XVinvV1VinvX[[d]] %*% Q
    tr.XVinvV2VinvV2VinvXQ <- tr.XVinvV2VinvV2VinvXQ +
             sum(diag(XVinvV2VinvV2VinvX[[d]] %*% Q))
    XVinvV2VinvXQ <- XVinvV2VinvXQ + XVinvV2VinvX[[d]] %*% Q
    tr.XVinvV1VinvV2VinvXQ <- tr.XVinvV1VinvV2VinvXQ +
             sum(diag(XVinvV1VinvV2VinvX[[d]] %*% Q))
}

tr.XVinvV3VinvXQ <- tr.XVinvV4VinvXQ  <-
tr.XVinvV3VinvV3VinvXQ <- XVinvV3VinvXQ <-
tr.XVinvV4VinvV4VinvXQ <- XVinvV4VinvXQ <-
tr.XVinvV3VinvV4VinvXQ <- 0

for(d in 1:Db){
    tr.XVinvV3VinvXQ <- tr.XVinvV3VinvXQ +
             sum(diag(XVinvV3VinvX[[d]] %*% Q))
    tr.XVinvV4VinvXQ <- tr.XVinvV4VinvXQ +
             sum(diag(XVinvV4VinvX[[d]] %*% Q))
    tr.XVinvV3VinvV3VinvXQ <- tr.XVinvV3VinvV3VinvXQ +
             sum(diag(XVinvV3VinvV3VinvX[[d]] %*% Q))
    XVinvV3VinvXQ <- XVinvV3VinvXQ + XVinvV3VinvX[[d]] %*% Q
    tr.XVinvV4VinvV4VinvXQ <- tr.XVinvV4VinvV4VinvXQ +
             sum(diag(XVinvV4VinvV4VinvX[[d]] %*% Q))
    XVinvV4VinvXQ <- XVinvV4VinvXQ + XVinvV4VinvX[[d]] %*% Q
    tr.XVinvV3VinvV4VinvXQ <- tr.XVinvV3VinvV4VinvXQ +
             sum(diag(XVinvV3VinvV4VinvX[[d]] %*% Q))
}

# Calculation of PV1, PV2, yPV1Py and yPV2Py --------- A

tr.XVinvV1VinvXQXVinvV1VinvXQ <-
             sum(diag(XVinvV1VinvXQ%*%XVinvV1VinvXQ))
tr.XVinvV2VinvXQXVinvV2VinvXQ <-
             sum(diag(XVinvV2VinvXQ%*%XVinvV2VinvXQ))
tr.XVinvV1VinvXQXVinvV2VinvXQ <-
             sum(diag(XVinvV1VinvXQ%*%XVinvV2VinvXQ))

tr.PV1 <- tr.VinvV1 - tr.XVinvV1VinvXQ
tr.PV2 <- tr.VinvV2 - tr.XVinvV2VinvXQ
tr.PV1PV1 <- tr.VinvV1VinvV1 - 2 * tr.XVinvV1VinvV1VinvXQ +
             tr.XVinvV1VinvXQXVinvV1VinvXQ
tr.PV2PV2 <- tr.VinvV2VinvV2 - 2 * tr.XVinvV2VinvV2VinvXQ +
             tr.XVinvV2VinvXQXVinvV2VinvXQ
tr.PV1PV2 <- tr.VinvV1VinvV2 - 2 * tr.XVinvV1VinvV2VinvXQ +
             tr.XVinvV1VinvXQXVinvV2VinvXQ

yPV1Py <- yVinvV1Vinvy - yVinvV1VinvX %*% Q %*% t(yVinvXa) -
        yVinvXa %*% Q %*% t(yVinvV1VinvX) + yVinvXa %*% Q %*%
        SumXVinvV1VinvX %*% Q %*% t(yVinvXa)

yPV2Py <- yVinvV2Vinvy - yVinvV2VinvX %*% Q %*% t(yVinvXa) -
```

```
            yVinvXa %*% Q %*% t(yVinvV2VinvX) + yVinvXa %*% Q %*%
            SumXVinvV2VinvX %*% Q %*% t(yVinvXa)

    # Calculation of PV3, PV4, yPV3Py and yPV4Py --------- B

    tr.XVinvV3VinvXQXVinvV3VinvXQ <-
                 sum(diag(XVinvV3VinvXQ %*% XVinvV3VinvXQ))
    tr.XVinvV4VinvXQXVinvV4VinvXQ <-
                 sum(diag(XVinvV4VinvXQ %*% XVinvV4VinvXQ))
    tr.XVinvV3VinvXQXVinvV4VinvXQ <-
                 sum(diag(XVinvV3VinvXQ %*% XVinvV4VinvXQ))

    tr.PV3 <- tr.VinvV3 - tr.XVinvV3VinvXQ
    tr.PV4 <- tr.VinvV4 - tr.XVinvV4VinvXQ
    tr.PV3PV3 <- tr.VinvV3VinvV3 - 2 * tr.XVinvV3VinvV3VinvXQ +
                 tr.XVinvV3VinvXQXVinvV3VinvXQ
    tr.PV4PV4 <- tr.VinvV4VinvV4 - 2 * tr.XVinvV4VinvV4VinvXQ +
                 tr.XVinvV4VinvXQXVinvV4VinvXQ
    tr.PV3PV4 <- tr.VinvV3VinvV4 - 2 * tr.XVinvV3VinvV4VinvXQ +
                 tr.XVinvV3VinvXQXVinvV4VinvXQ

    yPV3Py <- yVinvV3Vinvy - yVinvV3VinvX %*% Q %*% t(yVinvXb) -
           yVinvXb %*% Q %*% t(yVinvV3VinvX) + yVinvXb %*% Q %*%
              SumXVinvV3VinvX %*% Q %*% t(yVinvXb)

    yPV4Py <- yVinvV4Vinvy - yVinvV4VinvX %*% Q %*% t(yVinvXb) -
           yVinvXb %*% Q %*% t(yVinvV4VinvX) + yVinvXb %*% Q %*%
              SumXVinvV4VinvX %*% Q %*% t(yVinvXb)

    # Scores and elements of Fisher information Matrix
    # F11, F22, F12, F33, F44, F34  for A / B

    S1 <- -0.5 * tr.PV1 + 0.5 * yPV1Py
    S2 <- -0.5 * tr.PV2 + 0.5 * yPV2Py
    S3 <- -0.5 * tr.PV3 + 0.5 * yPV3Py
    S4 <- -0.5 * tr.PV4 + 0.5 * yPV4Py

    Ssig <- c(S1, S2, S3, S4)


    F11  <- 0.5 * tr.PV1PV1
    F22  <- 0.5 * tr.PV2PV2
    F12  <- 0.5 * tr.PV1PV2

    F33  <- 0.5 * tr.PV3PV3
    F44  <- 0.5 * tr.PV4PV4
    F34  <- 0.5 * tr.PV3PV4

 # Calculation of no-diagonal elements of Fisher information Matrix
```

```
# F13, F23, F14, F24   for A and B

   tr.XVinvV1VinvXQXVinvV3VinvXQ <-
                sum(diag(XVinvV1VinvXQ %*% XVinvV3VinvXQ))
   tr.XVinvV2VinvXQXVinvV3VinvXQ <-
                sum(diag(XVinvV2VinvXQ %*% XVinvV3VinvXQ))
   tr.XVinvV1VinvXQXVinvV4VinvXQ <-
                sum(diag(XVinvV1VinvXQ %*% XVinvV4VinvXQ))
   tr.XVinvV2VinvXQXVinvV4VinvXQ <-
                sum(diag(XVinvV2VinvXQ %*% XVinvV4VinvXQ))

   tr.PV1PV3 <- tr.XVinvV1VinvXQXVinvV3VinvXQ
   tr.PV2PV3 <- tr.XVinvV2VinvXQXVinvV3VinvXQ
   tr.PV1PV4 <- tr.XVinvV1VinvXQXVinvV4VinvXQ
   tr.PV2PV4 <- tr.XVinvV2VinvXQXVinvV4VinvXQ

   F13  <- 0.5 * tr.PV1PV3
   F23  <- 0.5 * tr.PV2PV3
   F14  <- 0.5 * tr.PV1PV4
   F24  <- 0.5 * tr.PV2PV4

   # FINAL Fisher information Matrix
   Fsig <- matrix(c(F11, F12, F13, F14, F12, F22,
                    F23, F24, F13, F23, F33, F34,
                    F14, F24, F34, F44),ncol=4)
   SumFsig <- abs(sum(Fsig))
   # print(det(Fsig))
   if (abs(det(Fsig))<0.000000001 || SumFsig > 10000000) {
       # print(ITER)
       ITER <- MAXITER
       Bad <- Bad + 1
       break }

   # print(ITER)

   Fsig.inv <- solve(Fsig)

   # Fisher-Scoring Algorithm
   dif <- Fsig.inv %*% Ssig

   #################################################################

   theta.f <- theta.f + dif

   sigma.fa <- theta.f[1,1]
   sigma.fb <- theta.f[3,1]
   rho.fa <- theta.f[2,1]
   rho.fb <- theta.f[4,1]
```

```
        # Stopping criterion
        # print(theta.f)

        if(abs(dif[1,1])<0.00001 && abs(dif[2,1])<0.00001 &&
           abs(dif[3,1])<0.00001 && abs(dif[4,1])<0.00001)
        break

        # print(Q)
        # print(sigma.fa)
        # print(sigma.fb)
        # print(rho.fa)
        # print(rho.fb)
        # results3 <- data.frame(theta.f, Fsig)
        # results3 <- as.data.frame(t(results3))
        # write.table(results3, file="REML.txt", sep="\t")

        # x<-"ITER greater than 49"
        # z<-"ITER smaller than 49"
        # if (ITER>49) print(x) else print(z)

    }

    if (sigma.fa < 0 || sigma.fb < 0 || rho.fa < -1 ||
        rho.fa > 1 || rho.fb < -1 || rho.fb > 1) {
        ITER <- MAXITER
        Bad <- Bad + 1 }

    return(list(as.vector(theta.f), Fsig, ITER, Bad))
}
```

18.3.3 R code of BETA.U.area.2corr

The R code of the function **BETA.U.area.2corr** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 3
### Author: Maria Chiara Pagliarella
### File name: EstimationBETA2corr.R
### Updated: May 2010
###
########################################################################


BETA.U.area.2corr <- function(X, ydt, D, Da, Db, md, mda, sigma2edt,
                        sigmaua, sigmaub, rhoa, rhob) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
```

```
Db <- D-Da

for(d in 2:D)
    i[[d]] <- (mdcum[d-1]+1):mdcum[d]

ia <- i[1:Da]
ib <- i[(Da+1):D]


yda <- Xda <- list()
for(d in 1:Da) {
    yda[[d]] <- ydt[ia[[d]]]
    Xda[[d]] <- X[ia[[d]],]
}
ydb <- Xdb <- list()
for(d in 1:Db) {
    ydb[[d]] <- ydt[ib[[d]]]
    Xdb[[d]] <- X[ib[[d]],]
}
yd <- Xd <- list()
for(d in 1:D) {
    yd[[d]] <- ydt[i[[d]]]
    Xd[[d]] <- X[i[[d]],]
}


Vd.inv <- list()
Q.inv <- matrix(0, nrow=p, ncol=p)
XVy <-0

for (d in 1:D) {
    ## Elements of the variance matrix TOTAL (A & B)
    if (d <= Da) {
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <-
                 rhoa^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
        Omega.a <- (1/(1-rhoa^2)) * Omega.a}
    else {
        Omega.b <- matrix(0,nrow=md[d],ncol=md[d])
        Omega.b[lower.tri(Omega.b)] <-
                 rhob^sequence((md[d]-1):1)
        Omega.b <- Omega.b + t(Omega.b)
        diag(Omega.b) <- 1
        Omega.b <- (1/(1-rhob^2)) * Omega.b}

    if (d <= Da)
        vd <-(sigmaua * Omega.a + diag(sigma2edt[ia[[d]]]))
        else
            vd <-(sigmaub * Omega.b + diag(sigma2edt[i[[d]]]))

        ### Inverse matrix of the variance and submatrices
        Vd.inv[[d]] <- solve(vd)

        ### Product between X^t_d,  V^-1_d and y_d for all d submatrices
        XVy <- XVy + t(Xd[[d]]) %*% Vd.inv[[d]] %*% yd[[d]]

        ### Inverse of Q. Next we calculate Q
        Q.inv <- Q.inv + t(Xd[[d]]) %*% Vd.inv[[d]] %*% Xd[[d]]
    }
    # print(Omega.a)
    # print(Omega.b)
    # print(vd)

    Q <- solve(Q.inv)

    beta <- Q %*% XVy

    ua <- ub <- list()
    for(d in 1:Da){
        ua[[d]] <- sigmaua * Omega.a %*% Vd.inv[[d]] %*% (yda[[d]] -
                   Xda[[d]] %*% beta)
    }
    for(d in 1:Db){
        ub[[d]] <- sigmaub * Omega.b %*% Vd.inv[[d]] %*% (ydb[[d]] -
                   Xdb[[d]] %*% beta)
    }
    ua<-as.matrix(unlist(ua))
    ub<-as.matrix(unlist(ub))
    u <- c(ua,ub)

    return(c(beta,u))
}
```

18.3.4 R code of mse.area.2corr

The R code of the function **mse.area.2corr** is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                      Pagliarella model 3
### Author: Maria Chiara Pagliarella
### File name: EstimationMSE2corr.R
### Updated: May 2010
###
########################################################################

mse.area.2corr <- function(X, D, Da, Db, md, mda, mdb, sigma2edt,
                     sigmaua, sigmaub, rhoa, rhob,
                     F11, F22, F12, F33, F44, F34) {

    p <- ncol(X)
    i <- list(1:md[1])
    mdcum <- cumsum(md)
    Db <- D-Da

    for(d in 2:D){
        i[[d]] <- (mdcum[d-1]+1):mdcum[d]
        if (d<= Da) ia <- i[1:Da]
        else ib <- i[(Da+1):D]
    }


    Xda <- list()
    for(d in 1:Da) {
        Xda[[d]] <- X[ia[[d]],]
    }
    Xdb <- list()
    for(d in 1:Db) {
        Xdb[[d]] <- X[ib[[d]],]
    }
    Xd <- list()
    for(d in 1:D) {
        Xd[[d]] <- X[i[[d]],]
    }


    ########################################################
    ###       Calculation of elements of g1, g2 and g3 FOR A

    Omega.a <- OmegaFirst.a <- Vda.inv <- VinvOmega.a <-
    VinvOmegaFirst.a <- OmegaVinvOmega.a <- OmegaVinvOmegaFirst.a <-
    OmegaFirstVinvOmegaFirst.a <- g1.a <- Sinv.a <- SinvXda <-
    OmegaVinvOmegaSinvXda <- g2.1a <- q11 <- q12 <- q22 <- list()
    for(d in 1:Da) {

        ### Elements of the variance matrix
        Omega.a[[d]] <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[[d]][lower.tri(Omega.a[[d]])] <-
                     rhoa^sequence((mda[d]-1):1)
        Omega.a[[d]] <- Omega.a[[d]] + t(Omega.a[[d]])
        diag(Omega.a[[d]]) <- 1
        Omega.a[[d]] <- (1/(1-rhoa^2)) * Omega.a[[d]]

        vda <-(sigmaua * Omega.a[[d]] + diag(sigma2edt[ia[[d]]]))

        OmegaFirst.a[[d]] <- matrix(0,nrow=mda[d],ncol=mda[d])
        OmegaFirst.a[[d]][lower.tri(OmegaFirst.a[[d]])] <-
                     sequence((mda[d]-1):1)*
                     rhoa^(sequence((mda[d]-1):1)-1)
        OmegaFirst.a[[d]] <- OmegaFirst.a[[d]] + t(OmegaFirst.a[[d]])
        OmegaFirst.a[[d]] <- (1/(1-rhoa^2)) * OmegaFirst.a[[d]]
        OmegaFirst.a[[d]] <- OmegaFirst.a[[d]] +
                     (2*rhoa / (1-rhoa^2)) * Omega.a[[d]]

        ### Inverse matrix of the variance and submatrices
        Vda.inv[[d]] <- solve(vda)

        ### Product between V^-1_da and Omega_a
        VinvOmega.a[[d]] <- Vda.inv[[d]] %*% Omega.a[[d]]
        ### Product between V^-1_da and OmegaFirst_a
        VinvOmegaFirst.a[[d]] <- Vda.inv[[d]] %*% OmegaFirst.a[[d]]
        ### Product between Omega.a, V^-1_da and Omega.a
        OmegaVinvOmega.a[[d]] <- t(VinvOmega.a[[d]]) %*% Omega.a[[d]]
        ### Product of Omega_a with V^-1_da and OmegaFirst_a
        OmegaVinvOmegaFirst.a[[d]] <- Omega.a[[d]] %*% Vda.inv[[d]] %*%
                                      OmegaFirst.a [[d]]
        ### Product of OmegaFirst_a with V^-1_da and OmegaFirst_a
        OmegaFirstVinvOmegaFirst.a[[d]] <- OmegaFirst.a[[d]] %*%
                     Vda.inv[[d]] %*% OmegaFirst.a[[d]]

        ### Calculation of g1_a
        g1.a[[d]] <- (sigmaua * Omega.a[[d]]) -
                     ((sigmaua)^2 * OmegaVinvOmega.a[[d]])

        ### Elements of the variance matrix Sigma_ed_a
        sed.a <- sigma2edt[ia[[d]]]
        ### Inverse matrix of Sigma_ed for all d submatrices_a
        Sinv.a[[d]] <- diag(1/sed.a)
        ### Product between Sigma_a^-1_ed and X_da for all d submatrices_a
        SinvXda[[d]] <- Sinv.a[[d]] %*% Xda[[d]]
        ### Product Omega.a with V^-1_da for Omega.a and Sigma^-1_ed
        ### for  X_da for all submatrices
        OmegaVinvOmegaSinvXda[[d]] <- OmegaVinvOmega.a[[d]]
                                      %*% SinvXda[[d]]

        ### First part of g2_a (the second is its transpose)
        g2.1a[[d]] <- Xda[[d]] - sigmaua * Omega.a[[d]] %*%
                      SinvXda[[d]] + (sigmaua^2) *
                      OmegaVinvOmegaSinvXda[[d]]

        ### Elements q11, q22 and q12 for calcultion of g3_a
        q11[[d]] <- OmegaVinvOmega.a[[d]] - 2 * sigmaua *
                    OmegaVinvOmega.a[[d]] %*% VinvOmega.a[[d]] +
                    (sigmaua)^2 * OmegaVinvOmega.a[[d]] %*% Vda.inv[[d]]
                    %*% OmegaVinvOmega.a[[d]]
```

```
    q12[[d]] <- sigmaua * OmegaVinvOmegaFirst.a[[d]] -
                (sigmaua)^2 * OmegaVinvOmegaFirst.a[[d]] %*%
                VinvOmega.a[[d]] - (sigmaua)^2 *
                OmegaVinvOmega.a[[d]] %*% VinvOmegaFirst.a[[d]] +
                (sigmaua)^3 * OmegaVinvOmega.a[[d]] %*%
                VinvOmegaFirst.a[[d]] %*% VinvOmega.a[[d]]
    q22[[d]] <- (sigmaua)^2 * OmegaFirstVinvOmegaFirst.a[[d]] -
                2 * (sigmaua)^3 * OmegaVinvOmegaFirst.a[[d]] %*%
                VinvOmegaFirst.a[[d]] + (sigmaua)^4 *
                OmegaVinvOmegaFirst.a[[d]] %*% Vda.inv[[d]] %*%
                t(OmegaVinvOmegaFirst.a[[d]])
}

########################################################
###       Calculation of elements of g1, g2 and g3 FOR B

Omega.b <- OmegaFirst.b <- Vdb.inv <- VinvOmega.b <-
VinvOmegaFirst.b <- OmegaVinvOmega.b <- OmegaVinvOmegaFirst.b <-
OmegaFirstVinvOmegaFirst.b <- g1.b <- Sinv.b <- SinvXdb <-
OmegaVinvOmegaSinvXdb <- g2.1b <- q33 <- q34 <- q44 <- list()
for(d in 1:Db) {

    ### Elements of the variance matrix
    Omega.b[[d]] <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    Omega.b[[d]][lower.tri(Omega.b[[d]])] <-
                 rhob^sequence((mdb[d]-1):1)
    Omega.b[[d]] <- Omega.b[[d]] + t(Omega.b[[d]])
    diag(Omega.b[[d]]) <- 1
    Omega.b[[d]] <- (1/(1-rhob^2)) * Omega.b[[d]]

    vdb <-(sigmaub * Omega.b[[d]] + diag(sigma2edt[ib[[d]]]))

    OmegaFirst.b[[d]] <- matrix(0,nrow=mdb[d],ncol=mdb[d])
    OmegaFirst.b[[d]][lower.tri(OmegaFirst.b[[d]])] <-
                 sequence((mdb[d]-1):1)*rhob^(sequence((mdb[d]-1):1)-1)
    OmegaFirst.b[[d]] <- OmegaFirst.b[[d]] + t(OmegaFirst.b[[d]])
    OmegaFirst.b[[d]] <- (1/(1-rhob^2)) * OmegaFirst.b[[d]]
    OmegaFirst.b[[d]] <- OmegaFirst.b[[d]] +
                 (2*rhob / (1-rhob^2)) * Omega.b[[d]]


    ### Inverse matrix of the variance and submatrices
    Vdb.inv[[d]] <- solve(vdb)

    ### Product between V^-1_db and Omega_b
    VinvOmega.b[[d]] <- Vdb.inv[[d]] %*% Omega.b[[d]]
    ### Product between V^-1_db and OmegaFirst_b
    VinvOmegaFirst.b[[d]] <- Vdb.inv[[d]] %*% OmegaFirst.b[[d]]
    ### Product between Omega.b, V^-1_db and Omega.b
    OmegaVinvOmega.b[[d]] <- t(VinvOmega.b[[d]]) %*% Omega.b[[d]]
    ### Product of Omega_b with V^-1_db and OmegaFirst_b
    OmegaVinvOmegaFirst.b[[d]] <- Omega.b[[d]] %*%
                 Vdb.inv[[d]] %*% OmegaFirst.b [[d]]
    ### Product of OmegaFirst_b with V^-1_db and OmegaFirst_b
    OmegaFirstVinvOmegaFirst.b[[d]] <- OmegaFirst.b[[d]] %*%
                 Vdb.inv[[d]] %*% OmegaFirst.b[[d]]

    ### Calculation of g1_b
    g1.b[[d]] <- (sigmaub * Omega.b[[d]]) - ((sigmaub)^2 *
                  OmegaVinvOmega.b[[d]])

    ### Elements of the variance matrix Sigma_ed_b
    sed.b <- diag(sigma2edt[ib[[d]]])
    ### Inverse matrix of Sigma_ed for all d submatrices_b
    Sinv.b[[d]] <- solve(sed.b)
    ### Product between Sigma_b^-1_ed and X_db for all d submatrices_b
    SinvXdb[[d]] <- Sinv.b[[d]] %*% Xdb[[d]]
    ### Product Omega.b with V^-1_db for Omega.b and Sigma^-1_ed
    ### for X_db for all submatrices
    OmegaVinvOmegaSinvXdb[[d]] <- OmegaVinvOmega.b[[d]] %*% SinvXdb[[d]]

    ### First part of g2_b (the second is its transpose)
    g2.1b[[d]] <- Xdb[[d]] - sigmaub * Omega.b[[d]] %*%
                  SinvXdb[[d]] + (sigmaub^2) *
                  OmegaVinvOmegaSinvXdb[[d]]

    ### Elements q33, q44 and q34 for calcultion of g3_b
    q33[[d]] <- OmegaVinvOmega.b[[d]] - 2 * sigmaub *
                OmegaVinvOmega.b[[d]] %*% VinvOmega.b[[d]] +
                (sigmaub)^2 * OmegaVinvOmega.b[[d]] %*%
                Vdb.inv[[d]] %*% OmegaVinvOmega.b[[d]]
    q34[[d]] <- sigmaub * OmegaVinvOmegaFirst.b[[d]] -
                (sigmaub)^2 * OmegaVinvOmegaFirst.b[[d]] %*%
                VinvOmega.b[[d]] - (sigmaub)^2 *
                OmegaVinvOmega.b[[d]] %*% VinvOmegaFirst.b[[d]] +
                (sigmaub)^3 * OmegaVinvOmega.b[[d]] %*%
                VinvOmegaFirst.b[[d]] %*% VinvOmega.b[[d]]
    q44[[d]] <- (sigmaub)^2 * OmegaFirstVinvOmegaFirst.b[[d]] -
                2 * (sigmaub)^3 * OmegaVinvOmegaFirst.b[[d]] %*%
                VinvOmegaFirst.b[[d]] + (sigmaub)^4 *
                OmegaVinvOmegaFirst.b[[d]] %*% Vdb.inv[[d]] %*%
                t(OmegaVinvOmegaFirst.b[[d]])
}


### Calculation of Q

Vd.inv <- VinvX <- list()
```

```
Q.inv <- matrix(0, nrow=p, ncol=p)

for(d in 1:D) {

    ### Elements of the variance matrix TOTAL (A & B)
    if (d <= Da) {
        Omega.a <- matrix(0,nrow=mda[d],ncol=mda[d])
        Omega.a[lower.tri(Omega.a)] <- rhoa^sequence((mda[d]-1):1)
        Omega.a <- Omega.a + t(Omega.a)
        diag(Omega.a) <- 1
        Omega.a <- (1/(1-rhoa^2)) * Omega.a
        vd <-(sigmaua * Omega.a + diag(sigma2edt[ia[[d]]]))}
    else {
        Omega.b <- matrix(0,nrow=md[d],ncol=md[d])
        Omega.b[lower.tri(Omega.b)] <- rhob^sequence((md[d]-1):1)
        Omega.b <- Omega.b + t(Omega.b)
        diag(Omega.b) <- 1
        Omega.b <- (1/(1-rhob^2)) * Omega.b
        vd <-(sigmaub * Omega.b + diag(sigma2edt[i[[d]]]))}

    ### Inverse matrix of the variance and submatrices
    Vd.inv[[d]] <- solve(vd)

    ### Inverse of Q. Next we calculate Q
    Q.inv <- Q.inv + t(Xd[[d]]) %*% Vd.inv[[d]] %*% Xd[[d]]
}

Q <- solve(Q.inv)


### Calculation of MSE_A

Fsig.a <- matrix(c(F11, F12, F12, F22),ncol=2)
g1a <- g2a <- g3a <- list()
for(d in 1:Da){
    g1a[[d]] <- diag(g1.a[[d]])
    g2a[[d]] <- diag(g2.1a[[d]] %*% Q %*% t(g2.1a[[d]]))
    q11[[d]] <- diag(q11[[d]])
    q12[[d]] <- diag(q12[[d]])
    q22[[d]] <- diag(q22[[d]])
}
for(d in 1:Da){
g3a[[d]] <- vector()
    for(i in 1:mda[d]){
        g3a[[d]][i] <- sum(diag(matrix(c(q11[[d]][i],
                       rep(q12[[d]][i],2),q22[[d]][i]),
                       nrow=2) %*% solve(Fsig.a)))
    }
}
```

```
    ### Calculation of MSE_B

    Fsig.b <- matrix(c(F33, F34, F34, F44),ncol=2)
    g1b <- g2b <- g3b <- list()
    for(d in 1:Db){
        g1b[[d]] <- diag(g1.b[[d]])
        g2b[[d]] <- diag(g2.1b[[d]] %*% Q %*% t(g2.1b[[d]]))
        q33[[d]] <- diag(q33[[d]])
        q34[[d]] <- diag(q34[[d]])
        q44[[d]] <- diag(q44[[d]])
    }
    for(d in 1:Db){
    g3b[[d]] <- vector()
        for(i in 1:mdb[d]){
            g3b[[d]][i] <- sum(diag(matrix(c(q33[[d]][i],
                           rep(q34[[d]][i],2),q44[[d]][i]),
                           nrow=2) %*% solve(Fsig.b)))
        }
    }

    g1a <- unlist(g1a)
    g1b <- unlist(g1b)
    g2a <- unlist(g2a)
    g2b <- unlist(g2b)
    g3a <- unlist(g3a)
    g3b <- unlist(g3b)

    ### Calculation of MSE

    mse.a <- g1a + g2a + 2 * g3a
    mse.b <- g1b + g2b + 2 * g3b
    mse <- c(mse.a, mse.b)

    return(mse)
}
```

18.3.5 R code of Interval.2corr

The R code of the function **Interval.2corr** for the confidence intervals and *p*-values is listed bellow.

