Package: smallarea (via r-universe)

September 6, 2024

Type Package

Title Fits a Fay Herriot Model

Version 0.1

Date 2015-10-03

Author Abhishek Nandy

Maintainer Abhishek Nandy <nandy006@umn.edu>

Description Inference techniques for Fay Herriot Model.

License GPL(>=2)

Depends MASS

Repository https://nandy006.r-universe.dev

RemoteUrl https://github.com/nandy006/smallarea

RemoteRef HEAD

RemoteSha 319c2e512fa24e3e4ce201b76fea6988c870aea6

Contents

`smallarea-package` *Fits a Fay Herriot model*

Description

It has some useful functions which the users might find convenient for fitting Fay Herriot Model. Details are included in package vignette.

Details

Package:	smallarea
Type:	Package
Version:	0.1
Date:	2013-04-25
License:	GPL(>=2)

Author(s)

Abhishek Nandy

Maintainer: Abhishek nandy<nandy006@umn.edu>

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13. Small Area Estimation, JNK Rao, Wiley 2003 Variance Components, Wiley Series in Probability and Statistics,2006 Searle, Casella, Mc-Culloh

Examples

```
data=data.frame(response=c(1,2,3,4,8),D=c(0.2,0.5,0.1,0.9,1))
data
ans=smallareafit(response~D,data,method="FH")
ans1=smallareafit(response~D,data,method="REML")
ans2=smallareafit(response~D,data,method="PR")
ans3=smallareafit(response~D,data,method="ML")
```

`estimate.unknownsampvar`

Estimates of variance component, unknown sampling variance, regression coefficients and small area means in Fay Herriot model with unknown sampling variance.

Description

The function returns a list of 5 elements. The first element is an estimate of the variance component , the second element is an estimate of the parameter related to sampling variance, the third element is a vector of estimates of the regression coefficients in the Fay-Herriot model, the fourth element is a vector of the predictors pf the small area means and last element is the design matrix, the first column being a column of ones and the remaining columns represent the values of the covariates for different small areas. See details below.

Usage

```
estimate.unknownsampvar(response, mat.design, sample.size)
```

Arguments

`response`	A numeric vector. It represents the response or the direct survey based estimators in the Fay Herriot Model
`mat.design`	A numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.
`sample.size`	A numeric vector. The known sample sizes used to calculate the direct survey based estimators.

Details

For more details please see the package vignette.

Value

`psi.hat`	Estimate of the variance component
`del.hat`	Estimate of the parameter for sampling variance
`beta.hat`	Estimate of the unknown regression coefficients
`theta.hat`	Predictors of the small area means
`mat.design`	design matrix

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13.

See Also

prasadraoest fayherriot

Examples

```
set.seed( 55 )                  # setting a random seed
require(MASS)                   # the function mvrnorm requires MASS
x1 <- rep( 1, 10 )              # vector of ones representing intercept
x2 <- rnorm( 10 )               # vector of covariates randomly generated
x <- cbind( x1, x2 )            # design matrix
x <- as.matrix( x )             # coercing into class matrix
n <- rbinom (10, 20, 0.4)       # generating sample sizes for each small area
psi <- 1                        # true value of the psi parameter
delta <- 1                      # true value of the delta parameter
beta <- c( 1, 0.5 )             # true values of the regression parameters
theta <- mvrnorm( 1, as.vector( x %*% beta ), diag(10) )  # true small area means
y <- mvrnorm( 1, as.vector( theta ), diag( delta/n ) )  # design based estimators
estimate.unknownsampvar( y, x, n )
```

---

fayherriot *Estimate of the variance component in Fay Herriot Model using Fay Herriot Method*

Description

This function returns a list with one element in it which is the estimate of the variance component in the Fay Herriot Model. The estimate is found by solving an equation (for details see vignette) and is due to Fay Herriot. The uniroot in the stats package is used to find the root. uniroot searches for a root of that equation in a particular interval the lower bound is 0 and the upper bound is set to estimate of the variance component using Prasad Rao method + three times the square root of the number of observation. It depends on the function prasadraoest in the same package. Note that our function does not accept missing values.

