The *meerva* R-package for analysis of data subject to measurement error with an internal validation subsample

Walter K. Kremers, Mayo Clinic, Rochester MN

26 October 2021

The Package

Sometimes data for analysis are obtained using more convenient or less expensive means yielding "surrogate" variables for what could be obtained more accurately, albeit with less convenience; or less conveniently or at more expense yielding "reference" variables, thought of as being measured without error. Analysis of the surrogate variables measured with error generally yields biased estimates when the objective is to make inference about the reference variables. Often it is thought that ignoring the measurement error in surrogate variables only biases effects toward the null hypothesis, but this need not be the case. Measurement errors may bias parameter estimates either toward or away from the null hypothesis.

If one has a data set with surrogate variable data from the full sample, and also reference variable data from a randomly selected subsample, then one can assess the bias introduced by measurement error in parameter estimation, and use this information to derive improved estimates based upon all available data. Formulaically these estimates based upon the reference variables from the validation subsample combined with the surrogate variables from the whole sample can be interpreted as starting with the estimate from reference variables in the validation subsample, and "augmenting" this with additional information from the surrogate variables. This suggests the term "augmented" estimate.

The *meerva* package calculates these augmented estimates in the regression setting when there is a randomly selected subsample with both surrogate and reference variables. Measurement errors may be differential or non-differential, in any or all predictors (simultaneously) as well as outcome. The augmented estimates derive, in part, from the multivariate correlation between regression model parameter estimates from the reference variables and the surrogate variables, both from the validation subset. Because the validation subsample is chosen at random any biases imposed by measurement error, whether non-differential or differential, are reflected in this correlation and these correlations can be used to derive estimates for the reference variables using data from the whole sample. The augmented estiamtes, as well as how measurement error may arise in practice, is described in more detail by Kremers WK (2021) (arXiv:2106.14063, https://arxiv.org/abs/2106.14063) and is an extension of the works by Chen Y-H, Chen H. (2000) doi:10.1111/1467-9868.00243, Chen Y-H. (2002) doi:10.1111/1467-9868.00324, Wang X, Wang Q (2015) doi:10.1016/j.jmva.2015.05.017 and Tong J, Huang J, Chubak J, et al. (2020) doi:10.1093/jamia/ocz180.

Using the *meerva* package

The main function in the *meerva* package is meerva.fit() which calculates estimates from the input dataset. We provide no example datasets with *meerva* but instead provide programs to generate example data. In fact there are three functions to generate simulated data with measurement error of appropriate form for analysis 1) meerva.sim.brn() for binomial outcome and the logistic regression setting 2) meerva.sim.cox() for time to event data and Cox regression and 3) meerva.sim.nrm() for normal data and linear regression. Additionally meerva.sim.block() performs a simulation study generating multiple datasets and stores the results for inspection in a list object. There are no other functions in the package intended for direct usage by the user barring the print(), summary() and plot() functions.

As with all R-packages one must first install the package, and "add" to the working library. This can be done by running the two lines of code **install.packages("meerva")** and **library(meerva)** from the R "console".

Data requirements

The basic data elements for input to the *meerva* package for analysis are three matrices of predictors and three vectors of outcomes. When fitting a model in the Cox regression framework there will be three additional vectors for event vs censoring information. These are submitted as input to the meerva.fit() function. This may be somewhat awkward but these data elements are used internally for calculations so it is quite natural from a data flow perspective. The intent is to eventually write the function meerva(), which would serve as a wrapper for meerva.fit(), and parse a data frame into the pieces described here and pass to meerva.fit(). Many of the R functions for statistical analysis work in this manner, e.g. the commonly used glm() function can read from a data frame and parse the predictor variables and outcome variable before passing to glm.fit(), and similarly for coxph() and coxph.fit().

The three input matrices for the predictor variables are x_val, xs_val and xs_non. The first contains the information on the reference predictors from the validation subsample. The next, xs_val contains the information on the surrogate predictors from the validation subsample. xs_non contains information on the surrogate predictors form the non-validation subsample, that is the sample units not included in the validation subsample. Together xs_val and xs_non constitute the surrogate predictors for the whole sample. The three vectors for the predictor variables are y_val, ys_val and ys_non, with a naming convention parallel to that of the predictor variable matrices.

By default, each row in all of x_val, xs_val, y_val and ys_val has to include data for the same sample unit, e.g. row i has to include the data for same patient for the matrices and column vectors. Similarly, each row in xs_non and ys_non has to include data for the same sample unit. Importantly then, before passing data to meerva.fit one may need to sort the data and check for consistency of the rows for these different data units.

An example dataset

To demonstrate usage of meerva.fit() we first generate a dataset for analysis, discuss essential characteristics of the data as required for analysis, and then shown an example analysis. First after running the analysis do we go into detail on the program used to generate the simulated data. Generation of simulated data may be less of interest to those wanting to start with an analysis.

