EUROPEAN COMMISSION
Community Research

SEVENTH FRAMEWORK PROGRAMME

AMELI
Advanced Methodology for European Laeken Indicators

Deliverable 2.2

Small Area Estimation of Indicators on Poverty and Social Exclusion

Supplement: Manual of R codes

Version: 2011

Ari Veijanen and Risto Lehtonen

The project FP7-SSH-2007-217322 AMELI is supported by European Commission funding from the Seventh Framework Programme for Research.

http://ameli.surveystatistics.net/

Contributors to Deliverable 2.2, Supplement:

Chapter 1: Ari Veijanen and Risto Lehtonen, University of Helsinki.

Chapter 2: Ari Veijanen, University of Helsinki.

Chapter 3: Ari Veijanen, University of Helsinki.

Chapter 4: Ari Veijanen, University of Helsinki.

Main responsibility

Ari Veijanen, University of Helsinki

Evaluators

Internal evaluator: Matthias Templ, Vienna University of Technology.

Contents

1 Introduction

Domain estimators are implemented for at-risk-of-poverty rate, poverty gap, quintile share and Gini coefficient. Methodology and technical details are presented in Deliverable 2.2 (Lehtonen et al., 2011). All equation references refer to that document. For poverty rate, we have implemented HT-based equation (24), GREG estimator (26), which is assisted by a model provided by the user, and EBP(Y) estimator (16). Other indicators, such as the share of persons with low educational attainment, can be estimated with the estimators of poverty rate. Poverty gap, quintile share and Gini coefficient require special attention, as they cannot be estimated by methods designed for estimation of totals or shares. Their default estimators defined by equations (27), (30) and (33) are available.

Predictors of poverty gap (31), quintile share (34) and Gini coefficient (28) are implemented. However, it is preferable to apply the expansion technique (18) with log-transformation log(x+c+1) taking negative incomes into account. It is available for each predictor. Zero incomes are not processed separately in contrast with our simulation experiments. If the user has no unit-level population information about auxiliary variables, it is possible to use a frequency-calibrated predictor (19).

Composite estimators (Eqs. 20 and 21) are constructed from the default (direct) estimator and corresponding ordinary, expanded or frequency-calibrated predictor. The variance of the direct estimator is calculated by bootstrap. A summary of equation references is in table below.

Indicator	**Estimator type**	**Equation in Deliverable 2.2**
Poverty rate	Direct (default)	(24)
	GREG	(26)
	EBP(Y)	(16)
Poverty gap	Direct (default)	(27)
	Predictor	(31)
	Expanded predictor	(18)
Quintile share	Direct (default)	(30)
	Predictor	(34)
	Expanded predictor	(18)
Gini coefficient	Direct (default)	(33)
	Predictor	(28)
	Expanded predictor	(18)
Generic for poverty gap, quintile share and Gini coefficient	Frequency calibrated predictor	(19)
Generic for poverty gap, quintile share and Gini coefficient	Composite	(20) or (21)

2 Implementation

Our collection of R functions contains separate functions for default estimators (such as `direct_gini`) and predictors (e.g. `predictor_quintile_share`) in files `gini.r`, `poverty_rate_estimator.r`, `poverty_gap_estimator.r` and `quintile_share.r`, but the user does not have to call these directly (see next section for interface). Direct estimators and ordinary predictors are implemented by a call of function `domain_estimators` (in `domain_estimators.r`). Expanded predictors (Eq. 18) are implemented by a call of function `expanded_domain_predictors` (in `domain_estimators.r`) with the required predictor function as one of the arguments. The expanded predictions are calculated in function `log_expanded_predictions` (`expanded_predictions.r`) A frequency-calibrated predictor is obtained by function `calibrated_predictors` (in `calibrated_predictions.r`), with predictor function as argument. A composite estimator is obtained by function `composite_estimators` (in `composite_estimators.r`), whose arguments include the direct estimator, the predictor, and the type of predictor (expanded, calibrated, or ordinary).

To support domain estimation, class Domain (`domain.r`) has methods finding all domains in a data set, and methods calculating domain indicators or domain sums, for example. The file `estimated_distribution_function.r` contains functions for calculating percentiles, among others.

Some special cases of data require somewhat arbitrary decisions. In the direct poverty gap estimator, all poor people of the sample are used if there are no poor in a domain. Similarly, the value of the poverty gap predictor is calculated from all predictions, if all predicted incomes in a domain exceed the poverty line. If a sample domain does not contain any observations, direct estimator is invalid, and the direct estimate is replaced by an estimate calculated with a predictor specified by the user.

Bootstrap samples are drawn by SRSWOR (R function `sample`) from a bootstrap population. The bootstrap population can be regarded as created by cloning each observation in the original sample with frequency equal to downwards rounded design weight. The bootstrap variance of a domain estimator is calculated as sample variance over bootstrap samples. The final composite weights are equal to the median over all domain-specific composite weights, irrespective of domain size.

3 Interface

The complexities of the implementation are hidden from an ordinary user. All the estimators of poverty indicators can be invoked through a single function `domain_estimate_data` (in interface.r). It creates a data set (R data frame) containing domain estimates for each domain.

The user has to fit a model to the sample and provide a function transforming the predictions to the original scale. Our R code assumes that the predicted values of a model can be obtained by calling generic R function `predict` with the model as the first argument. This is possible with models fitted by `lm`, `glm`, `lme` and `nlme` (library `nlme`), but not necessarily with models of package `lme4`, for example.

Our R functions do not perform classification of variables. As an example, age classes must be created prior to domain estimation.

Poverty rate estimators are based on poverty indicators. They are first created by function `create_poverty_indicator` (in `poverty_rate_estimator.r`) which has the following arguments: sample, name of y variable, name of weights and the data set determining the poverty line (typically the sample). Then a logistic fixed-effects model is fitted by `glm` with option `family=binomial` or a logistic mixed model is fitted by `nlme`.

In the case of poverty gap, quintile share and Gini coefficient, a mixed model is usually fitted to log-transformed equivalized incomes by `lme`, for example. For log-transformation, the package includes functions `logp` and `expm`. `logp(c)` returns a

function $f(x) = \log(x + c)$, and `expm(c)` returns its inverse function $f^{-1}(x) = \exp(x) - c$. If the model has been fitted to observations transformed by *`logp`*`(c)`, then the corresponding back-transformation function is `expm(c)`.

The estimators are specified by a list of names (argument estimator_descriptions of `domain_estimate_data`). The name of an estimator consists of the name of the poverty indicator and the type of the estimator. Names of the poverty indicators are ”poverty rate”, ”gini”, ”poverty gap” and ”quintile share”. Default estimators are identified by ”direct”, and predictors are identified by ”predictor”. Special cases of predictors are ”expanded” for predictors incorporating expanded predictions (18) and ”calibrated” for predictors based on the frequency-calibration (n-calibration) technique (19). In the case of poverty rate, it is also possible to use “greg” for GREG or MLGREG estimation and “ebp” for EBP estimation. Examples of estimator names are ”direct poverty rate”, ”greg poverty rate”, ”ebp poverty rate”, ”expanded gini predictor”, ”poverty gap predictor” and ”calibrated quintile share predictor”. The name of a composite estimator consists of the name of the unbiased component and the name of the predictor, separated by a ”+”. An example is ”direct quintile share + expanded quintile share predictor”.

The domains are defined by a cross-tabulation of variables. A list of variable names hast to be provided (argument domain_variables). The list can contain a single name, if the values of a variable are interpreted as domains. The domain variables must be present both in sample and in population.