Usage

```
fayherriot(response, designmatrix, sampling.var)
```

Arguments

response	a numeric vector. It represents the response or the observed value in the Fay Herriot Model
designmatrix	a numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.
sampling.var	a numeric vector consisting of the known sampling variances of each of the small area levels.

Details

For more details please see the attached vignette

Value

estimate	estimate of the variance component

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13.

See Also

prasadraoest maximlikelihood resimaxilikelihood

Examples

```
response=c(1,2,3,4,5)
designmatrix=cbind(c(1,1,1,1,1),c(1,2,4,4,1),c(2,1,3,1,5))
randomeffect.var=c(0.5,0.7,0.8,0.4,0.5)
fayherriot(response,designmatrix,randomeffect.var)
```

---

maximlikelihood *Maximum likelihood estimates of the variance components and the unknown regression coefficients in Fay Herriot Model.*

Description

This function returns a list of three elements the first one is the maximum likelihood estimate of the variance component,the second one is a vector of the maximum likelihood estimate of the unknown regression coefficients the first one being the coefficient of the intecept and the remaining ones are in the same order as the columns of the design matrix and the last one being the value of the maximized loglikelihood function in Fay Herriot model. it uses the optim in the stats package and the BFGS algorithm to minimize the negative loglikelihood. The initial value for this iterative proceedure of maximization are chosen as follows. The initial value for the variance component is the fay Prasad-rao estimate of the variance component, the initial value for the regression coefficients are the estimates of the regression coefficients using the multiple linear regression and ignoring the random effects.(For more details see vignette). Note that our function does not accept any missing values.

Usage

```
maximlikelihood(response, designmatrix, sampling.var)
```

Arguments

`response`	A numeric vector. It represents the response or the direct survey based estimators in the Fay Herriot Model
`designmatrix`	A numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.
`sampling.var`	A numeric vector consisting of the known sampling variances of each of the small area levels.

Details

For more details please see the package vignette

Value

`estimate`	Maximum likelihood estimate of the variance component
`reg.coefficients`	Maximum likelihood estimate of the unknown regression coefficients
`loglikeli.optimum`	The maximized value of the log likelihood function

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13.

See Also

prasadraoest fayherriot resimaxilikelihood

Examples

```
response=c(1,2,3,4,5)
designmatrix=cbind(c(1,1,1,1,1),c(1,2,4,4,1),c(2,1,3,1,5))
randomeffect.var=c(0.5,0.7,0.8,0.4,0.5)
maximlikelihood(response,designmatrix,randomeffect.var)
```

---

prasadraoest *Estimate of the variance component in Fay Herriot Model using Prasad Rao Method*

Description

This function returns a list with one element in it which is the estimate of the variance component in the Fay Herriot Model. The method used to get the estimate is the prasad rao method also known as BLUP.(for details see vignette). Note that our function does not accept any missing values.

Usage

```
prasadraoest(response, designmatrix, sampling.var)
```

Arguments

response	a numeric vector. It represents the response or the observed value in the Fay Herriot Model
designmatrix	a numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.
sampling.var	a numeric vector consisting of the known sampling variances of each of the small area levels.

Details

For more details see the package vignette

Value

estimate	estimate of the variance component

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13.

See Also

fayherriot maximlikelihood resimaxilikelihood

Examples

```
response=c(1,2,3,4,5)
designmatrix=cbind(c(1,1,1,1,1),c(1,2,4,4,1),c(2,1,3,1,5))
randomeffect.var=c(0.5,0.7,0.8,0.4,0.5)
prasadraoest(response,designmatrix,randomeffect.var)
```

---

prasadraomult *Estimate of the matrix of variance component in Fay Herriot Model using Prasad Rao Method*

Description

This function returns a matrix of the variance component in the multivariate Fay Herriot Model. It is an unbiased estimator based on method of moments and is analogous to the Univariate Fay herriot estimator. Note that our function does not accept any missing values. Also this function should not be use if you have only one small area attribute. Use the function prasadrao instead.