The code

```
# Simulate logistic regression data with measurement error
simd = meerva.sim.brn(n=4000, m=400,
     beta = c(-0.5, 0.5, 0.2, 1, 0.5) ,
     alpha1 = c(0.95, 0.90, 0.90, 0.95) ,
     alpha2 = c(0.98,0.94,0.95,0.95) ,
     bx3s1 = c(0.05, 0, 0, NA, NA) ,
     bx3s2 = c(NA,NA,NA) )
```

generates simulated data with measurement error with binomial outcomes and intended for analysis using the logistic regression framework. Here, the randomly generated data are stored in the output object simd. This full dataset has sample size of 4000 and the validation subsample has size 400. We extract data in the format required for input to the analysis program meerva.fit() using

```
# Read the simulated data to input data format
x_val  = simd$x_val   # reference predictors from the validation subset
y_val  = simd$y_val   # reference outcome from the validation subset
xs_val = simd$xs_val  # surrogate predictors from the validation subset
ys_val = simd$ys_val  # surrogate outcome from the validation subset
xs_non = simd$xs_non  # surrogate predictors from the non-validation
                      # sample units
ys_non = simd$ys_non  # surrogate outcome from the non-validation
                      # sample units
```

Inspecting the first few rows of the reference predictor variable matrix we see

```
# Print the first 10 elements of the y validaiton vector
print(x_val[1:10,])

##       x1 x2         x3          x4
##  [1,]  0  0  0.3627536  1.12215988
##  [2,]  1  0 -1.2701380 -1.58351783
##  [3,]  0  0  0.2678254 -1.21744969
##  [4,]  0  0  0.8093085 -0.58526992
##  [5,]  0  0 -0.3055455 -0.95833154
##  [6,]  1  0 -1.0052097  0.06790784
##  [7,]  1  0 -0.3890501 -1.25104539
##  [8,]  0  0  1.9126599 -0.29773421
##  [9,]  0  0  0.7984876  0.57395013
## [10,]  0  0  0.5495988 -0.22199711
```

As suggested by their appearance x1 and x2 are binomial predictors, and x3 and x4 are quantitative. Inspecting the first few elements of the reference outcome vector we see

```
# Print the first 10 elements of the y validaiton vector
print(y_val[1:10])

##  [1] 0 0 0 1 0 1 0 1 1 0
```

In their form xs_val and ys_val are similar to x_val and y_val but include the surrogate variables instead of the reference measures. xs_non and ys_non have analogous form but include variables form the non-validation subsample.

Analysis

To perform an analysis using either logistic, Cox or linear regression the user need only pass the data to meerva.fit() as the program first inspects the data to determine an appropriate model framework. The user may override the model framework for analysis selected by meerva.fit()by specifying this with the input variables faimlyr and familys for the reference and surrogate data, but generally will not need to do this. Current family options are binomial for logistic, Cox for Cox regression and gaussian for linear regression. The user may also provide vectors for sample unit id's (identifier or index) (e.g. patient id) and weights for sample units but this example does not involve these. Weights may be used in case of sampling with unequal probabilities or for of propensity scores. The sample unit id can be used in case of repeat records for sample units. This assures use of sandwich estimators of variances, or similar, accounting for within sample unit (patient) dependencies. The user may also specify whether to use sandwich variances methods in model derivation or jackknife estimates, i.e. choose faster or more accurate calculations. Here we use the default of faster calculations.

An example analysis can then be performed and results printed by

```
Ex1 = meerva.fit(x_val  = x_val , y_val  = y_val ,
                 xs_val = xs_val, ys_val = ys_val,
                 xs_non = xs_non, ys_non = ys_non )
print(Ex1)

##
## meerva.fit(x_val = x_val, y_val = y_val, xs_val = xs_val, ys_val = ys_val,
##     xs_non = xs_non, ys_non = ys_non)
##
##      familyr    familys    compare comparec vmethod vmethodc    vmethodr
## [1,] "binomial" "binomial" "1"     "ful"    NA      "ijk (alt)" "2"
##      vmethods n_val n_ful  dim_beta dim_gamma
## [1,] "2"      "400" "4000" "5"      "5"
##
##  Confidence intervals are for alpha = 0.05
##
##  Estimates for Beta using beta_aug
##               estimate         se        lcl        ucl           z
## (Intercept) -0.6268576 0.10099079 -0.8247959 -0.4289193 -6.2070768 5.40e-10
## x1          0.7436406 0.20911103  0.3337905  1.1534907  3.5561999 3.76e-04
## x2          0.1629424 0.23472323 -0.2971067  0.6229914  0.6941893 4.88e-01
## x3          1.2740382 0.12226244  1.0344083  1.5136682 10.4205201 2.00e-25
## x4          0.6054943 0.08634431  0.4362626  0.7747261  7.0125558 2.34e-12
```

The first output element is basically a readback of the call used to perform the analysis. Next are a number of model fit parameters which are either determined by the data or specified by the user. The first 2, familyr and familys, are the model families used to model the reference data (therefore the r in familayr) and surrogate data (therefore the s in familys). Usually, the reference and surrogate model families will be the same but they can be different, for example surrogate data might fail to record time to event and simply record whether or not an event was recorded. Here, since we did not specify the model family the meerva.fit() function identified the binomial outcomes fron the data and fit a logistic regression model. If we wanted to override this and fit a linear model we could have specified family=gaussian and familys=gaussian when calling meerva.fit(). The other elements n_val, n_ful, dim_beta, dim_gamma describe the validation sample size, the full sample size, the dimension of the model based upon the reference variables and the dimension of the model based upon the surrogate variables. The model dimension includes that of the interecept except for the Cox model where there is no intercept. In many cases, possibly most cases, there will be one surrogate variable for each reference variable. The method though allows for multiple surrogates for any or all of the reference variables, meaning the models based upon surrogates may have more parameters than the model based upon reference variables. Interestingly, the surrogate set may have fewer variables than the reference set. One way this may arise is in the special case of a missing data where a common set of variables is missing in a random subset of the data. This though would not work if different sample units had different variables missing, unless one were to discard some data to obtain the needed data structure. Finally, the print function provides the augmented estimates for the regression model, based upon both the reference and surrogate variables, along with standard errors, confidence intervals and p-values.