```
########################################################################
###   Area level Partitioned F-H model with correlated time effects
###                     Pagliarella model 3
### Author: Maria Chiara Pagliarella
### File name: IC2corr.R
### Updated: July 2010
###
########################################################################
```

```
Interval <- function(Fisher, conf=0.95) {

    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- solve(Fisher[[2]])

    sigma.a.std.err <- z*sqrt(Finv[1,1])
    sigma.b.std.err <- z*sqrt(Finv[3,3])
    sigma.ab.std.err <- z*sqrt(Finv[1,1] + Finv[3,3] - 2*Finv[1,3])

    rho.a.std.err <- z*sqrt(Finv[2,2])
    rho.b.std.err <- z*sqrt(Finv[4,4])
    rho.ab.std.err <- z*sqrt(Finv[2,2] + Finv[4,4] - 2*Finv[2,4])

    beta.std.err <- z*sqrt(as.vector(diag(Fisher[[5]])))

    infbeta <- beta0.hat - beta.std.err
    supbeta <- beta0.hat + beta.std.err
    testbeta <- beta0.hat - beta.std.err < 0
                & beta0.hat + beta.std.err > 0

    infsigmaua <- sigmaua.hat - sigma.a.std.err
    supsigmaua <- sigmaua.hat + sigma.a.std.err
    testsigmaua <- sigmaua.hat - sigma.a.std.err < 0
                   & sigmaua.hat + sigma.a.std.err > 0

    infsigmaub <- sigmaub.hat - sigma.b.std.err
    supsigmaub <- sigmaub.hat + sigma.b.std.err
    testsigmaub <- sigmaub.hat - sigma.b.std.err < 0
                   & sigmaub.hat + sigma.b.std.err > 0

    infdif.sigma <- (sigmaua.hat - sigmaub.hat)
                    - sigma.ab.std.err
    supdif.sigma <- (sigmaua.hat - sigmaub.hat)
                    + sigma.ab.std.err
    testdif.sigma <- (sigmaua.hat - sigmaub.hat)
                     - sigma.ab.std.err < 0
                     & (sigmaua.hat - sigmaub.hat)
                     + sigma.ab.std.err > 0


    infrhoa <- rhoa.hat - rho.a.std.err
    suprhoa <- rhoa.hat + rho.a.std.err
    testrhoa <- rhoa.hat - rho.a.std.err < 0
                & rhoa.hat + rho.a.std.err > 0

    infrhob <- rhob.hat - rho.b.std.err
    suprhob <- rhob.hat + rho.b.std.err
    testrhob <- rhob.hat - rho.b.std.err < 0
                & rhob.hat + rho.b.std.err > 0

    infdif.rho <- (rhoa.hat - rhob.hat) - rho.ab.std.err
    supdif.rho <- (rhoa.hat - rhob.hat) + rho.ab.std.err
    testdif.rho <- (rhoa.hat - rhob.hat) - rho.ab.std.err < 0
                   & (rhoa.hat - rhob.hat) + rho.ab.std.err > 0


    return(list(
    sigma.a.std.err, sigma.b.std.err, sigma.ab.std.err,
    rho.a.std.err, rho.a.std.err, rho.ab.std.err,

    beta.std.err,
    infbeta, supbeta, testbeta,

    infsigmaua, supsigmaua, testsigmaua,
    infsigmaub, supsigmaub, testsigmaub,
    infdif.sigma, supdif.sigma, testdif.sigma,

    infrhoa, suprhoa, testrhoa,
    infrhob, suprhob, testrhob,
    infdif.rho, supdif.rho, testdif.rho))
}

pvalueBeta.2corr <- function(beta0.hat, Fisher) {

    z <- abs(beta0.hat) / sqrt(as.vector(diag(Fisher[[5]])))
    p.beta <- pnorm(z, lower.tail=F)

    return( 2*p.beta )
}
```

Chapter 19

Appendix 5: R code for the area-level spatio-temporal models

19.1 R code for the area-level spatio-temporal models

19.1.1 R code of FitSpatioTemporalFH

```
######################################################################
###            Spatio-temporal Fay Herriot Models
###                    SAMPLE Project
###
### Author:    Yolanda Marhuenda (y.marhuenda@umh.es)
### File name: FitSpatioTemporalFH.R
### Updated:   January 20th, 2011
###
######################################################################

diagonalizematrix <- function(A,ntimes)
{
   nrowA <- nrow(A)
   ncolA <- ncol(A)

   Adiag <- matrix(0,nrow=nrowA*ntimes, ncol=ncolA*ntimes)

   firsti <- 1
   firstj <- 1
   for (n in 1:ntimes)
   {
      lasti <- firsti+nrowA-1
      lastj <- firstj+ncolA-1
      Adiag[firsti:lasti,firstj:lastj]<-A
      firsti <- lasti+1
      firstj <- lastj+1
   }
   return (Adiag)
```

```
}

FitSpatioTemporalFH <- function(model,X,y,nD,nT,sigma2dt,theta0,
                                      W,MAXITER,PRECISION,confidence)
{
   result <- list(model=model, convergence=TRUE, iterations=0,
                  validtheta=FALSE, theta=0, beta=0,
                  goodnessoffit=0, estimates=0)

   if (model!="A" && model!="B")
   {
      cat("Error REML_Model: Model must be A or B:",model)
      result$convergence <- FALSE
      return (result)
   }

   M <- nD*nT
   nparam <- nrow(theta0)
                                      # Initialization
   invA   <- matrix(0,nrow=M,ncol=M)
   invAZ1 <- matrix(0,nrow=M,ncol=nD)
   tZ1PZ1 <- matrix(0,nrow=nD, ncol=nD)
   tZ1P   <- matrix(0,nrow=nD,ncol=M)
   S <- trPV   <- matrix(0,nrow=nparam,ncol=1)
   F <- trPVPV <- matrix(0,nrow=nparam,ncol=nparam)

   Va     <- list()
                                       ### Calculate Z1
   vector1T <- matrix(1,nrow=nT, ncol=1)
   Z1       <- matrix(0,nrow=nD*nT, ncol=nD)
   first <- 1
   for (d in 1:nD)
   {
      last <- first+nT-1
      Z1[first:last,d]<-vector1T
      first <- last+1
   }

   tZ1 <- t(Z1)
   ty  <- t(y)
   tX  <- t(X)
   tWW <- crossprod(W)
   p1derivrho1 <- - W - t(W)
   Id      <- diag(1,nrow=nD,ncol=nD)
   Tmen1   <- nT-1

   if (model=="A")
      Va[[3]] <- diag(1,nrow=M,ncol=M)
   else
```

```
{
   PV <- list()
   Omega2drho2 <- derivOmega2drho2 <- matrix(0,nrow=nT,ncol=nT)
   seqTmen1_1 <- sequence(Tmen1:1)
   Ve <- diag(sigma2dt)
}

thetakmas1 <- thetak <- theta0
k <- 0
diff <- PRECISION+1
while (diff>PRECISION & k<MAXITER)
{
   k <- k+1
   thetak    <- thetakmas1
   sigma21_k <- thetak[1]
   rho1_k    <- thetak[2]
   sigma22_k <- thetak[3]

   Omega1rho1_k <- solve(crossprod(Id-rho1_k*W))
   Vu1 <- sigma21_k*Omega1rho1_k

   if (model=="A")
   {
      invAvec <- 1/(sigma22_k+sigma2dt)
      invA <- diag(invAvec)
      first <- 1
      for (i in 1:nD)
      {
         last <- first+Tmen1
         firstlast <- first:last
         invAZ1[firstlast,i]<-invAvec[firstlast]
         first <- first + nT
      }
   }
   else
   {
      rho2_k        <- thetak[4]
      Unomenrho22_k <- 1-(rho2_k^2)
      Omega2drho2_k <- matrix(0,nrow=nT,ncol=nT)
      Omega2drho2_k[lower.tri(Omega2drho2_k)] <- rho2_k^seqTmen1_1
      Omega2drho2_k        <- Omega2drho2_k+t(Omega2drho2_k)
      diag(Omega2drho2_k)  <- 1
      Omega2drho2_k        <- (1/Unomenrho22_k)*Omega2drho2_k
      sigma22Omega2drho2_k <- sigma22_k*Omega2drho2_k

                                   #inverse in blocks
      first <- 1
      for (i in 1:nD)
      {
         last <- first+Tmen1
         firstlast <- first:last
         Ved <- Ve[first:last,first:last]
         Ad <- sigma22Omega2drho2_k + Ved
         invAd <- solve(Ad)
         invA[first:last,first:last]<-invAd
         first <- first + nT
      }
      invAZ1 = invA%*%Z1
   }
   invVu1 <- solve(Vu1)

   invV <- invA - invAZ1%*%solve(invVu1+tZ1%*%invAZ1)%*%t(invAZ1)
   tXinvV     <- tX %*% invV
   inv_tXinVX <- solve(tXinvV %*% X)
   P          <- invV - t(tXinvV) %*% inv_tXinVX %*% tXinvV

                                          # calculate S and F
   derivrho1_k <- p1derivrho1 + 2*rho1_k*tWW
                                          # calculate Va
   sigmaOmegaderivrho1Omega <- (-sigma21_k)*(Omega1rho1_k %*%
                                   derivrho1_k %*% Omega1rho1_k)

   Va[[1]] <- Z1 %*% Omega1rho1_k %*% tZ1
   Va[[2]] <- Z1 %*% sigmaOmegaderivrho1Omega %*% tZ1

   if (model=="A")
   {
      tZ1P    <- tZ1%*%P
      tZ1PZ1  <- tZ1P%*%Z1
      auxV1   <- tZ1PZ1%*%Omega1rho1_k
      trPV[1] <- sum(diag(auxV1))
      auxV2   <- tZ1PZ1%*%sigmaOmegaderivrho1Omega
      trPV[2] <- sum(diag(auxV2))
      trPV[3] <- sum(diag(P))
      Py      <- P %*% y
      tyP     <- t(Py)                      # P is symmetric
      trPVPV[1,1] <- sum(auxV1*t(auxV1))
      trPVPV[2,2] <- sum(auxV2*t(auxV2))
      trPVPV[3,3] <- sum(P*t(P))
      trPVPV[1,2] <- sum(auxV1*t(auxV2))
      tZ1PPZ1     <- tZ1P%*%t(tZ1P)
      trPVPV[1,3] <- sum(tZ1PPZ1*t(Omega1rho1_k))
      trPVPV[2,3] <- sum(tZ1PPZ1*t(sigmaOmegaderivrho1Omega))
   }
   else
   {
      Va[[3]] <- diagonalizematrix(Omega2drho2_k,ntimes=nD)
      derivOmega2drho2_k <- matrix(0,nrow=nT,ncol=nT)
      derivOmega2drho2_k[lower.tri(derivOmega2drho2_k)]<-seqTmen1_1*
                                      rho2_k^(seqTmen1_1-1)
      derivOmega2drho2_k <- derivOmega2drho2_k +
                            t(derivOmega2drho2_k)
      derivOmega2drho2_k <- (1/Unomenrho22_k)*derivOmega2drho2_k +
                            (2*rho2_k/Unomenrho22_k)*Omega2drho2_k
      sigma22derivOmega2drho2_k <- sigma22_k * derivOmega2drho2_k

      Va[[4]] <- diagonalizematrix(sigma22derivOmega2drho2_k,
                                   ntimes=nD)

      for (i in 1:nparam)
      {
         PV[[i]] <- P %*% Va[[i]]
         trPV[i] <- sum(diag(PV[[i]]))
      }

      for (j in 1:nparam)
      {
         tPVj <- t(PV[[j]])
         for (i in 1:j)
            trPVPV[i,j] <- sum(PV[[i]]*tPVj)
      }
      Py  <- P %*% y
      tyP <- t(Py)
   }

   for (a in 1:nparam)
   {
      S[a] <- (-0.5)*trPV[a] + 0.5*(tyP %*% Va[[a]] %*% Py)
      for (b in a:nparam)
         F[a,b] <- 0.5*trPVPV[a,b]
   }

   for (a in 2:nparam)     # symmetric
   {
      for (b in 1:(a-1))
         F[a,b] <- F[b,a]
   }

   Finv <- ginv(F)
   thetakmas1 <- thetak + Finv %*% S

         # Test values!=0 to avoid division errors
   if (any(thetak==0))
   {
      for (i in 1:nparam)
      {
         if (thetak[i]==0)
            thetak[i]<- 0.0001
      }
   }
   diff <- max( abs(thetak - thetakmas1)/thetak )

} #while (diff>PRECISION & k<MAXITER)

result$iterations <- k

                                          # validate output
if (k>=MAXITER && diff>=PRECISION)
{
   result$convergence <- FALSE
   return (result)
}
niter     <- k
sigma21_k <- thetakmas1[1]
rho1_k    <- thetakmas1[2]
sigma22_k <- thetakmas1[3]

if (sigma21_k<0 || rho1_k < (-1) || rho1_k>1 || sigma22_k<0 ||
    (model=="B" && (thetakmas1[4]<(-1) || thetakmas1[4]>1)) )
{
   result$theta <- thetakmas1
   return(result)
}

                                # calculate estimates beta, u, mu
Omega1rho1_k <- solve(crossprod(Id-rho1_k*W))
Vu1 <- sigma21_k*Omega1rho1_k

if (model=="A")
{
   invAvec <- 1/(sigma22_k+sigma2dt)
   invA <- diag(invAvec)
   first <- 1
   for (i in 1:nD)
   {
      last <- first+Tmen1
      firstlast <- first:last
      invAZ1[firstlast,i]<-invAvec[firstlast]
      first <- first + nT
   }
}
else
{
   rho2_k        <- thetakmas1[4]
   Unomenrho22_k <- 1-(rho2_k^2)
```

```
      Omega2drho2_k <- matrix(0,nrow=nT,ncol=nT)
      Omega2drho2_k[lower.tri(Omega2drho2_k)] <- rho2_k^seqTmen1_1
      Omega2drho2_k        <- Omega2drho2_k+t(Omega2drho2_k)
      diag(Omega2drho2_k)  <- 1
      Omega2drho2_k        <- (1/Unomenrho22_k)*Omega2drho2_k
      sigma22Omega2drho2_k <- sigma22_k*Omega2drho2_k

      first <- 1
      for (i in 1:nD)
      {
         last <- first+Tmen1
         firstlast <- first:last
         Ad <- sigma22Omega2drho2_k + Ve[first:last,first:last]
         invAd <- solve(Ad)
         invA[first:last,first:last]<-invAd
         first <- first + nT
      }
      invAZ1 <- invA%*%Z1
   }
   invVu1  <- solve(Vu1)
   invV    <- invA - invAZ1%*%solve(invVu1+tZ1%*%invAZ1)%*%t(invAZ1)

   tXinvV  <- tX %*% invV
   Q       <- solve(tXinvV %*% X)
   betaest <- Q %*% (tXinvV %*% y)
   ymenXbetaest  <- ( y - X %*% betaest )
   invVymenXBest <- invV %*% ymenXbetaest

   parte1  <- Vu1 %*% tZ1
   if (model=="A")
      parte2 <- diag(sigma22_k,nrow=M)
   else
      parte2 <- diagonalizematrix(sigma22Omega2drho2_k,ntimes=nD)

   u1est   <- parte1 %*% invVymenXBest
   u2dtest <- parte2 %*% invVymenXBest
   u1dtest <- matrix(data=rep(u1est, each=nT),nrow=M,ncol=1)
   mudtest <- X%*%betaest + u1dtest + u2dtest

   V       <- solve(invV)
   loglike <- (-0.5) * ( M*log(2*pi) +
              determinant(V,logarithm=TRUE)$modulus +
              t(ymenXbetaest)%*%invV%*%ymenXbetaest )
   AIC     <- (-2)*loglike + 2*(length(betaest)+nparam)
   BIC     <- (-2)*loglike + log(M)*(length(betaest)+nparam)

                        # calculate confidence intervals and pvalues
   alfa <- 1-confidence
   k    <- 1-alfa/2
   z    <- qnorm(k)
   sqrtQvector <- sqrt(diag(Q))
   intconfidencebeta  <- z*sqrtQvector
   intconfidencetheta <- z*sqrt(diag(Finv))

   z <- abs(betaest)/sqrtQvector
   p <- pnorm(z, lower.tail=FALSE)
   pvalue <- 2*p

   result$validtheta   <- TRUE
   result$theta        <- data.frame(estimate=thetakmas1,
                                     std.error=intconfidencetheta)
   result$beta         <- data.frame(coef=betaest,
                                     std.error=intconfidencebeta,
                                     tvalue=betaest/intconfidencebeta,
                                     pvalue=pvalue,
                                     greater.alfa=pvalue>alfa)
   result$goodnessoffit<- c(loglike=loglike, AIC=AIC, BIC=BIC)
   result$estimates    <- mudtest

   return (result)
}
```

19.1.2 R code of BootMSE.SpatioTemporalFH

```
#####################################################################
###           Spatio-temporal Fay Herriot Models
###                   SAMPLE Project
###
### Author:    Yolanda Marhuenda (y.marhuenda@umh.es)
### File name: BootMseSpatioTemporalFH.R
### Updated:   January 20th, 2011
###
#####################################################################

BootMSE.SpatioTemporalFH <- function(model,nB,Xdt,nD,nT,sigma2dt,
                                     beta,theta,rho1_0_b,rho2_0_b,
                                     W,MAXITER,PRECISION,confidence)
{
   if (model!="A" && model!="B")
   {
      print("Error: Model must be A or B.")
      return;
   }
   msedt <- 0
   Id    <- diag(1,nrow=nD, ncol=nD)

   sigma21 <- theta[1]
   rho1    <- theta[2]
   sigma22 <- theta[3]
```

```
Omega1rho1 <- solve(crossprod(Id-rho1*W))
sigma21Omega1rho1 <- sigma21*Omega1rho1
M <- nD*nT

if (model=="B")
{
   rho2   <- theta[4]
   Unomenrho22_05 <- (1-rho2^2)^(-0.5)
   u2dt_b <- matrix(0, nrow=M, ncol=1)
}

b<-1
while (b<=nB)
{
   u1_b   <- matrix(data=mvrnorm(n=1, mu=rep(0,nD),
                    Sigma=sigma21Omega1rho1), nrow=nD, ncol=1)
   u1dt_b <- matrix(data=rep(u1_b, each=nT),nrow=M,ncol=1)

   if (model=="A")
      u2dt_b <- matrix(data=rnorm(M, mean=0, sd=sqrt(sigma22)),
                       nrow=M, ncol=1)
   else
   {
      epsilondt_b <- matrix(data=rnorm(M, mean=0, sd=sqrt(sigma22)),
                            nrow=M, ncol=1)
      i <- 1
      for (d in 1:nD)
      {
         u2dt_b[i] <- Unomenrho22_05*epsilondt_b[i]
         for (t in 2:nT)
         {
            i <- i+1
            u2dt_b[i] <- rho2*u2dt_b[i-1]+epsilondt_b[i]
         }
         i<-i+1
      }
   }

   edt_b  <- matrix(data=rnorm(M, mean=0, sd=sqrt(sigma2dt)),
                    nrow=M,ncol=1)
   ydt_b  <- Xdt%*%beta + u1dt_b + u2dt_b + edt_b
   mudt_b <- ydt_b - edt_b

                                          ### fitting the model
   seedsigma_b <- Henderson(Xdt,ydt_b,sigma2dt)
   if (seedsigma_b<0)
   {
      cat("Warning: Assigning Henderson seed for sigma in
               bootstrap sample b=", b,".\n")
          seedsigma_b<-min(sigma2dt)
          cat("sigma is established to the minimum value of sigma2dt:",
               seedsigma_b,".\n")
       }
       sigma21_0_b <- sigma22_0_b <- 0.5*seedsigma_b

       if (model=="A")
          theta0_b <- rbind(sigma21_0_b, rho1_0_b, sigma22_0_b)
       else
          theta0_b <- rbind(sigma21_0_b, rho1_0_b, sigma22_0_b, rho2_0_b)

       result <- FitSpatioTemporalFH(model,X,ydt_b,nD,nT,sigma2dt,theta0_b,
                                     W,MAXITER,PRECISION,confidence)

       if (result$convergence==FALSE || result$validtheta==FALSE)
       {
          next
       }
                                                 ### calculate msedt
       difference <- result$estimates - mudt_b
       msedt      <- msedt + difference^2
       b <- b+1
   } #while (b<=nB)

   msedt <- msedt/nB
   return (msedt)
}
```

19.1.3 R code of Henderson

```
#####################################################################
###           Spatio-temporal Fay Herriot Models
###                   SAMPLE Project
###
### Author:    Yolanda Marhuenda (y.marhuenda@umh.es)
### File name: Henderson.R
### Updated:   January 20th, 2011
###
#####################################################################

Henderson <- function(X,y,sigma2dt)
{
   invVe    <- 1/sigma2dt
   invVeX   <- invVe*X
   tXinvVeX <- t(X)%*%invVeX
   P        <- diag(invVe)-invVeX%*%solve(tXinvVeX)%*%t(invVeX)
   trtZPZ   <- sum(diag(P))
   tyPy     <- t(y)%*%P%*%y    # for vectors t(y)%*%P%*%y=sum(y*P%*%y)
   sigma2   <- as.numeric((tyPy-(nrow(X)-ncol(X)))/trtZPZ)

   return (sigma2)
}
```

19.1.4 R code of MessageErrorFitting

```
########################################################################
###           Spatio-temporal Fay Herriot Models
###                    SAMPLE Project
###
### Author:    Yolanda Marhuenda (y.marhuenda@umh.es)
### File name: MessageErrorFitting.R
### Updated:   January 20th, 2011
###
########################################################################

MessageErrorFitting <- function(model,nsample,convergence,niter,
                                validtheta,theta)
{
   linea <- paste("Warning: Sample=",nsample,"\n")
   cat(linea)

   if (convergence==FALSE)
   {
      linea <- paste("Maximum number of iterations is reached.\n")
      cat(linea)
   }
   else
   {
      sigma21_REML <- theta[1]
      rho1_REML    <- theta[2]
      sigma22_REML <- theta[3]

      if (sigma21_REML<0)
      {
         linea <- paste(" sigma21<0 (",sigma21_REML,")" )
         cat(linea)
      }
      if (rho1_REML<(-1) || rho1_REML>1)
      {
         linea <- paste(" rho1 must be in [-1,1] (",rho1_REML,")" )
         cat(linea)
      }
      if (sigma22_REML<0)
      {
         linea <- paste(" sigma22<0 (",sigma22_REML,")" )
         cat(linea)
      }
      if (model=="B")
      {
         rho2_REML <- theta[4]
         if (rho2_REML<(-1) || rho2_REML>1)
         {
            linea <- paste(" rho2 must be in [-1,1] (",rho2_REML,")" )
            cat(linea)
         }
      }
      linea <- paste("\nTotal number of iterations:",niter,".\n")
      cat (linea)
   }
}
```

Chapter 20

Appendix 6: R code for the unit-level time models

20.1 R code for the unit-level models with independent time effects

20.1.1 R code of REML.individual.indep

The R code of the function **REML.individual.indep** is listed bellow.

```
####################################################################
### Unit level model with independent time effects              ###
### SAMPLE project                                              ###
### Author: Laureano Santamaria Arana                           ###
####################################################################

REML.individual.indep <- function(X, Y, W, D, md, ndi,
sigma0,sigma1, sigma2, MAXITER = 500) {

    n <- nrow(X)
    p <- ncol(X)

    Sd.inv<-matrix(0, nrow=n, ncol=n)

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)


    for(ITER in 1:MAXITER){

        F1 <- sigma1/sigma0
        F2 <- sigma2/sigma0

        mR <- 0
        mF <- 0
        mJ <- 0
        for(d in 1:D) {
```

```
        if (d==1) {
            Inicio <-1
            Pr <-1
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        Wd <- W[Pr:Fin,Pr:Fin]
        yd <- Y[Pr:Fin]
        Xd <- X[Pr:Fin,]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])

        sld.inv <- solve(Imd+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd

        UnoNd <-as.matrix(rep(1,Nd))

        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)

        Sigmad.inv <- Ld.inv - (T1/T2[1,1])
        Sd.inv[Pr:Fin,Pr:Fin] <- Sigmad.inv

        mR <- mR + t(Xd)%*%Sigmad.inv%*%Xd
        mF <- mF + t(Xd)%*%Sigmad.inv%*%UnoNd%*%t(UnoNd)%*%
              Sigmad.inv%*%Xd
        mJ <- mJ + t(Xd)%*%Sigmad.inv%*%D1Nd%*%t(D1Nd)%*%
              Sigmad.inv%*%Xd
}

R <- solve(mR)

mA <- 0
mB <- 0
mC <- 0
mD <- 0
```

```
mE <- 0
mG <- 0
mH <- 0
mK <- 0
mL <- 0
mM <- 0
mN <- 0
mP <- 0
mQ <- 0
mR <- 0
mS <- 0
mT <- 0
mV <- 0
mW <- 0

for(d in 1:D) {
    if (d==1) {
        Inicio <-1
        Pr <-1
    }
    if (d!=1) {
        Pr <- (ndcum[mdcum[d-1]]+1)
        Inicio <-mdcum[d-1]+1
    }
    Fin  <- ndcum[mdcum[d]]
    Nd <- Fin-Pr+1

    Wd <- W[Pr:Fin,Pr:Fin]
    yd <- Y[Pr:Fin]
    Xd <- X[Pr:Fin,]

    D1Nd <- matrix(0,Nd,md[d])
    i <- 1
    for(k in 1:md[d]) {
        for(j in 1:ndi[Inicio+k-1]) {
            D1Nd[i,k]<-1
            i<- i+1
        }
    }

    UnoNd <-as.matrix(rep(1,Nd))

    mA <- mA + t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%yd
    mB <- mB + t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd
    mC <- mC + t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%yd

    T3 <- t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd%*%t(UnoNd)%*%
          Sd.inv[Pr:Fin,Pr:Fin]
    mD <- mD + T3%*%yd
    mE <- mE + T3%*%Xd

    T4 <- Sd.inv[Pr:Fin,Pr:Fin] - Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%
          R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]

    mG <- mG + t(UnoNd)%*%T4%*%UnoNd
    mH <- mH + sum(diag(t(D1Nd)%*%T4%*%D1Nd))

    T5 <- t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%t(D1Nd)%*%
          Sd.inv[Pr:Fin,Pr:Fin]

    mK <- mK + T5%*%yd
    mL <- mL + T5%*%Xd

    T6 <- t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd

mM <- mM + T6%*%T6
mN <- mN + T6%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R%*%
      t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd
mP <- mP + t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R%*%mF%*%
      R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd

T7 <- t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R

T8 <- t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd%*%t(UnoNd)%*%
      Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd

mQ <- mQ + sum(diag(T8))
mR <- mR + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
      UnoNd%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd))
mS <- mS + sum(diag(T7%*%mF%*%R%*%t(Xd)%*%
  Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd))

mT <- mT + sum(diag(t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
      t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd))
mV <- mV + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd
      %*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd))
mW <- mW + sum(diag(T7%*%mJ%*%R%*%t(Xd)%*%
       Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd))

}

S <- matrix(0, nrow=2, ncol=1)
F <- matrix(0, nrow=2, ncol=2)
F3 <- matrix(0, nrow=3, ncol=3)

sigma0S <- 1/(n-p)*(mA-(mB%*%R%*%mC))
```

```
        S[1,1] <- - mG/2 + mD/(2*sigma0) - (mE%*%R%*%mC)/sigma0 +
                  (mB%*%R%*%mF%*%R%*%mC)/(2*sigma0)
        S[2,1] <- - mH/2 + mK/(2*sigma0) - (mL%*%R%*%mC)/sigma0 +
                  (mB%*%R%*%mJ%*%R%*%mC)/(2*sigma0)

        F[1,1] <- mM/2 - mN + mP/2
        F[1,2] <- mQ/2 - mR + mS/2
        F[2,1] <- F[1,2]
        F[2,2] <- mT/2 -mV + mW/2

        F.inv <- solve(F)
        d <- F.inv%*%S
        F1 <- F1 + d[1]
        F2 <- F2 + d[2]

        dif <- rbind(sigma0S-sigma0,d*as.vector(sigma0S))
        sigma <- rbind(sigma0S, sigma0S*F1,sigma0S*F2)

        ### Fisher information matrix

        F3[1,1] <- (n-p)/(2*sigma0*sigma0)
        F3[1,2] <- mG/(2*sigma0)
        F3[1,3] <- mH/(2*sigma0)
        F3[2,1] <- F3[1,2]
        F3[2,2] <- mM/2 - mN + mP/2
        F3[2,3] <- mQ/2 - mR + mS/2
        F3[3,1] <- F3[1,3]
        F3[3,2] <- F3[2,3]
        F3[3,3] <- mT/2 -mV + mW/2

        F3.inv <- solve(F3)

        ### Scores

        sigma0 <- sigma[1]
        sigma1 <- sigma[2]
        sigma2 <- sigma[3]

    ### Stopping criterion

    if(abs(dif[1])<0.00001 & abs(dif[2])<0.00001 & abs(dif[3])<0.00001)
        break

    }
    return(list(sigma0,sigma1,sigma2,F3.inv,ITER,R))

}
```

20.1.2 R code of BETA.U.individual.indep

The R code of the function **BETA.U.individual.indep** is listed bellow.

```
###################################################################
### Unit level model with independent time effects             ###
### SAMPLE project                                              ###
### Author: Laureano Santamaria Arana                           ###
###################################################################


BETA.U.individual.indep <- function(X, Y, W, D, md, ndi,
sigma0,sigma1, sigma2) {


    n <- nrow(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0


    Sd.inv<-matrix(0, nrow=n, ncol=n)

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)
    B1 <- 0
    B2 <- 0

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            P <-1
        }
        if (d!=1) {
            P <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-P+1

        Wd <- W[P:Fin,P:Fin]
        yd <- Y[P:Fin]
        Xd <- X[P:Fin,]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])

        sld.inv <- solve(Imd+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd

        UnoNd <-as.matrix(rep(1,Nd))

        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)

        Sigmad.inv <- Ld.inv - (T1/T2[1,1])
        Sd.inv[P:Fin,P:Fin] <- Sigmad.inv

        B1 <- B1 + t(Xd)%*%Sigmad.inv%*%Xd
        B2 <- B2 + t(Xd)%*%Sigmad.inv%*%yd


    }

    Beta <- solve(B1)%*%B2

    return(Beta)

}
```

20.1.3 R code of mse.individual.indep

The R code of the function **mse.individual.indep** is listed bellow.