Usage

```
prasadraomult(directest.mat, samplingvar.mat, samplingcov.mat, design.mat)
```

Arguments

directest.mat	An nxq matrix of direct survey based estimates of q small area attributes on n small areas
samplingvar.mat	An nxq matrix of known sampling variances of the q small area attributes on n small areas. Note that the i-th row of this matrix represents the sampling variances of the q small area attributes associated with the i-th small area.
samplingcov.mat	An nx(q(q-1)/2) matrix of known sampling covariances of the q small area attributes on n small areas. Note that the i-th row of this matrix represents the sampling covariances of the q small area attributes associated with the i-th small area. The covariance terms in the matrix should occur in natural order, for example with 4 attributes the 6 columns of this matrix are cov(y1,y2), cov(y1,y3), cov(y1,y4), cov(y2,y3), cov(y2,y4),cov(y3,y4) respectively.
design.mat	A numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.

Value

`estimate` estimate of the matrix of variance component

Author(s)

Abhishek Nandy

See Also

prasadraoest

Examples

```
set.seed(55)
n <- 500                                          # Number of small areas
p <- 1                                      # Number of covariates (1 implies intercept only)
q <- 5                                    # Number of small area attributes for each small area
rho <- 0.3
psi <- matrix( c( 1, rho, rho^2, rho^3, rho^4,    # Matrix of variance component
                  rho, 1, rho,rho^2, rho^3,
                  rho^2, rho, 1 , rho, rho^2,
                  rho^3, rho^2, rho, 1, rho^3,
                  rho^4, rho^3,rho^2, rho, 1), q, q )
require( MASS )
Var <- matrix( rep( 1 , times = n*q ), n, q )     # Matrix of Sampling variances
samplingvar.mat <- Var

rho <- c( runif( n/2 , 0.8, 0.9 ), runif( n/2, -0.9, -0.8 ) )
rho2 <- rho^2
rho3 <- rho^3
rho4 <- rho^4

Cov <- as.matrix( cbind( rho, rho2, rho3, rho4, rho, rho2, rho3, rho, rho2, rho ) ) # Matrix of sampling covariances
samplingcov.mat <- Cov

theta <- rep( 0, times = n*q )
directest <- rep( 0, times = n*q )




for( i in 1:n ){                              # Simulating unknown small area means
  theta[ ( ( i -1 )*q + 1 ):( i*q ) ]  <- mvrnorm( 1, mu =  rep(0, q) ,
                                                   Sigma = psi )
}




  directest <- rep( 0, times = n*q )
  for( i in 1:n ){                           # Simulating direct survey based estimators
   directest[ ( ( i -1 )*q + 1 ):( i*q ) ] <- mvrnorm( 1, mu = theta[ ( ( i -1 )*q + 1 ):( i*q ) ],
```

```
                                        Sigma = matrix( c( 1, rho[i], rho[i]^2, rho[i]^3, rho[i]^4,
                                                      rho[i], 1, rho[i],rho[i]^2, rho[i]^3,
                                                     rho[i]^2, rho[i], 1 , rho[i], rho[i]^2,
                                                     rho[i]^3, rho[i]^2, rho[i], 1, rho[i]^3,
                                                     rho[i]^4, rho[i]^3,rho[i]^2, rho[i], 1)
                                                                    , q, q ))
}


directest.mat <- matrix( directest, n, q, byrow = TRUE )
prasadraomult( directest.mat,  samplingvar.mat, samplingcov.mat )
```

`resimaxilikelihood` *Estimate of the variance component in Fay Herriot Model using Residual Maximum Likelihood, REML.*

Description

This function returns a list with one element in it which is the estimate of the variance component in the Fay Herriot Model using residual maximum likelihood method. The estimates are obtained as a solution of equations known as REML equations. The solution is obtained numerically using Fisher-scoring algorithm. For more details please see the package vignette and the references. Note that our function does not accept any missing values.