The summary() function, provides additional output beyond that of the print() function. In particular, summary() provides model fit information when regressing y_val on x_val, ys_val on xs_val and ys_non on xs_non, as in

```
summary(Ex1)

##
## meerva.fit(x_val = x_val, y_val = y_val, xs_val = xs_val, ys_val = ys_val,
##     xs_non = xs_non, ys_non = ys_non)
##
##      familyr    familys    compare comparec vmethod vmethodc    vmethodr
## [1,] "binomial" "binomial" "1"     "ful"    NA      "ijk (alt)" "2"
##      vmethods n_val n_ful  dim_beta dim_gamma
## [1,] "2"      "400" "4000" "5"      "5"
##
##  Confidence intervals are for alpha = 0.05
##
##  Estimates for Beta using beta_aug (references augmented with surrogates)
##                estimate         se        lcl        ucl          z
## (Intercept) -0.6268576 0.10099079 -0.8247959 -0.4289193 -6.2070768 5.40e-10
## x1           0.7436406 0.20911103  0.3337905  1.1534907  3.5561999 3.76e-04
## x2           0.1629424 0.23472323 -0.2971067  0.6229914  0.6941893 4.88e-01
## x3           1.2740382 0.12226244  1.0344083  1.5136682 10.4205201 2.00e-25
## x4           0.6054943 0.08634431  0.4362626  0.7747261  7.0125558 2.34e-12
##
##  Estimates for Beta using beta_val (references alone
##                 estimate        se        lcl        ucl          z
## (Intercept) -0.512907988 0.1573809 -0.8213688 -0.2044472 -3.2590239 1.12e-03
## x1           0.711948072 0.2847698  0.1538095  1.2700867  2.5000825 1.24e-02
## x2           0.007773179 0.3592342 -0.6963128  0.7118592  0.0216382 9.83e-01
## x3           1.445910903 0.1573970  1.1374185  1.7544033  9.1863958 4.06e-20
## x4           0.732347710 0.1205835  0.4960084  0.9686871  6.0733652 1.25e-09
##
##  Effective multiplicative increase in sample size by using augmented estimates
## (Intercept)     x_valx1     x_valx2     x_valx3     x_valx4
##    2.428512    1.854531    2.342303    1.657321    1.950331
##
##  Estimates for Gamma using gamma_non (surrogates alone)
##  not for direct comparisons
##               estimate         se         lcl        ucl          z
## (Intercept) -0.5725799 0.04480599 -0.66039798 -0.4847617 -12.779093 2.15e-37
## x1s          0.6890032 0.07680289  0.53847226  0.8395341   8.971058 2.94e-19
## x2           0.1270045 0.09739016 -0.06387674  0.3178857   1.304079 1.92e-01
## x3s          0.8090865 0.04024644  0.73020489  0.8879680  20.103305 6.90e-90
## x4           0.4086580 0.03559780  0.33888755  0.4784284  11.479866 1.67e-30
##
##  Correlations between beta_val and gamma_val
##             [,1]         [,2]        [,3]       [,4]        [,5]
## [1,]  0.80367933 -0.376471875 -0.31345870 -0.1774925  0.04358301
## [2,] -0.39055533  0.706565488  0.05592196  0.1147770  0.02155382
## [3,] -0.31205859  0.078158138  0.78865862  0.1008101 -0.05238810
## [4,] -0.17820540  0.111477966  0.11389238  0.6517229  0.11937274
## [5,]  0.06078147 -0.008493862 -0.05009483  0.1205559  0.73004945
```

The augmented estimates displayed using print() are actually derived in part on these other regression model fits. The summary() function also describes the correlation between parameter estimates from the models based upon reference and surrogate data in the validation subsample. These individual model fits and correlation may be useful for data inspection and to see just how similar or different the estimates may be from the models based upon reference or surrogate data.

Example binomial data

In the section above "**An example dataset**" we used the code

```
# Simulate logistic regression data with measurement error
simd = meerva.sim.brn(n=4000, m=400,
     beta = c(-0.5, 0.5, 0.2, 1, 0.5) ,
     alpha1 = c(0.95, 0.90, 0.90, 0.95) ,
     alpha2 = c(0.98,0.94,0.95,0.95) ,
     bx3s1 = c(0.05, 0, 0, NA, NA) ,
     bx3s2 = c(NA,NA,NA) )
```

to generate a simulated data and store the data in an object, here named simd. The full dataset for this example has size 4000 as specified by the input n=4000. The validation subsample, with reference and surrogate variables, has size 400 as specified by m=400. We will ignore the other input parameters for the moment but note the meerva.sim.brn() generates data with 4 predictors, both for the surrogate data subject to measurement error and for the reference data not subject to error. We next extract the data for analysis from this object with the code (same as in the section "**An example dataset**")

```
# Read the simulated data to input data format
x_val  = simd$x_val   # reference predictors from the validation subset
y_val  = simd$y_val   # reference outcome from the validation subset
xs_val = simd$xs_val  # surrogate predictors from the validation subset
ys_val = simd$ys_val  # surrogate outcome from the validation subset
xs_non = simd$xs_non  # surrogate predictors from the non-validation
                      # sample units
ys_non = simd$ys_non  # surrogate outcome from the non-validation
                      # sample units
```