If frequency-calibrated predictors are used, the population data set is still unit-level but one observation in each domain is chosen to contain the domain sums of those auxiliary variables that are used in calibration; the other observations of such auxiliary variables are zeroes.

The arguments of the function `domain_estimate_data` are as follows.

Argument	**Description**
estimator_descriptions	List of names of estimators
sample	Sample data (data frame)
population	Population data (data frame)
y	Name of the y variable
model	Model object. Function calls predict(model, newdata=population) and predict(model, newdata=sample) must work
back_transformation	Function back-transforming the predictions
x_list	List of names of quantitative x-variables used in n-calibration (or empty list)
xq_list	List of names of qualitative x-variables used in n-calibration (or empty list)
unknown	List of names of x-variables whose domain totals are estimated by GREG in n-calibration (or empty list)
domain_variables	List of names of variables determining the domains (crosstabulation)
weight	Name of the design weight variable in sample
reference_set	Data set determining the poverty line, typically sample
percentages	Vector of percentage points used in the expansion of predictions (Eq. 18); default is 1:99
missing_handler	Name of the type of predictor used to replace invalid direct estimates; examples: “expanded predictor”, “calibrated predictor”. Such a predictor is created for each poverty indicator.

Next excerpt of code is an example of poverty rate estimation by EBP based on a logistic mixed model (variable y is the equivalized income, w is the weight variable, x is an auxiliary variable and domain is the domain variable; pop is the population data set; `invlogit` is the function $\exp(x)/(1+\exp(x))$ provided in the package). Note that the poverty indicator has to be created and added to the sample, and its name “ind” is used as argument y in the call of function `domain_estimate_data`.

```
sample = data.frame(y,w,x,domain)
ind = create_poverty_indicator(sample,"y","w",sample)
data[["ind"]] = ind
model <- nlme(ind ~ invlogit(fix+ran), fixed=fix~x,
random=ran~1|domain, start=c(0,0))

back_transformation=identity

estimator_data <- domain_estimate_data(list("ebp poverty rate"),
sample=sample, population=pop, y="ind", model, back_transformation,
domain_variables=list("domain"), weight="w", reference_set=sample)
```

In the following example the resulting data set contains domain estimates by direct quintile share estimator, expanded quintile share predictor and their composite. The example presumes variables y, x and domain and data sets sample and pop as in previous example.

```
logy <- logp(1)(y)
model <- lme(logy ~ x, random=~1|domain)
back_transformation=expm(1)

estimator_data <- domain_estimate_data(list("direct quintile share”,
"expanded quintile share predictor", ”direct quintile share +
expanded quintile share predictor”), sample=sample, population=pop,
y="y", model, back_transformation, domain_variables=list("domain"),
weight="w", reference_set=sample, missing_handler = “expanded
predictor”)
```

4 R codes

Instructions `readme.txt`

```
# To use the R files, define variable folder containing
# the files, and execute the following lines in R:

source(paste(folder,"generic_functions.r",sep=""))
source(paste(folder,"domain.r",sep=""))
source(paste(folder,"test_data.r",sep=""))
source(paste(folder,"estimated_distribution_function.r",sep=""))
source(paste(folder,"expanded_predictions.r",sep=""))
source(paste(folder,"composite_estimators.r",sep=""))
source(paste(folder,"domain_estimators.r",sep=""))
source(paste(folder,"gini.r",sep=""))
source(paste(folder,"poverty_rate_estimator.r",sep=""))
source(paste(folder,"poverty_gap_estimator.r",sep=""))
source(paste(folder,"quintile_share.r",sep=""))
source(paste(folder,"calibrated_predictions.r",sep=""))
source(paste(folder,"interface.r",sep=""))
```

Code generic_functions.r

```
#Generic functions for classes
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi

#Domain
setGeneric("region",function(self) standardGeneric("region")) #Region of a domain
setGeneric("region_domain",function(self) standardGeneric("region_domain")) #Domain
containing all units in same region as this domain
setGeneric("domain_indicator",function(self,dataset)
standardGeneric("domain_indicator")) #Indicator with true when observation in the
dataset belongs to this domain
setGeneric("set",function(self,dataset) standardGeneric("set")) #Subset (data.frame)
of observations of dataset in this domain
setGeneric("from_list",function(self,list_of_values) standardGeneric("from_list"))
#Creates a similar new domain using a prototype and a list of values
setGeneric("values_at",function(self,dataset,row) standardGeneric("values_at")) #List
of values of defining variables in a row of a data set
setGeneric("is_domain_variable",function(self,name)
standardGeneric("is_domain_variable")) #Returns true if variable <name> is used in
defining the domain
setGeneric("domain_vector",function(self,dataset,vec)
standardGeneric("domain_vector")) #Subvector in a domain of a dataset
setGeneric("domain_sum",function(self,dataset,vec) standardGeneric("domain_sum")) #Sum
of vector's values over a domain

#EstimatedDistributionFunction
setGeneric("value", function(self,index) standardGeneric("value"))
setGeneric("percentile", function(self,pct) standardGeneric("percentile"))
setGeneric("weighted_median", function(self) standardGeneric("weighted_median"))
setGeneric("poverty_threshold",function(self) standardGeneric("poverty_threshold"))
#At-risk-of-poverty threshold
```