```
#####################################################################
### Unit level model with independent time effects              ###
### SAMPLE project                                              ###
### Author: Laureano Santamaria Arana                           ###
#####################################################################



mse.individual.indep <- function(X, Y, W, D, md, ndi, MXm ,NDI, MXp,
sigma0, sigma1, sigma2, FInv) {

    g1 <- G1(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI)
    g2 <- G2(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI, MXp, MXm)
    g3 <- G3(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI, FInv)
    g4 <- G4(D, md, ndi, NDI, W, sigma0)

    return(g1+g2+2*g3+g4)
}

### Calculation of g1

G1 <- function(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI) {

    n <- nrow(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0

    # ndi muestral
    # NDI poblacional

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)

    fdi <- ndi/NDI

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        Wd <- W[Pr:Fin,Pr:Fin]
        yd <- Y[Pr:Fin]
        Xd <- X[Pr:Fin,]

        fd <- fdi[Inicio:F]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))

        s1d.inv <- solve(Imd+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%s1d.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        for(k in 1:md[d]) {
          M1 <- Imd[k,]*(1-fd[k])
          P1d <- sigma0*F1*((1-fd[k])^2)*(1-(F1*t(UnoNd)%*%
                 Sigmad.inv%*%UnoNd))
          P2d <- -sigma0*F1*F2*(1-fd[k])*(t(UnoNd)%*%
                  Sigmad.inv%*%D1Nd%*%M1)
          P3d <- (sigma0*F2*((1-fd[k])^2)) - (sigma0*F2*F2*(diag(t(M1)
                  %*%t(D1Nd)%*%Sigmad.inv%*%D1Nd%*%M1)))
          if (k==1) {
              g1d <- P1d+P2d+P3d
          }
          if (k!=1) {
              g1d <- c(g1d,P1d+P2d+P3d)
          }
        }
        if (d==1) {
            g1 <- g1d
        }
        if (d!=1) {
            g1 <- c(g1,g1d)
        }
    }
    return(g1)
}

### Calculation of g2

G2 <- function(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI,
MXp, MXm) {

    n <- nrow(X)
    p <- ncol(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0

    # ndi muestral
    # NDI poblacional
    # MXp Media poblacional
    # MXm Media muestral

    M <- sum(md)

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)

    fdi <- ndi/NDI
    D1 <- NDI-ndi
    K1 <- NDI/D1
    K2 <- ndi/D1
    Xast <- K1*MXp - K2*MXm

    mR <- 0

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        Wd <- W[Pr:Fin,Pr:Fin]
        yd <- Y[Pr:Fin]
        Xd <- X[Pr:Fin,]
        fd <- fdi[Inicio:F]
        Xdast <- Xast[Inicio:F]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))

        sld.inv <- solve(Imd+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        mR <- mR + t(Xd)%*%Sigmad.inv%*%Xd
    }
    Q <- sigma0*solve(mR)

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))

        Wd <- W[Pr:Fin,Pr:Fin]
        Xd <- X[Pr:Fin,]
        fd <- fdi[Inicio:F]
        Xdast <- Xast[Inicio:F]

        sld.inv <- solve(Imd+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        for(k in 1:md[d]) {
            G11d <- (F1*(1-fd[k]))*(1-(F1*t(UnoNd)%*%
                     Sigmad.inv%*%UnoNd))%*%t(UnoNd)%*%Wd%*%Xd
            G12d <- -F1*F2*(1-fd[k])*(t(UnoNd)%*%Sigmad.inv%*%
                     D1Nd%*%t(D1Nd)%*%Wd%*%Xd)
            G21d <- -F1*F2*(1-fd[k])*(t(Imd[k,])%*%t(D1Nd)%*%
                     Sigmad.inv%*%UnoNd%*%t(UnoNd)%*%Wd%*%Xd)
            T1   <- Imd - (F2*t(D1Nd)%*%Sigmad.inv%*%D1Nd)
            G22d <- F2*(1-fd[k])*t(Imd[k,])%*%T1%*%t(D1Nd)%*%Wd%*%Xd

            if (k==1) {
                G2d <- G11d + G12d + G21d + G22d
            }
            if (k!=1) {
                G2d <- c(G2d, G11d + G12d + G21d + G22d)
            }
        }
        if (d==1) {
            G2 <- G2d
        }
        if (d!=1) {
            G2 <- c(G2, G2d)
        }

    }

    at22 <- matrix(G2,nrow=M,ncol=p,byrow = T)
    at21 <- matrix((1-fdi)*Xast,nrow=M,ncol=p,byrow = T)

    g2 <- diag((at21-at22)%*%Q%*%(t(at21)-t(at22)))

    return(g2)

}

### Calculation of g3

G3 <- function(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, NDI,
Finv) {
```

```
n <- nrow(X)
F1 <- sigma1/sigma0
F2 <- sigma2/sigma0

# ndi muestral
# NDI poblacional

mdcum <- cumsum(md)
ndcum <- cumsum(ndi)
fdi <- ndi/NDI

mR <- 0

q<-matrix(0, nrow=3, ncol=3)

for(d in 1:D) {
    if (d==1) {
        Inicio <-1
        Pr <-1
        F <- md[d]
    }
    if (d!=1) {
        Pr <- (ndcum[mdcum[d-1]]+1)
        Inicio <-mdcum[d-1]+1
        F <- Inicio + md[d] - 1
    }
    Fin  <- ndcum[mdcum[d]]
    Nd <- Fin-Pr+1

    Wd <- W[Pr:Fin,Pr:Fin]
    yd <- Y[Pr:Fin]
    Xd <- X[Pr:Fin,]
    fd <- fdi[Inicio:F]

    D1Nd <- matrix(0,Nd,md[d])
    i <- 1
    for(k in 1:md[d]) {
        for(j in 1:ndi[Inicio+k-1]) {
            D1Nd[i,k]<-1
            i<- i+1
        }
    }
    Imd<-diag(md[d])
    UnoNd <-as.matrix(rep(1,Nd))

    Ads.inv <- solve(Imd+(F2*t(D1Nd)%*%Wd%*%D1Nd))
    Lds.inv <- Wd-(F2*Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd)
```

```
T1 <- F1*Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv
T2 <- 1+(F1*t(UnoNd)%*%Lds.inv%*%UnoNd)
Sigmad.inv <- Lds.inv - (T1/T2[1,1])

T1 <- -Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd
T2 <- F2*Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd%*%
      D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd
DpLdsF2 <- T1 + T2

T1 <- Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv
T2 <- 1+(F1*t(UnoNd)%*%Lds.inv%*%UnoNd)
DpSdsF1 <- (T1/(T2[1,1]^2))

T0 <- as.numeric((F1*F1*t(UnoNd)%*%DpLdsF2%*%UnoNd))
T1 <- (T0*(Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv))/(T2[1,1]^2)
T3 <- (DpLdsF2%*%UnoNd%*%t(UnoNd)%*%Lds.inv) +
      (Lds.inv%*%UnoNd%*%t(UnoNd)%*%DpLdsF2)
DpSdsF2 <- DpLdsF2 + T1 - (F1*T3)/(T2[1,1])

Sigmads <- solve(Sigmad.inv)




for(k in 1:md[d]) {
    DpB1dF1 <- (1-fd[k])*t(UnoNd)%*%(Sigmad.inv + F1*DpSdsF1)
    DpB1dF2 <- F1*(1-fd[k])*t(UnoNd)%*%DpSdsF2

    DpB2dF1 <- F2*(1-fd[k])*Imd[k,]%*%t(D1Nd)%*%DpSdsF1
    DpB2dF2 <- (1-fd[k])*Imd[k,]%*%t(D1Nd)%*%
               (Sigmad.inv + F2*DpSdsF2)

    T2 <- (DpB1dF1+DpB2dF1)
    T3 <- (DpB1dF2+DpB2dF2)


    q[1,1] <- 0
    q[1,2] <- 0
    q[1,3] <- 0
    q[2,1] <- 0
    q[2,2] <- T2%*%(sigma0*Sigmads)%*%t(T2)
    q[2,3] <- T2%*%(sigma0*Sigmads)%*%t(T3)
    q[3,1] <- 0
    q[3,2] <- T3%*%(sigma0*Sigmads)%*%t(T2)
    q[3,3] <- T3%*%(sigma0*Sigmads)%*%t(T3)

    g3dparcial <- sum(diag(q%*%Finv))
```

```
            if (k==1) {
                g3d <- g3dparcial
             }
            if (k!=1) {
                g3d <- c(g3d,g3dparcial)
            }
        }
        if (d==1) {
            g3 <- g3d
        }
        if (d!=1) {
            g3 <- c(g3,g3d)
        }
    }
    return(g3)
}

### Calculation of g4

G4 <- function(D, md, ndi, NDI, W, sigma0) {


    # ndi muestral
    # NDI poblacional

    g4 <- vector()

    Indice <- 1
    Inicio <- 1
    for(d in 1:D) {
        for (j in 1:md[d]) {
            g4[Indice] <- (sigma0/(NDI[Indice]^2))*
                              (NDI[Indice]-ndi[Indice])
            Indice <- Indice + 1
        }
    }

    return(g4)
}
```

20.1.4 R code of Interval.indep

The R code of the function **Interval.indep** is listed bellow.

```
###################################################################
### Unit level model with independent time effects             ###
### SAMPLE project                                             ###
### Author: Laureano Santamaria Arana                          ###
###################################################################


Interval.indep <- function(fit, conf=0.95) {
    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)

    Finv <- fit[[2]]
    sigma.std.err <- z*sqrt(Finv[1,1])
    sigma1.std.err <- z*sqrt(Finv[2,2])
    sigma2.std.err <- z*sqrt(Finv[3,3])
    beta.std.err <- z*sqrt(as.vector(diag(fit[[3]])))

    return(list(sigma.std.err, sigma1.std.err,
                sigma2.std.err, beta.std.err))
}
```

20.1.5 R code of pvalue

The R code of the function **pvalue** is listed bellow.

```
###################################################################
### Unit level model with independent time effects             ###
### SAMPLE project                                             ###
### Author: Laureano Santamaria Arana                          ###
###################################################################



pvalue <- function(beta0, fit) {

    z <- abs(beta0)/sqrt(as.vector(diag(fit[[3]])))
    pval <- 2*pnorm(z, lower.tail=F)

    return( pval )
}
```

20.2 R code for the unit-level models with correlated time effects

20.2.1 R code of REML.autocorr

The R code of the function **REML.individual.autocorr** is listed bellow.

```
####################################################################
### Unit level model with time correlated effects                ###
### SAMPLE project                                               ###
### Author: Laureano Santamaria Arana                            ###
####################################################################


source("Omega.R")

REML.individual.autocorr <- function(X, Y, W, D, md, ndi,
sigma0, sigma1, sigma2, rho, MAXITER = 500) {

    n <- nrow(X)
    p <- ncol(X)

    Sd.inv<-matrix(0, nrow=n, ncol=n)

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)


    for(ITER in 1:MAXITER){

        F1 <- sigma1/sigma0
        F2 <- sigma2/sigma0

        mR <- 0
        mF <- 0
        mJ <- 0
        mJD <- 0
        for(d in 1:D) {
            if (d==1) {
                Inicio <-1
                Pr <-1
            }
            if (d!=1) {
                Pr <- (ndcum[mdcum[d-1]]+1)
                Inicio <-mdcum[d-1]+1
            }
            Fin  <- ndcum[mdcum[d]]
            Nd <- Fin-Pr+1

            Wd <- W[Pr:Fin,Pr:Fin]
            yd <- Y[Pr:Fin]
            Xd <- X[Pr:Fin,]

            D1Nd <- matrix(0,Nd,md[d])
            i <- 1
            for(k in 1:md[d]) {
```

```
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        Omegad <- Calcula_Omegad(md[d], rho)
        OmegadDeriv <- Deriva_Omegad (md[d], rho, Omegad)
        OmegadInv <- Inversa_Omegad(md[d], rho)

        sld.inv <- solve(OmegadInv+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd

        UnoNd <-as.matrix(rep(1,Nd))

        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)

        Sigmad.inv <- Ld.inv - (T1/T2[1,1])
        Sd.inv[Pr:Fin,Pr:Fin] <- Sigmad.inv

        mR <- mR + t(Xd)%*%Sigmad.inv%*%Xd
        mF <- mF + t(Xd)%*%Sigmad.inv%*%UnoNd%*%
              t(UnoNd)%*%Sigmad.inv%*%Xd
        mJ <- mJ + t(Xd)%*%Sigmad.inv%*%D1Nd%*%
              Omegad%*%t(D1Nd)%*%Sigmad.inv%*%Xd
        mJD <- mJD + t(Xd)%*%Sigmad.inv%*%D1Nd%*%
              OmegadDeriv%*%t(D1Nd)%*%Sigmad.inv%*%Xd
    }

    R <- solve(mR)

    mA <- 0
    mB <- 0
    mC <- 0
    mD <- 0
    mE <- 0
    mG <- 0
    mH <- 0
    mK <- 0
    mL <- 0
    mHD <- 0
    mKD <- 0
    mLD <- 0
    mM <- 0
    mN <- 0
    mP <- 0
    mQ <- 0
    mR <- 0
```

```
mS <- 0
mQD <- 0
mRD <- 0
mSD <- 0
mT <- 0
mV <- 0
mW <- 0
mTD <- 0
mVD <- 0
mWD <- 0
mTDD <- 0
mVDD <- 0
mWDD <- 0

for(d in 1:D) {
    if (d==1) {
        Inicio <-1
        Pr <-1
    }
    if (d!=1) {
        Pr <- (ndcum[mdcum[d-1]]+1)
        Inicio <-mdcum[d-1]+1
    }
    Fin  <- ndcum[mdcum[d]]
    Nd <- Fin-Pr+1

    Wd <- W[Pr:Fin,Pr:Fin]
    yd <- Y[Pr:Fin]
    Xd <- X[Pr:Fin,]

    D1Nd <- matrix(0,Nd,md[d])
    i <- 1
    for(k in 1:md[d]) {
        for(j in 1:ndi[Inicio+k-1]) {
            D1Nd[i,k]<-1
            i<- i+1
        }
    }

    UnoNd <-as.matrix(rep(1,Nd))
    Omegad <- Calcula_Omegad(md[d], rho)
    OmegadDeriv <- Deriva_Omegad (md[d], rho, Omegad)

    mA <- mA + t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%yd
    mB <- mB + t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd
    mC <- mC + t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%yd

    T3 <- t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
          UnoNd%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]
```

```
mD <- mD + T3%*%yd
mE <- mE + T3%*%Xd

T4 <- Sd.inv[Pr:Fin,Pr:Fin] - Sd.inv[Pr:Fin,Pr:Fin]%*%
      Xd%*%R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]

mG <- mG + t(UnoNd)%*%T4%*%UnoNd
mH <- mH + sum(diag(t(D1Nd)%*%T4%*%D1Nd%*%Omegad))
mHD <- mHD + sum(diag(t(D1Nd)%*%T4%*%D1Nd%*%OmegadDeriv))

T5 <- t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%Omegad%*%
      t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]
mK <- mK + T5%*%yd
mL <- mL + T5%*%Xd

T5D <- t(yd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%OmegadDeriv%*%
       t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]
mKD <- mKD + T5D%*%yd
mLD <- mLD + T5D%*%Xd

T6 <- t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd

mM <- mM + T6%*%T6
mN <- mN + T6%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R%*%
      t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd
mP <- mP + t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R%*%mF%*%
      R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd

T7 <- t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%Xd%*%R

T8 <- t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd%*%t(UnoNd)%*%
      Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%Omegad

mQ <- mQ + sum(diag(T8))
mR <- mR + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
      UnoNd%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%Omegad))
mS <- mS + sum(diag(T7%*%mF%*%R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]
      %*%D1Nd%*%Omegad))

T8D <- t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%UnoNd%*%t(UnoNd)%*%
       Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%OmegadDeriv

mQD <- mQD + sum(diag(T8))
mRD <- mRD + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
       UnoNd%*%t(UnoNd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
       OmegadDeriv))
mSD <- mSD + sum(diag(T7%*%mF%*%R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]
       %*%D1Nd%*%OmegadDeriv))
```

```
    mT <- mT + sum(diag(t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
          Omegad%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%Omegad))
    mV <- mV + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
          Omegad%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%Omegad))
    mW <- mW + sum(diag(T7%*%mJ%*%R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]
          %*%D1Nd%*%Omegad))
    mTD <- mTD + sum(diag(t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
           Omegad%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%
           OmegadDeriv))
    mVD <- mVD + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd
           %*%Omegad%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd
           %*%OmegadDeriv))
    mWD <- mWD + sum(diag(T7%*%mJ%*%R%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]
           %*%D1Nd%*%OmegadDeriv))
    mTDD <- mTDD + sum(diag(t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
            D1Nd%*%OmegadDeriv%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]
            %*%D1Nd%*%OmegadDeriv))
    mVDD <- mVDD + sum(diag(T7%*%t(Xd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
            D1Nd%*%OmegadDeriv%*%t(D1Nd)%*%Sd.inv[Pr:Fin,Pr:Fin]%*%
            D1Nd%*%OmegadDeriv))
    mWDD <- mWDD + sum(diag(T7%*%mJD%*%R%*%t(Xd)%*%
            Sd.inv[Pr:Fin,Pr:Fin]%*%D1Nd%*%OmegadDeriv))

}

S <- matrix(0, nrow=4, ncol=1)
Fisher <- matrix(0, nrow=4, ncol=4)

S[1,1] <- - (n-p)/(2*sigma0) + mA/(2*sigma0*sigma0) -
            (mB%*%R%*%mC)/(2*sigma0*sigma0)
S[2,1] <- - mG/2 + mD/(2*sigma0) - (mE%*%R%*%mC)/sigma0 +
            (mB%*%R%*%mF%*%R%*%mC)/(2*sigma0)
S[3,1] <- - mH/2 + mK/(2*sigma0) - (mL%*%R%*%mC)/sigma0 +
            (mB%*%R%*%mJ%*%R%*%mC)/(2*sigma0)
S[4,1] <- F2*(- mHD/2 + mKD/(2*sigma0) - (mLD%*%R%*%mC)/sigma0 +
          (mB%*%R%*%mJD%*%R%*%mC)/(2*sigma0))

Fisher[1,1] <- (n-p)/(2*sigma0*sigma0)
Fisher[1,2] <- mG/(2*sigma0)
Fisher[1,3] <- mH/(2*sigma0)
Fisher[1,4] <- F2*(mHD/(2*sigma0))
Fisher[2,1] <- Fisher[1,2]
Fisher[2,2] <- mM/2 - mN + mP/2
Fisher[2,3] <- mQ/2 - mR + mS/2
Fisher[2,4] <- F2*(mQD/2 - mRD + mSD/2)
Fisher[3,1] <- Fisher[1,3]
Fisher[3,2] <- Fisher[2,3]
Fisher[3,3] <- mT/2 -mV + mW/2
```

```
        Fisher[3,4] <- F2*(mTD/2 -mVD + mWD/2)
        Fisher[4,1] <- Fisher[1,4]
        Fisher[4,2] <- Fisher[2,4]
        Fisher[4,3] <- Fisher[3,4]
        Fisher[4,4] <- F2*F2*(mTDD/2 -mVDD + mWDD/2)

        DET <- det(Fisher)
        if (DET<0.00000000001) {
            ITER <- 500
            break
        }
        Fisher.inv <- solve(Fisher)
        dif <- Fisher.inv%*%S

        sigma0 <- sigma0 + dif[1]
        dif[2] <- dif[2] * sigma0
        dif[3] <- dif[3] * sigma0
        sigma1 <- sigma1 + dif[2]
        sigma2 <- sigma2 + dif[3]
        rho <- rho + dif[4]

        if(abs(dif[1])<0.00001 & abs(dif[2])<0.00001 & abs(dif[3])<0.00001 &
           abs(dif[4])<0.00001)
            break
    }
    return(list(sigma0,sigma1,sigma2,rho,Fisher.inv,ITER,R))
}
```

20.2.2 R code of BETA.U.individual.autocorr

The R code of the function **BETA.U.individual.autocorr** is listed bellow.

```
####################################################################
### Unit level model with time correlated effects               ###
### SAMPLE project                                              ###
### Author: Laureano Santamaria Arana                           ###
####################################################################

source("Omega.R")

BETA.U.individual.autocorr <- function(X, Y, W, D, md, ndi,
sigma0, sigma1, sigma2, rho) {
    n <- nrow(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)
    B1 <- 0
    B2 <- 0

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            P <-1
        }
        if (d!=1) {
            P <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-P+1

        Wd <- W[P:Fin,P:Fin]
        yd <- Y[P:Fin]
        Xd <- X[P:Fin,]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        OmegadInv <- Inversa_Omegad(md[d], rho)

        sld.inv <- solve(OmegadInv+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd

        UnoNd <-as.matrix(rep(1,Nd))

        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)

        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        B1 <- B1 + t(Xd)%*%Sigmad.inv%*%Xd
        B2 <- B2 + t(Xd)%*%Sigmad.inv%*%yd


    }
    Beta <- solve(B1)%*%B2
    return(Beta)
}
```

20.2.3 R code of mse.individual.autocorr

The R code of the function **mse.individual.autocorr** is listed bellow.

```
source("Omega.R")

mse.individual.autocorr <- function(X, Y, W, D, md, ndi, MXm ,NDI,
MXp, sigma0, sigma1, sigma2, rho, FInv) {

  g1 <- G1(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, rho, NDI)
  g2 <- G2(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, rho, NDI, MXp, MXm)
  g3 <- G3(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, rho, NDI, FInv)
  g4 <- G4(D, md, ndi, NDI, W, sigma0)

  return(g1+g2+2*g3+g4)
}

G1 <- function(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2, rho, NDI) {

    n <- nrow(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0

    # ndi muestral
    # NDI poblacional

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)

    fdi <- ndi/NDI

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        Wd <- W[Pr:Fin,Pr:Fin]
        yd <- Y[Pr:Fin]
        Xd <- X[Pr:Fin,]
        fd <- fdi[Inicio:F]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))
        OmegadInv <- Inversa_Omegad(md[d], rho)
        Omegad <- Calcula_Omegad(md[d], rho)

        sld.inv <- solve(OmegadInv+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        for(k in 1:md[d]) {
            Valor1 <- (1-fd[k])^2
            P1d <- sigma0*F1*Valor1*(1-(F1*t(UnoNd)%*%Sigmad.inv%*%UnoNd))
            P2d <- -sigma0*F1*F2*Valor1*(t(UnoNd)%*%Sigmad.inv%*%D1Nd%*%
                    Omegad%*%Imd[k,])
            Valor2 <- sigma0*F2*Valor1*(t(Imd[k,])%*%Omegad%*%Imd[k,])
            Valor3 <- sigma0*F2*F2*Valor1*(t(Imd[k,])%*%Omegad%*%t(D1Nd)%*%
                      Sigmad.inv%*%D1Nd%*%Omegad%*%Imd[k,])
            P3d <-  Valor2 - Valor3
            if (k==1) {
                g1d <- P1d+P2d+P3d
            }
            if (k!=1) {
                g1d <- c(g1d,P1d+P2d+P3d)
            }
        }
        if (d==1) {
            g1 <- g1d
        }
        if (d!=1) {
            g1 <- c(g1,g1d)
        }
    }
    return(g1)
}

G2 <- function(X, Y, W, D, md, ndi, sigma0, sigma1, sigma2,
rho, NDI, MXp, MXm) {

    n <- nrow(X)
    p <- ncol(X)
    F1 <- sigma1/sigma0
    F2 <- sigma2/sigma0

    # ndi muestral
    # NDI poblacional
    # MXp Media poblacional
    # MXm Media muestral

    M <- sum(md)

    mdcum <- cumsum(md)
    ndcum <- cumsum(ndi)

    fdi <- ndi/NDI
    D1 <- NDI-ndi
    K1 <- NDI/D1
    K2 <- ndi/D1
    Xast <- K1*MXp - K2*MXm

    mR <- 0

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        Wd <- W[Pr:Fin,Pr:Fin]
        yd <- Y[Pr:Fin]
        Xd <- X[Pr:Fin,]
        fd <- fdi[Inicio:F]
        Xdast <- Xast[Inicio:F,]

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }
        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))
        OmegadInv <- Inversa_Omegad(md[d], rho)
        Omegad <- Calcula_Omegad(md[d], rho)

        sld.inv <- solve(OmegadInv+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        mR <- mR + t(Xd)%*%Sigmad.inv%*%Xd
    }
    Q <- sigma0*solve(mR)

    for(d in 1:D) {
        if (d==1) {
            Inicio <-1
            Pr <-1
            F <- md[d]
        }
        if (d!=1) {
            Pr <- (ndcum[mdcum[d-1]]+1)
            Inicio <-mdcum[d-1]+1
            F <- Inicio + md[d] - 1
        }
        Fin  <- ndcum[mdcum[d]]
        Nd <- Fin-Pr+1

        D1Nd <- matrix(0,Nd,md[d])
        i <- 1
        for(k in 1:md[d]) {
            for(j in 1:ndi[Inicio+k-1]) {
                D1Nd[i,k]<-1
                i<- i+1
            }
        }

        Wd <- W[Pr:Fin,Pr:Fin]
        Xd <- X[Pr:Fin,]
        fd <- fdi[Inicio:F]
        Xdast <- Xast[Inicio:F,]

        Imd<-diag(md[d])
        UnoNd <-as.matrix(rep(1,Nd))
        OmegadInv <- Inversa_Omegad(md[d], rho)
        Omegad <- Calcula_Omegad(md[d], rho)

        sld.inv <- solve(OmegadInv+F2*t(D1Nd)%*%Wd%*%D1Nd)
        Ld.inv <- Wd-F2*Wd%*%D1Nd%*%sld.inv%*%t(D1Nd)%*%Wd
        T1 <- F1*Ld.inv%*%UnoNd%*%t(UnoNd)%*%Ld.inv
        T2 <- 1+(F1*t(UnoNd)%*%Ld.inv%*%UnoNd)
        Sigmad.inv <- Ld.inv - (T1/T2[1,1])

        for(k in 1:md[d]) {
            G11d <- (F1*(1-fd[k]))*(1-(F1*t(UnoNd)%*%Sigmad.inv%*%UnoNd))
                    %*%t(UnoNd)%*%Wd%*%Xd
            G12d <- -F1*F2*(1-fd[k])*(t(UnoNd)%*%Sigmad.inv%*%D1Nd%*%
                    Omegad%*%t(D1Nd)%*%Wd%*%Xd)
            G21d <- -F1*F2*(1-fd[k])*(t(Imd[k,])%*%Omegad%*%t(D1Nd)%*%
                    Sigmad.inv%*%UnoNd%*%t(UnoNd)%*%Wd%*%Xd)
            T1   <- Imd - (F2*t(D1Nd)%*%Sigmad.inv%*%D1Nd%*%Omegad)
            G22d <- F2*(1-fd[k])*t(Imd[k,])%*%Omegad%*%T1%*%t(D1Nd)%*%
                    Wd%*%Xd

            if (k==1) {
                G2d <- G11d + G12d + G21d + G22d
            }
            if (k!=1) {
                G2d <- c(G2d, G11d + G12d + G21d + G22d)
            }
        }
        if (d==1) {
            G2 <- G2d
        }
        if (d!=1) {
            G2 <- c(G2, G2d)
        }

    }

    at22 <- matrix(G2,nrow=M,ncol=p,byrow = T)
    at21 <- matrix((1-fdi)*Xast,nrow=M,ncol=p,byrow = T)

    g2 <- diag((at21-at22)%*%Q%*%(t(at21)-t(at22)))

    return(g2)

}

G3 <- function(X, Y, W, D, md, ndi, sigma0, sigma1,
sigma2, rho, NDI, Finv) {

    n <- nrow(X)
```

```
F1 <- sigma1/sigma0
F2 <- sigma2/sigma0

# ndi muestral
# NDI poblacional

mdcum <- cumsum(md)
ndcum <- cumsum(ndi)
fdi <- ndi/NDI

mR <- 0

q<-matrix(0, nrow=4, ncol=4)

for(d in 1:D) {
    if (d==1) {
        Inicio <-1
        Pr <-1
        F <- md[d]
    }
    if (d!=1) {
        Pr <- (ndcum[mdcum[d-1]]+1)
        Inicio <-mdcum[d-1]+1
        F <- Inicio + md[d] - 1
    }
    Fin  <- ndcum[mdcum[d]]
    Nd <- Fin-Pr+1

    Wd <- W[Pr:Fin,Pr:Fin]
    yd <- Y[Pr:Fin]
    Xd <- X[Pr:Fin,]
    fd <- fdi[Inicio:F]

    D1Nd <- matrix(0,Nd,md[d])
    i <- 1
    for(k in 1:md[d]) {
        for(j in 1:ndi[Inicio+k-1]) {
            D1Nd[i,k]<-1
            i<- i+1
        }
    }
    Imd<-diag(md[d])
    UnoNd <-as.matrix(rep(1,Nd))
    OmegadInv <- Inversa_Omegad(md[d], rho)
    Omegad <- Calcula_Omegad(md[d], rho)

    Ads.inv <- solve(OmegadInv+(F2*t(D1Nd)%*%Wd%*%D1Nd))
    Lds.inv <- Wd-(F2*Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd)
```

```
T1 <- F1*Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv
T2 <- 1+(F1*t(UnoNd)%*%Lds.inv%*%UnoNd)
Sigmad.inv <- Lds.inv - (T1/T2[1,1])

T1 <- -Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd
T2 <- F2*Wd%*%D1Nd%*%Ads.inv%*%t(D1Nd)%*%Wd%*%D1Nd%*%
      Ads.inv%*%t(D1Nd)%*%Wd
DpLdsF2 <- T1 + T2

DpLdsRho <- -F2*Wd%*%D1Nd%*%Ads.inv%*%OmegadInv%*%
            t(Omegad)%*%OmegadInv%*%Ads.inv%*%t(D1Nd)%*%Wd

T1 <- Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv
T2 <- 1+(F1*t(UnoNd)%*%Lds.inv%*%UnoNd)
DpSdsF1 <- (T1/(T2[1,1]^2))

T0 <- as.numeric((F1*F1*t(UnoNd)%*%DpLdsF2%*%UnoNd))
T1 <- (T0*(Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv))/(T2[1,1]^2)
T3 <- (DpLdsF2%*%UnoNd%*%t(UnoNd)%*%Lds.inv) +
      (Lds.inv%*%UnoNd%*%t(UnoNd)%*%DpLdsF2)
DpSdsF2 <- DpLdsF2 + T1 - (F1*T3)/(T2[1,1])

T0 <- as.numeric((F1*F1*t(UnoNd)%*%DpLdsRho%*%UnoNd))
T1 <- (T0*(Lds.inv%*%UnoNd%*%t(UnoNd)%*%Lds.inv))/(T2[1,1]^2)
T3 <- (DpLdsRho%*%UnoNd%*%t(UnoNd)%*%Lds.inv) +
      (Lds.inv%*%UnoNd%*%t(UnoNd)%*%DpLdsRho)
DpSdsRho <- DpLdsRho + T1 - (F1*T3)/(T2[1,1])

Sigmads <- solve(Sigmad.inv)
for(k in 1:md[d]) {
    DpB1dF1 <- (1-fd[k])*t(UnoNd)%*%(Sigmad.inv + F1*DpSdsF1)
    DpB1dF2 <- F1*(1-fd[k])*t(UnoNd)%*%DpSdsF2
    DpB1dRho <- F1*(1-fd[k])*t(UnoNd)%*%DpSdsRho

    DpB2dF1 <- F2*(1-fd[k])*Imd[k,]%*%Omegad%*%t(D1Nd)%*%DpSdsF1
    DpB2dF2 <- (1-fd[k])*Imd[k,]%*%Omegad%*%t(D1Nd)%*%
               (Sigmad.inv + F2*DpSdsF2)
    DpB2dRho <- F2*(1-fd[k])*Imd[k,]%*%(OmegadInv%*%t(D1Nd)%*%
                Sigmad.inv + Omegad%*%t(D1Nd)%*%Sigmad.inv)

    T2 <- (DpB1dF1+DpB2dF1)
    T3 <- (DpB1dF2+DpB2dF2)
    T4 <- (DpB1dRho+DpB2dRho)

    q[1,1] <- 0
    q[1,2] <- 0
    q[1,3] <- 0
    q[1,4] <- 0
    q[2,1] <- 0
```

```
            q[2,2] <- T2%*%(sigma0*Sigmads)%*%t(T2)
            q[2,3] <- T2%*%(sigma0*Sigmads)%*%t(T3)
            q[2,4] <- T2%*%(sigma0*Sigmads)%*%t(T4)
            q[3,1] <- 0
            q[3,2] <- T3%*%(sigma0*Sigmads)%*%t(T2)
            q[3,3] <- T3%*%(sigma0*Sigmads)%*%t(T3)
            q[3,4] <- T3%*%(sigma0*Sigmads)%*%t(T4)
            q[4,1] <- 0
            q[4,2] <- T4%*%(sigma0*Sigmads)%*%t(T2)
            q[4,3] <- T4%*%(sigma0*Sigmads)%*%t(T3)
            q[4,4] <- T4%*%(sigma0*Sigmads)%*%t(T4)

            if (k==1) {
                g3d <- g3dparcial
             }
            if (k!=1) {
                g3d <- c(g3d,g3dparcial)
            }
        }

        if (d==1) {
            g3 <- g3d
        }
        if (d!=1) {
            g3 <- c(g3,g3d)
        }

    }

    return(g3)

}


G4 <- function(D, md, ndi, NDI, W, sigma0) {


    # ndi muestral
    # NDI poblacional

    g4 <- vector()

    Indice <- 1
    Inicio <- 1
    for(d in 1:D) {
        for (j in 1:md[d]) {
            g4[Indice] <- (sigma0/(NDI[Indice]^2))*
                              (NDI[Indice]-ndi[Indice])
            Indice <- Indice + 1
        }
    }
    return(g4)
}
```

20.2.4 R code of Interval.autocorr

The R code of the function **Interval.autocorr** is listed bellow.