Usage

```
resimaxilikelihood(response, designmatrix, sampling.var,maxiter)
```

Arguments

`response`	a numeric vector. It represents the response or the observed value in the Fay Herriot Model
`designmatrix`	a numeric matrix. The first column is a column of ones(also called the intercept). The other columns consist of observations of each of the covariates or the explanatory variable in Fay Herriot Model.
`sampling.var`	a numeric vector consisting of the known sampling variances of each of the small area levels.
`maxiter`	maximum number of iterations of fisher scoring

Details

For more details see the package vignette

Value

`estimate`	estimate of the variance component

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13. Small Area Estimation, JNK Rao,Wiley 2003 Variance Components, Wiley Series in Probability and Statistics,2006 Searle, Casella, Mc-Culloh

See Also

prasadraoest maximlikelihood fayherriot

Examples

```
response=c(1,2,3,4,5)
designmatrix=cbind(c(1,1,1,1,1),c(1,2,4,4,1),c(2,1,3,1,5))
randomeffect.var=c(0.5,0.7,0.8,0.4,0.5)
resimaxilikelihood(response,designmatrix,randomeffect.var,100)
```

---

smallareafit *Fits a Univariate Fay Herriot Model to data*

Description

Fits a Fay Herriot model to the data and returns a list of items which area estimates of different paramaters and mse of the estimates of the small area means the details of which are provided in the value section.

Usage

```
smallareafit(formula, data, method)
```

Arguments

formula	an object of class formula. a formula similar in appearance to that of in lm function in R. It has to be ascertained that the data contains a column of the sampling variances, and that while specifying the formula the the name of the variable that contains the sampling variances should preceede the variables which are the covariates. e.g response~D+x1+x2 is a correct way of specifying the formula where as response~x1+D+x2 is not.(note D is the variabe that contains the values of sampling variances and x1 and x2 are covariates). In general the first of the variables on the right hand side of ~ will be treated as the vector of sampling variance. Note that our function does not accept any missing values.
data	an optional data.frame. containg the variable names and data. in absence of this arguments the function will accept the corresponding things from the global environment.
method	method can be from "PR", "FH", "ML", "REML"

Details

for more details see the vignette

Value

smallmean.est	Estimates of the small area mean
smallmean.mse	Mean Square Prediction error of the estimates of the small area mead
var.comp	an estimate of the variance components
est.coef	an estimate of the regression coefficients

Author(s)

Abhishek Nandy

References

On measuring the variability of small area estimators under a basic area level model. Datta, Rao, Smith. Biometrika(2005),92, 1,pp. 183-196 Large Sample Techniques for Statistics, Springer Texts in Statistics. Jiming Jiang. Chapters - 4,12 and 13. Small Area Estimation, JNK Rao, Wiley 2003 Variance Components, Wiley Series in Probability and Statistics,2006 Searle, Casella, Mc-Culloh

See Also

prasadraoest fayherriot resimaxilikelihood maximlikelihood

Examples

```
data=data.frame(response=c(1,2,3,4,8),D=c(0.2,0.5,0.1,0.9,1))
data
ans=smallareafit(response~D,data,method="FH")
ans1=smallareafit(response~D,data,method="REML")
ans2=smallareafit(response~D,data,method="PR")
ans3=smallareafit(response~D,data,method="ML")
```

---

smallareafit.mult *Fits a multivariate Fay-Herriot model to data*

Description

Fits a Fay Herriot model to the data and returns a list of items which area estimates of different paramaters and mse of the estimates of the small area means the details of which are provided in the value section.

Usage

```
smallareafit.mult(directest.mat, samplingvar.mat, samplingcov.mat, design.mat)
```