Code `domain.r`

```
#Domain is defined using a list of variable names and corresponding list of values.
#Definition is independent of a dataset.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi
#============================================================================
# Subject: Class Domain defining domains
#----------------------------------------------------------------------------------------
----------------------------------------
# SLOTS
# region_id     Name of the region variable, such as "nuts3"
# variables      Names of variables defining the domain
# values           Values of each variable, in same order as variables
#----------------------------------------------------------------------------------------
----------------------------------------
setClass("Domain",representation(
    region_id="character",
    variables="list",
    values="list"
  )
)

#============================================================================
# Subject: Function creating a domain in class Domain
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# region_id     Name of the region variable, such as "nuts3"
# variables      Names of variables defining the domain
# values           Values of each variable, in same order as variables
#----------------------------------------------------------------------------------------
----------------------------------------
setMethod("initialize","Domain",
function(.Object,region_id,variables,values)
{
.Object@region_id=region_id
.Object@variables=variables
.Object@values=values
callNextMethod(.Object,region_id=region_id,variables=variables,values=values)
}
)

#============================================================================
# Subject: Value of the region variable in a domain.
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# self     Domain in question
```

```
#-------------------------------------------------------------------------------------
----------------------------------------
setMethod("region","Domain",
function(self)
{
as.numeric(self@values[self@variables==self@region_id])
}
)


#=====================================================================
# Subject: Domain created from given list of variables and their values
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# region_id         Name of the region variable, such as "nuts3"
# variables          List of names of variables
# values               List of values of variables (in same order as variables)
#-------------------------------------------------------------------------------------
----------------------------------------
domain_from_list <- function(region_id,variables,list_of_values)
{
new("Domain", region_id, variables,values=list_of_values)
}

#=====================================================================
# Subject: A domain that always contains all observations
#-------------------------------------------------------------------------------------
----------------------------------------
any_domain <- function()
{
new("Domain","",list(),list())
}

#=====================================================================
# Subject: Distinct domains of a data set
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# region_id       Name of the region variable, such as "nuts3"
# variables        List of variable names whose values define the domains
# dataset           A data set
# NOTE: synchronize maintenance with data_of_distinct_domains
#-------------------------------------------------------------------------------------
----------------------------------------
all_domains <-  function(region_id,variables,dataset)
{
used_cols = dataset[,names(dataset)==variables]
if (length(variables)==1)
    {
    value_list = used_cols
```

```
    }
else
    {
    index=1
    value_list = list()
    for(row in 1:dim(dataset)[1])
        {
          value_list[[index]]=as.numeric(used_cols[row,])
          index=index+1
        }
    }
domains = list()
ind=1
for(list_of_values in unique(value_list))
    {
    domains[ind] = domain_from_list(region_id,variables,as.list(list_of_values))
    ind=ind+1
    }
domains
}

#============================================================================
# Subject: Function returning all domains with same domain variables and values
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# domain    Instance of Domain
# dataset    data frame
#----------------------------------------------------------------------------------------
----------------------------------------
all_similar_domains <-  function(domain,dataset)
    {
    all_domains(slot(domain,"region_id"),slot(domain,"variables"),dataset)
    }

#============================================================================
# Subject: Function determining when two domains are identical
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# d1    First domain
# d2    Second domain
#----------------------------------------------------------------------------------------
----------------------------------------
is_same_domain <- function(d1,d2)
{
 (identical(d1@values,d2@values)) && (identical(d1@variables,d2@variables))
}

#============================================================================
# Subject: Indicator of a domain in a data frame
```

```
#-------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# self              Domain in question
# dataset       A data set
#-------------------------------------------------------------------------------------
---------------------------------------
setMethod("domain_indicator","Domain",
function(self,dataset)
{
rows=dim(dataset)[1]
indicator = seq(rows)>0 #initialized to TRUE
for(var in self@variables)
    {
    expected = self@values[self@variables==var]
    col = names(dataset)==var
    indicator = indicator & (dataset[,col]==expected)
    }
indicator
}
)

#=====================================================================
# Subject: Dataset of units in a domain
#-------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# self              Domain in question
# dataset       A data set
#-------------------------------------------------------------------------------------
---------------------------------------
setMethod("set","Domain",
function(self,dataset)
{
dataset[domain_indicator(self,dataset),TRUE]
}
)

#=====================================================================
# Subject: Subset of a vector containing values of units in a domain
#-------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# self              Domain in question
# dataset       A data set
# vec               A vector
#-------------------------------------------------------------------------------------
---------------------------------------
setMethod("domain_vector","Domain",
function(self,dataset,vec)
{
```

```
vec[domain_indicator(self,dataset)]
}
)

#============================================================================
# Subject: Sum of vector's values over a domain
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# self              Domain in question
# dataset       A data set
# vec               A vector
#----------------------------------------------------------------------------
----------------------------------------
setMethod("domain_sum","Domain",
function(self,dataset,vec)
{
sum(domain_vector(self,dataset,vec))
}
)

#============================================================================
# Subject: Indicator vector (with elements TRUE or FALSE) for domain in sample
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# domain  domain for which the indicator is created
# s              sample
#----------------------------------------------------------------------------
----------------------------------------
s_d <- function(domain,s)
{
domain_indicator(domain,s)
}

#============================================================================
# Subject: Domain subset of sample
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# domain  domain for which the data set is created
# s              sample
#----------------------------------------------------------------------------
----------------------------------------
data_s_d <- function(domain,s)
{
s[s_d(domain,s),]
}

#============================================================================
# Subject: Values of y in a sample domain
```

```
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# s              sample data set
# y               y vector or the name of the y variable
# domain  domain
#-------------------------------------------------------------------------------------
----------------------------------------
y_d <- function(s,y,domain)
{
if (is.numeric(y))
    y
else
    {
    s[[y]][s_d(domain,s)] #Here y is the name of the y variable
    }
}

#=====================================================================
# Subject: Values of weights in a sample domain
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# s              sample data set
# weight    vector of weights or the name of the weight variable
# domain  domain
#-------------------------------------------------------------------------------------
----------------------------------------
w_sd <- function(s,weight,domain)
{
if (is.numeric(weight))
    weight
else
    s[[weight]][s_d(domain,s)]
}

#=====================================================================