```
##################################################################
### Unit level model with time correlated effects              ###
### SAMPLE project                                             ###
### Author: Laureano Santamaria Arana                          ###
##################################################################

Interval.autocorr <- function(fit, conf=0.95) {
    alfa <- 1-conf
    k <- 1-alfa/2
    z <- qnorm(k)
    Finv <- fit[[2]]
    sigma.std.err <- z*sqrt(Finv[1,1])
    sigma1.std.err <- z*sqrt(Finv[2,2])
    sigma2.std.err <- z*sqrt(Finv[3,3])
    rho.std.err <- z*sqrt(Finv[4,4])
    beta.std.err <- z*sqrt(as.vector(diag(fit[[3]])))
    return( list(sigma.std.err, sigma1.std.err, sigma2.std.err,
            rho.std.err, beta.std.err) )
}
```

20.2.5 R code of Omega calculation

```
##################################################################
### Unit level model with time correlated effects              ###
### SAMPLE project                                             ###
### Author: Laureano Santamaria Arana                          ###
##################################################################

Calcula_Omegad <- function(Elem, rho) {
    Omegad<-matrix(0,nrow=Elem,ncol=Elem)
    Omegad[lower.tri(Omegad)]<-rho^sequence((Elem-1):1)
    Omegad<-Omegad+t(Omegad)
    diag(Omegad)<-1
    Omegad <- (1/(1-rho^2))*Omegad
    return(as.matrix(Omegad))
}

Deriva_Omegad <- function(Elem, rho, Omegad) {
    OmegadPrima<-matrix(0,nrow=Elem,ncol=Elem)
    OmegadPrima[lower.tri(OmegadPrima)] <- sequence((Elem-1):1)*
                                    rho^(sequence((Elem-1):1)-1)
    OmegadPrima <- OmegadPrima+t(OmegadPrima)
    OmegadPrima <- (1/(1-rho^2))*OmegadPrima
    OmegadPrima <- OmegadPrima + (2*rho/(1-rho^2))*Omegad

    return(as.matrix(OmegadPrima))
}

Inversa_Omegad <- function(Elem, rho) {
    Imd <- diag(Elem)
    E <- diag(Elem)
    E[1,1] <- 0
    E[Elem,Elem] <- 0
    F <- diag(Elem)
    for(i in 1:Elem) {
        F[i,i] <- 0
        if (i>1) {
            F[i,i-1] <- 1
        }
        if (i<Elem) {
            F[i,i+1] <- 1
        }
    }
    OmegadInversa <- matrix(0,nrow=Elem,ncol=Elem)
    OmegadInversa <- Imd + (rho*rho*E) - (rho*F)
    return(as.matrix(OmegadInversa))
}
```

Chapter 21

Appendix 7: R code for M-quantile small area estimators of the mean

21.1 R code of mq.sae

The R code of the function **mq.sae** is listed bellow.

```
###################################################################
###
###             M-quantile estimators for the mean
###                              SAMPLE project
###
### Authors: N. Salvati, N.Tzavidis, C. Giusti,
###          S. Marchetti and M. Pratesi
### File name: MQ_FUNCTION_MEAN.R
### Updated: February 2nd, 2010
###
###################################################################

library(MASS)

#M-quantile function
QRLM <- function (x, y, case.weights = rep(1, nrow(x)),
var.weights = rep(1, nrow(x)), w = rep(1, nrow(x)), init = "ls",
psi = psi.huber, scale.est = c("MAD", "Huber", "proposal 2"),
k2 = 1.345, method = c("M", "MM"), maxit = 20, acc = 1e-04,
test.vec = "resid", q = 0.5)
{
irls.delta <- function(old,new)
              sqrt(sum((old-new)^2)/max(1e-20,sum(old^2)))
irls.rrxwr <- function(x, w, r) {
w <- sqrt(w)
max(abs((matrix(r*w,1,length(r))%*% x)/sqrt(matrix(w,1,length(r)) %*%
%*% (x^2))))/sqrt(sum(w*r^2))
}
```

```
method <- match.arg(method)
nmx <- deparse(substitute(x))
if (is.null(dim(x))) {
x <- as.matrix(x)
colnames(x) <- nmx
}
else x <- as.matrix(x)
if (is.null(colnames(x)))
colnames(x) <- paste("X", seq(ncol(x)), sep = "")
if (qr(x)$rank < ncol(x))
stop("x is singular: singular fits are not implemented in rlm")
if (!(any(test.vec == c("resid", "coef", "w", "NULL")) ||
is.null(test.vec)))
stop("invalid testvec")
if (length(var.weights) != nrow(x))
stop("Length of var.weights must equal number of observations")
if (any(var.weights < 0))
stop("Negative var.weights value")
if (length(case.weights) != nrow(x))
stop("Length of case.weights must equal number of observations")
w <- (w * case.weights)/var.weights
if (method == "M") {
scale.est <- match.arg(scale.est)
if (!is.function(psi))
psi <- get(psi, mode = "function")
arguments <- list(...)
if (length(arguments)) {
pm <- pmatch(names(arguments), names(formals(psi)), nomatch = 0)
if (any(pm == 0))
warning(paste("some of ... do not match"))
pm <- names(arguments)[pm > 0]
formals(psi)[pm] <- unlist(arguments[pm])
}
if (is.character(init)) {
if (init == "ls")
temp <- lm.wfit(x, y, w, method = "qr")
else if (init == "lts")
temp <- lqs.default(x, y, intercept = FALSE, nsamp = 200)
else stop("init method is unknown")
coef <- temp$coef
resid <- temp$resid
}
else {
if (is.list(init))
coef <- init$coef
else coef <- init
resid <- y - x %*% coef
}
}
```

```
else if (method == "MM") {
scale.est <- "MM"
temp <- lqs.default(x, y, intercept=FALSE, method="S", k0=1.548)
coef <- temp$coef
resid <- temp$resid
psi <- psi.bisquare
if (length(arguments <- list(...)))
if (match("c", names(arguments), nomatch = FALSE)) {
c0 <- arguments$c
if (c0 > 1.548) {
psi$c <- c0
}
else warning("c must be at least 1.548 and has been ignored")
}
scale <- temp$scale
}
else stop("method is unknown")
done <- FALSE
conv <- NULL
n1 <- nrow(x) - ncol(x)
if (scale.est != "MM")
scale <- mad(resid/sqrt(var.weights), 0)
theta <- 2 * pnorm(k2) - 1
gamma <- theta + k2^2 * (1 - theta) - 2 * k2 * dnorm(k2)
qest <- matrix(0, nrow = ncol(x), ncol = length(q))
qwt <- matrix(0, nrow = nrow(x), ncol = length(q))
qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
qres <- matrix(0, nrow = nrow(x), ncol = length(q))
for(i in 1:length(q)) {
for (iiter in 1:maxit) {
if (!is.null(test.vec))
testpv <- get(test.vec)
if (scale.est != "MM") {
if (scale.est == "MAD")
scale <- median(abs(resid/sqrt(var.weights)))/0.6745
else scale <- sqrt(sum(pmin(resid^2/var.weights,
              (k2*scale)^2))/(n1*gamma))
if (scale == 0) {
done <- TRUE
break
}
}
w <- psi(resid/(scale * sqrt(var.weights))) * case.weights
ww <- 2 * (1 - q[i]) * w
ww[resid > 0] <- 2 * q[i] * w[resid > 0]
w <- ww
temp <- lm.wfit(x, y, w, method = "qr")
coef <- temp$coef
resid <- temp$residuals
```

```
if (!is.null(test.vec))
convi <- irls.delta(testpv, get(test.vec))
else convi <- irls.rrxwr(x, wmod, resid)
conv <- c(conv, convi)
done <- (convi <= acc)
if (done)
break
}
if (!done)
warning(paste("rlm failed to converge in",maxit,"steps at q=",q[i]))
qest[, i] <- coef
qwt[, i] <- w
qfit[, i] <- temp$fitted.values
qres[,i] <- resid
}
list(fitted.values = qfit, residuals = qres, q.values = q,
q.weights = qwt, coefficients = qest)
}

# COMPUTE THE QUANTILE ORDER

# COMPUTING OF THE QUANTILE-ORDERS
"zerovalinter"<-function(y, x)
{
        if(min(y) > 0) {
                xmin <- x[y == min(y)]
                if(length(xmin) > 0)
                        xmin <- xmin[length(xmin)]
                xzero <- xmin
        }

else {
                if(max(y) < 0) {
                        xmin <- x[y == max(y)]
                        if(length(xmin) > 0)
                                xmin <- xmin[1]
                        xzero <- xmin
                }
                else {
                        y1 <- min(y[y > 0])
                        if(length(y1) > 0)
                                y1 <- y1[length(y1)]
                        y2 <- max(y[y < 0])
                        if(length(y2) > 0)
                                y2 <- y2[1]
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}

# Function for Finding the Quantile Orders by Linear Interpolation
# Assumes that "zerovalinter" function has been already loaded

"gridfitinter"<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y by
# interpolation
{
nq<-length(Q)
  diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
 vectordest <- apply(diff, 1, zerovalinter,Q)

#print(vectordest)
#qord<-list(ord=c(vectordest))
#qord
}


#y: study variable
#x: set of covariates without the intercept for sampled units
#regioncode.s: area code for sampled units
#x.r: set of covariates for out of sample units
#regioncode.r: area code for out of sample units
#p size of x +1 (intercept)


mq.sae=function(y,x,regioncode.s,m,p,x.outs,regioncode.r,
tol.value=0.0001,maxit.value=100,k.value=1.345)

{

MQE<-c(rep(0,m))
MQNAIVE<-c(rep(0,m))


datanew=cbind(y,x,regioncode.s)

ni=as.numeric(table(regioncode.s))
```

```
sample.sizer<-as.numeric(table(regioncode.r))

Ni=sample.sizer+ni


N<-sum(Ni)
n<-sum(ni)

x=matrix(x,n,p-1)
x.r=matrix(x.outs,(N-n),p-1)
x.t=rbind(x,x.r)
x.c<-rep(1,n)
x.design<-cbind(x.c,x)

p=ncol(x.design)

ob<-QRLM(x.design, y,q=sort(c(seq(0.006,0.99,0.045),0.5,0.994,
0.01,0.02,0.96,0.98)),k = k.value,maxit=maxit.value,acc=tol.value)

qo<-matrix(c(gridfitinter(y,ob$fitted.values,ob$q.values)),
           nrow=n,ncol=1)

qmat<-matrix(c(qo,regioncode.s),nrow=sum(ni),ncol=2)

mqo<-aggregate(qmat[,1],list(d2=qmat[,2]),mean)[,2]

uar<-sort(unique(regioncode.s))
saq<-matrix(c(mqo,uar),nrow=m,ncol=2)

saq<-rbind(saq,c(0.5,9999))

beta.stored=matrix(0,m,2)
res.s=NULL
tttmp1<-NULL
ci<-array(rep(0,n*m),dim=c(n,m,1))
ci1<-array(rep(0,n*m),dim=c(n,m,1))
prs<-NULL
prr<-NULL
wi<-matrix(0,n,m)

for(i in 1:m){

ob1<-QRLM(x.design,y,q=mqo[i],psi=psi.huber,k = k.value,
maxit=maxit.value,acc=tol.value)

wd<-diag(c(ob1$q.weights))

# Regional parameters from multiquantile model
```

```
coef<-matrix(c(t(ob1$coefficients)),nrow=1,ncol=p)
# need to be ordered by area

coef<-t(coef)

meat<-wd%*%x.design%*%solve(t(x.design)%*%wd%*%x.design)

x1<-c(rep(1,(Ni[i]-ni[i])))

ir<-rep(0,n)

ir[regioncode.s==uar[i]]<-1

rj1<-sample.sizer[i]

r=NULL

for (kk in 1:(p-1))
{
r<-c(r,sum(x.r[,kk][regioncode.r==uar[i]]))}

r=c(rj1,r)


sj1<-sum(rep(1,ni[i]))

tss=NULL

for (kk in 1:(p-1))
{
tss<-c(tss,sum(x[,kk][regioncode.s==uar[i]]))}

tss<-c(sj1,tss)

w.welsh<-((Ni[i])/(ni[i]))*ir+meat%*%(r-((Ni[i]-ni[i])/ni[i])*tss)

MQE[i]<-sum(w.welsh*y)/sum(w.welsh)
y.i<-y[regioncode.s==uar[i]]

y.pred.s<-cbind(1,x[regioncode.s==uar[i],])%*%coef
residual<-y.i-y.pred.s

tttmp1[i]<-(sample.sizer[i]/ni[i])*sum(residual)

prs<-c(prs,y.pred.s)
res.s<-c(res.s,residual)
y.pred<-cbind(1,x.r[regioncode.r==uar[i],])%*%coef
prr<-c(prr,y.pred)
```

```
data<-cbind(regioncode.s,w.welsh)

for (kk in 1:n){

if (data[kk,1]==uar[i]) ci[kk,i,1]<-data[kk,2]-1
else if (data[kk,1]!=uar[i]) ci[kk,i,1]<-data[kk,2]  }

MQNAIVE[i]<-(1/Ni[i])*as.real(sum(y.i)+sum(y.pred))

#f<-(sample.sizer[i])/ni[i]
ai<-r
bi<-ai%*%solve(t(x.design)%*%wd%*%x.design)%*%t(x.design)%*%wd
bi<-c(bi)
wi[,i]<-c(ir+bi)

datanaive<-cbind(regioncode.s,wi[,i])
for (kk in 1:n){
if (datanaive[kk,1]==uar[i]) ci1[kk,i,1]<-datanaive[kk,2]-1
else if (datanaive[kk,1]!=uar[i]) ci1[kk,i,1]<-datanaive[kk,2]  }

}

res.d=res.s^2

res.d<-cbind(res.d,regioncode.s,ci[,,1])

v<-NULL
for (oo in 1:m){
v[oo]<-1/Ni[oo]^2*(sum((res.d[,(oo+2)][res.d[,2]==uar[oo]]^2+
+(sample.sizer[oo])/ni[oo])*res.d[,1][res.d[,2]==uar[oo]])+
+sum(res.d[,(oo+2)][res.d[,2]!=uar[oo]]^2*res.d[,1][res.d[,2]!=uar[oo]]))

}

res.d1<-cbind(res.s^2,regioncode.s,ci1[,,1])
v1<-NULL
bias<-NULL
mse<-NULL
for (oo in 1:m){
v1[oo]<-1/Ni[oo]^2*(sum((res.d1[,(oo+2)][res.d1[,2]==uar[oo]]^2+
+(sample.sizer[oo])/n)*res.d1[,1][res.d1[,2]==uar[oo]])+
+sum(res.d1[,(oo+2)][res.d1[,2]!=uar[oo]]^2*res.d1[,1]
     [res.d1[,2]!=uar[oo]]))

bias[oo]<-(1/Ni[oo])*(sum(wi[,oo]*prs)-sum(c(prs[regioncode.s==uar[oo]],
prr[regioncode.r==uar[oo]])))

mse[oo]<-v1[oo]+(bias[oo])^2
```

```
}

list(mq.cd=MQE,mq.naive=MQNAIVE,mse.cd=v,mse.naive=mse,code.area=uar)
}
```

Chapter 22

Appendix 8: R code for nonparametric M-quantile small area estimators of the mean

22.1 R code of npmq.sae

The R code of the function **npmq.sae** is listed bellow.

```
##################################################################
###
###              M-quantile estimators for the mean
###                                SAMPLE project
###
### Authors: N. Salvati, N.Tzavidis, C. Giusti,
###          S. Marchetti and M. Pratesi
### File name: npmq.sae.R
### Updated: February 10th, 2010
###
##################################################################

# Model-Based Simulations
#rm(list=ls(all=TRUE))
library(MASS)
library(SemiPar)
library(nlme)

my.default.knots.2D<-function (x1, x2, num.knots)
{
    require("cluster")
    if (missing(num.knots))
        num.knots <- max(10, min(50, round(length(x1)/4)))
    X <- cbind(x1, x2)
    dup.inds <- (1:nrow(X))[dup.matrix(X) == T]
    if (length(dup.inds) > 0)
```

```
        X <- X[-dup.inds, ]
    knots <- clara(X, num.knots)$medoids
    return(knots)
}
#
#
#
Z.matrix<-function(lon,lat,knots){
K<-nrow(knots)
dist.knot<-matrix(0,K,K)
dist.knot[lower.tri(dist.knot)]<-dist(knots)
dist.knot<-dist.knot+t(dist.knot)
Omega<-tps.cov(dist.knot)
dist.lon<-outer(lon,knots[,1],"-")
dist.lat<-outer(lat,knots[,2],"-")
dist.x<-sqrt(dist.lon^2+dist.lat^2)
svd.Omega<-svd(Omega)
sqrt.Omega<-t(svd.Omega$v %*% (t(svd.Omega$u) * sqrt(svd.Omega$d)))
Z<- t(solve(sqrt.Omega,t(tps.cov(dist.x))))
return(Z)
}

npqrlm<-function(Y.n,X,Z.spline,quantile,tol=0.001,maxit=100,theta=2,
kk=1.345){

# prototype function for panel data fitting of MQ models
# the matrix X is assumed to contain an intercept
# the vector s is a strata indicator assumed (so far) to be a one-way layout
# theta : GCV parameter, defaulted to 2

require(SemiPar)
require(MASS)
require(splines)
assign("tol",tol,pos=1)
assign("maxit",maxit,pos=1)
assign("Y.n",Y.n,pos=1)
assign("X",X,pos=1)
assign("kk",kk,pos=1)
assign("Z.spline",Z.spline,pos=1)
assign("theta",theta,pos=1)
assign("quantile",quantile,pos=1)

n<-length(Y.n)
X<-as.matrix(X)
p1<-ncol(X)
p2<-ncol(Z.spline)
X.n<-cbind(X,Z.spline)
p=p1+p2
```

```
b=rep(1,p)

my.psi.q<-function(u,q,c){
s<-median(abs(u))/0.6745
w <- psi.huber((u/s),c)
ww <- 2 * (1 - q) * w
ww[u> 0] <- 2 * q * w[u > 0]
w <- ww
w*u
}

my.b<-function(X,Y,W,lambda)
{G<-as.matrix(diag(c(rep(0,p1),rep(1,p2)),p))
solve(t(X)%*%W%*%X+G*lambda)%*%t(X)%*%W%*%Y
}


stima<-function(l){
# l : coefficiente di penalizzazione nell'IRPLS
n<-nrow(X.n)
diff<-1
iter<-0
while (diff>tol)
{#inizia procedura di stima
iter<-iter+1
res<-Y.n-X.n%*%b #calcolo residui
W.n<-as.matrix(diag(c(my.psi.q(as.matrix(res),qtl,kk)/as.matrix(res)),n))
assign("W.n", W.n, pos=1)
b.ott<-my.b(X.n,Y.n,W.n,l)
diff<-sum((as.matrix(b)-as.matrix(b.ott))^2)
b<-b.ott
if (iter>maxit)
{warning(paste("failed to converge in", maxit, "steps at q = ", qtl))
break}
}
y.hat=X.n%*%b
list(fitted.values=as.matrix(y.hat),coef=as.matrix(b),
we=as.matrix(diag(W.n)))
}

my.GCV<-function(l)
{
G<-as.matrix(diag(c(rep(0,p1),rep(1,p2)),p))
tmp<-stima(l)
y.hat<-tmp$fitted.values
S<-(X.n)%*%solve(t(X.n)%*%W.n%*%X.n+G*l)%*%t(X.n)%*%W.n
sum((Y.n-y.hat)^2)/((1-theta*sum(diag(S))/n)^2)
}
```

```
length.q<-length(quantile)
y.fit<-matrix(0,n,length.q)
y.coef<-matrix(0,p,length.q)
y.weight<-matrix(0,n,length.q)
lambda.ott<-NULL
for (k in 1:length.q)
{
qtl<-quantile[k]
qtl<-assign("qtl",qtl,pos=1)
tmp<-optimize(my.GCV,c(0,50))
l.ott<-tmp$minimum
lambda.ott[k]<-l.ott
y.stim<-stima(l.ott)
y.fit[,k]<-y.stim$fitted.values
y.coef[,k]=y.stim$coef
y.weight[,k]=y.stim$we
}
list(hat.values=y.fit,b.stim=y.coef,q.weights=y.weight,lambda.q=lambda.ott)
}

# COMPUTE THE QUANTILE ORDER

# COMPUTING OF THE QUANTILE-ORDERS
"zerovalinter"<-function(y, x)
{
        if(min(y) > 0) {
                xmin <- x[y == min(y)]
                if(length(xmin) > 0)
                        xmin <- xmin[length(xmin)]
                xzero <- xmin
        }



else {
                if(max(y) < 0) {
                        xmin <- x[y == max(y)]
                        if(length(xmin) > 0)
                                xmin <- xmin[1]
                        xzero <- xmin
                }
                else {
                        y1 <- min(y[y > 0])
                        if(length(y1) > 0)
                                y1 <- y1[length(y1)]
                        y2 <- max(y[y < 0])
                        if(length(y2) > 0)
                                y2 <- y2[1]
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}

# Function for Finding the Quantile Orders by Linear Interpolation
# Assumes that "zerovalinter" function has been already loaded

"gridfitinter"<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y
by interpolation
{
nq<-length(Q)
  diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
 vectordest <- apply(diff, 1, zerovalinter,Q)

#print(vectordest)
#qord<-list(ord=c(vectordest))
#qord
}


#y: study variable
#x: set of covariates without the intercept for sampled units
#regioncode.s: area code for sampled units
#x.r: set of covariates for out of sample units
#regioncode.r: area code for out of sample units
#p size of x +1 (intercept)
#z.spline: the spline matrix
sae.npmq=function(y,x,z.spline,z.spline.r,regioncode.s,m,p,x.outs,
regioncode.r,tol.value=0.0001,maxit.value=100,k.value=1.345)
{

MQE<-c(rep(0,m))
MQNAIVE<-c(rep(0,m))


datanew=cbind(y,x,regioncode.s)
```

```
ni=as.numeric(table(regioncode.s))

sample.sizer<-as.numeric(table(regioncode.r))

Ni=sample.sizer+ni


N<-sum(Ni)
n<-sum(ni)

x=matrix(x,n,p-1)
x.r=matrix(x.outs,(N-n),p-1)
x.t=rbind(x,x.r)
x.c<-rep(1,n)
x.design<-cbind(x.c,x)
p=ncol(x.design)
p2<-ncol(z.spline)
G.matrix=diag(c(rep(0,p),rep(1,p2)))

ob<-npqrlm(y, x.design, z.spline, quantile=sort(c(seq(0.006,0.99,0.045),
0.5,0.994,0.01,0.02,0.96,0.98)),kk = k.value,maxit=maxit.value,tol=tol.value)

q.values<-sort(c(seq(0.006,0.99,0.045),0.5,0.994,0.01,0.02,0.96,0.98))

qo<-matrix(c(gridfitinter(y,ob$hat.values,q.values)),nrow=n,ncol=1)

qmat<-matrix(c(qo,regioncode.s),nrow=sum(ni),ncol=2)

mqo<-aggregate(qmat[,1],list(d2=qmat[,2]),mean)[,2]

uar<-sort(unique(regioncode.s))
saq<-matrix(c(mqo,uar),nrow=m,ncol=2)

saq<-rbind(saq,c(0.5,9999))

beta.stored=matrix(0,m,2)
res.s=NULL
tttmp1<-NULL
ci<-array(rep(0,n*m),dim=c(n,m,1))
ci1<-array(rep(0,n*m),dim=c(n,m,1))
prs<-NULL
prr<-NULL
wi<-matrix(0,n,m)

for(i in 1:m){


ob1<-npqrlm(y, x.design, z.spline, quantile=mqo[i],kk = k.value,
maxit=maxit.value,tol=tol.value)
```

```
wd<-diag(c(ob1$q.weights))

# Regional parameters from multiquantile model

coef<-matrix(c(t(ob1$b.stim)),nrow=1,ncol=p+p2) # need to be ordered by area

coef<-t(coef)

meat<-wd%*%cbind(x.design,z.spline)%*%solve(t(cbind(x.design,z.spline))
%*%wd%*%cbind(x.design,z.spline)+ob1$lambda.q*G.matrix)

x1<-c(rep(1,(Ni[i]-ni[i])))

ir<-rep(0,n)

ir[regioncode.s==uar[i]]<-1

rj1<-sample.sizer[i]

r=NULL

for (kk in 1:(p-1))
{
r<-c(r,sum(x.r[,kk][regioncode.r==uar[i]]))}

r=c(rj1,r,c(as.numeric(apply(z.spline.r[regioncode.r==uar[i],],2,sum))))


sj1<-sum(rep(1,ni[i]))

tss=NULL

for (kk in 1:(p-1))
{
tss<-c(tss,sum(x[,kk][regioncode.s==uar[i]]))}


tss<-c(sj1,tss,c(as.numeric(apply(z.spline[regioncode.s==uar[i],],2,sum))))

w.welsh<-((Ni[i])/(ni[i]))*ir+meat%*%(r-((Ni[i]-ni[i])/ni[i])*tss)


MQE[i]<-sum(w.welsh*y)/sum(w.welsh)
y.i<-y[regioncode.s==uar[i]]

y.pred.s<-cbind(1,x[regioncode.s==uar[i],],z.spline[regioncode.s==uar[i],])
%*%coef
```

```
residual<-y.i-y.pred.s

tttmp1[i]<-(sample.sizer[i]/ni[i])*sum(residual)

prs<-c(prs,y.pred.s)
res.s<-c(res.s,residual)
y.pred<-cbind(1,x.r[regioncode.r==uar[i],],z.spline.r[regioncode.r==uar[i],])
%*%coef
prr<-c(prr,y.pred)

data<-cbind(regioncode.s,w.welsh)

for (kk in 1:n){

if (data[kk,1]==uar[i]) ci[kk,i,1]<-data[kk,2]-1
else if (data[kk,1]!=uar[i]) ci[kk,i,1]<-data[kk,2]  }

MQNAIVE[i]<-(1/Ni[i])*as.real(sum(y.i)+sum(y.pred))

#f<-(sample.sizer[i])/ni[i]
ai<-r
bi<-ai%*%solve(t(cbind(x.design,z.spline))%*%wd%*%cbind(x.design,z.spline))
%*%t(cbind(x.design,z.spline))%*%wd
bi<-c(bi)
wi[,i]<-c(ir+bi)

datanaive<-cbind(regioncode.s,wi[,i])
for (kk in 1:n){
if (datanaive[kk,1]==uar[i]) ci1[kk,i,1]<-datanaive[kk,2]-1
else if (datanaive[kk,1]!=uar[i]) ci1[kk,i,1]<-datanaive[kk,2]  }



}

res.d=res.s^2

res.d<-cbind(res.d,regioncode.s,ci[,,1])

v<-NULL
for (oo in 1:m){
v[oo]<-1/Ni[oo]^2*(sum((res.d[,(oo+2)][res.d[,2]==uar[oo]]^2+
(sample.sizer[oo])/n)*res.d[,1][res.d[,2]==uar[oo]])+
sum(res.d[,(oo+2)][res.d[,2]!=uar[oo]]^2*res.d[,1][res.d[,2]!=uar[oo]]))

}

res.d1<-cbind(res.s^2,regioncode.s,ci1[,,1])
v1<-NULL
```

```
bias<-NULL
mse<-NULL
for (oo in 1:m){
v1[oo]<-1/Ni[oo]^2*(sum((res.d1[,(oo+2)][res.d1[,2]==uar[oo]]^2+
(sample.sizer[oo])/n)*res.d1[,1][res.d1[,2]==uar[oo]])+sum(res.d1[,(oo+2)]
[res.d1[,2]!=uar[oo]]^2*res.d1[,1][res.d1[,2]!=uar[oo]]))

bias[oo]<-(1/Ni[oo])*(sum(wi[,oo]*prs)-sum(c(prs[regioncode.s==uar[oo]],
prr[regioncode.r==uar[oo]])))
mse[oo]<-v1[oo]+(bias[oo])^2
}


list(npmq.cd=MQE,npmq.naive=MQNAIVE,mse.cd=v,mse.naive=mse,code.area=uar)
}
```

Chapter 23

Appendix 9: R code for M-quantile Geographically Weighted Regression

23.1 R code of mqgwr.sae

The R code of the function **mqgwr.sae** is listed bellow.