With the data elements described after each line of code. Because we specified a validation subsample of size 400, x_val is a matrix 400 rows tall and 4 columns wide. Inspecting the first 10 rows (same as in the section "**An example dataset**")

```
# Print the first 10 elements of the y validaiton vector
x_val[1:10,]

##       x1 x2          x3          x4
##  [1,]  0  0  1.68763730 -0.67781913
##  [2,]  0  1 -0.39802125  1.31151150
##  [3,]  0  0  1.71577182 -0.32623965
##  [4,]  0  0  1.25245033  0.02819455
##  [5,]  1  0 -0.03601293 -0.04438002
##  [6,]  1  0 -2.28411085 -0.79361748
##  [7,]  0  0  1.48743603  2.38535937
##  [8,]  1  0  0.29237729  1.03242762
##  [9,]  1  0  2.22198131  0.55704673
## [10,]  0  0  2.14268805  2.11336517
```

we see x1 and x2 appear to be binomial predictors, and x3 and x4 to the quantitative. This is indeed the case and is explained in detail in the *meerva* "Reference manual" (at https://cran.r-project.org/web/packages/meerva/index.html) for the meerva.sim.brn(). The outcome is binomial and so y_val is a vector with length 400. Inspecting the first 10 elements we see (same as in the section "An example dataset")

```
# Print the first 10 elements of the y validation vector
y_val[1:10]
```

```
##  [1] 1 1 1 1 1 0 1 1 1 1
```

xs_val and ys_val are similar to x_val and y_val but include the surrogate mearues instead of the reference variables (thus the s in xs_val and ys_val). The matrix xs_non and vector ys_non contain the surrogate variable data for the sample units not included in the validation subsample. For this example there are 4000 – 400 = 3600 non-validation units in this set.

In this example, after specifying sample and validation subsample size, we specified alpha1 which describes the misclassification error probabilities for the surrogate binomial outcome variable ys in a differential manner, that is with different misclassification probabilities dependent on the predictors. Specifically,

P(ys = y | y = 1, x1=1) = alpha1[1]
P(ys = y | y = 0, x1=1) = alpha1[2]
P(ys = y | y = 1, x1=0) = alpha1[3]
P(ys = y | y = 0, x1=0) = alpha1[4]

We then specified alpha2 which describes the differential misclassification probabilities for x1s, the surrogate for x1 predictor. That is the misclassification probabilities for the surrogate X1 are dependent on the true outcome Y. Specifically,

P(x1s = x1 | x1 = 1, y=1) = alpha2[1]
P(x1s = x1 | x1 = 0, y=1) = alpha2[2]
P(x1s = x1 | x1 = 1, y=0) = alpha2[3]
P(x1s = x1 | x1 = 0, y=0) = alpha2[4]

We then specified bx3s1 and bx3s2 which describe the probabilistic relation between x3 and the surrogates x3s and potentially x3s2. Here we specified bx3s1[1]=0.5 with which a normal random error with SD 0.5 is added to x3 to obtain x3s. All other elements of bx3x1 and bx3s2 are either 0 or NA and thus no other error structure is imposed by measurement error. Terms can be given so that the mean of x3s has mean non linear in X3 and so that x3s2 is generated, a second surrogate variable for x3. As described in the *meerva* Reference manual, more possibilities are possible when generating surrogate markers for x3.