# Subject: Size of population domain
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# domain          domain
# population     population data set
#-------------------------------------------------------------------------------------
----------------------------------------
pop_n_d <- function(domain,population)
{
sum(domain_indicator(domain,population))
}

#=====================================================================
```

```
# Subject: Estimated size of population domain
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# s                     sample data set
# weight           weight vector or name of weight variable
# domain         domain
#-------------------------------------------------------------------------------------
----------------------------------------
pop_nhat_d <- function(s,weight,domain)
{
sum(w_sd(s,weight,domain))
}

#=====================================================================
# Subject: Indexes of domains in a data frame.
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# data                     data set
# domain               Name of created variable
# variables             List or vector of variables determining the domains
#
# Returns the vector of indices of domains, order is as returned by function
all_domains
#-------------------------------------------------------------------------------------
----------------------------------------
domain_indices <- function(data,domains)
{
indicators = sapply(domains,function(d){Position(function(x){identical(x,d)}, domains)
* domain_indicator(d,data)})
rowSums(indicators)
}

#=====================================================================
# Subject: Add domain variable (indices of domains) to a data frame. Returns the data.
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# data                                    data set
# domain_variable              Name of created variable
# variables                           List or vector of variables determining the
domains
#-------------------------------------------------------------------------------------
----------------------------------------
add_domain_variable <- function(data,domain_variable,variables)
{
domains <- all_domains("",variables,data)
data[[domain_variable]] <- domain_indices(data,domains)
data
}
```

```
#============================================================================
# Subject: Values of domain variables of distinct domains as a data frame.
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# region_id       Name of the region variable, such as "nuts3"
# variables        List of variable names whose values define the domains
# dataset           A data set
# NOTE: in maintanence, synchronize code changes with all_domains
#----------------------------------------------------------------------------
----------------------------------------
data_of_distinct_domains <-  function(region_id,variables,dataset)
{
used_cols = dataset[,names(dataset)==variables]
if (length(variables)==1)
    {
    value_list = used_cols
    }
else
    {
    index=1
    value_list = list()
    for(row in 1:dim(dataset)[1])
        {
        value_list[[index]]=as.numeric(used_cols[row,])
        index=index+1
        }
    }
ind=1
data = data.frame(matrix(nrow=1,ncol=length(value_list[[1]])))
for(list_of_values in unique(value_list))
    {
    data[ind,] = list_of_values
    ind=ind+1
    }
names(data) <- variables
data
}

#============================================================================
# Subject: Index of domain in a list (-1 if not found)
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# d     domain
# list  list of domains
#----------------------------------------------------------------------------
----------------------------------------
#Returns index of domain in a list, or -1 if not found
index_of_domain_in_list <- function(d, list)
```

```
{
for(idx in 1:length(list))
    {
    if (is_same_domain(d,list[[idx]]))
        return(idx)
    }
return(-1)
}

#============================================================================
# Subject: Returns TRUE if domain does not contain observation in the data set
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# d        domain
# data data set
#----------------------------------------------------------------------------------------
----------------------------------------
is_empty_domain <- function(d,data)
{
sum(domain_indicator(d,data))==0
}

############################
#TESTS
############################
test_region <- function()
{
dom1 <- new("Domain","d",variables=list("d"),values=list(1))
dom2 <- new("Domain","d",variables=list("d"),values=list(2))
dom3 <- new("Domain","d",variables=list("d"),values=list(3))
domn1 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(1,1))
domn2 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(2,2))
domn3 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(2,3))

stopifnot(region(dom1)==1)
stopifnot(region(dom2)==2)
stopifnot(region(dom3)==3)
stopifnot(region(domn1)==1)
stopifnot(region(domn2)==2)
stopifnot(region(domn3)==2)
}

############################
test_domain_indicator <- function()
{
dom1 <- new("Domain","d",variables=list("d"),values=list(1))
dom2 <- new("Domain","d",variables=list("d"),values=list(2))
dom3 <- new("Domain","d",variables=list("d"),values=list(3))
domn1 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(1,1))
domn2 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(2,2))
```

```
domn3 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(2,3))
d=c(1,1,1,2,3)
nuts3=c(1,1,1,2,2)
data<-data.frame(d,nuts3)
stopifnot(seq(5)[domain_indicator(dom1,data)]==c(1,2,3))
stopifnot(seq(5)[domain_indicator(dom2,data)]==c(4))
stopifnot(seq(5)[domain_indicator(dom3,data)]==c(5))
stopifnot(seq(5)[domain_indicator(domn1,data)]==c(1,2,3))
stopifnot(seq(5)[domain_indicator(domn2,data)]==c(4))
stopifnot(seq(5)[domain_indicator(domn3,data)]==c(5))

dom <- new("Domain","nuts3",variables=list("nuts3","d","d2"),values=list(1,1,2))
d=c(1,1,1,2,3)
d2=c(1,2,1,1,2)
nuts3=c(1,1,1,2,2)
data<-data.frame(d,d2,nuts3)
stopifnot(seq(5)[domain_indicator(dom,data)] == c(2))
}

##########################
test_data_set <- function()
{
dom1 <- new("Domain","d",variables=list("d"),values=list(1))
domn1 <- new("Domain","nuts3",variables=list("nuts3","d"),values=list(2,3))
y=c(1,2,3,4,5)
d=c(1,1,1,2,3)
nuts3=c(1,1,1,2,2)
data<-data.frame(y,d,nuts3)
data1 <- set(dom1,data)
stopifnot(identical(data1$y, c(1,2,3)))
data2 <- set(domn1,data)
stopifnot(identical(data2$y, c(5)))
}

##########################
test_all_domains <- function()
{
dom1 <- new("Domain","d",variables=list("d"),values=list(1))
domn1 <- new("Domain","nuts3",variables=list("d","nuts3"),values=list(1,2))
d =   c(1,1,2,2,3)
nuts3=c(2,2,3,3,1)
data <- data.frame(d,nuts3)
dom2 <- new("Domain","d",variables=list("d"),values=list(2))
dom3 <- new("Domain","d",variables=list("d"),values=list(3))
expected = list(dom1,dom2,dom3)
assert_equal(expected, all_domains("d",list("d"),data))
domn2 <- new("Domain","d",variables=list("d","nuts3"),values=list(2,3))
domn3 <- new("Domain","d",variables=list("d","nuts3"),values=list(3,1))

expected2 = list(domn1,domn2,domn3)
for(index in 1:3)
```

```
    {
    assert_equal(expected2[[index]]@values,
all_domains("d",list("d","nuts3"),data)[[index]]@values)
    assert_equal(expected2[[index]]@variables,
all_domains("d",list("d","nuts3"),data)[[index]]@variables)
    }
}

############################
test_all_similar_domains <- function()
{
dom1 <- new("Domain","d",variables=list("d"),values=list(1))
d =   c(1,1,2,2,3)
nuts3=c(2,2,3,3,1)
data <- data.frame(d,nuts3)
dom2 <- new("Domain","d",variables=list("d"),values=list(2))
dom3 <- new("Domain","d",variables=list("d"),values=list(3))
expected = list(dom1,dom2,dom3)
assert_equal(expected, all_similar_domains(dom1,data))
}

############################
test_any_domain <- function()
{
d =   c(1,1,2,2,3)
nuts3=c(2,2,3,3,1)
data <- data.frame(d,nuts3)
assert_equal(c(1,2,3,4,5),domain_vector(any_domain(),data,c(1,2,3,4,5)))
}

############################
test_add_to_data <- function()
{
d =   c(1,1,2,2,2)
d2 =   c(1,1,1,2,2)
data <- data.frame(d,d2)
new_data <- add_domain_variable(data,"domain",c(d,d2))
assert_equal(c(1,1,2,3,3),new_data[["domain"]])
}

############################################
test_domain_estimator_helpers <- function()
{
y = c(3,4, 5,5,6)
x = c(2,5, 4,5,6)
w= c(3,2, 3,2,2)
domain = c(1,1, 2,2,2)
data = data.frame(y,w,x,domain)

x = c(3,5, 1,2,4,2,4,5)
domain = c(2,2,1,1,1,2,2,2)
```

```
pop = data.frame(x,domain)

dom1 = new("Domain",region="domain",
variables=list("domain"),
values = list(1)
)
dom2 = new("Domain",region="domain",
variables=list("domain"),
values = list(2)
)

assert_equal(s_d(dom1,data),c(TRUE,TRUE,FALSE,FALSE,FALSE))
assert_equal(s_d(dom2,data),c(FALSE,FALSE,TRUE,TRUE,TRUE))
assert_equal(data_s_d(dom1,data)[["y"]],c(3,4))
assert_equal(data_s_d(dom2,data)[["y"]],c(5,5,6))
assert_equal(w_sd(data,"w",dom1),c(3,2))
assert_equal(w_sd(data,"w",dom2),c(3,2,2))
assert_equal(pop_nhat_d(data,"w",dom1),3+2)
assert_equal(pop_nhat_d(data,"w",dom2),3+2+2)
assert_equal(pop_n_d(dom1,pop),3)
assert_equal(pop_n_d(dom2,pop),5)

}

test_domain <- function()
{
test_region()
test_domain_indicator()
test_data_set()
test_all_domains()
test_all_similar_domains()
test_any_domain()
test_domain_estimator_helpers()
print("Domain tests pass")
}
```