```
##############################################################
###
###             M-quantile GWR estimators for the mean
###                              SAMPLE project
###
### Authors: N. Salvati, N.Tzavidis, C. Giusti,
###          S. Marchetti and M. Pratesi
### File name: mqgwr-R
### Updated: March 17th, 2010
###
################################################################

library(MASS)
library(nlme)
library(sp)
library(spgwr)

#M-estimator with GWR weights

QRLM <- function (x, y, case.weights = rep(1, nrow(x)),
var.weights = rep(1, nrow(x)), ..., w = rep(1, nrow(x)),
init="ls", psi=psi.huber, scale.est=c("MAD", "Huber", "proposal 2"),
k2 = 1.345, method = c("M", "MM"), maxit = 20, acc = 1e-04,
test.vec = "resid", q = 0.5,w1)

{
irls.delta <- function(old,new)sqrt(sum((old-new)^2)/
                       max(1e-20,sum(old^2)))
irls.rrxwr <- function(x, w, r) {
w <- sqrt(w)
max(abs((matrix(r*w,1,length(r))%*% x)/sqrt(matrix(w,1,length(r)) %*%
%*% (x^2))))/sqrt(sum(w*r^2))
}
method <- match.arg(method)
nmx <- deparse(substitute(x))
if (is.null(dim(x))) {
x <- as.matrix(x)
colnames(x) <- nmx
}
else x <- as.matrix(x)
if (is.null(colnames(x)))
colnames(x) <- paste("X", seq(ncol(x)), sep = "")
if (qr(x)$rank < ncol(x))
stop("x is singular: singular fits are not implemented in rlm")
if (!(any(test.vec == c("resid", "coef", "w", "NULL")) ||
    is.null(test.vec)))
stop("invalid testvec")
if (length(var.weights) != nrow(x))
stop("Length of var.weights must equal number of observations")
if (any(var.weights < 0))
stop("Negative var.weights value")
if (length(case.weights) != nrow(x))
stop("Length of case.weights must equal number of observations")
w <- (w * case.weights)/var.weights
if (method == "M") {
scale.est <- match.arg(scale.est)
if (!is.function(psi))
psi <- get(psi, mode = "function")
arguments <- list(...)
if (length(arguments)) {
pm <- pmatch(names(arguments), names(formals(psi)), nomatch = 0)
if (any(pm == 0))
warning(paste("some of ... do not match"))
pm <- names(arguments)[pm > 0]
formals(psi)[pm] <- unlist(arguments[pm])
}
if (is.character(init)) {
if (init == "ls")
temp <- lm.wfit(x, y, w, method = "qr")
else if (init == "lts")
temp <- lqs.default(x, y, intercept = FALSE, nsamp = 200)
else stop("init method is unknown")
coef <- temp$coef
resid <- temp$resid
}
else {
```

```
if (is.list(init))
coef <- init$coef
else coef <- init
resid <- y - x %*% coef
}
}
else if (method == "MM") {
scale.est <- "MM"
temp <- lqs.default(x, y, intercept=FALSE, method="S", k0=1.548)
coef <- temp$coef
resid <- temp$resid
psi <- psi.bisquare
if (length(arguments <- list(...)))
if (match("c", names(arguments), nomatch = FALSE)) {
c0 <- arguments$c
if (c0 > 1.548) {
psi$c <- c0
}
else warning("c must be at least 1.548 and has been ignored")
}
scale <- temp$scale
}
else stop("method is unknown")
done <- FALSE
conv <- NULL
n1 <- nrow(x) - ncol(x)
if (scale.est != "MM")
scale <- mad(resid/sqrt(var.weights), 0)
theta <- 2 * pnorm(k2) - 1
gamma <- theta + k2^2 * (1 - theta) - 2 * k2 * dnorm(k2)
qest <- matrix(0, nrow = ncol(x), ncol = length(q))
qwt <- matrix(0, nrow = nrow(x), ncol = length(q))
qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
qres <- matrix(0, nrow = nrow(x), ncol = length(q))
for(i in 1:length(q)) {
for (iiter in 1:maxit) {
if (!is.null(test.vec))
testpv <- get(test.vec)
if (scale.est != "MM") {
if (scale.est == "MAD")
scale <- median(abs(resid/sqrt(var.weights)))/0.6745
else scale <- sqrt(sum(pmin(resid^2/var.weights,(k2*scale)^2))/
              (n1*gamma))
if (scale == 0) {
done <- TRUE
break
}
}
w <- psi(resid/(scale * sqrt(var.weights))) * case.weights
```

```
ww <- 2 * (1 - q[i]) * w
ww[resid > 0] <- 2 * q[i] * w[resid > 0]
w <- ww*diag(w1)
temp <- lm.wfit(x, y, w, method = "qr")
coef <- temp$coef
resid <- temp$residuals
if (!is.null(test.vec))
convi <- irls.delta(testpv, get(test.vec))
else convi <- irls.rrxwr(x, wmod, resid)
conv <- c(conv, convi)
done <- (convi <= acc)
if (done)
break
}
if (!done)
warning(paste("rlm failed to converge in",maxit,"steps at q = ",q[i]))
qest[, i] <- coef
qwt[, i] <- w
qfit[, i] <- temp$fitted.values
qres[,i] <- resid
}
list(fitted.values = qfit, residuals = qres, q.values = q,
q.weights = qwt, coefficients = qest)
}

# COMPUTE THE QUANTILE ORDER
#
zerovalinter<-function(y, x)
{
       if(min(y) > 0) {
               xmin <- x[y == min(y)]
               if(length(xmin) > 0)
                       xmin <- xmin[length(xmin)]
               xzero <- xmin
       }
else {
               if(max(y) < 0) {
                       xmin <- x[y == max(y)]
                       if(length(xmin) > 0)
                               xmin <- xmin[1]
                       xzero <- xmin
               }
               else {
                       y1 <- min(y[y > 0])
                       if(length(y1) > 0)
                               y1 <- y1[length(y1)]
                       y2 <- max(y[y < 0])
                       if(length(y2) > 0)
                               y2 <- y2[1]
```

```
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}
#
# Function for Finding the Quantile Orders by Linear Interpolation
#
gridfitinter<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y by
# interpolation
{
nq<-length(Q)
 diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
 vectordest <- apply(diff, 1, zerovalinter,Q)
}

##M-estimator original

QRLM1 <- function (x, y, case.weights = rep(1, nrow(x)),
var.weights = rep(1, nrow(x)), ..., w = rep(1, nrow(x)), init = "ls",
psi = psi.huber, scale.est = c("MAD", "Huber", "proposal 2"),
k2 = 1.345, method = c("M", "MM"), maxit = 20, acc = 1e-04,
test.vec = "resid", q = 0.5)
{
irls.delta <- function(old,new)sqrt(sum((old-new)^2)/
               max(1e-20,sum(old^2)))
irls.rrxwr <- function(x, w, r) {
w <- sqrt(w)
max(abs((matrix(r*w,1,length(r)) %*% x)/sqrt(matrix(w,1,length(r)) %*%
 %*% (x^2))))/sqrt(sum(w*r^2))
}
method <- match.arg(method)
nmx <- deparse(substitute(x))
if (is.null(dim(x))) {
x <- as.matrix(x)
colnames(x) <- nmx
}
else x <- as.matrix(x)
```

```
if (is.null(colnames(x)))
colnames(x) <- paste("X", seq(ncol(x)), sep = "")
if (qr(x)$rank < ncol(x))
stop("x is singular: singular fits are not implemented in rlm")
if (!(any(test.vec == c("resid", "coef", "w", "NULL")) ||
      is.null(test.vec)))
stop("invalid testvec")
if (length(var.weights) != nrow(x))
stop("Length of var.weights must equal number of observations")
if (any(var.weights < 0))
stop("Negative var.weights value")
if (length(case.weights) != nrow(x))
stop("Length of case.weights must equal number of observations")
w <- (w * case.weights)/var.weights
if (method == "M") {
scale.est <- match.arg(scale.est)
if (!is.function(psi))
psi <- get(psi, mode = "function")
arguments <- list(...)
if (length(arguments)) {
pm <- pmatch(names(arguments), names(formals(psi)), nomatch = 0)
if (any(pm == 0))
warning(paste("some of ... do not match"))
pm <- names(arguments)[pm > 0]
formals(psi)[pm] <- unlist(arguments[pm])
}
if (is.character(init)) {
if (init == "ls")
temp <- lm.wfit(x, y, w, method = "qr")
else if (init == "lts")
temp <- lqs.default(x, y, intercept = FALSE, nsamp = 200)
else stop("init method is unknown")
coef <- temp$coef
resid <- temp$resid
}
else {
if (is.list(init))
coef <- init$coef
else coef <- init
resid <- y - x %*% coef
}
}
else if (method == "MM") {
scale.est <- "MM"
temp <- lqs.default(x, y, intercept=FALSE, method="S", k0=1.548)
coef <- temp$coef
resid <- temp$resid
psi <- psi.bisquare
if (length(arguments <- list(...)))
if (match("c", names(arguments), nomatch = FALSE)) {
c0 <- arguments$c
if (c0 > 1.548) {
psi$c <- c0
}
else warning("c must be at least 1.548 and has been ignored")
}
scale <- temp$scale
}
else stop("method is unknown")
done <- FALSE
conv <- NULL
n1 <- nrow(x) - ncol(x)
if (scale.est != "MM")
scale <- mad(resid/sqrt(var.weights), 0)
theta <- 2 * pnorm(k2) - 1
gamma <- theta + k2^2 * (1 - theta) - 2 * k2 * dnorm(k2)
qest <- matrix(0, nrow = ncol(x), ncol = length(q))
qwt <- matrix(0, nrow = nrow(x), ncol = length(q))
qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
qres <- matrix(0, nrow = nrow(x), ncol = length(q))
for(i in 1:length(q)) {
for (iiter in 1:maxit) {
if (!is.null(test.vec))
testpv <- get(test.vec)
if (scale.est != "MM") {
if (scale.est == "MAD")
scale <- median(abs(resid/sqrt(var.weights)))/0.6745
else scale <- sqrt(sum(pmin(resid^2/var.weights,(k2*scale)^2))/
              (n1*gamma))
if (scale == 0) {
done <- TRUE
break
}
}
w <- psi(resid/(scale * sqrt(var.weights))) * case.weights
ww <- 2 * (1 - q[i]) * w
ww[resid > 0] <- 2 * q[i] * w[resid > 0]
w <- ww
temp <- lm.wfit(x, y, w, method = "qr")
coef <- temp$coef
resid <- temp$residuals
if (!is.null(test.vec))
convi <- irls.delta(testpv, get(test.vec))
else convi <- irls.rrxwr(x, wmod, resid)
conv <- c(conv, convi)
done <- (convi <= acc)
if (done)
break
}
if (!done)
warning(paste("rlm failed to converge in",maxit,"steps at q = ",q[i]))
qest[, i] <- coef
qwt[, i] <- w
qfit[, i] <- temp$fitted.values
qres[,i] <- resid
}
list(fitted.values = qfit, residuals = qres, q.values = q,
q.weights = qwt, coefficients = qest)
}

mqgwr.sae=function(x,y,m,area,lon,lat,x.r,area.r,
lon.r,lat.r,method="mqgwr",k.value=1.345, mqgwrweight=TRUE)
{

id.area<- sort(unique(area))
m<-length(id.area)

id.area.r=sort(unique(area.r))
m.r<-length(id.area.r)

tmp.cont=rep(0,m.r)
for (i in 1:m.r)
{for (j in 1:m)
{
if (id.area.r[i]==id.area[j])tmp.cont[i]=1
}

}

tmp0=which(tmp.cont==0)
id.area.out=id.area.r[tmp0]
id.area.in=id.area

ni<- rep(0,m)
for(i in 1:m)ni[i]<- sum(area==id.area.in[i])
n<- sum(ni)

ri<- rep(0,m)
for(i in 1:m)ri[i]<- sum(area.r==id.area.in[i])
r<- sum(ri)

Ni=ri+ni

RI.MQGWR_CD.Mean=rep(0,m)
if (m.r>m)RI.MQGWR_CD.Mean.out=rep(0,(m.r-m))
if (m.r==m)RI.MQGWR_CD.Mean.out=NULL
mse=rep(0,m)
```

```
# Compute the Distance Matrix

eu_dist=as.matrix(dist(cbind(as.vector(lon),as.vector(lat))))
x.design=cbind(1,x)
p=ncol(x.design)
q.value=sort(c(seq(0.002,0.99,0.045),0.5,0.994,0.01,0.02,0.96,0.98))

if (method=="mqgwr")
{

ob.trad<-QRLM1(x.design, y,maxit=100,q=q.value,k=k.value)
qo.trad<-matrix(c(gridfitinter(y,ob.trad$fitted.values,q.value)),
nrow=n,ncol=1)

if (mqgwrweight==TRUE)band=gwr.sel(y ~ 1 + x, coords=cbind(lon, lat),
gweight=gwr.gauss)

if (mqgwrweight==FALSE)band=gwr.sel(y ~ 1 + x, coords=cbind(lon, lat),
gweight=gwr.bisquare)

if (mqgwrweight==TRUE)w.sp<-gwr.gauss((eu_dist^2),band)
if (mqgwrweight==FALSE)w.sp<-gwr.bisquare((eu_dist^2),band)
    q.new=0
    for(ii in 1:n){
    w.new<-diag(w.sp[,ii])
    ob<-QRLM(x.design, y,maxit=100,q=sort(c(seq(0.002,0.99,0.045),0.5,
    0.994,0.01,0.02,0.96,0.98)),w1=w.new,k=k.value)

    qo<-matrix(c(gridfitinter(y,ob$fitted.values,ob$q.values)),
    nrow=n,ncol=1)
    q.new[ii]=as.real(qo[ii,1])
    if (is.na(q.new[ii])) q.new[ii]=as.real(qo.trad[ii,1])
    }


    qmat1<-matrix(c(q.new,area),nrow=n,ncol=2)

    mqo1=tapply(qmat1[,1],qmat1[,2],mean)

    saq<-matrix(0,nrow=m,ncol=2)

    saq[,1]=mqo1

    saq[,2]=sort(unique(qmat1[,2]))

    ci=array(rep(0,n*m),dim=c(n,m,1))

    res.s=NULL
```

```
for (i in 1:m)
   {
   pred.medr=0
   x.r.area<-matrix(cbind(1,x.r)[area.r==id.area.in[i]],ri[i],p)
   lon.gwr=lon.r[area.r==id.area.in[i]]
   lat.gwr=lat.r[area.r==id.area.in[i]]

   tmp=matrix(0,n,1)
   tmp1=matrix(0,n,1)

   for (j in 1:ri[i]){
   dbase=as.matrix(rbind(cbind(lon.gwr[j],lat.gwr[j]),
   cbind(as.vector(lon),as.vector(lat))))

   dist.r=(as.matrix(dist(dbase))[-1,1])
   if (mqgwrweight==TRUE)w.new=gwr.gauss((dist.r)^2,band)
   if (mqgwrweight==FALSE)w.new=gwr.bisquare((dist.r)^2,band)
   w.new=diag(w.new)
   ob1=QRLM(x.design, y,maxit = 100,q=c(saq[i,1]),w1=w.new,k=k.value)
   coef<-matrix(c(t(ob1$coef)),nrow=1,ncol=p)
   # need to be ordered by area

   coef<-t(coef)
   W_star=diag(c(ob1$q.weight),n,n)
   S=W_star%*%x.design%*%solve(t(x.design)%*%W_star%*%x.design)
   xir=x.r.area[j,]
   tmp=tmp+S%*%xir
   pred.medr[j]<-(x.r.area[j,]%*%coef[,1])
   }

   pred.meds=0
   sj<-matrix(x.design[area==id.area.in[i]],ni[i],p)

   lon.gwr=(lon)[area==id.area.in[i]]
   lat.gwr=(lat)[area==id.area.in[i]]
   for (j in 1:ni[i]){
   dbase=as.matrix(rbind(cbind(lon.gwr[j],lat.gwr[j]),
   cbind(as.vector(lon),as.vector(lat))))

   dist.r=(as.matrix(dist(dbase))[-1,1])
   if (mqgwrweight==TRUE)w.new=gwr.gauss((dist.r)^2,band)
   if (mqgwrweight==FALSE)w.new=gwr.bisquare((dist.r)^2,band)
   w.new=diag(w.new)
   ob1=QRLM(x.design, y,maxit = 100,q=c(saq[i,1]),w1=w.new,k=k.value)
   coef<-matrix(c(t(ob1$coef)),nrow=1,ncol=p)
   # need to be ordered by area
   coef<-t(coef)
```

```
    W_star=diag(c(ob1$q.weight),n,n)
    S=W_star%*%x.design%*%solve(t(x.design)%*%W_star%*%x.design)
    xis=sj[j,]
    tmp1=tmp1+S%*%xis
    pred.meds[j]<-(sj[j,]%*%coef[,1])
    }


   f1<-y[area==id.area.in[i]]
   res.s<-c(res.s,(f1-pred.meds))

   ir=rep(0,n)
   ir[area==id.area.in[i]]<-1
   welsh.cd=ir+ir*((ri[i]+ni[i])/ni[i])+tmp-((ri[i]+ni[i])/ni[i])*tmp1
   data<-cbind(as.vector(area),welsh.cd)

   for (kk in 1:n)
   {
   if (data[kk,1]==id.area.in[i]) ci[kk,i,1]=data[kk,2]-1
   else if (data[kk,1]!=id.area.in[i]) ci[kk,i,1]=data[kk,2]}
   RI.MQGWR_CD.Mean[i]=as.real(1/(ri[i]+ni[i])*(t(welsh.cd)%*%
   %*%as.vector(y)))
   }
   res.s=res.s^2
   res.d=cbind(res.s,as.vector(area),ci[,,1])

 for (oo in 1:m)
 {mse[oo]=(1/(ni[oo]+ri[oo])^2)*(sum((res.d[,(oo+2)][res.d[,2]==oo]^2+
 +(ri[oo])/n)*res.d[,1][res.d[,2]==oo])+
 +sum(res.d[,(oo+2)][res.d[,2]!=oo]^2*res.d[,1][res.d[,2]!=oo]))}
}

  if (method=="mqgwr-li")

  {
   if (mqgwrweight==TRUE)band=gwr.sel(y ~ 1 + x,
coords=cbind(lon, lat),gweight=gwr.gauss)

if (mqgwrweight==FALSE)band=gwr.sel(y ~ 1 + x, coords=cbind(lon, lat),
gweight=gwr.bisquare)

if (mqgwrweight==TRUE)w.sp<-gwr.gauss((eu_dist^2),band)
if (mqgwrweight==FALSE)w.sp<-gwr.bisquare((eu_dist^2),band)
     q.new=0
     n.q=length(q.value)
     fitted=matrix(0,n,n.q)
     for (qj in 1:n.q)
     {
     ob.trad<-QRLM1(x.design, y,maxit=100,q=q.value[qj])
     for(ii in 1:n){
     w.new<-(w.sp[,ii])
     err<-sum(w.new*ob.trad$q.weights*ob.trad$residuals)/
     /sum(w.new*ob.trad$q.weights)

     fitted[ii,qj]=ob.trad$fitted.values[ii]+err}
     }
     q.new<-matrix(c(gridfitinter(y,fitted,q.value)),nrow=n,ncol=1)

     qmat1<-matrix(c(q.new,area),nrow=n,ncol=2)

     mqo1=tapply(qmat1[,1],qmat1[,2],mean)

     saq<-matrix(0,nrow=m,ncol=2)

     saq[,1]=mqo1

     saq[,2]=sort(unique(qmat1[,2]))

     ci=array(rep(0,n*m),dim=c(n,m,1))

     res.s=NULL

for (i in 1:m)
  {
  pred.medr=0
  x.r.area<-matrix(cbind(1,x.r)[area.r==id.area.in[i]],ri[i],p)
  lon.gwr=lon.r[area.r==id.area.in[i]]
  lat.gwr=lat.r[area.r==id.area.in[i]]

  tmp=matrix(0,1,n)
  tmp1=matrix(0,1,n)

  ob.trad<-QRLM1(x.design, y,maxit=100,q=c(saq[i,1]))
  coef<-matrix(c(t(ob.trad$coef)),nrow=1,ncol=2)
  # need to be ordered by area
  coef<-t(coef)
  wd<-diag(c(ob.trad$q.weights))
  meat<-wd%*%x.design%*%solve(t(x.design)%*%wd%*%x.design)
  meat1=(diag(1,n,n)-x.design%*%solve(t(x.design)%*%wd%*%x.design)%*%
  %*%t(x.design)%*%wd)

  for (j in 1:ri[i]){
  dbase=as.matrix(rbind(cbind(lon.gwr[j],lat.gwr[j]),
  cbind(as.vector(lon),as.vector(lat))))

  dist.r=(as.matrix(dist(dbase))[-1,1])
  if (mqgwrweight==TRUE)w.new=gwr.gauss((dist.r)^2,band)
  if (mqgwrweight==FALSE)w.new=gwr.bisquare((dist.r)^2,band)
```

```
 uno=matrix(1,n,1)
 err1=((t(uno)%*%(diag(c(w.new),n,n)%*%wd%*%meat1))*
 *as.real(solve(t(uno)%*%diag(c(w.new),n,n)%*%wd%*%uno)))
 tmp=tmp+err1
 pred.medr[j]<-(x.r.area[j,]%*%coef[,1]+as.real(err1%*%matrix(y,n,1)))
 }

 pred.meds=0
 sj<-matrix(x.design[area==id.area.in[i]],ni[i],p)

 lon.gwr=(lon)[area==id.area.in[i]]
 lat.gwr=(lat)[area==id.area.in[i]]
 for (j in 1:ni[i]){
 dbase=as.matrix(rbind(cbind(lon.gwr[j],lat.gwr[j]),
 cbind(as.vector(lon),as.vector(lat))))

 dist.r=(as.matrix(dist(dbase))[-1,1])
 if (mqgwrweight==TRUE)w.new=gwr.gauss((dist.r)^2,band)
 if (mqgwrweight==FALSE)w.new=gwr.bisquare((dist.r)^2,band)

 err2=((t(uno)%*%(diag(c(w.new),n,n)%*%wd%*%meat1))*
 *as.real(solve(t(uno)%*%diag(c(w.new),n,n)%*%wd%*%uno)))
  tmp1=tmp1+err2
 pred.meds[j]<-(sj[j,]%*%coef[,1]+as.real(err2%*%matrix(y,n,1)))
 }


f1<-y[area==id.area.in[i]]
res.s<-c(res.s,(f1-pred.meds))

sj.tot<-apply(sj,2,sum)
rj.tot<-apply(x.r.area,2,sum)

ir=rep(0,n)
ir[area==id.area.in[i]]<-1


welsh.cd=ir*((ri[i]+ni[i])/ni[i])+meat%*%
%*%(rj.tot-(ri[i]/ni[i])*sj.tot)+t(tmp)-((ri[i])/ni[i])*t(tmp1)

data<-cbind(as.vector(area),welsh.cd)

for (kk in 1:n)
{
if (data[kk,1]==id.area.in[i]) ci[kk,i,1]=data[kk,2]-1
else if (data[kk,1]!=id.area.in[i]) ci[kk,i,1]=data[kk,2]}
RI.MQGWR_CD.Mean[i]=as.real(1/(ri[i]+ni[i])*(t(welsh.cd)%*%as.vector(y)))
 }
```

```
res.s=res.s^2
res.d=cbind(res.s,as.vector(area),ci[,,1])

for (oo in 1:m)
{mse[oo]=(1/(ni[oo]+ri[oo])^2)*(sum((res.d[,(oo+2)][res.d[,2]==oo]^2+
+(ri[oo])/n)*res.d[,1][res.d[,2]==oo])+
+sum(res.d[,(oo+2)][res.d[,2]!=oo]^2*res.d[,1][res.d[,2]!=oo]))}

 }
      #out of sample area
      if (m.r>m){
       rr.sample=m.r-m
       for (i in 1:rr.sample)
       {
      Ri=sum(area.r==id.area.out[i])
       x.r.area<-matrix(cbind(1,x.r)[area.r==id.area.out[i]],Ri,p)
       pred.medr=0
       lon.gwr=lon.r[area.r==id.area.out[i]]
       lat.gwr=lat.r[area.r==id.area.out[i]]
       for (j in 1:Ri){
       dbase=as.matrix(rbind(cbind(lon.gwr[j],lat.gwr[j]),
       cbind(as.vector(lon),as.vector(lat))))
       dist.r=(as.matrix(dist(dbase))[-1,1])
       if (mqgwrweight==TRUE)w.new=gwr.gauss((dist.r)^2,band)
       if (mqgwrweight==FALSE)w.new=gwr.bisquare((dist.r)^2,band)
       w.new=diag(w.new)
       ob1=QRLM(x.design, y,maxit = 100,q=0.5,w1=w.new)
       coef<-matrix(c(t(ob1$coef)),nrow=1,ncol=2)
       # need to be ordered by area
       coef<-t(coef)
       pred.medr[j]<-(x.r.area[j,]%*%coef[,1])
       }


        RI.MQGWR_CD.Mean.out[i]<-1/(Ri)*(sum(pred.medr))}
        }

 list(Area.code.in=id.area.in,Area.code.out=id.area.out,
 Est.Mean.in=RI.MQGWR_CD.Mean,
 Est.Mean.out=RI.MQGWR_CD.Mean.out,Est.mse.in=mse)

}
```

Chapter 24

Appendix 10: R code for M-quantile CD estimators of the CDF

24.1 R code of mq.sae.quant

The R code of the function **mqcd.sae** is listed bellow.