Arguments

directest.mat
samplingvar.mat

samplingcov.mat

design.mat

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (directest.mat, samplingvar.mat, samplingcov.mat, design.mat)
{
    require("Rmpfr")
    options(digits = 16)
    if (missing(design.mat)) {
        n <- nrow(directest.mat)
        design.mat <- matrix(1, n, 1)
        p <- 1
        q <- ncol(directest.mat)
        prasadraomult.nocov <- function(directest.mat, samplingvar.mat,
            samplingcov.mat) {
            areas <- nrow(directest.mat)
            residual.transpose <- t(directest.mat) - rowMeans(t(directest.mat))
            leverage <- rep(1/areas, areas)
            adjust.varmat <- replicate(ncol(samplingvar.mat),
                1 - leverage) * samplingvar.mat
            adjust.var <- colSums(adjust.varmat)
            adjust.covmat <- replicate(ncol(samplingcov.mat),
                1 - leverage) * samplingcov.mat
            adjust.cov <- colSums(adjust.covmat)
            if (length(adjust.cov) == 1) {
                adjust.cov = as.numeric(adjust.cov)
            }
            adjustment.mat <- diag(adjust.var)
            adjustment.mat[lower.tri(adjustment.mat, diag = FALSE)] <- adjust.cov
            adjustment.mat <- forceSymmetric(adjustment.mat,
                uplo = "L")
            estimate <- (residual.transpose %*% t(residual.transpose) -
                adjustment.mat)/(areas - 1)
            if (det(estimate) <= 0) {
                return(1e-04 * diag(q))
            }
            else {
                return(estimate)
            }
        }
```

```
require(Matrix)
x <- 1:n
psi.hat <- prasadraomult.nocov(directest.mat, samplingvar.mat,
    samplingcov.mat)
options(digits = 16)
Sigmainv.list <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    D <- D + psi.hat
    D <- forceSymmetric(D, uplo = "L")
    return(solve(D))
})
inv.Sigma <- bdiag(Sigmainv.list)
designmatexpanded.list <- lapply(x, function(x) {
    as.matrix(bdiag(replicate(q, t(as.matrix(design.mat[x,
        ])), simplify = F)))
})
designmat.expanded <- do.call(rbind, designmatexpanded.list)
designmat.expanded <- Matrix(designmat.expanded, sparse = TRUE)
directest.vec <- as.vector(t(directest.mat))
beta.estimate <- solve(t(designmat.expanded) %*% inv.Sigma %*%
    designmat.expanded, sparse = TRUE) %*% (t(designmat.expanded) %*%
    directest.vec)
options(digits = 16)
B.est <- matrix(beta.estimate, q, p, byrow = TRUE)
if (det(psi.hat) <= 0.01) {
    thetaest.list <- lapply(x, function(x) {
        D <- diag(samplingvar.mat[x, ])
        D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
          ]
        D <- forceSymmetric(D, uplo = "L")
        B.est %*% design.mat[x, ]
    })
    theta.est <- t(do.call(cbind, thetaest.list))
}
else {
    thetaest.list <- lapply(x, function(x) {
        D <- diag(samplingvar.mat[x, ])
        D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
          ]
        D <- forceSymmetric(D, uplo = "L")
        psi.hat <- prasadraomult.nocov(directest.mat,
          samplingvar.mat, samplingcov.mat)
        solve((solve(D) + solve(psi.hat))) %*% (solve(D) %*%
          directest.mat[x, ] + solve(psi.hat) %*% B.est %*%
          design.mat[x, ])
    })
    theta.est <- t(do.call(cbind, thetaest.list))
}
g1 <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
```

```
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    g1 <- sum(diag(psi.hat %*% solve(D + psi.hat) %*%
        D))
    return(g1)
})
g1 <- unlist(g1)
mymat <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    return(solve(psi.hat + D))
})
mymat <- solve(Reduce("+", mymat))
g2 <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    mymat1 <- psi.hat %*% solve(D + psi.hat) %*% D
    g2 <- sum(diag(solve(psi.hat) %*% mymat1 %*% mymat1 %*%
        solve(psi.hat) %*% mymat))
    return(g2)
})
g2 <- unlist(g2)
z <- 1:n
fun5 <- function(z) {
    u <- matrix(NA, q, q)
    for (index1 in 1:q) {
        for (index2 in 1:q) {
          fun4 <- function(x) {
            D <- diag(samplingvar.mat[x, ])
            D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
              ]
            D <- forceSymmetric(D, uplo = "L")
            mat <- psi.hat + D
            fun1 <- function(index1, index2, index3,
              index4) {
              mat[index1, index2] * mat[index3, index4] +
                mat[index3, index2] * mat[index1, index4]
            }
            fun3 <- function(index3) {
              index4 <- 1:q
              fun2 <- function(index4) {
                fun1(index1, index2, index3, index4)
              }
              u <- lapply(index4, fun2)
              u <- unlist(u)
              return(u)
            }
            index3 <- 1:q
```