Code `test_data.r`

```
#Test data and functions for assertion testing
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi

testsample1 <- function()
{
y=c(1,3,2,4,5)
x1=c(2,2,5,6,8)
x2=c(3,4,6,7,11)
yc=c(0,1,1,0,0)
w=c(2,3,4,2,3)
d=c(1,1,1,2,3)
nut3=c(1,1,1,2,2)
data.frame(y,x1,x2,yc,w,d,nut3)
}

testpop1 <- function()
{
x1= c(2,2,5,6,8,2,3)
x2=c(3,4,6,7,11,7,1)
d= c(1,1,1,1,1,2,3)
nut3=c(1,1,1,1,1,2,2)
data.frame(x1,x2,d,nut3)
}

nearly_equal <- function(vec1,vec2,tol)
{
diff = abs(as.numeric(vec1)-as.numeric(vec2))
length(as.numeric(vec1))>0 & max(diff)<=tol
}


assert_nearly_equal <- function(vec1,vec2,tol)
{
if (is.numeric(vec1) & is.numeric(vec2))
    {
    equals = nearly_equal(vec1,vec2,tol)
    }
else
    {
    equals = identical(vec1,vec2)
    }
if (!equals)
    {
    print("not equal:")
    print(vec1)
    print(vec2)
```

```
    stopifnot(equals)
    }
}

assert_equal <- function(vec1,vec2) {assert_nearly_equal(vec1,vec2,1e-6)}

############################
# Test variables

test_setup <- function()
{
    y = c(1,0,0,3,6,100,4,5,5,6,1,2,3,2,1)*10
    x = c(1,1,0,2,5,100,3,3,5,6,1,2,3,2,1)
    w = c(5,2,3,3,5,2,2,2,2,2,2,2,4,3,5)
    domain = c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2)
}

############################
test_s <- function()
{
    y = c(1,0,0,3,6,100,4,5,5,6,1,2,3,2,1)*10
    x = c(1,1,0,2,5,100,3,3,5,6,1,2,3,2,1)
    w = c(5,2,3,3,5,2,2,2,2,2,2,2,4,3,5)
    domain = c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2)
    data.frame(y,x,w,domain)
}

test_U <- function()
{
    x = c(2,1,2,1,5,3,4,4,5,5,1,2,1,5,3,4,4,5,5,6,6,6,1.1,5,5)
    domain = c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2)
    x2 = c(2,1,2,1,5,3,4,4,5,5,1,2,1,5,3,4,4,5,5,6,6,6,1.1,5,5)
    domain2 = c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2)
    data.frame(x,domain,x2,domain2)
}

dom1 = new("Domain",region="domain",
variables=list("domain"),
values = list(1)
)

dom2 = new("Domain",region="domain",
variables=list("domain"),
values = list(2)
)
```