More example data

Similar to the meerva.sim.brn() the meerva.sim.cox() and meerva.sim.nrm() functions generate simulated data with measurement error for analysis with the Cox and linear regression model framework. For Cox model framework, measurement error may occur in the time to event variable and the event indicator variable too may be misclassified. As with all model families analyzed by *meerva*, any or all predictors may be measured with error. (Here we consider misclassification as a type of measurement error.) For the Cox regression framework we include the time to event or censoring variable in the vectors y_val, ys_val, ys_non and the event vs. censor indicator variable in e_val, es_val and es_non. If e_val, es_val and es_non are specified then meerva.fit() will default to analysis using the Cox regression framework. The user can override this by specifying the familyr or familys to be either binomial or gaussian. Example code generating simulated time to event data with measurement error and anlzying these data is

```
#=======================================================
# Simulate Cox regression data with measurement error
simd = meerva.sim.cox(n=4000, m=400,
     beta   = c(-0.5, 0.5, 0.2, 1, 0.5) ,
     alpha1 = c(0.95,0.90,0.90,0.95)  ,
     alpha2 = c(0.98,0.94,0.94,0.98) ,
     bx3s1  = c(0.05,0,0,NA,NA) ,
     bx3s2  = c(1.1, NA, NA) ,
     sd=0.1)

# Read the simulated data to input data format
x_val  = simd$x_val
y_val  = simd$y_val
xs_val = simd$xs_val
ys_val = simd$ys_val
xs_non = simd$xs_non
ys_non = simd$ys_non
e_val  = simd$e_val
es_val = simd$es_val
es_non = simd$es_non

# Analyze the data and print
cox.me = meerva.fit(x_val, y_val, xs_val, ys_val, xs_non, ys_non,
                    e_val, es_val, es_non)
cox.me

##
## meerva.fit(x_val = x_val, y_val = y_val, xs_val = xs_val, ys_val = ys_val,
##     xs_non = xs_non, ys_non = ys_non, e_val = e_val, es_val = es_val,
##     es_non = es_non)
##
##      familyr familys compare comparec vmethod vmethodc    vmethodr vmethods
## [1,] "Cox"   "Cox"   "1"     "ful"    NA      "ijk (alt)" "1"      "1"
##      n_val n_ful  dim_beta dim_gamma
## [1,] "400" "4000" "4"      "4"
##
##  Confidence intervals are for alpha = 0.05
##
##  Estimates for Beta using beta_aug
##     estimate         se       lcl       ucl         z
## x1 0.4576975 0.07831787 0.3041973 0.6111977  5.844101  5.09e-09
## x2 0.2596587 0.07494578 0.1127677 0.4065497  3.464621  5.31e-04
## x3 0.9784529 0.04291347 0.8943440 1.0625617 22.800602 4.52e-115
## x4 0.4597069 0.03433951 0.3924027 0.5270111 13.387113  7.19e-41
```

Example code generating normal data with measurement error and analyzing these data is

```
# Simulate linear regression data with measurement error
simd = meerva.sim.nrm(n=4000, m=400,
     beta=c(-0.5,0.5,0.2,1,0.5),
     alpha1=c(-0.05,0.1,0.05,0.1),
     alpha2=c(0.95,0.91,0.9,0.9),
     bx3s1= c(0.05, 0, 0, NA, NA),
     bx3s2=c(1.1,0.9,0.05),
     sd=5)

# Read the simulated data to input data format
x_val  = simd$x_val
y_val  = simd$y_val
xs_val = simd$xs_val
ys_val = simd$ys_val
xs_non = simd$xs_non
ys_non = simd$ys_non

# Analyze the data and print
nrm.me = meerva.fit(x_val, y_val, xs_val, ys_val, xs_non, ys_non)
nrm.me

##
## meerva.fit(x_val = x_val, y_val = y_val, xs_val = xs_val, ys_val = ys_val,
##     xs_non = xs_non, ys_non = ys_non)
##
##      familyr    familys    compare comparec vmethod vmethodc    vmethodr
## [1,] "gaussian" "gaussian" "1"     "ful"    NA      "ijk (alt)" "1"
##      vmethods n_val n_ful  dim_beta dim_gamma
## [1,] "1"      "400" "4000" "5"      "6"
##
##  Confidence intervals are for alpha = 0.05
##
##  Estimates for Beta using beta_aug
##                estimate         se        lcl        ucl         z
## (Intercept) -0.5160102 0.13585249 -0.7822762 -0.2497442 -3.798312 1.46e-04
## x1           0.6415720 0.41625605 -0.1742748  1.4574189  1.541292 1.23e-01
## x2           0.8126396 0.22195328  0.3776191  1.2476600  3.661309 2.51e-04
## x3           1.0497514 0.07691983  0.8989913  1.2005115 13.647344 2.09e-42
## x4           0.5894433 0.08055363  0.4315611  0.7473255  7.317402 2.53e-13
```

Use of meerva.sim.cox() and meerva.sim.nrm() is further described in the *meerva* Reference manual.

