1 Introduction

Model-based small area estimators (for example [1], chapters 5ff.) depend on model assumptions to hold. This dependency doesn’t make them very attractive for official statistics.

Model-assisted small area estimators do not depend on the model assumptions to hold, albeit their variances will be higher if the model is inappropriate (see [2], chapter 6.7). The synthetic-regression estimator (SRE) (for example [1], chapter 4.2.2) is biased, and the variance of its biased-corrected version, the generalized regression estimator (for example [1], chapter 2.5), “depends crucially on the […] residuals in the small area” ([3], p. 444), which basically means that its variance will be unacceptably high in many applications.

Daniel Mandallaz and others ([4], [3], [5], [6], [7] and [8]) propose an unbiased extension of the SRE for two- and three-phase sampling designs with or without clustering. The variance of the extended SRE is, like that of the SRE, based on all residuals in the second (or third) phase and asymptotically equivalent to the SRE’s variance (see [3], p. 444).

2 Non-Exhaustive Auxiliary Information

2.1 Two-Phase Sampling

Let us suppose we have a two-phase sampling design, and the sample data can be loaded via

```r
> data("s2")
> data("s1")
```

We now add sampling phase indicators to the data and join (from a database point of view we do a union, but I’ll keep calling it join) it into a single data.frame s12:

```r
> s12 <- bind_data(s1, s2)
```

We build a small area Object

```r
> saeO <- saObj(data = s12, f = y ~ x1 + x2 + x3 | g,
+ s2 = "phase2")
```
and get the small area estimations for the non-exhaustive case as

\[
\text{predict(saeO)}
\]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>378.8590</th>
<th>487.3680</th>
<th>367.2796</th>
<th>303.1129</th>
<th>378.7573</th>
<th>533.7463</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>391.8262</td>
<td>417.3442</td>
<td>385.1562</td>
<td>314.9198</td>
<td>391.8016</td>
<td>455.7217</td>
</tr>
</tbody>
</table>

### 2.2 Three-Phase Sampling

Suddenly we stumble across data from a third sampling phase, the null phase \(s0\) has all the predictors of \(s2\), but we keep only one – if we kept all of them, we'd be back to two-phase sampling with more observations), and we join all three phases into \(s012\):

\[
\text{data("s0")} \\
\text{s0$x1 <- s0$x3 <- NULL} \\
\text{s012 <- bind_data(s1, s2, s0)}
\]

from which we predict again:

\[
\text{predict(saObj(data = s012, f = y ~x1 + x2 + x3 | g,} \\
\text{+ s2 = "phase2", s1 = "phase1")}
\]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>397.1866</th>
<th>311.8078</th>
<th>385.5963</th>
<th>82.87085</th>
<th>397.0740</th>
<th>313.5043</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>404.2197</td>
<td>271.9036</td>
<td>397.5939</td>
<td>82.71266</td>
<td>404.2394</td>
<td>223.5145</td>
</tr>
</tbody>
</table>

Note the drop in variance induced by the extensive null phase sampling.

### 3 Partially Exhaustive Auxiliary Information

Let us suppose we knew the estimated small area means of the fixed effect sampled in all three phases to be the true small area means:

\[
\text{tm1 <- as.data.frame(tapply(s012$x2, s012$g, mean))} \\
\text{names(tm1)[1] <- c("x2"); tm1$g <- row.names(tm1)} \\
\text{predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g,} \\
\text{+ s2 = "phase2", smallAreaMeans = tm1})
\]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>397.1866</th>
<th>295.8078</th>
<th>385.5963</th>
<th>66.28085</th>
<th>397.0740</th>
<th>313.5043</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>404.2197</td>
<td>254.9036</td>
<td>397.5939</td>
<td>64.71266</td>
<td>404.2394</td>
<td>223.5145</td>
</tr>
</tbody>
</table>

Again, the variance estimation is reduced, but not as markedly as before: due to the extensive null phase the mean estimations had very small variances (which added to the small area estimation variances).
4 Exhaustive Auxiliary Information

Of course we could also take our estimated small area means of all fixed effects from the first and second phase to be the true means:

```r
> preds <- paste("x",1:3, sep="")
> tm <- as.data.frame(rbind(colMeans(subset(s12, g == "a")[, preds]),
+ colMeans(subset(s12, g == "b")[, preds])
+ ))
+ ); tm$g=c("a", "b")
```

That would give us the smallest variance estimates:

```r
> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g,
+ s2 = "phase2", smallAreaMeans = tm))
```