Code EstimatedDistributionFunction.r

```
#EstimatedDistributionFunction is the HT estimate of the cumulative distribution
function.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi

#============================================================================
# Subject: Class EstimatedDistributionFunction, HT estimate of the cumulative
distribution function.
#----------------------------------------------------------------------------------------
----------------------------------------
# SLOTS
# y                                  Name of the y variable
# w                                  Name of design weight variable
# data                              Sample data set
# w_cum                          Vector of cumulative weights
# y_sorted                       Sorted y vector
# nhat                               Sum of weights
#----------------------------------------------------------------------------------------
----------------------------------------

setClass("EstimatedDistributionFunction",
representation(
y = "character",
w= "character",
data= "data.frame",
w_cum = "numeric",
y_sorted ="numeric",
nhat = "numeric" #sum of weights
)
)

#============================================================================
# Subject: Function initializing objects of class EstimatedDistributionFunction
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# data                               Sample data set
# y                                     Name of the y variable
# w                                   Name of design weight variable
#----------------------------------------------------------------------------------------
----------------------------------------
setMethod("initialize","EstimatedDistributionFunction",
function(.Object,data=data,y=y,w=w)
{
.Object@data=data
.Object@w=w
.Object@y=y
```

```
if (is.numeric(w))
    wgt=w
else
    wgt = data[[w]]
if (is.numeric(y))
    yvec=y
else
    yvec =  data[[y]]
w_sorted_by_y = wgt[order(yvec)]
.Object@y_sorted = sort(yvec)
.Object@w_cum = cumsum(w_sorted_by_y)
.Object@nhat = sum(wgt)
callNextMethod(.Object,data=data,y=y,w=w)
}
)

#============================================================================
# Subject: Interval of indices where the percentage point is found
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# self                          Instance of EstimatedDistributionFunction
# pct                           Percentage point (in [0,1])
#----------------------------------------------------------------------------------------
----------------------------------------
inv_interval <- function(self,pct)
{
search = pct*self@nhat
below = findInterval(search,self@w_cum,rightmost.closed=TRUE)
c(below,below+1)
}

#============================================================================
# Subject: Value of the estimated cumulative distribution function at an index
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# self                          Instance of EstimatedDistributionFunction
# index                        Index of a sample unit
#----------------------------------------------------------------------------------------
----------------------------------------
setMethod("value","EstimatedDistributionFunction",
function(self,index)
{
self@w_cum[index]/self@nhat
}
)



#============================================================================
```

```
# Subject: Percentile derived from the HT estimate of cumulative distribution function
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# self                          Instance of EstimatedDistributionFunction
# pct                           Percentage point in [0,1]
#-------------------------------------------------------------------------------------
----------------------------------------
setMethod("percentile","EstimatedDistributionFunction",
function(self,pct)
{
sorted=self@y_sorted
if (pct==0) return (sorted[1])
if (pct==1) return (sorted[length(sorted)])
range = inv_interval(self,pct)
if (range[1]==0)
    point = sorted[1]
else
    {
    if (pct==0.5 && value(self,range[1])==pct)
               {
              point = mean(sorted[range])
               }
    else
        {
        if (value(self,range[1])>=pct)
        point = sorted[range[1]]
        else
        point = sorted[range[2]]
        }
    }
point
}
)

############################
#TESTS
############################

test_estimated_distribution_function <- function()
{
y=  c(1,3,2,4,5)
w=c(2,3,4,2,3)
data <- data.frame(y,w)
df = new("EstimatedDistributionFunction",data=data,y="y",w="w")


stopifnot(inv_interval(df,4.0/14.0)==c(1,2))
stopifnot(inv_interval(df,0.5)==c(2,3))
stopifnot(inv_interval(df,6.0/14.0+1e-5)==c(2,3))
stopifnot(inv_interval(df,9.0/14.0-1e-5)==c(2,3))
```

```
stopifnot(inv_interval(df,12.0/14.0)==c(4,5))


stopifnot(percentile(df,0.25)==2)
stopifnot(percentile(df,6.0/df@nhat)==2)
stopifnot(percentile(df,0.5)==3)
stopifnot(percentile(df,0)==1)
stopifnot(percentile(df,1)==5)

y =  c(1,2,4,5)
w = c(2,2,2,2)
data <- data.frame(y,w)
df = new("EstimatedDistributionFunction",data=data,y="y",w="w")
stopifnot(percentile(df,0.5)==3) #Average of two values (2,4)

print("EstimatedDistributionFunction tests passed")
}
```