Simulation studies

Simulation studies can be helpful in understanding how different statistical methods may perform for different scenario. A number of simulations are provided in the paper describing augmented estimates in (arXiv:2106.14063, https://arxiv.org/abs/2106.14063). We provide with the *meerva* package the program used to perform the simulation studies in that paper. With this program the user can for themselves investigate the properties of the augmented estimators for various scenario. The programs simulate data with measurement error in both predictors and outcomes for the logistic, Cox and linear regression models. Very simplistically the user may first try this program, meerva.sim.block(), without specifying any of the input variables and a default simulation setting will be run. Because the default is provided more as a starting point than for inference, by default only 100 data sets are simulated and analyzed. To draw stronger conclusions one should probably simulate 1000 data sets or more by specifying the input variable nsim (for number of simulations) as in meerva.sim.block(nsim=1000). Simulating 1000 datasets, as determined by the parameter nsim, suppressing a log of time as the program runs through the simulations, as determined by teh paramter simtime, and viewing the summary information we have, for example,

```
simex = meerva.sim.block(nsims=1000, simtime=0)
summary(simex)

## 
## ======================= Simulation parameters ==================================
## 
## list name =  simex
## 
## R glm family = gaussian
## 
## VCOV method vmethod =  1 , dfbeta
## 
## Comparison group =  1 , ful
## 
## Number of simulations = 1000
## 
## seed = 633677224
## 
## Full and val sample sizes of 4000 and 400
## 
##      [,1] [,2] [,3] [,4] [,5]
## beta -0.5  0.5  0.2    1  0.5
## 
##         [,1] [,2] [,3] [,4]
## alpha1 -0.05  0.1 0.05  0.1
## 
##        [,1] [,2] [,3] [,4]
## alpha2 0.98 0.98 0.95 0.95
## 
##       [,1] [,2] [,3] [,4] [,5]
## bx3s1 0.05    0    0   NA   NA
## 
##       [,1] [,2] [,3]
## bx3s2 0.95   NA   NA
## 
##      [,1] [,2]
## bx12 0.25 0.15
## 
##  SD = 1
## 
```

```
##  mncor = 0
##
## ================================================================================
## ====================== Estimate averages =======================================
##
##          (Intercept)   x_valx1   x_valx2   x_valx3   x_valx4
## beta      -0.5000000 0.5000000 0.2000000 1.0000000 0.5000000
## beta_val  -0.5022569 0.5036400 0.2009553 0.9993991 0.4983573
## beta_aug  -0.5008796 0.5015795 0.2013922 0.9998795 0.4995057
##
##           (Intercept) xs_fulx1s  xs_fulx2 xs_fulx3s  xs_fulx4
## gamma_ful  -0.4372526 0.3251148 0.2007229  1.049886 0.4997596
## gamma_val  -0.4385054 0.3271122 0.2004235  1.049366 0.4986172
##
## ====================== Bias ====================================================
##
##                 (Intercept)     x_valx1      x_valx2       x_valx3
## beta_val_bias -0.0022569178 0.003640022 0.0009553472 -0.0006009482
## beta_aug_bias -0.0008795792 0.001579478 0.0013922206 -0.0001205343
##                     x_valx4
## beta_val_bias -0.0016427232
## beta_aug_bias -0.0004942831
##
##                (Intercept)  xs_fulx1s     xs_fulx2  xs_fulx3s      xs_fulx4
## gamma_ful_bias  0.06274740 -0.1748852 0.0007229432 0.04988557 -0.0002403569
## gamma_val_bias  0.06149457 -0.1728878 0.0004234665 0.04936587 -0.0013827686
##
## ====================== Standard Deviations =====================================
##
##             (Intercept)    x_valx1    x_valx2    x_valx3    x_valx4
## beta_val_sd  0.06107792 0.11972736 0.14012782 0.05080355 0.05080432
## beta_aug_sd  0.02305151 0.06164541 0.05003283 0.01696226 0.01706642
##
##              (Intercept)  xs_fulx1s   xs_fulx2  xs_fulx3s   xs_fulx4
## gamma_ful_sd  0.01920507 0.03517745 0.04587256 0.01696276 0.01617495
## gamma_val_sd  0.06246692 0.12213634 0.14114124 0.05415981 0.05181284
##
## ====================== Average Standard Errors  ================================
##
##             (Intercept)    x_valx1    x_valx2    x_valx3    x_valx4
## beta_val_sd  0.06174022 0.11696770 0.14193863 0.05076439 0.05048094
## beta_aug_sd  0.02331342 0.06150785 0.04793484 0.01732204 0.01712690
##
##               (Intercept)  xs_fulx1s   xs_fulx2  xs_fulx3s   xs_fulx4
## gamma_ful_sdj  0.01977944 0.03645767 0.04481982 0.01681767 0.01598942
## gamma_val_sd   0.06289383 0.11481727 0.14268464 0.05355446 0.05086724
## gamma_ful_sd   0.01985192 0.03611491 0.04481597 0.01681043 0.01598961
##
## ================================================================================
## ====================== Square Root MSEs ========================================
##
##                 (Intercept)    x_valx1    x_valx2    x_valx3    