```
###########################################################
###
###             M-quantile CD estimators for the quantiles
###                              SAMPLE project
###
### Authors: N. Salvati, N.Tzavidis, C. Giusti,
###          S. Marchetti and M. Pratesi
### File name: MQ.SAE.quant.R
### Updated: March 17th, 2010
###
############################################################

library(MASS)
library(np)


QRLM <- function (x, y, case.weights = rep(1, nrow(x)),
var.weights = rep(1, nrow(x)), ..., w = rep(1, nrow(x)),
init="ls",psi=psi.huber, cale.est=c("MAD","Huber","proposal 2"),
k2=1.345,method= ("M","MM"),maxit=20, acc=1e-04,test.vec="resid",q=0.5)
{
irls.delta <- function(old,new)sqrt(sum((old-new)^2)/
               max(1e-20,sum(old^2)))
irls.rrxwr <- function(x, w, r) {
w <- sqrt(w)
```

```
max(abs((matrix(r*w,1,length(r)) %*% x)/sqrt(matrix(w,1,length(r)) %*%
%*%(x^2))))/sqrt(sum(w*r^2))
}
method <- match.arg(method)
nmx <- deparse(substitute(x))
if (is.null(dim(x))) {
x <- as.matrix(x)
colnames(x) <- nmx
}
else x <- as.matrix(x)
if (is.null(colnames(x)))
colnames(x) <- paste("X", seq(ncol(x)), sep = "")
if (qr(x)$rank < ncol(x))
stop("x is singular: singular fits are not implemented in rlm")
if (!(any(test.vec == c("resid", "coef", "w", "NULL")) ||
        is.null(test.vec)))
stop("invalid testvec")
if (length(var.weights) != nrow(x))
stop("Length of var.weights must equal number of observations")
if (any(var.weights < 0))
stop("Negative var.weights value")
if (length(case.weights) != nrow(x))
stop("Length of case.weights must equal number of observations")
w <- (w * case.weights)/var.weights
if (method == "M") {
scale.est <- match.arg(scale.est)
if (!is.function(psi))
psi <- get(psi, mode = "function")
arguments <- list(...)
if (length(arguments)) {
pm <- pmatch(names(arguments), names(formals(psi)), nomatch = 0)
if (any(pm == 0))
warning(paste("some of ... do not match"))
pm <- names(arguments)[pm > 0]
formals(psi)[pm] <- unlist(arguments[pm])
}
if (is.character(init)) {
if (init == "ls")
temp <- lm.wfit(x, y, w, method = "qr")
else if (init == "lts")
temp <- lqs.default(x, y, intercept = FALSE, nsamp = 200)
else stop("init method is unknown")
coef <- temp$coef
resid <- temp$resid
}
else {
if (is.list(init))
coef <- init$coef
else coef <- init
```

```
resid <- y - x %*% coef
}
}
else if (method == "MM") {
scale.est <- "MM"
temp <- lqs.default(x, y, intercept=FALSE, method="S", k0=1.548)
coef <- temp$coef
resid <- temp$resid
psi <- psi.bisquare
if (length(arguments <- list(...)))
if (match("c", names(arguments), nomatch = FALSE)) {
c0 <- arguments$c
if (c0 > 1.548) {
psi$c <- c0
}
else warning("c must be at least 1.548 and has been ignored")
}
scale <- temp$scale
}
else stop("method is unknown")
done <- FALSE
conv <- NULL
n1 <- nrow(x) - ncol(x)
if (scale.est != "MM")
scale <- mad(resid/sqrt(var.weights), 0)
theta <- 2 * pnorm(k2) - 1
gamma <- theta + k2^2 * (1 - theta) - 2 * k2 * dnorm(k2)
qest <- matrix(0, nrow = ncol(x), ncol = length(q))
qwt <- matrix(0, nrow = nrow(x), ncol = length(q))
qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
qres <- matrix(0, nrow = nrow(x), ncol = length(q))
for(i in 1:length(q)) {
for (iiter in 1:maxit) {
if (!is.null(test.vec))
testpv <- get(test.vec)
if (scale.est != "MM") {
if (scale.est == "MAD")
scale <- median(abs(resid/sqrt(var.weights)))/0.6745
else scale <- sqrt(sum(pmin(resid^2/var.weights,(k2*scale)^2))/
              (n1*gamma))
if (scale == 0) {
done <- TRUE
break
}
}
w <- psi(resid/(scale * sqrt(var.weights))) * case.weights
ww <- 2 * (1 - q[i]) * w
ww[resid > 0] <- 2 * q[i] * w[resid > 0]
w <- ww
```

```
temp <- lm.wfit(x, y, w, method = "qr")
coef <- temp$coef
resid <- temp$residuals
if (!is.null(test.vec))
convi <- irls.delta(testpv, get(test.vec))
else convi <- irls.rrxwr(x, wmod, resid)
conv <- c(conv, convi)
done <- (convi <= acc)
if (done)
break
}
if (!done)
warning(paste("rlm failed to converge in", maxit, "steps at q = ", q[i]))
qest[, i] <- coef
qwt[, i] <- w
qfit[, i] <- temp$fitted.values
qres[,i] <- resid
}
list(fitted.values = qfit, residuals = qres, q.values = q,
q.weights = qwt, coefficients = qest)
}

# COMPUTE THE QUANTILE ORDER

# COMPUTING OF THE QUANTILE-ORDERS
"zerovalinter"<-function(y, x)
{
        if(min(y) > 0) {
                xmin <- x[y == min(y)]
                if(length(xmin) > 0)
                        xmin <- xmin[length(xmin)]
                xzero <- xmin
        }

else {
                if(max(y) < 0) {
                        xmin <- x[y == max(y)]
                        if(length(xmin) > 0)
                                xmin <- xmin[1]
                        xzero <- xmin
                }
                else {
                        y1 <- min(y[y > 0])
                        if(length(y1) > 0)
                                y1 <- y1[length(y1)]
                        y2 <- max(y[y < 0])
                        if(length(y2) > 0)
                                y2 <- y2[1]
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}

# Function for Finding the Quantile Orders by Linear Interpolation
# Assumes that "zerovalinter" function has been already loaded

"gridfitinter"<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y by
# interpolation
{
nq<-length(Q)
  diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
 vectordest <- apply(diff, 1, zerovalinter,Q)

#print(vectordest)
#qord<-list(ord=c(vectordest))
#qord
}

mq.coef<-function(myx,myy,myregioncode,maxiter=100){

#This function estimate the m-quantile regression coefficients
#myx<- x sample matrix of auxiliary variables
#myy<- y vector
#mynumauxvar<- number of auxiliary variables (include constant)
#myregioncode<- area code for y and x units, data must be ordered by
area code
#maxiter<- OPTIONAL, number of maximum iteration for ob algorithm

myar<-unique(myregioncode)
myareas<-length(myar)
mysamplesize<-sum(as.numeric(table(myregioncode)))
mynumauxvar<-dim(myx)[2]

ob<-QRLM(myx, myy, maxit=maxiter,q=sort(c(seq(0.006,0.99,0.045),0.5,
0.994,0.01,0.02,0.96,0.98)))
qo<-matrix(c(gridfitinter(myy,ob$fitted.values,ob$q.values)),
nrow=mysamplesize,ncol=1)
qmat<-matrix(c(qo,myregioncode),nrow=length(myregioncode),ncol=2)
mqo<-aggregate(qmat[,1],list(d2=qmat[,2]),mean)[,2]
saq<-matrix(c(mqo,myar),nrow=myareas,ncol=2)
saq<-rbind(saq,c(0.5,9999))
ob1<-QRLM(myx, myy,maxit = maxiter,q=c(mqo[1:myareas]))
mycoef<-matrix(c(t(ob1$coefficients)),nrow=myareas,ncol=mynumauxvar)
# need to be ordered by area
mycoef<-t(mycoef)
mycoef
}


intsolver<-function(myqest,myyboot,myX,myregioncodepop,mypopsize,myar,
myareas,adjseed,mysboot,mymaxit=100){

myres<-array(0,dim=c(myareas))

myy<-myyboot[mysboot]
myx<-myX[mysboot,]
myregioncode<-myregioncodepop[mysboot]
myregioncoder<-myregioncodepop[-mysboot]
mysamplesizer<-as.numeric(table(myregioncoder))
myX.r<-myX[-mysboot,]

# M-quantiles
coef.boot<-mq.coef(myx,myy,myregioncode)

# Quantile Estimation Using Chambers Dunstan Estimator
for(i in 1:myareas){
f1<-myy[myregioncode==myar[i]]
X.aux.i<-as.matrix(myX.r[myregioncoder==myar[i],])
pred.medr<-(X.aux.i%*%coef.boot[,i])
x.design.i<-as.matrix(myx[myregioncode==myar[i],])
pred.meds<-(x.design.i%*%coef.boot[,i])
res.s<-f1-pred.meds
z<-sample(res.s,mysamplesizer[i],replace=TRUE)
z<-z+pred.medr
comb<-c(f1,pred.medr)
start0<-quantile(comb,prob=c(myqest))

sameside<-T
myiter<-0
while (sameside & myiter<mymaxit){
ff2<-sum(c(z)<=start0)
ff1<-sum(c(f1)<=start0)
f0<-1/(mypopsize[i])*(ff1+ff2)
if (f0<=myqest) start1<-start0+adjseed
if (f0>myqest) start1<-start0-adjseed
```

```
ff2<-sum(c(z)<=start1)
ff1<-sum(c(f1)<=start1)
f.new<-1/(mypopsize[i])*(ff1+ff2)
start.bef<-start0
start.aft<-start1
if (f0<=myqest & f.new>=myqest) sameside<-F
if (f0>=myqest & f.new<=myqest) sameside<-F
start0<-start1
myiter<-myiter+1
}
if(myiter>=100) warning("intsolver sameside did not converge in
"mymaxit" iteration")

ff2<-sum(c(z)<=start.bef)
ff1<-sum(c(f1)<=start.bef)
f.bef<-1/(mypopsize[i])*(ff1+ff2)
ff2<-sum(c(z)<=start.aft)
ff1<-sum(c(f1)<=start.aft)
f.aft<-1/(mypopsize[i])*(ff1+ff2)
fdif<-abs(f.bef-f.aft)
if (fdif>=0.01) {
start.med<-(start.bef+start.aft)/2

eps<-0.001
tol<-50
myiter<-0
while (abs(tol)>=0.05 & myiter<mymaxit){
ff2<-sum(c(z)<=start.med)
ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff1+ff2)
tol<-(fmed-myqest)
fmedl<-1/(mypopsize[i])*(ff1+ff2)-eps
fmedu<-1/(mypopsize[i])*(ff1+ff2)+eps
if (fmed<myqest & fmedl<fmedu) start.bef<-start.med
if (fmed<myqest & fmedl>fmedu) start.aft<-start.med
if (fmed>myqest & fmedl<fmedu) start.aft<-start.med
if (fmed>myqest & fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
myiter<-myiter+1
}
if(myiter>=100) warning("intsolver fdif>0.01 tol did not converge in
"mymaxit" iteration")
}

if (fdif<0.01) {
start.med<-(start.bef+start.aft)/2

eps<-0.001
tol<-50
```

```
myiter<-0
while (abs(tol)>0.05 & myiter<mymaxit){
ff2<-sum(c(z)<=start.med)
ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff1+ff2)
tol<-(fmed-myqest)
fmedl<-1/(mypopsize[i])*(ff1+ff2)-eps
fmedu<-1/(mypopsize[i])*(ff1+ff2)+eps
if (fmed<myqest & fmedl<fmedu) start.bef<-start.med
if (fmed<myqest & fmedl>fmedu) start.aft<-start.med
if (fmed>myqest & fmedl<fmedu) start.aft<-start.med
if (fmed>myqest & fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
myiter<-myiter+1
}
if(myiter>=100) warning("intsolver fdif<0.01 tol did not converge in
"mymaxit" iteration")
}

myres[i]<-start.med
}#Iteration i in bootstrap ends here
myres
}


boot.CD.R<-function(myqest,myyboot,myX,myregioncodepop,mypopsize,
mysamplesize,myar,myareas,myadjseed,myR,myid){

myproc.a<-array(0,dim=c(myareas,myR))

#Sampling from bootstrap population
for (r in 1:myR){
mysboot<-NULL
s.boot.i<-NULL
for (i in 1:myareas){
s.boot.i<-sample(myid[myregioncodepop==myar[i]],mysamplesize[i])
mysboot<-c(mysboot,s.boot.i)
}

myproc.a[,r]<-intsolver(myqest,myyboot,myX,myregioncodepop,mypopsize,myar,
myareas,myadjseed,mysboot)
}#R ends here
myproc.a
}


MQ.SAE.quant<-function(myqgrid,myy,myx,myX,myregioncode,myregioncodepop,
adjseed=max(0.15,mean(myy)/500),myMSE=FALSE,B=1,R=400,method="su",
mymaxit=100){
```

```
#This function estimate quantiles via CD estimator when n/N -> p
with p very small
#myqgrid<- quantiles order to be estimated (i.e. 0.25,0.50,0.75)
#myy<- y vector
#myx<- x sample matrix of auxiliary variables
#myX<- X population matrix of auxiliary variables
#myregioncode<- area code for y and x units, data must be ordered by
area code
#myregioncodepop<- area code for X units (population), data must be ordered
by area code
#adjseed<- OPTIONAL, tune the value used to find two good starting point to
solve the integral
#myMSE<-TRUE compute the MSE of the CD quantiles estimate via bootstrap,
FALSE does not compute any MSE
#B<- number of bootstrap population
#R<- number of bootstrap samples
#method<- which method to be used to estimate residuals distribution,
choice= "su" (smooth unconditional),"eu" (emprirical unconditional),
"sc" (smooth conditional),"ec" (empirical coditional)
#mymaxit: maximum iteration allowed in the while routines

myar<-unique(myregioncode)
myareas<-length(myar)
mypopsize<-as.numeric(table(myregioncodepop))
mysamplesize<-as.numeric(table(myregioncode))
myquantnum<-length(myqgrid)
id<-seq(1:sum(mypopsize))

myarea.q<-matrix(0,myquantnum,myareas)
myq.true.boot<-array(0,dim=c(myquantnum,myareas,B))
myarea.q.boot.r<-array(0,dim=c(myquantnum,myareas,B,R))
myarea.q.boot<-array(0,dim=c(myquantnum,myareas,B))
myq.true.boot.m<-array(0,dim=c(myquantnum,myareas))
myarea.q.boot.m<-array(0,dim=c(myquantnum,myareas))
BIAS.boot<-matrix(0,myquantnum,myareas)
VAR.boot<-matrix(0,myquantnum,myareas)
MSE.boot<-matrix(0,myquantnum,myareas)
kk<-0

mycoef<-mq.coef(myx,myy,myregioncode)

for(qq in myqgrid){
qest<-qq
kk<-kk+1
myres<-NULL

for(i in 1:myareas){
f1<-myy[myregioncode==myar[i]]
```

```
X.aux.i<-as.matrix(myX[myregioncodepop==myar[i],])
pred.medtot<-(X.aux.i%*%mycoef[,i])
x.design.i<-as.matrix(myx[myregioncode==myar[i],])
pred.meds<-(x.design.i%*%mycoef[,i])
res.s<-f1-pred.meds
myres[i]<-list(res.s)
z<-sample(res.s,mypopsize[i],replace=TRUE)
z<-z+pred.medtot
comb<-c(pred.medtot)
start0<-quantile(comb,prob=c(qest))

sameside<-T
myiter<-0
while (sameside & myiter<mymaxit){
ff2<-sum(c(z)<=start0)
f0<-1/(mypopsize[i])*(ff2)
if (f0<=qest) start1<-start0+adjseed
if (f0>qest) start1<-start0-adjseed
ff2<-sum(c(z)<=start1)
f.new<-1/(mypopsize[i])*(ff2)
start.bef<-start0
start.aft<-start1
if (f0<=qest & f.new>=qest) sameside<-F
if (f0>=qest & f.new<=qest) sameside<-F
start0<-start1
myiter<-myiter+1
}
if(myiter>=100) warning("CD.quant sameside did not converge in "mymaxit"
iteration")

ff2<-sum(c(z)<=start.bef)
f.bef<-1/(mypopsize[i])*(ff2)
ff2<-sum(c(z)<=start.aft)
f.aft<-1/(mypopsize[i])*(ff2)
fdif<-abs(f.bef-f.aft)
if (fdif>=0.01) {
start.med<-(start.bef+start.aft)/2

eps<-0.001
tol<-50
myiter<-0
while (abs(tol)>=0.05 & myiter<mymaxit){
ff2<-sum(c(z)<=start.med)
fmed<-1/(mypopsize[i])*(ff2)
tol<-(fmed-qest)
fmedl<-1/(mypopsize[i])*(ff2)-eps
fmedu<-1/(mypopsize[i])*(ff2)+eps
if (fmed<qest & fmedl<fmedu) start.bef<-start.med
if (fmed<qest & fmedl>fmedu) start.aft<-start.med
if (fmed>qest & fmedl<fmedu) start.aft<-start.med
if (fmed>qest & fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
myiter<-myiter+1
}
if(myiter>=100) warning("CD.quant fdif>0.01 tol did not converge in
"mymaxit" iteration")
}

if (fdif<0.01) {
start.med<-(start.bef+start.aft)/2

eps<-0.001
tol<-50
myiter<-0
while (abs(tol)>0.05 & myiter<mymaxit){
ff2<-sum(c(z)<=start.med)
fmed<-1/(mypopsize[i])*(ff2)
tol<-(fmed-qest)
fmedl<-1/(mypopsize[i])*(ff2)-eps
fmedu<-1/(mypopsize[i])*(ff2)+eps
if (fmed<qest & fmedl<fmedu) start.bef<-start.med
if (fmed<qest & fmedl>fmedu) start.aft<-start.med
if (fmed>qest & fmedl<fmedu) start.aft<-start.med
if (fmed>qest & fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
myiter<-myiter+1
}
if(myiter>=100) warning("CD.quant fdif<0.01 tol did not converge in
"mymaxit" iteration")
}
myarea.q[kk,i]<-start.med
}#Iteration i ends here

if(myMSE){
######Bootstrap######

#Generate B bootstrap Population (size N)

if(method=="sc"){
#Centering residuals in each areas (use this for area conditioned approach)
res.s.centered<-NULL
for (i in 1:myareas){
res.s.centered[i]<-list(myres[[i]]-mean(myres[[i]]))
}

#smoothed density of residuals areas conditioned
Fhat.ord<-NULL
res.ord<-NULL
```

```
for (i in 1:myareas){
bw<-npudensbw(~res.s.centered[[i]],ckertype="epanechnikov")
Fhat<-fitted(npudist(bws=bw))
res.ord[i]<-list(sort(res.s.centered[[i]]))
Fhat.ord[i]<-list(sort(Fhat))
}
}

if(method=="su"){
#Centering residuals for the whole sample
(use this for area unconditioned approach)
res.s.centered<-NULL
for (i in 1:myareas){
res.s.centered<-c(res.s.centered,myres[[i]])
}
res.s.centered<-sort(res.s.centered-mean(res.s.centered))

#smoothed density of residuals areas unconditioned
Fhat.ord<-NULL
bw<-npudensbw(~res.s.centered,ckertype="epanechnikov")
Fhat<-fitted(npudist(bws=bw))
Fhat.ord<-sort(Fhat)
}

if(method=="ec"){
#Centering residuals in each areas (use this for area conditioned approach)
res.s.centered<-NULL
for (i in 1:myareas){
res.s.centered[i]<-list(myres[[i]]-mean(myres[[i]]))
}
}

if(method=="eu"){
#Centering residuals for the whole sample
(use this for area unconditioned approach)
res.s.centered<-NULL
for (i in 1:myareas){
res.s.centered<-c(res.s.centered,myres[[i]])
}
res.s.centered<-sort(res.s.centered-mean(res.s.centered))
}


for (b in 1:B){

if(method=="sc"){
#Sample from kernel density areas conditioned
samp.boot<-NULL
for (i in 1:myareas){
```

```
s.boot<-NULL
for (g in 1:mypopsize[i]){
s.boot[g]<-which(Fhat.ord[[i]]==quantile(Fhat.ord[[i]],prob=runif(1),
type=3))
}
samp.boot[i]<-list(s.boot)
}
#Population smoothed density of residuals area conditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:myareas){
y.boot.i<-myX[myregioncodepop==myar[i],]%*%mycoef[,i]+
res.ord[[i]][samp.boot[[i]]]
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="su"){
#Sample from kernel density areas unconditioned
samp.boot<-NULL
for (i in 1:myareas){
s.boot<-NULL
for (g in 1:mypopsize[i]){
s.boot[g]<-which(Fhat.ord==quantile(Fhat.ord,prob=runif(1),type=3))
}
samp.boot[i]<-list(s.boot)
}
#Population smoothed density of residuals areas unconditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:myareas){
y.boot.i<-myX[myregioncodepop==myar[i],]%*%mycoef[,i]+
res.s.centered[samp.boot[[i]]]
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="ec"){
#Population empirical density of residuals area conditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:myareas){
y.boot.i<-myX[myregioncodepop==myar[i],]%*%mycoef[,i]+
sample(res.s.centered[[i]],mypopsize[i],replace=TRUE)
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="eu"){
```

```
#Population empirical density of residuals area unconditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:myareas){
y.boot.i<-myX[myregioncodepop==myar[i],]%*%mycoef[,i]+
sample(res.s.centered,mypopsize[i],replace=TRUE)
y.boot<-c(y.boot,y.boot.i)
}
}


for (ii in 1:myareas){
myq.true.boot[kk,ii,b]<-quantile(y.boot[myregioncodepop==myar[ii]],
prob=c(qest))
}

myarea.q.boot.r[kk,,b,1:R]<-boot.CD.R(qest,y.boot,myX,myregioncodepop,
mypopsize,mysamplesize,myar,myareas,adjseed,R,id)
#print(myarea.q.boot.r[kk,,b,])
#myarea.q.boot.r[kk,,b,(R/2+1):R]<-mycollect[[2]]
#print(myarea.q.boot.r[kk,,b,])

for (i in 1:myareas){
myarea.q.boot[kk,i,b]<-mean(myarea.q.boot.r[kk,i,b,])
}

}#B ends here

for (i in 1:myareas){
myq.true.boot.m[kk,i]<-mean(myq.true.boot[kk,i,])
myarea.q.boot.m[kk,i]<-mean(myarea.q.boot[kk,i,])
}

for (i in 1:myareas){
BIAS.boot[kk,i]<-myarea.q.boot.m[kk,i]-myq.true.boot.m[kk,i]
aux<-matrix(0,B,1)
for (b in 1:B){
aux[b,1]<-(1/R)*sum((myarea.q.boot.r[kk,i,b,]-myarea.q.boot[kk,i,b])^2)
}
VAR.boot[kk,i]<-(1/B)*sum(aux[,1])
}

for (i in 1:myareas){
MSE.boot[kk,i]<-((BIAS.boot[kk,i])^2)+VAR.boot[kk,i]
}

}#end if myMSE
}#Iteration k ends here
RMSE.CD<-as.data.frame(sqrt(MSE.boot))
```

```
names(RMSE.CD)<-myar
row.names(RMSE.CD)<-myqgrid
quantiles<-as.data.frame(myarea.q)
names(quantiles)<-myar
row.names(quantiles)<-myqgrid
rest<-list(quantiles=quantiles,rmse=RMSE.CD,Area.Code=myar)
}
```

Chapter 25

Appendix 11: R code for nonparametric M-quantile CD estimators of the CDF

25.1 R code of npmq.sae.quant

The R code of the function **npmqcd.sae** is listed bellow.

```
###########################################################
###
### Nonparametric M-quantile CD estimators for the quantiles
###                 SAMPLE project
###
### Authors: N. Salvati, N.Tzavidis, C. Giusti,
###          S. Marchetti and M. Pratesi
### File name: MQ.SAE.quant.R
### Updated: February 2nd, 2010
###
############################################################

rm(list=ls(all.names=TRUE))


set.seed(1977)

library(MASS)
library(np)
library(SemiPar)
library(splines)


npqrlm<-function(Y.n,X,Z.spline,quantile,tol=0.001,maxit=100,theta=2,
kk=1.345){
```

```
# prototype function for panel data fitting of MQ models
# the matrix X is assumed to contain an intercept
# the vector s is a strata indicator assumed (so far) to be a one-way layout
#theta : GCV parameter, defaulted to 2

require(SemiPar)
require(MASS)
require(splines)
assign("tol",tol,pos=1)
assign("maxit",maxit,pos=1)
assign("Y.n",Y.n,pos=1)
assign("X",X,pos=1)
assign("kk",kk,pos=1)
assign("Z.spline",Z.spline,pos=1)
assign("theta",theta,pos=1)
assign("quantile",quantile,pos=1)

n<-length(Y.n)
X<-as.matrix(X)
p1<-ncol(X)
Z.spline<-as.matrix(Z.spline)
p2<-ncol(Z.spline)
X.n<-cbind(X,Z.spline)
p=p1+p2

b=rep(1,p)

my.psi.q<-function(u,q,c){
s<-median(abs(u))/0.6745
w <- psi.huber((u/s),c)
ww <- 2 * (1 - q) * w
ww[u> 0] <- 2 * q * w[u > 0]
w <- ww
w*u
}

my.b<-function(X,Y,W,lambda)
{G<-as.matrix(diag(c(rep(0,p1),rep(1,p2)),p))
solve(t(X)%*%W%*%X+G*lambda)%*%t(X)%*%W%*%Y
}


stima<-function(l){
# l : coefficiente di penalizzazione nell'IRPLS
n<-nrow(X.n)
diff<-1
iter<-0
while (diff>tol)
{#inizia procedura di stima
iter<-iter+1
res<-Y.n-X.n%*%b #calcolo residui
W.n<-as.matrix(diag(c(my.psi.q(as.matrix(res),qtl,kk)/as.matrix(res)),n))
assign("W.n", W.n, pos=1)
b.ott<-my.b(X.n,Y.n,W.n,l)
diff<-sum((as.matrix(b)-as.matrix(b.ott))^2)
b<-b.ott
if (iter>maxit)
{warning(paste("failed to converge in", maxit, "steps at q = ", qtl))
break}
}
y.hat=X.n%*%b
list(fitted.values=as.matrix(y.hat),coef=as.matrix(b),
we=as.matrix(diag(W.n)))
}

my.GCV<-function(l)
{
G<-as.matrix(diag(c(rep(0,p1),rep(1,p2)),p))
tmp<-stima(l)
y.hat<-tmp$fitted.values
S<-(X.n)%*%solve(t(X.n)%*%W.n%*%X.n+G*l)%*%t(X.n)%*%W.n
sum((Y.n-y.hat)^2)/((1-theta*sum(diag(S))/n)^2)
}

length.q<-length(quantile)
y.fit<-matrix(0,n,length.q)
y.coef<-matrix(0,p,length.q)
y.weight<-matrix(0,n,length.q)
lambda.ott<-NULL
for (k in 1:length.q)
{
qtl<-quantile[k]
qtl<-assign("qtl",qtl,pos=1)
tmp<-optimize(my.GCV,c(0,50))
l.ott<-tmp$minimum
lambda.ott[k]<-l.ott
y.stim<-stima(l.ott)
y.fit[,k]<-y.stim$fitted.values
y.coef[,k]=y.stim$coef
y.weight[,k]=y.stim$we
}
list(hat.values=y.fit,b.stim=y.coef,q.weights=y.weight,lambda.q=lambda.ott)
}

# COMPUTE THE QUANTILE ORDER

# COMPUTING OF THE QUANTILE-ORDERS
```

```
"zerovalinter"<-function(y, x)
{
        if(min(y) > 0) {
                xmin <- x[y == min(y)]
                if(length(xmin) > 0)
                        xmin <- xmin[length(xmin)]
                xzero <- xmin
        }



else {
                if(max(y) < 0) {
                        xmin <- x[y == max(y)]
                        if(length(xmin) > 0)
                                xmin <- xmin[1]
                        xzero <- xmin
                }
                else {
                        y1 <- min(y[y > 0])
                        if(length(y1) > 0)
                                y1 <- y1[length(y1)]
                        y2 <- max(y[y < 0])
                        if(length(y2) > 0)
                                y2 <- y2[1]
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}

# Function for Finding the Quantile Orders by Linear Interpolation
# Assumes that "zerovalinter" function has been already loaded

"gridfitinter"<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y by interpolation
{
nq<-length(Q)
  diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
```

```
 vectordest <- apply(diff, 1, zerovalinter,Q)

#print(vectordest)
#qord<-list(ord=c(vectordest))
#qord
}



mq.coef<-function(myx,myy,myzspline,mykvalue=1.345,myregioncode,
maxiter=100){

#This function estimate the m-quantile regression coefficients
#myx<- x sample matrix of auxiliary variables
#myy<- y vector
#mynumauxvar<- number of auxiliary variables (include constant)
#myregioncode<- area code for y and x units, data must be ordered by
area code
#maxiter<- OPTIONAL, number of maximum iteration for ob algorithm

myar<-sort(unique(myregioncode))
myzspline<-as.matrix(myzspline)
myareas<-length(myar)
mysamplesize<-sum(as.numeric(table(myregioncode)))
mynumauxvar<-dim(myx)[2]+ncol(myzspline)

ob<-npqrlm(myy, myx, myzspline,q=sort(c(seq(0.006,0.99,0.045),0.5,
0.994,0.01,0.02,0.96,0.98)),kk=mykvalue)
q.values<-sort(c(seq(0.006,0.99,0.045),0.5,0.994,0.01,0.02,0.96,0.98))
qo<-matrix(c(gridfitinter(y,ob$hat.values,q.values)),nrow=mysamplesize,
ncol=1)
qmat<-matrix(c(qo,myregioncode),nrow=length(myregioncode),ncol=2)
mqo<-aggregate(qmat[,1],list(d2=qmat[,2]),mean)[,2]
saq<-matrix(c(mqo,myar),nrow=myareas,ncol=2)
saq<-rbind(saq,c(0.5,9999))
ob1<-npqrlm(myy, myx, myzspline, quantile=c(mqo[1:myareas]), kk=mykvalue)
mycoef<-matrix(c(t(ob1$b.stim)),nrow=myareas,ncol= mynumauxvar)
# need to be ordered by area
mycoef<-t(mycoef)
mycoef
}


intsolver<-function(myqest,myyboot,myX,myzspline,myzsplinepop,
myregioncodepop,mypopsize,myar,myareas,adjseed,mysboot){

myres<-array(0,dim=c(myareas))

myy<-myyboot[mysboot]
```

```
myx<-myX[mysboot,]
myregioncode<-myregioncodepop[mysboot]
myregioncoder<-myregioncodepop[-mysboot]
mysamplesizer<-as.numeric(table(myregioncoder))
myX.r<-myX[-mysboot,]
myzspline<-myzsplinepop[mysboot,]
myzspliner<-myzsplinepop[-mysboot,]

# M-quantiles
coef.boot<-mq.coef(myx,myy,myregioncode)

# Quantile Estimation Using Chambers Dunstan Estimator
for(i in 1:myareas){
f1<-myy[myregioncode==myar[i]]
pred.medr<-cbind(myX.r[myregioncoder==myar[i],],
myzspliner[myregioncoder==myar[i],])%*%coef.boot[,i]
pred.meds<-cbind(myx[myregioncode==myar[i],],
myzspline[myregioncode==myar[i],])%*%coef.boot[,i]
res.s<-f1-pred.meds
#z<-matrix(rep(res.s,sample.sizer[i]),nrow=sample.sizer[i],
ncol=sample.size[i])
z<-sample(res.s,mysamplesizer[i],replace=TRUE)
z<-z+pred.medr
comb<-c(f1,pred.medr)
start0<-quantile(comb,prob=c(myqest))
sameside<-T

while (sameside){
ff2<-sum(c(z)<=start0)
ff1<-sum(c(f1)<=start0)
f0<-1/(mypopsize[i])*(ff1+ff2)
if (f0<=myqest) start1<-start0+adjseed
if (f0>myqest) start1<-start0-adjseed
ff2<-sum(c(z)<=start1)
ff1<-sum(c(f1)<=start1)
f.new<-1/(mypopsize[i])*(ff1+ff2)
start.bef<-start0
start.aft<-start1
if (f0<=myqest & f.new>=myqest) sameside<-F
if (f0>=myqest & f.new<=myqest) sameside<-F
start0<-start1
}

ff2<-sum(c(z)<=start.bef)
ff1<-sum(c(f1)<=start.bef)
f.bef<-1/(mypopsize[i])*(ff1+ff2)
ff2<-sum(c(z)<=start.aft)
ff1<-sum(c(f1)<=start.aft)
f.aft<-1/(mypopsize[i])*(ff1+ff2)
```