Code expanded_predictions.r

```
#Expanded predictions are transformed predictions so that their percentiles
#are closer to percentiles of income in sample. In contrast with the Ameli Deliverable
2.2, the logarithmic tranformation is of form log(x+c),
# where x+c is positive for all observations and predictions. Backtransformed
predictions can be negative.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi

#============================================================================
# Subject: Predictions of a domain transformed so that the percentiles of transformed
predictions are
# closer to percentiles of income in the whole sample
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                        The sample data set
# population                   The population data set
# predicted                     The original predictions
# domain                        Domain (instance of class Domain) where the
transformation is done
# y                                   Name of the y variable
# weight                          Name of design weight variable
# percentages              List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
#----------------------------------------------------------------------------------------
----------------------------------------
expanded_predictions_s <-
function(sample,population,predicted,domain,y,weight,percentages)
{
y_cdf <- new("EstimatedDistributionFunction", sample,y,weight)
y_pct <- sapply(percentages, function(pct){percentile(y_cdf,pct/100)})
domain_predictions <- domain_vector(domain,population,predicted)

pred_w <- array(1,length(domain_predictions))
pred_data <- data.frame(domain_predictions,pred_w)
pred_cdf <- new("EstimatedDistributionFunction",
pred_data,"domain_predictions","pred_w")
pred_pct <- sapply(percentages, function(pct){percentile(pred_cdf,pct/100)})

pct_data <- data.frame(y_pct,pred_pct)
model <- lm("y_pct ~ pred_pct",pct_data)
b0 <- as.numeric(coefficients(model)[1])
b1 <- as.numeric(coefficients(model)[2])
b1*domain_predictions + b0
}

#============================================================================
```

```
# Subject: Create function log(x+c)
logp <- function(c){function(x){log(x+c)}}

#=============================================================================
# Subject: Create function exp(x)-c
expm <- function(c){function(x){exp(x)-c}}

#=============================================================================
# Subject: Logarithms (log(x+c)) of percentiles of observations.
# INPUT
# sample                        The sample data set
# domain                        Domain (instance of class Domain)
# y                                    Name of the y variable
# weight                          Name of design weight variable
# percentages               List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
# c                                    Constant for log(x+c) -transformation

log_percentiles <- function(sample,domain,y,w,percentages,c)
{
    in_domain <- domain_indicator(domain,sample)
    yv <- sample[[y]][in_domain]
    wv <- sample[[w]][in_domain]
    data <- data.frame(yv,wv)
    cdf <- new("EstimatedDistributionFunction", data,"yv","wv")
    percentiles <- sapply(percentages, function(pct){percentile(cdf,pct/100)})
    sapply(percentiles,logp(c))
}

#=============================================================================
# Subject: Logarithms (log(x+c)) of percentiles of predictions.
# INPUT
# sample                        The sample data set
# population                   The population data set
# predicted                     The original predictions
# domain                        Domain (instance of class Domain) where the
transformation is done
# y                                    Name of the y variable
# weight                          Name of design weight variable
# percentages               List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
# c                                    Constant for log(x+c) -transformation

log_pred_percentiles <-
function(sample,population,predicted,domain,y,weight,percentages,c)
{
    pred <- domain_vector(domain,population,predicted)
    constant_w <- array(1,length(pred))
    pred_data <- data.frame(pred,constant_w)
    pred_cdf <- new("EstimatedDistributionFunction", pred_data,"pred","constant_w")
```

```
    log_pct <- sapply(sapply(percentages,
function(pct){percentile(pred_cdf,pct/100)}),logp(c))
    log_pct
}

#=========================================================================
# Subject: Model fitted to percentiles of observations and predictions from all
domains.
# The model contains domain-specific intercepts.
# Synchronize changes with percentile_transformation.
#-------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                        The sample data set
# population                   The population data set
# predicted                     The original predictions from all domains
# y                                    Name of the y variable
# weight                          Name of design weight variable
# percentages               List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
# all_domains               List of all domains
# c                                    Constant for log(x+c) -transformation
#-------------------------------------------------------------------------------
----------------------------------------
percentile_model <- function(sample,population,predicted,y,w,percentages,domains,c)
{
y_pct <- Reduce("c", sapply(domains,
function(d){log_percentiles(sample,d,y,w,percentages,c)}))
pred_pct <- Reduce("c", sapply(domains,
function(d){log_pred_percentiles(sample,population,predicted,d,y,weight,percentages,c)
}))
domain_vector <- Reduce("c", sapply(1:length(domains),
function(d){array(d,length(percentages))}))
domain_factor <- factor(domain_vector)
lm(y_pct ~ pred_pct + domain_factor)
}

#=========================================================================
# Subject: Create a transformation function of predictions in a domain using common
percentile_model. Slope is common to all domains,
# intercept depends on domain. See Equation (18) in Ameli deliverable.
# Returns a function of domain that returns transformed domain predictions.
# Synchronize changes with percentile_model.
#-------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                        The sample data set
# population                   The population data set
# predicted                     The original predictions
# y                                    Name of the y variable
# weight                          Name of design weight variable
```

```
# percentages               List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
# all_domains               List of all domains
# c                                   Constant for log(x+c) -transformation
#-------------------------------------------------------------------------------------
----------------------------------------
percentile_transformation <-
function(sample,population,predicted,y,weight,percentages,all_domains,c)
{
mod <-
percentile_model(sample,population,predicted,y,weight,percentages,all_domains,c)
function(domain)
    {
    b0 <- coefficients(mod)[1]  #Intercept
    if (is_same_domain(domain,all_domains[[1]]))
        b0d = 0     #Domain intercept for the first class
    else
        b0d = coefficients(mod)[1+Position(function(x){is_same_domain(x,domain)},
all_domains)] #Domain intercept
    b1 = coefficients(mod)[2]
    function(pred){b0 + b0d + b1*pred}
    }
}

#=====================================================================
# Subject: Value used in log(x+c)-transformation. If all values are positive or zero,
c=1, otherwise it is abs(min(x))+1.
# INPUT
# x             Vector
#-------------------------------------------------------------------------------------
----------------------------------------
pos_min <- function(x)
{
    if (min(x)>=0)
        1
    else
        abs(min(x))+1
}


#=====================================================================
# Subject: Predictions of a domain transformed so that the percentiles of logarithms
of transformed predictions are
# closer to percentiles of logarithms of income in domains. See Equation (18) in WP2
Deliverable.
# A model with domain-specific intercepts is fitted to the percentiles
# calculated separately in each domain.
# Returns a function of domain, which returns the transformed domain predictions.
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
```

```
# sample                        The sample data set
# population                   The population data set
# predicted                     All the original predictions, must be in same order as
observations in population.
# y                                   Name of the y variable
# weight                          Name of design weight variable
# percentages               List of percentage points in range [0,100] used to compare
percentiles of predictions and original y values
# all_domains               List of all domains
#-------------------------------------------------------------------------------------
----------------------------------------
log_expanded_predictions <-
function(sample,population,predicted,y,weight,percentages,all_domains)
{
c <- pos_min(c(y,predicted))
transformation <-
percentile_transformation(sample,population,predicted,y,weight,percentages,all_domains
,c) #Percentile model is fitted here

function(domain)
    {
    domain_pred <- domain_vector(domain,population,predicted)
    log_pred <- sapply(domain_pred, logp(c))
    sapply(transformation(domain)(log_pred), expm(c))
    }
}

#=====================================================================
# Subject: All predictions transformed so that the percentiles of transformed
predictions are
# closer to percentiles of income in sample
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                        The sample data set
# population                   The population data set
# predicted                     The original predictions
# y                                    Name of the y variable
# weight                          Name of design weight variable
# percentages                List of percentage points in range [0,100] used to
compare percentages of predictions and original y values
#-------------------------------------------------------------------------------------
----------------------------------------
all_expanded_predictions <- function(sample,population,predicted,y,weight,percentages)
{
expanded_predictions(sample,population,predicted,any_domain(),y,weight,percentages)
}


##########################
#TESTS
```

```
##########################

test_expanded_predictions_s <- function()
{
nuts = c(1,1,1,1,1,2,2,2,2,2)
income=c(1,5,3,4,2,2,3,5,6,4)
x=c(1,3,4,5,1,2,3,4,7,5)
pop = data.frame(nuts,income,x)

nuts=c(1,1,1,2,2)
income=c(1,5,2,6,4)
x=c(1,3,1,7,5)
w=c(2,2,2,2,2)
sample = data.frame(nuts,income,x,w)
dom1 = new("Domain","nuts",list("nuts"),list(1))
dom2 = new("Domain","nuts",list("nuts"),list(2))

mod <- lm(income~x)

exp1 <-
expanded_predictions_s(sample,pop,predict(mod,newdata=pop),dom1,"income","w",1:99)
exp2 <-
expanded_predictions_s(sample,pop,predict(mod,newdata=pop),dom2,"income","w",1:99)
assert_equal(exp1,c(1.5462590506838692, 3.8294448913917964, 4.97103781174576,
6.112630732099724, 1.5462590506838692))
assert_equal(exp2,c(1.305049088359136, 2.350631136044933, 3.396213183730731,
6.532959326788124, 4.441795231416528))
print("Expanded prediction test passed")
}
#=============================================================================
test_log_percentiles <- function()
{
nuts=c(1,1,1,2,2)
income=c(0,5,2,6,4)
x=c(1,3,1,7,5)
w=c(2,2,2,2,2)
sample = data.frame(nuts,income,x,w)
dom1 = new("Domain","nuts",list("nuts"),list(1))
dom2 = new("Domain","nuts",list("nuts"),list(2))

d1 <- log_percentiles(sample,dom1,"income","w",1:99,0)
assert_equal(log(2),d1[1])
assert_equal(log(5),d1[99])

d2 <- log_percentiles(sample,dom2,"income","w",1:99,0)
assert_equal(log(4),d2[1])
assert_equal(log(6),d2[99])
print("Tested log_percentiles")

}
```

```
#=========================================================================
test_percentile_model <- function()
{
income<- c(1,2,2,3,4, 1,4,5,5,6)
x <- c(1,2,3,2,5, 1,4,3,5,6)
w <- c(2,3,2,3,2, 2,3,3,2,2)
domain <- c(1,1,1,1,1, 2,2,2,2,2)
sample = data.frame(domain,income,x,w)
dom1 = new("Domain","domain",list("domain"),list(1))
dom2 = new("Domain","domain",list("domain"),list(2))
mod <- lm(income~x,weights=w)
pop <- test_U()
predicted <- predict(mod,newdata=pop)

mod<-percentile_model(sample,pop,predicted,"income","w",1:99,c(dom1,dom2),1)
assert_equal(c(-0.08924085199813607, 0.8746958476092522, 0.22201028495524608),
coefficients(mod))
print("Tested percentile_model")
}
#=========================================================================
test_percentile_transformation <- function()
{
income<- c(1,2,2,3,4, 1,4,5,5,6)
x <- c(1,2,3,2,5, 1,4,3,5,6)
w <- c(2,3,2,3,2, 2,3,3,2,2)
domain <- c(1,1,1,1,1, 2,2,2,2,2)
sample = data.frame(domain,income,x,w)
dom1 = new("Domain","domain",list("domain"),list(1))
dom2 = new("Domain","domain",list("domain"),list(2))
mod <- lm(income~x,weights=w)
pop <- test_U()
predicted <- predict(mod,newdata=pop)

f <- percentile_transformation(sample,pop,predicted,"income","w",1:99,c(dom1,dom2),1)
pred <- c(1,2,3,4,5,6)
assert_nearly_equal(-0.08924085199813607 + 0.8746958476092522*pred, f(dom1)(pred), 1e-
6)
assert_nearly_equal(0.13276943295711002 + 0.8746958476092522*pred, f(dom2)(pred), 1e-
6)
}

#=========================================================================
test_log_expanded_predictions <- function()
{
income<- c(1,2,2,3,4, 1,4,5,5,6)
x <- c(1,2,3,2,5, 1,4,3,5,6)
w <- c(2,3,2,3,2, 2,3,3,2,2)
domain <- c(1,1,1,1,1, 2,2,2,2,2)
sample = data.frame(domain,income,x,w)
```

```
dom1 = new("Domain","domain",list("domain"),list(1))
dom2 = new("Domain","domain",list("domain"),list(2))
mod <- lm(income~x,weights=w)
pop <- test_U()
predicted <- predict(mod,newdata=pop)

exp1<- c(1.027638956342164, 1.027638956342164, 1.027638956342164, 1.027638956342164,
1.6364845270781696, 1.6364845270781696,
     1.6364845270781696, 2.2256294228885887, 2.2256294228885887, 2.799653857665162,
2.799653857665162, 2.799653857665162, 2.799653857665162,
     3.361463617851996, 3.361463617851996, 3.361463617851996, 3.361463617851996,
3.361463617851996, 3.361463617851996, 3.9130606669526804)

exp2<-c(1.6090411542720713, 4.445654646872717, 4.445654646872717, 5.1343700178645735,
5.1343700178645735)

assert_equal(exp1,
sort(log_expanded_predictions(sample,pop,predicted,"income","w",1:99,c(dom1,dom2))(dom
1)))
assert_equal(exp2,
sort(log_expanded_predictions(sample,pop,predicted,"income","w",1:99,c(dom1,dom2))(dom
2)))
print("Expanded predictions tested")
}
```