x_valx4
## beta_val_rmse  0.06108908 0.11972283 0.14006100 0.05078169 0.05080548
## beta_aug_rmse  0.02305676 0.06163482 0.05002718 0.01695420 0.01706505
##
##                  (Intercept) xs_fulx1s   xs_fulx2  xs_fulx3s   xs_fulx4
## gamma_ful_rmse  0.06561785 0.1783845 0.04585532 0.05268793 0.01616864
## gamma_val_rmse  0.08763445 0.2116426 0.14107129 0.07326214 0.05180539
##
## ===============================================================================
## ======================= Compare means and squared errors =======================
##
##  Compare MSE beta_aug minus MSE bata_val (Paired t-test)
##                 n         mean beta0          sd          lcl          ucl
## (Intercept) 1000 -0.003200261     0 0.005364371 -0.003533145 -0.002867377
## x_valx1     1000 -0.010534704     0 0.019680734 -0.011755986 -0.009313422
## x_valx2     1000 -0.017114365     0 0.029658093 -0.018954788 -0.015273941
## x_valx3     1000 -0.002291335     0 0.003656269 -0.002518224 -0.002064447
## x_valx4     1000 -0.002289980     0 0.003479157 -0.002505878 -0.002074082
##                    t            p
## (Intercept) -18.86542 3.852487e-68
## x_valx1     -16.92704 1.056313e-56
## x_valx2     -18.24810 1.977107e-64
## x_valx3     -19.81758 5.648511e-74
## x_valx4     -20.81411 3.279030e-80
##
##  Bias in beta_val
##                 n          mean beta0         sd          lcl         ucl
## (Intercept) 1000 -0.0022569178     0 0.06107792 -0.006047089 0.001533253
## x_valx1     1000  0.0036400221     0 0.11972736 -0.003789621 0.011069666
## x_valx2     1000  0.0009553472     0 0.14012782 -0.007740241 0.009650935
## x_valx3     1000 -0.0006009482     0 0.05080355 -0.003753546 0.002551650
## x_valx4     1000 -0.0016427232     0 0.05080432 -0.004795369 0.001509923
##                     t         p
## (Intercept) -1.1685075 0.2428809
## x_valx1      0.9614144 0.3365765
## x_valx2      0.2155941 0.8293482
## x_valx3     -0.3740615 0.7084379
## x_valx4     -1.0225010 0.3067914
##
##  Bias in beta_aug
##                 n          mean beta0         sd          lcl          ucl
## (Intercept) 1000 -0.0008795792     0 0.02305151 -0.002310033 0.0005508748
## x_valx1     1000  0.0015794779     0 0.06164541 -0.002245909 0.0054048644
## x_valx2     1000  0.0013922206     0 0.05003283 -0.001712551 0.0044969921
## x_valx3     1000 -0.0001205343     0 0.01696226 -0.001173122 0.0009320532
## x_valx4     1000 -0.0004942831     0 0.01706642 -0.001553335 0.0005647685
##                     t         p
## (Intercept) -1.2066342 0.2278587
## x_valx1      0.8102384 0.4179962
## x_valx2      0.8799398 0.3791035
## x_valx3     -0.2247123 0.8222490
## x_valx4     -0.9158688 0.3599567
##
##  Bias in gamma_ful
##                n          mean beta0         sd          lcl           ucl
## (Intercept) 1000  0.0627473966     0 0.01920507  0.061555632  0.0639391612
## xs_fulx1s   1000 -0.1748851522     0 0.03517745 -0.177068077 -0.1727022268
## xs_fulx2    1000  0.0007229432     0 0.04587256 -0.002123664  0.0035695506
## xs_fulx3s   1000  0.0498855728     0 0.01696276  0.048832954  0.0509381913
## xs_fulx4    1000 -0.0002403569     0 0.01617495 -0.001244088  0.0007633743
##                        t         p
## (Intercept)  103.3189077 0.0000000
## xs_fulx1s   -157.2130670 0.0000000
## xs_fulx2       0.4983691 0.6183336
## xs_fulx3s     92.9990604 0.0000000
## xs_fulx4      -0.4699090 0.6385226
##
## ================================================================================
## ======== beta and gamma 95% Confidence Interval coverage probabilities =========
##
##  Coverage for beta_aug 95% CI
##               n_ coverage coverage_lcl coverage_ucl
## (Intercept) 1000    0.956        0.943        0.969
## x_valx1     1000    0.950        0.936        0.964
## x_valx2     1000    0.932        0.916        0.948
## x_valx3     1000    0.963        0.951        0.975
## x_valx4     1000    0.953        0.940        0.966
##
##  Coverage for beta_val 95% CI
##               n_ coverage coverage_lcl coverage_ucl
## (Intercept) 1000    0.953        0.940        0.966
## x_valx1     1000    0.942        0.928        0.956
## x_valx2     1000    0.947        0.933        0.961
## x_valx3     1000    0.941        0.926        0.956
## x_valx4     1000    0.946        0.932        0.960
##
##  Coverage for gamma_ful 95% CI
##               n_ coverage coverage_lcl coverage_ucl
## (Intercept) 1000    0.116        0.096        0.136
## xs_fulx1s   1000    0.003        0.000        0.006
## xs_fulx2    1000    0.940        0.925        0.955
## xs_fulx3s   1000    0.151        0.129        0.173
## xs_fulx4    1000    0.947        0.933        0.961
##
## ================================================================================
```