```
                v <- lapply(index3, fun3)
                return(matrix(unlist(v), nrow = q, byrow = TRUE))
            }
            v <- Reduce("+", lapply(x, fun4))
            D <- diag(samplingvar.mat[z, ])
            D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[z,
              ]
            D <- forceSymmetric(D, uplo = "L")
            psi.hat <- prasadraomult.nocov(directest.mat,
              samplingvar.mat, samplingcov.mat)
            dummy <- solve(D + psi.hat) %*% v
            u[index1, index2] <- sum(diag(dummy))/(nrow(design.mat) -
              ncol(design.mat))^2
        }
    }
    D <- diag(samplingvar.mat[z, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[z,
        ]
    D <- forceSymmetric(D, uplo = "L")
    g3 <- solve(psi.hat + D) %*% D %*% solve(psi.hat +
        D) %*% u
    g3 <- sum(diag(g3))
    return(g3)
}
g3 <- lapply(z, fun5)
g3 <- unlist(g3)
mspe <- g1 + 2 * g2 + g3
return(list(theta.est = theta.est, B.estimate = B.est,
    mspe = mspe, psi.hat = psi.hat))
}
else {
    n <- nrow(directest.mat)
    q <- ncol(directest.mat)
    p <- ncol(design.mat)
    prasadraomult <- function(directest.mat, samplingvar.mat,
        samplingcov.mat, design.mat) {
        require(Matrix)
        require("Rmpfr")
        projection <- function(mat) {
            mat %*% solve(t(mat) %*% mat) %*% t(mat)
        }
        areas <- nrow(directest.mat)
        proj <- projection(design.mat)
        fit <- lm(directest.mat ~ design.mat)
        residual.transpose <- t(fit$residuals)
        leverage <- as.vector(diag(proj))
        adjust.varmat <- replicate(ncol(samplingvar.mat),
            1 - leverage) * samplingvar.mat
        adjust.var <- colSums(adjust.varmat)
        adjust.covmat <- replicate(ncol(samplingcov.mat),
            1 - leverage) * samplingcov.mat
        adjust.cov <- colSums(adjust.covmat)
        if (length(adjust.cov) == 1) {
```

```
        adjust.cov = as.numeric(adjust.cov)
    }
    adjustment.mat <- diag(adjust.var)
    adjustment.mat[upper.tri(adjustment.mat, diag = FALSE)] <- adjust.cov
    adjustment.mat[lower.tri(adjustment.mat, diag = FALSE)] <- adjust.cov
    estimate <- (residual.transpose %*% t(residual.transpose) -
        adjustment.mat)/(areas - ncol(design.mat) - 1)
    return(estimate)
}
require(Matrix)
x <- 1:n
psi.hat <- prasadraomult(directest.mat, samplingvar.mat,
    samplingcov.mat, design.mat)
options(digits = 16)
Sigmainv.list <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    D <- D + psi.hat
    D <- forceSymmetric(D, uplo = "L")
    return(solve(D))
})
inv.Sigma <- bdiag(Sigmainv.list)
designmatexpanded.list <- lapply(x, function(x) {
    as.matrix(bdiag(replicate(q, t(as.matrix(design.mat[x,
        ])), simplify = F)))
})
designmat.expanded <- do.call(rbind, designmatexpanded.list)
designmat.expanded <- Matrix(designmat.expanded, sparse = TRUE)
directest.vec <- as.vector(t(directest.mat))
beta.estimate <- solve(t(designmat.expanded) %*% inv.Sigma %*%
    designmat.expanded, sparse = TRUE) %*% (t(designmat.expanded) %*%
    directest.vec)
options(digits = 16)
B.est <- matrix(beta.estimate, q, p, byrow = TRUE)
if (det(psi.hat) <= 0.01) {
    thetaest.list <- lapply(x, function(x) {
        D <- diag(samplingvar.mat[x, ])
        D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
          ]
        D <- forceSymmetric(D, uplo = "L")
        psi.hat <- prasadraomult(directest.mat, samplingvar.mat,
          samplingcov.mat, design.mat)
        B.est %*% design.mat[x, ]
    })
    theta.est <- t(do.call(cbind, thetaest.list))
}
else {
    thetaest.list <- lapply(x, function(x) {
        D <- diag(samplingvar.mat[x, ])
        D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
          ]
```