<table>
<thead>
<tr>
<th>small_area prediction variance</th>
<th>psynth</th>
<th>var_psynth</th>
<th>psmall</th>
<th>var_psmall</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>378.8590</td>
<td>223.1295</td>
<td>367.2796</td>
<td>39.66430</td>
</tr>
<tr>
<td>b</td>
<td>391.8262</td>
<td>142.0318</td>
<td>385.1562</td>
<td>38.60882</td>
</tr>
</tbody>
</table>

5 Model-Based Small Area Estimation

Wait! If I use the code in Appendix A to calculate the EBLUP for the basic unit level model given by [1], chapter 7.2, I get at least a much smaller mse1:

```r
> source("Rao.R")
> library(nlme)
> dat <- subset(s2, ! is.na(s2$g))
> dat <- dat[with(dat, order(g)), TRUE]
> aLmeObj <- lme(y ~x1 + x2 + x3 | g,
+ s2 = "phase2", smallAreaMeans = tm)
> foo <- new(Class = "sae", lmeObj = aLmeObj, domain.df = tm)
> sae(foo)
```

<table>
<thead>
<tr>
<th>eblup</th>
<th>mse1</th>
<th>mse2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>376.6563</td>
<td>98.99138</td>
</tr>
<tr>
<td>b</td>
<td>393.4619</td>
<td>101.4835</td>
</tr>
</tbody>
</table>

Of course you do, but you rely on the model’s assumptions to hold. Have you checked them?

No, I don’t even know what they are. But even if the tests would not reject the hypotheses of the assumptions being valid on, say, a .95 confidence level, I could never be sure.

That’s why I wrote maSAE.

I see, but wait again, what is . . .

6 Cluster Sampling

```r
> grep(".*clust", capture.output(str(s1)), value = TRUE)
```

```
```
Dear, I forgot that sampling for my completely made up example data was done with a cluster design! So all the above variances estimates are too optimistic. My fault. For the sake of CPU time on CRAN, I’ll leave the following to you:

```r
> predict(saObj(data = s12, f = y ~ x1 + x2 + x3 | g,
+ s2 = "phase2", cluster = "clustid"))
> predict(saObj(data = s012, f = y ~ x1 + x2 + x3 | g,
+ s2 = "phase2", s1 = "phase1",
+ cluster = "clustid"))
> predict(saObj(data = s12, f = y ~ x1 + x2 + x3 | g,
+ s2 = "phase2", smallAreaMeans = tm1,
+ cluster = "clustid"))
> predict(saObj(data = s12, f = y ~ x1 + x2 + x3 | g,
+ s2 = "phase2", smallAreaMeans = tm,
+ cluster = "clustid"))
```

References


A Rao.R

```r
library("methods")
setOldClass(c("lme"))
setClass(Class = "sae",
  representation = representation(
    lmeObj = "lme",
    domain.df = "data.frame"
  ),
  validity = function(object){
    if(object$lmeObj$method != 'REML')
      return("lme method wasn't REML, can't handle this")
    if(! all(c(attr(getGroups(object$lmeObj)), 'label')
      , dimnames(attr(terms(formula(object$lmeObj))
        "factors"))[[2]])
        %in% names(object@domain.df))
      return("Names in domain.df do not match names in lmeObj")
    if(! all(eval(parse(text=paste(sep = ' ',
        'object@domain.df$', unitName)))
        %in% unique(getGroups(object$lmeObj))
      )
      return("Unknown unit in domain.df")
  )
  })
setGeneric(name = "sae",
  def = function(object, ...){standardGeneric("sae")}
)
setMethod(f = "sae",
  signature(object = "sae"),
  function(object){
    lmeObj <- object$lmeObj
    domain.df <- object@domain.df
    eval(parse(text=paste(sep = ' ',
      'domain.df$'
    , setdiff(dimnames(attr(terms(formula(lmeObj))
        "factors"))[[1]]
      , dimnames(attr(terms(formula(lmeObj))
        "factors"))[[2]])
    , '<=' 1'))
  }
  )
unitName <- attributes(getGroups(lmeObj))$label
units <- unique(getGroups(lmeObj))
unitseq <- seq(along = units); names(unitseq) <- units
n <- tapply(getGroups(lmeObj), getGroups(lmeObj), length)
# the estimated Variances, see p.137, last paragraph
varV <- as.numeric(VarCorr(lmeObj)[1,1])
varE <- as.numeric(VarCorr(lmeObj)[2,1])
# Vector of the mean Residual per Unit
```