```
fdif<-abs(f.bef-f.aft)
if (fdif>=0.01) {
start.med<-(start.bef+start.aft)/2
eps<-0.001
tol<-50

while (abs(tol)>=0.5){
ff2<-sum(c(z)<=start.med)
ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff1+ff2)
tol<-(fmed-myqest)
fmedl<-1/(mypopsize[i])*(ff1+ff2)-eps
fmedu<-1/(mypopsize[i])*(ff1+ff2)+eps
if (fmed<myqest & fmedl<fmedu) start.bef<-start.med
if (fmed<myqest & fmedl>fmedu) start.aft<-start.med
if (fmed>myqest & fmedl<fmedu) start.aft<-start.med
if (fmed>myqest & fmedl>fmedu) start.bef<-start.med
#if (fmedl<fmedu) start.bef<-start.med
#if (fmedl>fmedu) start.aft<-start.med
#if (fmedl<fmedu) start.aft<-start.med
#if (fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
#print(tol)
}
}

if (fdif<0.01) {
start.med<-(start.bef+start.aft)/2
eps<-0.001
tol<-50

while (abs(tol)>0.5){
ff2<-sum(c(z)<=start.med)
ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff1+ff2)
tol<-(fmed-myqest)
fmedl<-1/(mypopsize[i])*(ff1+ff2)-eps
fmedu<-1/(mypopsize[i])*(ff1+ff2)+eps
if (fmed<myqest & fmedl<fmedu) start.bef<-start.med
if (fmed<myqest & fmedl>fmedu) start.aft<-start.med
if (fmed>myqest & fmedl<fmedu) start.aft<-start.med
if (fmed>myqest & fmedl>fmedu) start.bef<-start.med
#if (fmedl<fmedu) start.bef<-start.med
#if (fmedl>fmedu) start.aft<-start.med
#if (fmedl<fmedu) start.aft<-start.med
#if (fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
#print(tol)
}
```

```
}

myres[i]<-start.med
#print(i)
}#Iteration i in bootstrap ends here
myres
}


NPMQ.SAE.quant<-function(myqgrid,myy,myx,myX,myzspline,myzsplinepop,myregioncode,
myregioncodepop,adjseed=max(0.15,mean(myy)/500)){

#This function estimate quantiles via CD estimator when n/N -> p with
p very small
#myqgrid<- quantiles order to be estimated (i.e. 0.25,0.50,0.75)
#myy<- y vector
#myx<- x sample matrix of auxiliary variables
#myX<- X population matrix of auxiliary variables
#myregioncode<- area code for y and x units, data must be ordered
by area code
#myregioncodepop<- area code for X units (population), data must be ordered
by area code
#adjseed<- OPTIONAL, tune the value used to find two good starting point to
solve the integral

myar<-unique(myregioncode)
myareas<-length(myar)
mypopsize<-as.numeric(table(myregioncodepop))
mysamplesize<-as.numeric(table(myregioncode))
myquantnum<-length(myqgrid)
id<-seq(1:sum(mypopsize))

myarea.q<-matrix(0,myquantnum,myareas)
myq.true.boot.m<-array(0,dim=c(myquantnum,myareas))
myarea.q.boot.m<-array(0,dim=c(myquantnum,myareas))
kk<-0
mycoef<-mq.coef(myx,myy,myzspline,mykvalue=1.345,myregioncode)

for(qq in myqgrid){
qest<-qq
kk<-kk+1
myres<-NULL

for(i in 1:myareas){
f1<-myy[myregioncode==myar[i]]
pred.medtot<-cbind(myX[myregioncodepop==myar[i],],
myzsplinepop[myregioncodepop==myar[i],])%*%mycoef[,i]
x.design.i<-as.matrix(myx[myregioncode==myar[i],])
pred.meds<-cbind(myx[myregioncode==myar[i],],
myzspline[myregioncode==myar[i],])%*%mycoef[,i]
res.s<-f1-pred.meds
myres[i]<-list(res.s)
# z<-matrix(rep(res.s,pop.size[i]),nrow=pop.size[i],ncol=sample.size[i])
z<-sample(res.s,mypopsize[i],replace=TRUE)
z<-z+pred.medtot
comb<-c(pred.medtot)
start0<-quantile(comb,prob=c(qest))
sameside<-T

while (sameside){
ff2<-sum(c(z)<=start0)
# ff1<-sum(c(f1)<=start0)
f0<-1/(mypopsize[i])*(ff2)
if (f0<=qest) start1<-start0+adjseed
if (f0>qest) start1<-start0-adjseed
ff2<-sum(c(z)<=start1)
# ff1<-sum(c(f1)<=start1)
f.new<-1/(mypopsize[i])*(ff2)
start.bef<-start0
start.aft<-start1
if (f0<=qest & f.new>=qest) sameside<-F
if (f0>=qest & f.new<=qest) sameside<-F
start0<-start1
}

ff2<-sum(c(z)<=start.bef)
# ff1<-sum(c(f1)<=start.bef)
f.bef<-1/(mypopsize[i])*(ff2)
ff2<-sum(c(z)<=start.aft)
# ff1<-sum(c(f1)<=start.aft)
f.aft<-1/(mypopsize[i])*(ff2)
fdif<-abs(f.bef-f.aft)
if (fdif>=0.01) {
start.med<-(start.bef+start.aft)/2
eps<-0.001
tol<-50

while (abs(tol)>=0.5){
ff2<-sum(c(z)<=start.med)
# ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff2)
tol<-(fmed-qest)
fmedl<-1/(mypopsize[i])*(ff2)-eps
fmedu<-1/(mypopsize[i])*(ff2)+eps
if (fmed<qest & fmedl<fmedu) start.bef<-start.med
if (fmed<qest & fmedl>fmedu) start.aft<-start.med
if (fmed>qest & fmedl<fmedu) start.aft<-start.med
if (fmed>qest & fmedl>fmedu) start.bef<-start.med
#if (fmedl<fmedu) start.bef<-start.med
#if (fmedl>fmedu) start.aft<-start.med
#if (fmedl<fmedu) start.aft<-start.med
#if (fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
#print(tol)
}
}

if (fdif<0.01) {
start.med<-(start.bef+start.aft)/2
eps<-0.001
tol<-50

while (abs(tol)>0.5){
ff2<-sum(c(z)<=start.med)
# ff1<-sum(c(f1)<=start.med)
fmed<-1/(mypopsize[i])*(ff2)
tol<-(fmed-qest)
fmedl<-1/(mypopsize[i])*(ff2)-eps
fmedu<-1/(mypopsize[i])*(ff2)+eps
if (fmed<qest & fmedl<fmedu) start.bef<-start.med
if (fmed<qest & fmedl>fmedu) start.aft<-start.med
if (fmed>qest & fmedl<fmedu) start.aft<-start.med
if (fmed>qest & fmedl>fmedu) start.bef<-start.med
#if (fmedl<fmedu) start.bef<-start.med
#if (fmedl>fmedu) start.aft<-start.med
#if (fmedl<fmedu) start.aft<-start.med
#if (fmedl>fmedu) start.bef<-start.med
start.med<-(start.bef+start.aft)/2
#print(tol)
}
}
#print(i)
myarea.q[kk,i]<-start.med
}#Iteration i ends here
}
quantiles<-myarea.q
for(i in 1:myareas){
check<-sort(quantiles[,i])-quantiles[,i]
check<-sum(abs(check))
if(check!=0){
warning("Quantile crossing produced in area ",i)
}
}
rest<-list(quantiles=myarea.q,Area.Code=myar)
}
```

Chapter 26

Appendix 12: R code for M-quantile poverty indicators estimators

26.1 R code of mq.sae.poverty

The R code of the function **mq.sae.poverty** is listed bellow.

```
#########################################################################
###
###             M-quantile poverty indicators estimators
###                             SAMPLE project
###
### Author: N. Salvati, N.Tzavidis, C. Giusti, S. Marchetti, M. Pratesi
### File name: MQ.SAE.poverty.R
### Updated: February 2nd, 2010
###
########################################################################

###########LIBRARY
library(MASS)
library(np)


QRLM <- function (x, y, case.weights = rep(1, nrow(x)), var.weights =
rep(1, nrow(x)), ..., w = rep(1, nrow(x)), init = "ls", psi = psi.huber,
scale.est = c("MAD", "Huber", "proposal 2"), k2 = 1.345,
method = c("M", "MM"), maxit = 20, acc = 1e-04, test.vec = "resid",
q = 0.5)
{
irls.delta <- function(old, new) sqrt(sum((old - new)^2)/
max(1e-20, sum(old^2)))
irls.rrxwr <- function(x, w, r) {
```

```
w <- sqrt(w)
max(abs((matrix(r*w,1,length(r)) %*% x)/sqrt(matrix(w,1,length(r)) %*%
(x^2))))/sqrt(sum(w*r^2))
}
method <- match.arg(method)
nmx <- deparse(substitute(x))
if (is.null(dim(x))) {
x <- as.matrix(x)
colnames(x) <- nmx
}
else x <- as.matrix(x)
if (is.null(colnames(x)))
colnames(x) <- paste("X", seq(ncol(x)), sep = "")
if (qr(x)$rank < ncol(x))
stop("x is singular: singular fits are not implemented in rlm")
if (!(any(test.vec == c("resid", "coef", "w", "NULL")) ||
is.null(test.vec)))
stop("invalid testvec")
if (length(var.weights) != nrow(x))
stop("Length of var.weights must equal number of observations")
if (any(var.weights < 0))
stop("Negative var.weights value")
if (length(case.weights) != nrow(x))
stop("Length of case.weights must equal number of observations")
w <- (w * case.weights)/var.weights
if (method == "M") {
scale.est <- match.arg(scale.est)
if (!is.function(psi))
psi <- get(psi, mode = "function")
arguments <- list(...)
if (length(arguments)) {
pm <- pmatch(names(arguments), names(formals(psi)), nomatch = 0)
if (any(pm == 0))
warning(paste("some of ... do not match"))
pm <- names(arguments)[pm > 0]
formals(psi)[pm] <- unlist(arguments[pm])
}
if (is.character(init)) {
if (init == "ls")
temp <- lm.wfit(x, y, w, method = "qr")
else if (init == "lts")
temp <- lqs.default(x, y, intercept = FALSE, nsamp = 200)
else stop("init method is unknown")
coef <- temp$coef
resid <- temp$resid
}
else {
if (is.list(init))
coef <- init$coef
```

```
else coef <- init
resid <- y - x %*% coef
}
}
else if (method == "MM") {
scale.est <- "MM"
temp <- lqs.default(x, y, intercept = FALSE, method = "S", k0 = 1.548)
coef <- temp$coef
resid <- temp$resid
psi <- psi.bisquare
if (length(arguments <- list(...)))
if (match("c", names(arguments), nomatch = FALSE)) {
c0 <- arguments$c
if (c0 > 1.548) {
psi$c <- c0
}
else warning("c must be at least 1.548 and has been ignored")
}
scale <- temp$scale
}
else stop("method is unknown")
done <- FALSE
conv <- NULL
n1 <- nrow(x) - ncol(x)
if (scale.est != "MM")
scale <- mad(resid/sqrt(var.weights), 0)
theta <- 2 * pnorm(k2) - 1
gamma <- theta + k2^2 * (1 - theta) - 2 * k2 * dnorm(k2)
qest <- matrix(0, nrow = ncol(x), ncol = length(q))
qwt <- matrix(0, nrow = nrow(x), ncol = length(q))
qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
qres <- matrix(0, nrow = nrow(x), ncol = length(q))
for(i in 1:length(q)) {
for (iiter in 1:maxit) {
if (!is.null(test.vec))
testpv <- get(test.vec)
if (scale.est != "MM") {
if (scale.est == "MAD")
scale <- median(abs(resid/sqrt(var.weights)))/0.6745
else scale <- sqrt(sum(pmin(resid^2/var.weights,(k2*scale)^2))/(n1*gamma))
if (scale == 0) {
done <- TRUE
break
}
}
w <- psi(resid/(scale * sqrt(var.weights))) * case.weights
ww <- 2 * (1 - q[i]) * w
ww[resid > 0] <- 2 * q[i] * w[resid > 0]
w <- ww
```

```
temp <- lm.wfit(x, y, w, method = "qr")
coef <- temp$coef
resid <- temp$residuals
if (!is.null(test.vec))
convi <- irls.delta(testpv, get(test.vec))
else convi <- irls.rrxwr(x, wmod, resid)
conv <- c(conv, convi)
done <- (convi <= acc)
if (done)
break
}
if (!done)
warning(paste("rlm failed to converge in", maxit, "steps at q = ", q[i]))
qest[, i] <- coef
qwt[, i] <- w
qfit[, i] <- temp$fitted.values
qres[,i] <- resid
}
list(fitted.values = qfit, residuals = qres, q.values = q, q.weights = qwt,
coefficients = qest)
}



# COMPUTE THE QUANTILE ORDER

# COMPUTING OF THE QUANTILE-ORDERS
"zerovalinter"<-function(y, x)
{
        if(min(y) > 0) {
                xmin <- x[y == min(y)]
                if(length(xmin) > 0)
                        xmin <- xmin[length(xmin)]
                xzero <- xmin
        }



else {
                if(max(y) < 0) {
                        xmin <- x[y == max(y)]
                        if(length(xmin) > 0)
                                xmin <- xmin[1]
                        xzero <- xmin
                }
                else {
                        y1 <- min(y[y > 0])
                        if(length(y1) > 0)
                                y1 <- y1[length(y1)]
                        y2 <- max(y[y < 0])
                        if(length(y2) > 0)
                                y2 <- y2[1]
                        x1 <- x[y == y1]
                        if(length(x1) > 0)
                                x1 <- x1[length(x1)]
                        x2 <- x[y == y2]
                        if(length(x2) > 0)
                                x2 <- x2[1]
                        xzero <- (x2 * y1 - x1 * y2)/(y1 - y2)
                        xmin <- x1
                        if(abs(y2) < y1)
                                xmin <- x2
                }
        }
        resu <-  xzero
        resu
}

# Function for Finding the Quantile Orders by Linear Interpolation
# Assumes that "zerovalinter" function has been already loaded

"gridfitinter"<-function(y,expectile,Q)
# computing of the expectile-order of each observation of y
by interpolation
{
nq<-length(Q)
  diff <- y %*% t(as.matrix(rep(1, nq))) - expectile
 vectordest <- apply(diff, 1, zerovalinter,Q)

#print(vectordest)
#qord<-list(ord=c(vectordest))
#qord
}



mq.coef<-function(myx,myy,myregioncode,maxiter=100){

#This function estimate the m-quantile regression coefficients
#myx<- x sample matrix of auxiliary variables
#myy<- y vector
#mynumauxvar<- number of auxiliary variables (include constant)
#myregioncode<- area code for y and x units, data must be ordered by
area code
#maxiter<- OPTIONAL, number of maximum iteration for ob algorithm

myar<-unique(myregioncode)
myareas<-length(myar)
```

```
mysamplesize<-sum(as.numeric(table(myregioncode)))
mynumauxvar<-dim(myx)[2]

ob<-QRLM(myx, myy, maxit=maxiter,q=sort(c(seq(0.006,0.99,0.045),0.5,
0.994,0.01,0.02,0.96,0.98)),k=1.345)
qo<-matrix(c(gridfitinter(myy,ob$fitted.values,ob$q.values)),
nrow=mysamplesize,ncol=1)
qmat<-matrix(c(qo,myregioncode),nrow=length(myregioncode),ncol=2)
mqo<-aggregate(qmat[,1],list(d2=qmat[,2]),mean)[,2]
saq<-matrix(c(mqo,myar),nrow=myareas,ncol=2)
saq<-rbind(saq,c(0.5,9999))
ob1<-QRLM(myx, myy,maxit = maxiter,q=c(mqo[1:myareas]),k=1.345)
mycoef<-matrix(c(t(ob1$coefficients)),nrow=myareas,ncol=mynumauxvar)
# need to be ordered by area
mycoef<-t(mycoef)
list(q.mean=mycoef,q.unit=qmat)
}



compute.hcr.pg<-function(my.ys,my.x.s,my.X.pop,myregioncode,myregioncodepop,
L,areas,ar,pop.size,sample.size,myz){

f.MQ.0<-array(0,dim=c(areas))
f.MQ.1<-array(0,dim=c(areas))

res.mq<-NULL

#Fit the model M-quantile
tmp=mq.coef(my.x.s,my.ys,myregioncode)
beta.mq<-tmp$q.mean
for(i in 1:areas){
#MQ
ysd<-my.ys[myregioncode==ar[i]]
x.sd<-my.x.s[myregioncode==ar[i],]
Eds.mq<-x.sd%*%beta.mq[,i]
res.d.mq<-ysd-Eds.mq
res.mq<-c(res.mq,res.d.mq)
}


for(i in 1:areas){

x.rd<-my.X.pop[myregioncodepop==ar[i],]

#Monte Carlo approximation to the best predictor of yi
F.0.hl.mq<-matrix(0,L,1)
F.1.hl.mq<-matrix(0,L,1)
```

```
for (l in 1:L){
#MQ
Ehl.mc<-x.rd%*%beta.mq[,i]+sample(res.mq,pop.size[i],replace=TRUE)
I.mc.mq<-Ehl.mc<myz
F.0.hl.mq[l,1]<-sum(I.mc.mq)/pop.size[i]
Ehl.mc[Ehl.mc<0]<-0
F.1.hl.mq[l,1]<-(1/pop.size[i])*sum((1-Ehl.mc/myz)*I.mc.mq)
}
f.MQ.0[i]<-mean(F.0.hl.mq[,1])
f.MQ.1[i]<-mean(F.1.hl.mq[,1])

}#i ends here

f.MQ.0[which(f.MQ.0>1)]<-1
f.MQ.1[which(f.MQ.1>1)]<-1

res<-list(HCR.MQ=f.MQ.0,PG.MQ=f.MQ.1,res.mq=res.mq,mycoef=beta.mq)
res

}#Function compute.hcr.pg ends here



MQ.SAE.poverty<-function(my.ys,my.x.s,my.X.pop,myregioncode,
myregioncodepop,L=50,myMSE=TRUE,myB=1,myR=400,method="eu",pov.l=NULL){

#This function compute the HCR and the PovertyGap statistics for
Small Area

areas<-length(unique(myregioncode))
ar<-unique(myregioncode)
pop.size<-table(myregioncodepop)
sample.size<-table(myregioncode)
myid<-1:sum(pop.size)
if(is.null(pov.l)) z<-0.6*median(my.ys)
if(!is.null(pov.l)) z<-povl

myq.true.boot<-array(0,dim=c(2,areas,myB))
myarea.q.boot.r<-array(0,dim=c(2,areas,myB,myR))
myarea.q.boot<-array(0,dim=c(2,areas,myB))
myq.true.boot.m<-array(0,dim=c(2,areas))
myarea.q.boot.m<-array(0,dim=c(2,areas))
BIAS.boot<-matrix(0,2,areas)
VAR.boot<-matrix(0,2,areas)
MSE.boot<-matrix(0,2,areas)
CI.boot.hcr<-matrix(0,areas,2)
CI.boot.pg<-matrix(0,areas,2)

estimate<-compute.hcr.pg(my.ys,my.x.s,my.X.pop,myregioncode,myregioncodepop,
L,areas,ar,pop.size,sample.size,z)
res.mq<-estimate$res.mq


if(myMSE){

#Generate B bootstrap Population (size N)

if(method=="sc"){
#Centering residuals in each areas (use this for area conditioned approach)
res.s.centered<-NULL
for (i in 1:areas){
res.s.centered[i]<-list(res.mq[myregioncode==ar[i]]-
mean(res.mq[myregioncode==ar[i]]))
}

#smoothed density of residuals areas conditioned
Fhat.ord<-NULL
res.ord<-NULL
for (i in 1:areas){
bw<-npudensbw(~res.s.centered[[i]],ckertype="epanechnikov")
Fhat<-fitted(npudist(bws=bw))
res.ord[i]<-list(sort(res.s.centered[[i]]))
Fhat.ord[i]<-list(sort(Fhat))
}
}

if(method=="su"){
#Centering residuals for the whole sample
(use this for area unconditioned approach)
res.s.centered<-sort(res.mq-mean(res.mq))

#smoothed density of residuals areas unconditioned
Fhat.ord<-NULL
bw<-npudensbw(~res.s.centered,ckertype="epanechnikov")
Fhat<-fitted(npudist(bws=bw))
Fhat.ord<-sort(Fhat)
}

if(method=="ec"){
#Centering residuals in each areas (use this for area conditioned approach)
res.s.centered<-NULL
for (i in 1:areas){
res.s.centered[i]<-list(res.mq[myregioncode==ar[i]]-
mean(res.mq[myregioncode==ar[i]]))
}
}

if(method=="eu"){
```

```
#Centering residuals for the whole sample
(use this for area unconditioned approach)
res.s.centered<-sort(res.mq-mean(res.mq))
}

for (b in 1:myB){

if(method=="sc"){
#Sample from kernel density areas conditioned
samp.boot<-NULL
for (i in 1:areas){
s.boot<-NULL
for (g in 1:pop.size[i]){
s.boot[g]<-which(Fhat.ord[[i]]==quantile(Fhat.ord[[i]],prob=runif(1),
type=3))
}
samp.boot[i]<-list(s.boot)
}
#Population smoothed density of residuals area conditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:areas){
y.boot.i<-my.X.pop[myregioncodepop==ar[i],]%*%estimate$mycoef[,i]+
res.ord[[i]][samp.boot[[i]]]
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="su"){
#Sample from kernel density areas unconditioned
samp.boot<-NULL
for (i in 1:areas){
s.boot<-NULL
for (g in 1:pop.size[i]){
s.boot[g]<-which(Fhat.ord==quantile(Fhat.ord,prob=runif(1),type=3))
}
samp.boot[i]<-list(s.boot)
}
#Population smoothed density of residuals areas unconditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:areas){
y.boot.i<-my.X.pop[myregioncodepop==ar[i],]%*%estimate$mycoef[,i]+
res.s.centered[samp.boot[[i]]]
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="ec"){
```

```
#Population empirical density of residuals area conditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:areas){
y.boot.i<-my.X.pop[myregioncodepop==ar[i],]%*%estimate$mycoef[,i]+
sample(res.s.centered[[i]],pop.size[i],replace=TRUE)
y.boot<-c(y.boot,y.boot.i)
}
}

if(method=="eu"){
#Population empirical density of residuals area unconditioned
y.boot<-NULL
y.boot.i<-NULL
for (i in 1:areas){
y.boot.i<-my.X.pop[myregioncodepop==ar[i],]%*%estimate$mycoef[,i]+
sample(res.s.centered,pop.size[i],replace=TRUE)
y.boot<-c(y.boot,y.boot.i)
}
}


for (ii in 1:areas){
y.d.boot<-y.boot[myregioncodepop==ar[ii]]
myq.true.boot[1,ii,b]<-sum(y.d.boot<z)/pop.size[ii]
y.d.boot[y.d.boot<0]<-0
myq.true.boot[2,ii,b]<-(1/pop.size[ii])*sum((1-y.d.boot/z)*(y.d.boot<z))
}


for(rr in 1:myR){
mysboot<-NULL
s.boot.i<-NULL
for(ii in 1:areas){
s.boot.i<-sample(myid[myregioncodepop==ar[ii]],sample.size[ii])
mysboot<-c(mysboot,s.boot.i)
}
ys.boot<-y.boot[mysboot]
x.s.boot<-my.X.pop[mysboot,]
estimate.boot<-compute.hcr.pg(ys.boot,x.s.boot,my.X.pop,myregioncode,
myregioncodepop,L,areas,ar,pop.size,sample.size,z)
myarea.q.boot.r[1,,b,rr]<-estimate.boot$HCR.MQ
myarea.q.boot.r[2,,b,rr]<-estimate.boot$PG.MQ
}


for (ii in 1:areas){
myarea.q.boot[1,ii,b]<-mean(myarea.q.boot.r[1,ii,b,])
myarea.q.boot[2,ii,b]<-mean(myarea.q.boot.r[2,ii,b,])
```

```
}

}#B ends here

for (i in 1:areas){
myq.true.boot.m[1,i]<-mean(myq.true.boot[1,i,])
myq.true.boot.m[2,i]<-mean(myq.true.boot[2,i,])
myarea.q.boot.m[1,i]<-mean(myarea.q.boot[1,i,])
myarea.q.boot.m[2,i]<-mean(myarea.q.boot[2,i,])
}

for (i in 1:areas){
BIAS.boot[1,i]<-myarea.q.boot.m[1,i]-myq.true.boot.m[1,i]
BIAS.boot[2,i]<-myarea.q.boot.m[2,i]-myq.true.boot.m[2,i]
aux.0<-matrix(0,myB,1)
aux.1<-matrix(0,myB,1)
for (b in 1:myB){
aux.0[b,1]<-(1/myR)*sum((myarea.q.boot.r[1,i,b,]-myarea.q.boot[1,i,b])^2)
aux.1[b,1]<-(1/myR)*sum((myarea.q.boot.r[2,i,b,]-myarea.q.boot[2,i,b])^2)
}
VAR.boot[1,i]<-(1/myB)*sum(aux.0[,1])
VAR.boot[2,i]<-(1/myB)*sum(aux.1[,1])
}

for (i in 1:areas){
MSE.boot[1,i]<-((BIAS.boot[1,i])^2)+VAR.boot[1,i]
MSE.boot[2,i]<-((BIAS.boot[2,i])^2)+VAR.boot[2,i]
CI.boot.hcr[i,]<-quantile(c(myarea.q.boot.r[1,i,,]),prob=c(0.025,0.975))
CI.boot.pg[i,]<-quantile(c(myarea.q.boot.r[2,i,,]),prob=c(0.025,0.975))
}

rmse.hcr.mq<-sqrt(MSE.boot[1,])
rmse.pg.mq<-sqrt(MSE.boot[2,])

}#end if myMSE

if(myMSE==FALSE){
rmse.hcr.mq<-NULL
rmse.pg.mq<-NULL
}

res<-list(HCR.MQ=estimate$HCR.MQ,PG.MQ=estimate$PG.MQ,
RMSE.HCR.MQ=rmse.hcr.mq,RMSE.PG.MQ=rmse.pg.mq,
Area.Code=unique(myregioncode),Pov.Line=z)

}#Function ends here
```

Chapter 27

Appendix 13: R code for the EB method for poverty estimation

27.1 R code of FGTpovertyEB

The R code of the function `FGTpovertyEB()` is listed bellow.

```
###############################################################################
###
### This function fits a unit level model to the log of the welfare variable
### and obtains EB estimators of FGT poverty measures of order 1 and 2
### (poverty incidence and poverty gap) when the out-of-sample values of
### auxiliary variables are available.
###
### Work for European project SAMPLE
###
### Author: Isabel Molina
### File name: FGTpovertyEB.R
### Updated: February 8th, 2011
###
###############################################################################

FGTpovertyEB<-function(dom,seldomain=unique(dom),Xrdtot,welfare,Xs,weight,
z=0.6*median(welfare),L=50,seed=Sys.time()){

# Load library required to fit a mixed model

library(nlme)

# Set the seed for random number generation,
# required for the Monte Carlo approximation of the EB method.

set.seed(seed)

# Number of domains for which EB estimators of poverty measures are required
# called (target domains)

I<-length(unique(seldomain))

# Total number of domains with sample data

D<-length(unique(dom))

# Number of auxiliary variables (including the constant)

p<-dim(Xs)[2]

# Sample sizes and out-of-sample sizes of target domains

nd<-rep(0,D)
rd<-rep(0,D)

for (i in 1:I){
  nd[seldomain[i]]<-sum(dom==seldomain[i])
  rd[seldomain[i]]<-dim(Xrdtot[[i]])[1]
}

# Take logarithm of welfare after adding a constant to make it positive

m<-abs(min(welfare))+1
welfaret<-welfare+m
ys<-log(welfaret)

# Fit the nested-error model to sample data by REML method using function lme
# from library nlme.

fit.EB<-lme(ys~-1+Xs,random=~1|as.factor(dom),method="REML")

# Create a list object containing different results from the model fit.

Resultsfit<-list(Summary=summary(fit.EB),FixedEffects=fixed.effects(fit.EB),
RandomEffects=random.effects(fit.EB),ResVar=fit.EB$sigma,
RandomEffVar=as.numeric(VarCorr(fit.EB)[1,1]),Loglike=fit.EB$logLik,
RawResiduals=fit.EB$residuals[1:n])

# Save some of the results of the fitting method in variables

betaest<-fixed.effects(fit.EB)# Vector of model coefficients (size p)
upred<-random.effects(fit.EB) # Matrix with predicted random effects in 1st col.
sigmae2est<-fit.EB$sigma^2    # Estimated error variance
sigmau2est<-as.numeric(VarCorr(fit.EB)[1,1]) # Estimated random effects var.

# EB method starts: Generate L vectors of non-sample values of the response
# from their conditional distribution given the sample data and calculate
# empirical values of EB estimators.

# Matrices with poverty incidences and gaps for the L simulations
# in the EB method
povinc<-matrix(0,nr=D,nc=L)
povgap<-matrix(0,nr=D,nc=L)

# Vectors with final EB estimators
povinc.EB<-rep(0,D)
povgap.EB<-rep(0,D)

# Time counter initialization
time1<-Sys.time()
time1

for (i in 1:I){    # Cycle for target domains

  # Print order of target domain
  cat("Domain num.",i,"\n")

  # Code of target domain
  d<-seldomain[i]

  # Matrix with the values of the p auxiliary variables for the
  # out-of-sample observations in the i-th target domain
  Xrd<-Xrdtot[[i]]

  # Get sample values for target domain
  ysd<-ys[dom==d]

  # Compute conditional means for out-of-sample units
  mudpred<-Xrd%*%matrix(betaest,nr=p,nc=1)+upred[d,1]

  # The conditional distribution of (non-sample data given sample data)
  # in the EB method can be expressed as a new nested-error model with
  # different random effects variance. We calculate this random effects
  # variance (called sigmav2)

  gammad<-sigmau2est/(sigmau2est+sigmae2est/nd[d])
  sigmav2<-sigmau2est*(1-gammad)

  for (ell in 1:L){   ### Start of Monte Carlo simulations for EB method

    # Generate random effect for target domain d
    vd<-rnorm(1,0,sqrt(sigmav2))

    # Generate random errors for all out-of-sample units in target domain d
    ed<-rnorm(rd[d],0,sqrt(sigmae2est))
    # Compute vector of out-if-sample responses
    yrdpred<-mudpred+vd+ed

    # Merge non-sample and sample values (full population or census)
    # for domain d
    ydnew<-c(ysd,yrdpred)

    # Compute domain poverty measures using population values
    Ednew<-exp(ydnew)-m
    povinc[d,ell]<-mean(Ednew<z)
    povgap[d,ell]<-mean((Ednew<z)*(z-Ednew)/z)

  }  # End of Monte Carlo simulations for EB method

  # EB predictors of poverty measures (averages over the L Monte
  # Carlo simulations)
  povinc.EB[d]<-mean(povinc[d,])
  povgap.EB[d]<-mean(povgap[d,])

} # End of cycle for province index
time2 <- Sys.time()   # Current time
ComputTime<-difftime(time2,time1,units="mins") # Total time spent by EB method

# Results
EstimatedPoverty<-data.frame(Domain=seldomain,PovInc=100*povinc.EB[seldomain],
PovGap=100*povgap.EB[seldomain])
return (list(EstimatedPoverty=EstimatedPoverty,ComputTime=ComputTime,
Resultsfit=Resultsfit))

} #End of function FGTpovertyEB
```

27.2 R code of PBMSE.EB

The R code of the function `PBMSE.EB()` is listed bellow.