```
        D <- forceSymmetric(D, uplo = "L")
        psi.hat <- prasadraomult(directest.mat, samplingvar.mat,
          samplingcov.mat, design.mat)
        solve((solve(D) + solve(psi.hat))) %*% (solve(D) %*%
          directest.mat[x, ] + solve(psi.hat) %*% B.est %*%
          design.mat[x, ])
    })
    theta.est <- t(do.call(cbind, thetaest.list))
}
g1 <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    g1 <- sum(diag(psi.hat %*% solve(D + psi.hat) %*%
        D))
    return(g1)
})
g1 <- unlist(g1)
mymat <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    design <- as.vector(design.mat[x, ])
    design <- design %o% design
    return(kronecker(solve(psi.hat + D), design))
})
mymat <- solve(Reduce("+", mymat))
g2 <- lapply(x, function(x) {
    D <- diag(samplingvar.mat[x, ])
    D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
        ]
    D <- forceSymmetric(D, uplo = "L")
    mymat1 <- psi.hat %*% solve(D + psi.hat) %*% D
    design <- as.vector(design.mat[x, ])
    mymat2 <- kronecker(solve(psi.hat), t(design))
    g2 <- sum(diag(t(mymat2) %*% mymat1 %*% mymat1 %*%
        mymat2 %*% mymat))
    return(g2)
})
g2 <- unlist(g2)
z <- 1:n
fun5 <- function(z) {
    u <- matrix(NA, q, q)
    for (index1 in 1:q) {
        for (index2 in 1:q) {
          fun4 <- function(x) {
            D <- diag(samplingvar.mat[x, ])
            D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[x,
              ]
            D <- forceSymmetric(D, uplo = "L")
            mat <- psi.hat + D
```

```
                    fun1 <- function(index1, index2, index3,
                      index4) {
                      mat[index1, index2] * mat[index3, index4] +
                        mat[index3, index2] * mat[index1, index4]
                    }
                    fun3 <- function(index3) {
                      index4 <- 1:q
                      fun2 <- function(index4) {
                        fun1(index1, index2, index3, index4)
                      }
                      u <- lapply(index4, fun2)
                      u <- unlist(u)
                      return(u)
                    }
                    index3 <- 1:q
                    v <- lapply(index3, fun3)
                    return(matrix(unlist(v), nrow = q, byrow = TRUE))
                  }
                  v <- Reduce("+", lapply(x, fun4))
                  D <- diag(samplingvar.mat[z, ])
                  D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[z,
                    ]
                  D <- forceSymmetric(D, uplo = "L")
                  dummy <- solve(D + psi.hat) %*% v
                  u[index1, index2] <- sum(diag(dummy))/(nrow(design.mat) -
                    ncol(design.mat))^2
                }
              }
              D <- diag(samplingvar.mat[z, ])
              D[lower.tri(D, diag = FALSE)] <- samplingcov.mat[z,
                  ]
              D <- forceSymmetric(D, uplo = "L")
              psi.hat <- prasadraomult(directest.mat, samplingvar.mat,
                  samplingcov.mat, design.mat)
              g3 <- solve(psi.hat + D) %*% D %*% solve(psi.hat +
                  D) %*% u
              g3 <- sum(diag(g3))
              return(g3)
            }
            g3 <- lapply(z, fun5)
            g3 <- unlist(g3)
            mspe <- g1 + 2 * g2 + g3
            return(list(theta.est = theta.est, B.estimate = B.est,
                mspe = mspe, psi.hat = psi.hat))
          }
        }
```

Index