[8] Daniel Mandallaz. Regression estimators in forest inventories with three-
phase sampling and two multivariate components of auxiliary information.
Technical report, Eidgenössische Technische Hochschule Zürich, Departement
Umwelt-systemwissenschaften, 2013.
mUnitRes <- tapply(resid(lmeObj, level = 0), names(resid(lmeObj, level = 0)), mean)[levels(units)]

## list of design matrices, multipurpose
lX <- lapply(X = unitsseq, FUN = function(i, lmeObj, unitName){
  unit <- units[i]
  X <- model.matrix(formula(lmeObj), subset(lmeObj$data, eval(parse(text = unit)) == unit))
  return(X)
}, lmeObj, unitName)

## list of R7.2.4, multipurpose
lR7.2.4 <- lapply(X = unitsseq, FUN = function(i, n, varV, varE){
  unit <- units[i]
  n.i <- n[unit]
  R7.2.4 <- varV / (varV + varE / n.i)
  return(R7.2.4)
}, n, varV, varE)

## list of R7.2.7 to build the sum in R7.2.12
lR7.2.7 <- lapply(X = unitsseq, FUN = function(i, n, lX, lR7.2.4, varV, varE){
  unit <- units[i]
  X <- lX[[unit]]
  n.i <- n[unit]
  R7.2.4 <- lR7.2.4[[unit]]
  R7.2.2 <- 1 / varE * (diag(n.i) - R7.2.4 / n.i * rep(1, n.i) %*% t(rep(1, n.i)))
  R7.2.7 <- t(X) %*% R7.2.2 %*% X
  return(R7.2.7)
}, n, lX, lR7.2.4, varV, varE)

sumAi <- Reduce('+', lR7.2.7)
lR7.2.30 <- lapply(X = unitsseq, FUN = function(i, n, varV, varE){
  unit <- units[i]
  n.i <- n[unit]
  R7.2.30 <- varE + varV * n.i
  return(R7.2.30)
}, n, varV, varE)

R7.2.27 <- 0.5 * sum(n^2 * unlist(lR7.2.30) - 2)
R7.2.28 <- 0.5 * sum((n - 1) * sqrt(varE) - 4 + unlist(lR7.2.30)^2 - 2)
R7.2.29 <- 0.5 * sum(n * unlist(1IR7.2.30)^-2)
informationMatrix <- matrix(data = c(R7.2.27, R7.2.29, R7.2.29, R7.2.28), ncol = 2)
asymtoticCovarianceMatrix <- solve(informationMatrix)
R7.2.23 <- varE^2 * asyntoticCovarianceMatrix[1, 1] + varV^2 * asyntoticCovarianceMatrix[2, 2] - 2 * varE * varV * asyntoticCovarianceMatrix[1, 2]
units <- eval(parse(text = paste(sep = ' ', 'domain.df$'), unitName))

unitsq <- seq(along = units)
names(unitsq) <- units

foo <- lapply(# over observations in domain.df
X = unitsq
, FUN = function(i, n, IX, lR7.2.4, varV, varE, sumAi, R7.2.23){
  unit <- units[i]
  X <- IX[[unit]]
  n.i <- n[unit]
  R7.2.4 <- lR7.2.4[[unit]]
  R7.2.11 <- R7.2.4 * varE / n.i
  Xbar <- model.matrix(formula(lmeObj), subset(domain.df, eval(parse(text = unitName)) == unit))
  xbar <- colMeans(X)
  tmp <- as.numeric(Xbar) - R7.2.4 * xbar
  R7.2.12 <- t(tmp) %*% solve(sumAi) %*% tmp
  R7.2.30 <- lR7.2.30[units[unit]]
  R7.2.22 <- n.i^-2 * (varV + varE / n.i)^-3 * R7.2.23
  R7.2.32 <- n.i^-2 * (varV + varE / n.i)^-4 * R7.2.23 * mUnitRes[unit]^2
  R7.2.33 <- R7.2.11 + R7.2.12 + 2 * R7.2.32
  R7.2.34 <- R7.2.11 + R7.2.12 + R7.2.22 + R7.2.32
  return(c(mse1 = R7.2.33
            , mse2 = R7.2.34
          )
        )
  }
, n
, IX
, lR7.2.4
, varV
, varE
, sumAi
, R7.2.23
)

## transform list to transposed data.frame
foo <- as.data.frame(t(as.data.frame(foo)))
return(cbind(ebUup = as.numeric(predict(lmeObj, newdata = domain.df)),
             , foo))
}