```
############################################################################
###
### This function obtains estimators of the MSEs of the EB estimators
### of FGT poverty measures of order 1 (poverty incidence) and order 2
### (poverty gap) by a parametric bootstrap method. Population values of
### auxiliary variables are required.
###
### Work for European project SAMPLE
###
### Author: Isabel Molina
### File name: PBMSE_EB.R
### Updated: February 8th, 2011
###
###########################################################################
```

```
PBMSE.EB<-function(dom,seldomain=unique(dom),Xrdtot,welfare,Xs,weight,
z=0.6*median(welfare),B=50,LB=50,seed=Sys.time()){

# Load library required to fit a mixed model

library(nlme)

# Set the seed for random number generation,
# required for the Monte Carlo approximation of the EB method.

set.seed(seed)

# Number of domains for which EB estimators of poverty measures are
# required called (target domains)

I<-length(unique(seldomain))

# Total number of sample observations

n<-length(welfare)

# Total number of domains with sample data

D<-length(unique(dom))

# Number of auxiliary variables (including the constant)

p<-dim(Xs)[2]

# Sample sizes and out-of-sample sizes of target domains

nd<-rep(0,D)
rd<-rep(0,D)

for (i in 1:I){
  nd[seldomain[i]]<-sum(dom==seldomain[i])
  rd[seldomain[i]]<-dim(Xrdtot[[i]])[1]
}

# Take logarithm of welfare after adding a constant to make it positive

m<-abs(min(welfare))+1
welfaret<-welfare+m
ys<-log(welfaret)

# Fit the nested-error model to sample data by REML method using
# function lme from library nlme.
```

```
fit.EB<-lme(ys~-1+Xs,random=~1|as.factor(dom),method="REML")

# Save some of the results of the fitting method in variables

betaest<-fixed.effects(fit.EB)# Vector of model coefficients (size p)
upred<-random.effects(fit.EB) # Matrix with predicted random effects in 1st col.
sigmae2est<-fit.EB$sigma^2    # Estimated error variance
sigmau2est<-as.numeric(VarCorr(fit.EB)[1,1]) # Estimated random effects var.

# Initialize vectors with the bootstrap MSEs of EB estimators

MSEpropsum.B<-rep(0,D)
MSEgapsum.B<-rep(0,D)
MSEpovincEB.B<-rep(0,D)
MSEpovgapEB.B<-rep(0,D)

# Time counter initialization
time1<-Sys.time()

for (b in 1:B){   ### Start of bootstrap cycle

  # Print bootstrap iteration

  cat("Bootstrap iteration",b,"\n")

  # We will generate a bootstrap population by generating sample and
  # out-of-sample elements and calculate true poverty measures.

  ys.B<-rep(0,n)   # Boostrap sample vector

  # Initialize vectors with true poverty measures
  truepovinc.B<-rep(0,D)
  truepovgap.B<-rep(0,D)

  # Initialize vectors with estimated poverty measures
  povinc.B<-matrix(0,nr=D,nc=LB)
  povgap.B<-matrix(0,nr=D,nc=LB)
  povinc.EB.B<-rep(0,D)
  povgap.EB.B<-rep(0,D)

  ud.B<-rep(0,D)    # Bootstrap random effects
  nd<-rep(0,D)      # Vector with sample sizes of provinces

  # Generate sample elements (for all domains)

  for (d in 1:D){

    # Take sample elements
    Xsd<-Xs[dom==d,]

    # Calculate the cummulated sample sizes

    nd[d]<-sum(dom==d)

    # Generate sample values of y from the fitted model

    esd.B<-rnorm(nd[d],0,sqrt(sigmae2est))
    ud.B[d]<-rnorm(1,0,sqrt(sigmau2est))
    musd.B<-Xsd%*%matrix(betaest,nr=p,nc=1)
    ys.B[dom==d]<-musd.B+ud.B[d]+esd.B
  }

  # Take the bootstrap sample data and fit the nested-error model to it

  fit.B<-lme(ys.B~-1+Xs,random=~1|as.factor(dom),method="REML")
  betaest.B<-fixed.effects(fit.B)
  upred.B<-random.effects(fit.B)
  sigmae2est.B<-fit.B$sigma^2
  sigmau2est.B<-as.numeric(VarCorr(fit.B)[1,1])

  # Generate non-sample values only for selected domains

  for (i in 1:I){

    # Print order of target domain
    cat("Domain num.",i,"\n")

    # Target domain
    d<-seldomain[i]

    # Matrix with the values of the p auxiliary variables for the
    # out-of-sample observations in i-th target domain
    Xrd<-Xrdtot[[i]]

    # Vector of out-of-sample random errors in target domain d
    erd.B<-rnorm(rd[d],0,sqrt(sigmae2est))

    # Vector of out-of-sample marginal means
    murd.B<-Xrd%*%matrix(betaest,nr=p,nc=1)

    # Vector of out-of-sample responses
    yrd.B<-murd.B+ud.B[d]+erd.B
    Erd.B<-exp(yrd.B)-m

    # Merge sample and non-sample elements for target domain d

    ysd.B<-ys.B[dom==d]
```

```
Esd.B<-exp(ysd.B)-m
Ed.B<-c(Erd.B,Esd.B)

# Calculate true domain poverty incidence and gap for target
# domain d

truepovinc.B[d]<-mean(Ed.B<z)
truepovgap.B[d]<-mean((Ed.B<z)*(z-Ed.B)/z)

# Compute EB predictors for each bootstrap sample

# Conditional mean for non-sample units
mudpred.B<-Xrd%*%matrix(betaest,nr=p,nc=1)+upred.B[d,1]

# The conditional distribution of (non-sample data given sample data)
# in the EB method can be expressed as a new nested-error model with
# different random effects variance. We calculate this random effects
# variance (called sigmav2).

gammad.B<-sigmau2est.B/(sigmau2est.B+sigmae2est.B/nd[d])
sigmav2.B<-sigmau2est.B*(1-gammad.B)

for (ellb in 1:LB){   ### Monte Carlo approximation of EB predictors

  # Generate random effect for target domain d
  vd.B<-rnorm(1,0,sqrt(sigmav2.B))

  # Generate random errors for all out-of-sample units in target domain d
  ed.B<-rnorm(rd[d],0,sqrt(sigmae2est.B))

  # Compute vector of out-if-sample responses
  yrdpred.B<-mudpred.B+vd.B+ed.B

  # Merge non-sample and sample values (full population or census)
  # for domain d.
  ydnew.B<-c(ysd.B,yrdpred.B)
  Ednew.B<-exp(ydnew.B)-m

  # EB predictors of poverty measures for each simulation

  povinc.B[d,ellb]<-mean(Ednew.B<z)
  povgap.B[d,ellb]<-mean((Ednew.B<z)*(z-Ednew.B)/z)

}  # End of Monte Carlo generations

# EB predictors of poverty measures

povinc.EB.B[d]<-mean(povinc.B[d,])
povgap.EB.B[d]<-mean(povgap.B[d,])
```

```
    # Cumulated squared errors for Bootstrap MSE
    MSEpropsum.B[d]<-MSEpropsum.B[d]+(povinc.EB.B[d]-truepovinc.B[d])^2
    MSEgapsum.B[d]<-MSEgapsum.B[d]+(povgap.EB.B[d]-truepovgap.B[d])^2
  } # En of cycle for target area
}  # End of the bootstrap cycle
time2 <- Sys.time()
time2<-difftime(time2,time1,units="mins")
print(time2)

# Bootstrap MSEs

for (i in 1:I){
  d<-seldomain[i]

  # Bootstrap MSS of EB estimators
  MSEpovincEB.B[d]<-MSEpropsum.B[d]/B
  MSEpovgapEB.B[d]<-MSEgapsum.B[d]/B
}

# Save results
Results<-data.frame(Domain=seldomain,SampSize=nd[seldomain],
PBMSEpovinc=MSEpovincEB.B[seldomain]*10000,
PBMSEpovgap=MSEpovgapEB.B[seldomain]*10000)

return (Results)
} # End of function PBMSE.EB
```

27.3 R code of FGTpovertyEBsample

The R code of the function `FGTpovertyEBsample()` is listed bellow.

```
############################################################################
###
### This function fits a unit level model to the log of the welfare variable
### and obtains approximate EB estimators of FGT poverty measures of order 1
### and 2 (poverty incidence and poverty gap) when the population values of
### auxiliary variables are not available and only sample data are available.
###
### Work for European project SAMPLE
###
### Author: Isabel Molina
### File name: FGTpovertyEBsample.R
### Updated: February 8th, 2011
###
############################################################################

FGTpovertyEBsample<-function(dom,seldomain=unique(dom),welfare,Xs,weight,
z=0.6*median(welfare),L=50,seed=Sys.time()){
```

```
# Load library required to fit a mixed model

library(nlme)

# Set the seed for random number generation,
# required for the Monte Carlo approximation of the EB method.
set.seed(seed)

# Number of domains for which EB estimators of poverty measures are required
# called (target domains)
I<-length(unique(seldomain))

# Total number of domains with sample data
D<-length(unique(dom))

# Sample and population sizes of target domains
nd<-rep(0,D)
Nd<-rep(0,D)

for (i in 1:I){
    nd[seldomain[i]]<-sum(dom==seldomain[i])
    weightd<-round(weight[dom==seldomain[i]])
    Nd[seldomain[i]]<-sum(weightd)
}

# Sizes of out-of-sample data in each domain
rd<-Nd-nd

# Take logarithm of welfare after adding a constant to make it positive
m<-abs(min(welfare))+1
welfaret<-welfare+m
ys<-log(welfaret)

# Fit the nested-error model to sample data by REML method using function lme
# from library nlme.

p<-dim(Xs)[2]
fit.EB<-lme(ys~-1+Xs,random=~1|as.factor(dom),method="REML")
Resultsfit<-list(Summary=summary(fit.EB),FixedEffects=fixed.effects(fit.EB),
RandomEffects=random.effects(fit.EB),ResVar=fit.EB$sigma,
RandomEffVar=as.numeric(VarCorr(fit.EB)[1,1]),Loglike=fit.EB$logLik,
RawResiduals=fit.EB$residuals[1:n])

# Save some of the results of the fitting method in variables

betaest<-fixed.effects(fit.EB)# Vector of model coefficients (size p)
upred<-random.effects(fit.EB) # Matrix with predicted random effects in 1st col.
sigmae2est<-fit.EB$sigma^2    # Estimated error variance
```

```
sigmau2est<-as.numeric(VarCorr(fit.EB)[1,1]) # Estimated random effects var.

# EB method starts: Generate L vectors of non-sample values of the response
# from their conditional distribution given the sample data and calculate
# empirical values of EB estimators.

# Matrices with poverty incidences and gaps for the L simulations in the
# EB method.
povinc<-matrix(0,nr=D,nc=L)
povgap<-matrix(0,nr=D,nc=L)

# Vectors with final EB estimators
povinc.EB<-rep(0,D)
povgap.EB<-rep(0,D)

# Time counter initialization
time1<-Sys.time()
time1

for (i in 1:I){    # Cycle for target domain

  # Print order of target domain
  cat("Domain num.",i,"\n")

  # Code of target domain
  d<-seldomain[i]

  # Obtain sample values for selected domain

  ysd<-ys[dom==d]
  Xsd<-Xs[dom==d,]
  wd<-round(weight[dom==d])

  # Compute conditional means for non-sample units. For this, generate
  # out-of-sample values of the means for domain d repeating the sample
  # values a number of times equal to their sampling weight minus one.

  musdpred<-Xsd%*%matrix(betaest,nr=p,nc=1)+upred[d,1]

  mudpred<-NULL
  for (i in 1:nd[d]){
    mudpred<-c(mudpred,rep(musdpred[i],(wd[i]-1)))
  }

  # The conditional distribution of (non-sample data given sample data)
  # in the EB method can be expressed as a new nested-error model with
  # different random effects variance. We calculate this random effects
  # variance (called sigmav2).
  gammad<-sigmau2est/(sigmau2est+sigmae2est/nd[d])
  sigmav2<-sigmau2est*(1-gammad)

  for (ell in 1:L){   ### Start of Monte Carlo simulations for EB method

    # Generate random effect for target domain d
    vd<-rnorm(1,0,sqrt(sigmav2))

    # Generate model random errors for all out-of-sample units in target
    # domain d.
    ed<-rnorm(rd[d],0,sqrt(sigmae2est))

    # Compute vectors of out-of-sample responses
    yrdpred<-mudpred+vd+ed

    # Merge non-sample and sample values (full population or census for
    # target domain d)

    ydnew<-c(ysd,yrdpred)

    # Compute province poverty measures with population values

    Ednew<-exp(ydnew)-m
    povinc[d,ell]<-mean(Ednew<z)
    povgap[d,ell]<-mean((Ednew<z)*(z-Ednew)/z)

  }  # End of simulations for EB method

  # EB predictors of poverty measures (averages over the L Monte Carlo
  # simulations)

  povinc.EB[d]<-mean(povinc[d,])
  povgap.EB[d]<-mean(povgap[d,])

} # End of cycle for target domain

time2 <- Sys.time()   # Current time
ComputTime<-difftime(time2,time1,units="mins")

# Results

EstimatedPoverty<-data.frame(Domain=seldomain,SampSz=nd[seldomain],
PovInc=100*povinc.EB[seldomain],PovGap=100*povgap.EB[seldomain])

return (list(EstimatedPoverty=EstimatedPoverty,ComputTime=ComputTime,
Resultsfit=Resultsfit))

} #End of function FGTpovertyEBsample
```

Chapter 28

Appendix 14: R code for the Fast EB method for poverty estimation

28.1 R code of FastEB

The R code of the function `FastEB` is listed bellow.

```
########################################################################
###
### This function fits a unit level model to the log of the welfare
### variable and the clog-log transformation of the score variable
### and obtains FAST-EB estimates of domain fuzzy poverty measures
### when the out-of-sample values of auxiliary variables are available.
###
### Work for European project SAMPLE
###
### Author: Caterina Ferretti
### File name: FUZZYpovertyFAST_EB.R
### Updated: February 18th, 2011
###
########################################################################

FUZZYpovertyFAST.EB<-function(welfare,score,Xs,dom,weight,
z=0.6*median(welfare),alpha=2,L=50){

  # Load library required to fit a unit level model

  library(nlme)

  # Vector of unique domains

  undom<-unique(dom)

  # Number of domains
```

```
D<-length(undom)

# Number of explanatory variables (including intercept)

p<-dim(Xs)[2]

# Domain sample and population sizes

nd<-rep(0,D)
Nd<-rep(0,D)

for (d in 1:D) {
  nd[d]<-sum(dom==d)
  Nd[d]<-sum(round(weight[dom==d]))
  }

# Sample size and estimated population size.

n<-sum(nd)
N<-sum(Nd)

# Head Count Ratio, FM and FS indicators,
# latent and manifest poverty indexes

HCRs<-matrix(0,nr=D,nc=1)
means<-matrix(0,nr=D,nc=1)
meanss<-matrix(0,nr=D,nc=1)
lats<-matrix(0,nr=D,nc=1)
mans<-matrix(0,nr=D,nc=1)

y1s<-NULL
v1s<-NULL
y2s<-NULL
v2s<-NULL
fms<-NULL
y1ss<-NULL
v1ss<-NULL
y2ss<-NULL
v2ss<-NULL
fss<-NULL
lat<-NULL
man<-NULL
sumnd<-0

for (i in 1:n){
      flags_i<- welfare > welfare[i]
      y1s[i]=sum(weight*flags_i)
      v1s[i]<-y1s[i]/sum(weight)
      y2s[i]=sum(weight*welfare*flags_i)
      v2s[i]<-y2s[i]/sum(weight*welfare)
      fms[i]<-v1s[i]^(alpha-1)*v2s[i]
      flagss_i<- score > score[i]
      y1ss[i]=sum(weight*flagss_i)
      v1ss[i]<-y1ss[i]/sum(weight)
      y2ss[i]=sum(weight*score*flagss_i)
      v2ss[i]<-y2ss[i]/sum(weight*score)
      fss[i]<-v1ss[i]^(alpha-1)*v2ss[i]
      lat[i]<-max(fms[i],fss[i])
      man[i]<-min(fms[i],fss[i])
      }

for (d in 1:D){ # For each domain d

     Esd<-welfare[dom==d]
     weightd<-weight[dom==d]

     HCRs[d,1]<-sum(weightd*(Esd<z))/sum(weightd)

     fmsd<-fms[dom==d]
     fssd<-fss[dom==d]
     latd<-lat[dom==d]
     mand<-man[dom==d]

     means[d,1]<-sum(weightd*fmsd)/sum(weightd)
     meanss[d,1]<-sum(weightd*fssd)/sum(weightd)
     lats[d,1]<-sum(weightd*latd)/sum(weightd)
     mans[d,1]<-sum(weightd*mand)/sum(weightd)

      sumnd<-sumnd+nd[d]
    }

# Log transformation of the welfare variable "welfare"

ys<-log(welfare)

# clog-log transformation of the non monetary variable "score"

yss<-log(-log(1-score))

# Fitting the nested-error model to sample data using R function lme
# from library nlme for the welfare variable

fit<-lme(ys~-1+Xs,random=~1|as.factor(dom),method="REML")

# Create a list object containing different results from the model fit

Resultsfit<-list(Summary=summary(fit),FixedEffects=fixed.effects(fit),
RandomEffects=random.effects(fit),ResVar=fit$sigma,
  RandomEffVar=as.numeric(VarCorr(fit)[1,1]),Loglike=fit$logLik,
  RawResiduals=fit$residuals[1:n])

  betaest<-fixed.effects(fit)  # Estimated model coefficients
  upred<-random.effects(fit)   # Predicted random domain effects
  sigmae2est<-fit$sigma^2      # Residual variance
  sigmau2est<-as.numeric(VarCorr(fit)[1,1]) # Cov. matrix of random effects

  # Fitting the nested-error model to sample data using R function lme
  # from library nlme for the non-monetary variable

  fits<-lme(yss~-1+Xs,random=~1|as.factor(dom),method="REML")

  # Create a list object containing different results from the model fits

  Resultsfits<-list(Summary=summary(fits),FixedEffects=fixed.effects(fits),
  RandomEffects=random.effects(fits),ResVar=fits$sigma,
  RandomEffVar=as.numeric(VarCorr(fits)[1,1]),Loglike=fit$logLik,
  RawResiduals=fits$residuals[1:n])

  betaests<-fixed.effects(fits) # Estimated model coefficients
  upreds<-random.effects(fits)  # Predicted random domain effects
  sigmae2ests<-fits$sigma^2     # Residual variance
  sigmau2ests<-as.numeric(VarCorr(fits)[1,1]) # Cov. matrix of random ef.

############################################################
# Generation of non-sample values and calculation of
# FASTEB and ELL province poverty fuzzy measures
############################################################

  # Initialize vectors

  # Matrix with domain poverty incidences for each Monte Carlo replication
  # in fast EB method
  povinc<-matrix(0,nr=D,nc=1)
  # Vector with estimated domain poverty incidences
  # by fast EB method
  povincfin<-rep(0,D)
  # Matrix with domain FM indicators for each Monte Carlo replication
  # in fast EB method
  povFM<-matrix(0,nr=D,nc=1)
  # Vector with estimated domain FM indicators by fast EB method
  povFMfin<-rep(0,D)
  # Matrix with domain FS indicators for each Monte Carlo replication
  # in fast EB method
  povFS<-matrix(0,nr=D,nc=1)
  # Vector with estimated domain FM indicators by fast EB method
  povFSfin<-rep(0,D)
  # Matrix with domain latent poverty for each Monte Carlo replication
  # in fast EB method
  LAT<-matrix(0,nr=D,nc=1)
  # Vector with estimated domain latent poverty by fast EB method
  LATfin<-rep(0,D)
  # Matrix with domain manifest poverty for each Monte Carlo replication
  # in fast EB method
  MAN<-matrix(0,nr=D,nc=1)
  # Vector with estimated domain manifest poverty by fast EB method
  MANfin<-rep(0,D)

    for (ell in 1:L){   ### Start of Monte Carlo generations

    samp<-NULL
    Enew<-NULL
    scnew<-NULL
    sumnd<-0

    for (d in 1:D){ # For each domain d

    # Drawing of a sample with the same size of the original sample
    # and probability proportional to sample weights

    weightd<-round(weight[dom==d])
    samp<-c(samp,sample((sumnd+1):(sumnd+nd[d]),size=nd[d],
    replace=TRUE,prob=weightd))
    Xr<-Xs[samp,]
    Xrd<-Xr[(sumnd+1):(sumnd+nd[d]),]

    # Compute conditional means for out-of-sample units

    mudpred<-Xrd%*%matrix(betaest,nr=p,nc=1)+upred[d,1]

    gammad<-sigmau2est/(sigmau2est+sigmae2est/nd[d])
    sigmav2<-sigmau2est*(1-gammad)

    # Generate random effect for target domain d
    vd<-rnorm(1,0,sqrt(sigmav2))

    # Generate random errors for all out-of-sample units in target
    # domain d
    ed<-rnorm(nd[d],0,sqrt(sigmae2est))

    # Compute vector of out-if-sample responses
    ydnew<-mudpred+vd+ed

    # Welfare variable, inverse tranformation of response

    Ednew<-exp(ydnew)
    Enew<-c(Enew,Ednew)
```

```
Xrs<-Xs[samp,]

Xrds<-Xrs[(sumnd+1):(sumnd+nd[d]),]

mudpreds<-Xrds%*%matrix(betaests,nr=p,nc=1)+upreds[d,1]

gammads<-sigmau2ests/(sigmau2ests+sigmae2ests/nd[d])
sigmav2s<-sigmau2ests*(1-gammads)

vds<-rnorm(1,0,sqrt(sigmav2s))
eds<-rnorm(nd[d],0,sqrt(sigmae2ests))

ydnews<-mudpreds+vds+eds

# Non-monetary variable, inverse tranformation of response

scdnew<-1-exp(-exp(ydnews))
scnew<-c(scnew,scdnew)

sumnd<-sumnd+nd[d]
    }

# FASTEB predictors of poverty measures

y1new<-NULL
v1new<-NULL
y2new<-NULL
v2new<-NULL
fmnew<-NULL

y1news<-NULL
v1news<-NULL
y2news<-NULL
v2news<-NULL
fsnew<-NULL

latnew<-NULL
mannew<-NULL

for (i in 1:n){

flagnew_i<- Enew > Enew[i]
y1new[i]=sum(flagnew_i)
v1new[i]<-y1new[i]/(n-1)
y2new[i]=sum(Enew*flagnew_i)
v2new[i]<-y2new[i]/sum(Enew)

fmnew[i]<-v1new[i]^(alpha-1)*v2new[i]
```

```
flagnews_i<- scnew > scnew[i]
y1news[i]=sum(flagnews_i)
v1news[i]<-y1news[i]/(n-1)
y2news[i]=sum(scnew*flagnews_i)
v2news[i]<-y2news[i]/sum(scnew)

fsnew[i]<-v1news[i]^(alpha-1)*v2news[i]

latnew[i]<-max(fmnew[i],fsnew[i])
mannew[i]<-min(fmnew[i],fsnew[i])
}

sumnd<-0

for (d in 1:D){

    Ednew<-Enew[(sumnd+1):(sumnd+nd[d])]
    fmdnew<-fmnew[(sumnd+1):(sumnd+nd[d])]
    fsdnew<-fsnew[(sumnd+1):(sumnd+nd[d])]
    latdnew<-latnew[(sumnd+1):(sumnd+nd[d])]
    mandnew<-mannew[(sumnd+1):(sumnd+nd[d])]

    # FASTEB predictors of poverty measures for each domain

    povinc[d,1]<-povinc[d,1]+mean(Ednew<z)
    povFM[d,1]<-povFM[d,1]+mean(fmdnew)
    povFS[d,1]<-povFS[d,1]+mean(fsdnew)
    LAT[d,1]<-LAT[d,1]+mean(latdnew)
    MAN[d,1]<-MAN[d,1]+mean(mandnew)

    sumnd<-sumnd+nd[d]

    }  # End of cycle for d
}  # End of generations


# FASTEB predictors of poverty measures (averages over the
# L Monte Carlo simulations)

povincfin<-povinc/L
povFMfin<-povFM/L
povFSfin<-povFS/L
LATfin<-LAT/L
MANfin<-MAN/L

#Results

EstimatedPoverty<-data.frame(Domain=undom,PopnSize=Nd,SampSize=nd,
    FASTEBPovinc=povincfin,FASTEBFM=povFMfin,FASTEBFS=povFSfin)
    return(list(EstimatedPoverty=EstimatedPoverty,Resultsfit=Resultsfit,
    Resultsfits=Resultsfits))

} # end of function FUZZYpovertyFAST.EB
```

Bibliography

ANSELIN, L. (1988).*Spatial Econometrics. Methods and Models*. Boston: Kluwer Academic Publishers.

ARORA, V. & LAHIRI, P. (1997). Bayesian method over the BLUP in small area estimation problems. *Statistica Sinica* **7**, 1053–1063.

BANERJEE, S., CARLIN, B. & GELFAND, A.(2004). *Hierarchical Modeling and Analysis for Spatial Data*. New York: Chapman and Hall.

CHAMBERS, R. & DUNSTAN (1986). Estimating distribution function from survey data. *Biometrika*, **73**, 597–604.

CHAMBERS, R. & TZAVIDIS, N. (2006). M-quantile models for small area estimation. *Biometrika*, **93**, 255–68.

CRESSIE, N. (1993). *Statistics for Spatial Data*.New York: John Wiley & Sons.

DATTA, G. S. & LAHIRI, P. (2000). A unified measure of uncertainty of estimated best linear unbiased predictors in small area estimation problems. *Statistica Sinica* **10**, 613–627.

DATTA, G. S. & RAO, J.N.K. & SMITH D.D.(2005). On measuring the variability od small area estimators under a basic area. *Biometrika* **92**, 183–196.

FAY, R. & HERRIOT, R. (1979). Estimates of income for small places: an application of James–Stein procedures to census data. *Journal of the American Statistical Association* **74**, 269–277.

GONZÁLEZ-MANTEIGA, W., LOMBARDÍA, M., MOLINA, I., MORALES, D. & SANTAMARÍA, L. (2008). Analytic and bootstrap approximations of prediction errors under a multivariate Fay–Herriot model. *Computational Statistics and Data Analysis*, **52**, 5242–5252.

JIANG, J. (1996). REML estimation: asymptotic behavior and related topics. *Annals of Statistics* **24**, 255-286.

LOMBARDIA, M., GONZALEZ-MANTEIGA, W. & PRADA-SANCHEZ, J. (2003). Bootstrapping the Chambers-Dunstan estimate of finite population distribution function. *Journal of Statistical Planning and Inference*, **116**, 367–388.

MARCHETTI, S. & textscTzavidis, N. $ PRATESI, M.(2011). Non-parametric Bootstrap Mean Squared Error Estimation for M-quantile Estimators of Small Area Averages, Quantiles and Poverty Indicators. *Department of Statistics and Mathematics Applied to Economics, University of Pisa*, Report 343

PRASAD, N. & RAO, J. (1990). The estimation of the mean squared error of small-area estimators. *Journal of the American Statistical Association* **85**, 163–171.

PETRUCCI, A. & SALVATI, N. (2006). Small area estimation for spatial correlation in watershed erosion assessment. *Journal of Agricultural, Biological and Environmental Statistics* **11**, 169–182.

PFEFFERMANN, D. & TILLER, R. (2005). Bootstrap approximation to prediction MSE for state-space models with estimated parameters. *Journal of Time Series Analysis* **26**, 893–916.

PRASAD, N. & RAO, J. (1990). The estimation of the mean squared error of small-area estimators. *Journal of the American Statistical Association* **85**, 163–171.

ROYALL, R. M. & CUMBERLAND, W. G. (1978). Variance estimation in finite population sampling. *Journal of the American Statistical Association* **73**, 351–358.

RUPPERT, D. & WAND, M. P. & textscCarrol, R. (2003). textitSemiparametric regression. Cambridge University Press, Cambridge, New York.

SALVATI, N., TZAVIDIS, N., PRATESI, M. & CHAMBERS, R. (2008) Small Area Estimation Via M-quantile Geographically Weighted Regression. *University of Manchester, CSSR*, paper submitted for publication, currently under review.

SALVATI, N., RANALLI, M. & PRATESI, M. (2010) Small area estimation of the mean using non-parametric M-quantile regression: a comparison when a linear mixed model does not hold. To appear in Journal of Statistical Computation and Simulation.

SINGH, B., SHUKLA, G. & KUNDU, D.(2005). Spatio-temporal models in small area estimation. *Survey Methodology* **31**, 183–195.

TZAVIDIS, N., MARCHETTI, S. & CHAMBERS, R. (2010). Robust estimation of small area means and quantiles. *Australian and New Zealand Journal of Statistics* **52**(2), 167-186.

YOU, Y. & CHAPMAN, B. (1979). Small Area Estimation Using Area Level Models and Estimated Sampling Variances. *Survey Methodology* **32**, 97–103.