The first few elements of the simulation parameter summary include

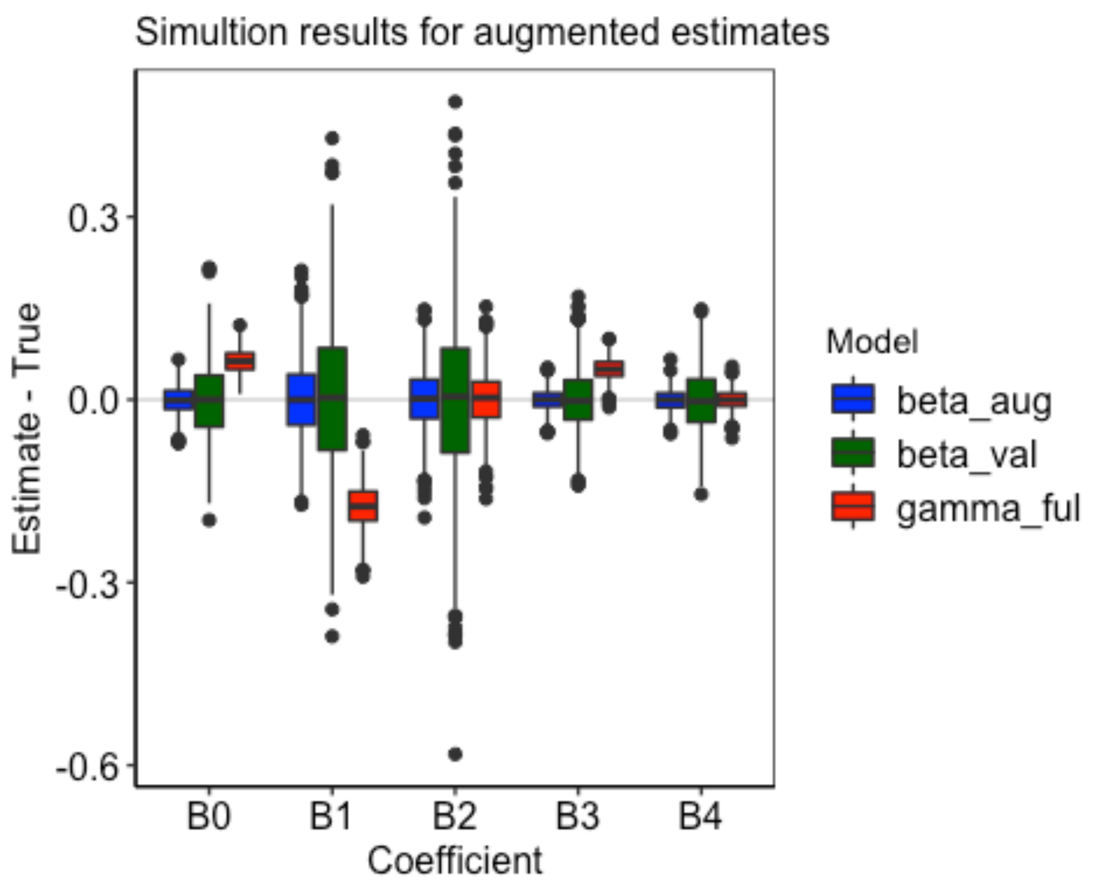
- list name – name of the list object in which results were stored. I found this helpful when reading summaries from multiple simulations.
- R glm family – the family of model for simulating datasets and analyzing data, here gaussian.
- VCOV method – the method used for estimating the variance covariance matrices needed for calculation of the augmented estimates, here by default using the dfbeta's (infinitesimal jackknife) from existing R functions or from Tong et al. doi:10.1093/jamia/ocz180 for logistic regression. The jackknife will provide more accurate variance estimates, e.g. meerva.sim.block(n=1000, vmethod=0), but will take longer to compute.
- Comparison group – In the standard formulation the augmented estimates involve the difference (gamma_ful – gamma_val) but the formulation involving (gamma_non – gamma_val) can also be used. Thus, either gamma_ful or gamma_val can serve as a comparison group. By default the programs uses gamma_ful as the comparison group but can use gamma_non as the comparison group (compare = 0). This will generally not have a large impact on the analysis and is discussed in our methods paper.
- seed – By default the program first generates a seed and stores this seed. One can give this seed value as input when running the simulation program again to replicate findings, e.g. meerva.sim.block(n=1000, seed = 648284311).

The other simulation parameters are as in the meerva.sim.brn(), meerva.sim.cox() and meerva.sim.nrm() functions for which we refer to the *meerva* Reference manual.

Summary statistics include various descriptions like averages, biases, standard deviations and mean square errors as well as average standard errors calculated form the individual simulations. We provide these summaries for the augmented estimates (based upon all reference and surrogate data) as well as estimates based upon either the reference or surrogate data alone using either the subsample or full sample. Because the models based upon surrogate data (alone) indeed estimate quantities different than estimates based upon the reference variables (alone) we use beta to denote estimates for reference variable models and gamma to denote estimates based upon surrogate variables. Additionally, we compare MSE's between augmented estimates and estimates based upon reference variables in the validation subsample, as well as test for bias of the various estimates. Finally, we calculate coverage probabilities for the different estimates. Though not shown in this example, for coverage probabilities of the augmented estimates one can compare using the jackknife with using the infinitesimal jackknife (using vmethod=3), typically finding better coverage probabilities with the jackknife. Fitting the data multiple times for the different methods and also estimating variances using the jackknife, this may require a bit of computing time.

Simulation results may be inspected visually by plotting as in

```
plot(simex)
```



Whereas the summary() and plot() functions provide numerous insights to the simulations the user may want to further inspect the estimator properties. Since meerva.sim.block() actually stores estimates from the individual simulated datasets, the user can extract these elements to inspect them. One can see the elements of the output list using the names() function, e.g.

```
# see the elements of the simulaiton output list
names(simex)

##  [1] "simfam"          "mncor"           "vmethod"         "jksize"
##  [5] "compare"         "nsims"           "seed"            "nm"
##  [9] "beta"            "alpha1"          "alpha2"          "bx3s1"
## [13] "bx3s2"           "bx12"            "sd"              "mncor"
## [17] "sigma"           "beta_augs"       "beta_vals"       "beta_aug_vars"
## [21] "beta_val_vars"   "gamma_bigs"      "gamma_vals"      "gamma_big_varjs"
## [25] "gamma_val_vars"  "gamma_big_vars"
```

This includes the regression parameter estimates from each simulated dataset. Because when deriving the augmented estimates one can compare either gamma_ful (regression parameter estimate derived from the full sample) or gamma_non (derived from the non-validation subsample) with gamma_val we denote the respective comparator by gamma_big. The s in beta_augs and similar terms simply signifies that the individual dataset estimates are stored, not just a summary. I may have used this convention as well out of programming convenience. Importantly the s here does not relate to surrogate.