Code composite_estimators.r

```
#A composite estimator is a linear combination of a direct estimator and a predictor.
Code for bootstrap variance estimation is in this file.
#Depends on R function sample.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#============================================================================
# Subject: Function creating a composite estimator
# Creates composite estimators and calculates composite weights for domains.
# The final composite estimators are returned as a function of domain.
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# unbiased                     The (nearly) unbiased component, such as direct_gini
# predictor                     The predictor estimator
# sample                        The sample data set
# population                   The population data set
# y                                    Name of the y variable
# model                           Model fitted to data by function lm, glm, lme, or
nlme. The model should yield predicted values by call of function predict.
# back_transformation Function back-transforming the predictions to the original
scale.
# x_list                             The list of names of quantitative x-variables,
such as list("age") or list(). Required only for n-calibrated predictors.
# xq_list                           The list of names of qualitative x-variables, such
as list("socstrat","lfs") or list(). Required only for n-calibrated predictors.
# unknown                       The list of names of x-variables not known in
population, such as list("socstrat") or list(). Required only for n-calibrated
predictors.
# domain_variables     List of names of variables defining the domains
# weight                           Name of design weight variable
# reference_set             Data set determining the poverty thresholds (usually whole
sample)
# percentages               List of percentage points used in the expanded predictors;
default is 1:99. Consider 1:50 for poverty gap, 1:99 for other statistics.
# missing_handler       Function of  domain, used when the direct estimate is not
defined (NaN).
#----------------------------------------------------------------------------------------
----------------------------------------
composite_estimators <-
function(unbiased,predictor,sample,population,y,model,back_transformation,x_list=list(
),xq_list=list(),unknown=list(),domain_variables,weight,reference_set,percentages=1:99
,missing_handler)
  {
    direct_domain_estimate <- function(s)
        {
```

```
        unbiased_estimators = domain_estimators(unbiased,
sample=s,population,y,model,back_transformation,list(),list(),list(),domain_variables,
weight,reference_set)
        function(domain)
            {
            est <- unbiased_estimators(domain)
            if (is.nan(est))
                {
                missing_handler(domain) #NOTE: does not depend on bootstrap sample
(that would require fitting a model to each bootstrap sample)
                }
            else
                est
            }
        }

    weights = individual_composite_weights(predictor,direct_domain_estimate,
sample,population,y,model,back_transformation,domain_variables,weight,reference_set)
    final_weight = median(weights)
    direct_estimate <- direct_domain_estimate(sample)
    function(domain)
        {
            (1 - final_weight)*predictor(domain) +
final_weight*direct_estimate(domain)
        }
    }


#============================================================================
# Subject: Function forcing weight to [0, 1]
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# wgt   Scalar weight
#----------------------------------------------------------------------------------------
----------------------------------------
checked_weight <- function(wgt)
{
if (is.nan(wgt))
    {
    return(1)
    }
if (wgt<0)
    {
    return(0)
    }
if (wgt>1)
    {
    return(1)
    }
wgt
```

```
}


#============================================================================
# Subject: List of composite weights for domains (Equations 20 and 21 in the Ameli
Deliverable).
#----------------------------------------------------------------------------------
----------------------------------------
# INPUT
# predictor                           Function of domain, returns predictor estimate
for the domain
# direct_domain_estimate Function of sample that returns a function of domain (domain
estimate in the sample's domain)
# sample                            Sample data set
# population                       The population data set
# y                                        Name of the y variable
# model                              Model fitted to data by function lm, glm, lmer or
glmer.
# back_transformation    Function back-transforming the predictions to the original
scale. Can be obtained by function backtransformation.
# domain_variables         List of names of variables defining the domains
# weight                              Name of design weight variable
# reference_set                Data set determining the poverty thresholds (usually
whole sample)
# percentages                  Vector of percentage points (such as 1:99) used to
compare percentiles of y and predictions.
#----------------------------------------------------------------------------------
----------------------------------------
individual_composite_weights <-
function(predictor,direct_domain_estimate,sample,population,y,model,back_transformatio
n,domain_variables,weight,reference_set)
{
var =
bootstrap_variances(direct_domain_estimate,sample,population,y,model,back_transformati
on,domain_variables,weight,reference_set)

direct_estimate <- direct_domain_estimate(sample)
wgt <- function(domain)
    {
    diff = predictor(domain) - direct_estimate(domain)
    syn_mse = diff*diff - var(domain)
    if (syn_mse<0)
       {
      syn_mse=0
       }
    checked_weight(syn_mse/(syn_mse + var(domain)))
    }

sapply(all_domains("",domain_variables,sample),wgt)
}
```

```
#============================================================================
# Subject: List of indices of sample units in bootstrap population
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                            Sample data set
# weight                              Name of weight variable in sample
#----------------------------------------------------------------------------
----------------------------------------
bootstrap_population_indices <- function(sample,weight)
{
wgt = sample[[weight]]
n = dim(sample)[1]
indices = c()
for (index in 1:n)
    {
    indices = append(indices, array(index,floor(wgt[index])))
    }
indices
}


#============================================================================
# Subject:  Inclusion probabilities of bootstrap population units
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                            Sample data set
# weight                              Name of weight variable in sample
#----------------------------------------------------------------------------
----------------------------------------
bootstrap_population_pi <- function(sample,weight)
{
wgt = sample[[weight]]
n = dim(sample)[1]
pi = c()
for (index in 1:n)
    {
    pi = append(pi, array(1/wgt[index],floor(wgt[index])))
    }
pi
}

#============================================================================
# Subject:  Number of bootstrap samples
#----------------------------------------------------------------------------
----------------------------------------
bootstrap_n <- function()
{
1000
}
```

```
#============================================================================
# Subject:  Bootstrap sample from a sample given indices of sample units in population
and inclusion probabilities
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# s                           Sample data set
# indices                List of indices of sample units in bootstrap population
# pi                          List of inclusion probabilities of bootstrap population
units
#----------------------------------------------------------------------------------------
----------------------------------------
bootstrap_sample <- function(s,indices,pi)
{
n = dim(s)[1]
boot_s <- s[sample(indices, n,prob=pi),]
boot_s
}

#============================================================================
# Subject:  Bootstrap variance of an estimator in a domain.
# Assumes that the returned function is called with a domain that is in the list
all_domains("",domain_variables,sample).
# Assumes that reference set is the bootstrap sample itself for each bootstrap sample.
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# direct_domain_estimate Function of sample that returns a function of domain (domain
estimate in the sample's domain)
# sample                            Sample data set
# population                       Population data set
# reference_set                Reference set for poverty threshold estimation
(usually whole sample)
# y                                        Name of y variable
# weight                               Name of weight variable in sample
# domain_variables         List of variables defining the domains
#----------------------------------------------------------------------------------------
----------------------------------------
bootstrap_variances <-
function(direct_domain_estimate,sample,population,y,model,back_transformation,domain_v
ariables,weight,reference_set)
{
domain_list = all_domains("",domain_variables,sample)

domain_results = array(0,c(length(domain_list), bootstrap_n()))
indices = bootstrap_population_indices(sample,weight)
pi = bootstrap_population_pi(sample,weight)
for (b in 1:bootstrap_n())
    {
    boot_s <- bootstrap_sample(sample,indices,pi)
```

```
    domain_results[,b] = sapply(domain_list,direct_domain_estimate(boot_s))
    }

variance_in_domain <- function(domain_idx){var(domain_results[domain_idx,])}
variances = sapply(1:length(domain_list),variance_in_domain)
print(paste("Bootstrap variances", sapply(variances, toString)))
function(domain) {
    idx=index_of_domain_in_list(domain,domain_list)
    if (idx<0)
        0 #domain not found, variance estimator is not well defined
    else
        variances[idx]
    }
}


############################
#TESTS
############################

test_composite_estimators <- function()
{
y = c(3,4, 5,5,6)
x = c(2,5, 4,5,6)
w= c(3,2, 3,2,2)
domain = c(1,1, 2,2,2)
data = data.frame(y,w,x,domain)

x = c(3,5, 1,2,4,2,4,5)
domain = c(2,2,1,1,1,2,2,2)
pop = data.frame(x,domain)

dom1 = new("Domain",region="domain",
variables=list("domain"),
values = list(1)
)
dom2 = new("Domain",region="domain",
variables=list("domain"),
values = list(2)
)
ind = data[["ind"]] = create_poverty_indicator(data,"y","w",data)
x <- data[["x"]]
mod <- lm(ind~x)

estimators <-
composite_estimators(direct_rate,predictor_rate,"",sample=data,population=pop,y="ind",
mod,identity,list(),list(),list(),domain_variables=list("domain"),weight="w",reference
_set=data)

estimator1 = estimators(dom1)
print("CompositeEstimators test passed")
```

```
}

################################################
test_bootstrap <- function()
{
y = c(3,4, 5,5,6)
x = c(2,5, 4,5,6)
w= c(3,2, 3,2,2)
domain = c(1,1, 2,2,2)
data = data.frame(y,w,x,domain)

x = c(3,5, 1,2,4,2,4,5)
domain = c(2,2,1,1,1,2,2,2)
pop = data.frame(x,domain)

domain_list = all_domains("",list("domain"),data)

dom1 = new("Domain",region="domain",
    variables=list("domain"),
    values = list(1)
    )
dom2 = new("Domain",region="domain",
    variables=list("domain"),
    values = list(2)
    )
print("bootstrap population")
print(bootstrap_population_indices(data,"w"))
indices=bootstrap_population_indices(data,"w")
pi = bootstrap_population_pi(data,"w")
print("index variable")
assert_equal(c(1,1,1,2,2,3,3,3,4,4,5,5),bootstrap_population_indices(data,"w"))

    direct_domain_estimate <- function(s)
        {
        unbiased_estimators = domain_estimators(direct_rate,
sample=s,population,y,NULL,NULL,list(),list(),list(),list("domain"),"w",s)
        function(domain)
            {
            est <- unbiased_estimators(domain)
            if (is.nan(est))
                0
            else
                est
            }
        }


variances <-
bootstrap_variances(direct_domain_estimate,data,pop,"y",NULL,NULL,list("domain"),"w",d
ata)
```

```
#bootstrap_variances <-
function(direct_domain_estimate,sample,population,y,model,back_transformation,domain_v
ariables,weight,reference_set)

print("bootstrap variances:")
print(variances(dom1))
print(variances(dom2))
stopifnot(variances(dom1)>0)
stopifnot(variances(dom2)>0)

print("Test of bootstrap passed")
}
```

Code domain_estimators.r

```
#Code for creating domain estimators of given type. Model is fitted to the whole
sample once for each estimator type.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#============================================================================
# Subject: Weighted residuals in a domain
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# predicted Function returning fitted values with arguments domain, sample
# domain    A Domain instance
# sample    data.frame object
# y                 name of the y variable
# weights    name of the design weights
#----------------------------------------------------------------------------------------
----------------------------------------
weighted_residuals <- function(predicted,domain,sample,y,weights)
  {
    in_domain = s_d(domain,sample)
    sample[[weights]][in_domain]*(sample[[y]][in_domain] - predicted(domain,sample))
  }


#============================================================================
# Subject: Function replacing negative input by 0.
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# x    value
#----------------------------------------------------------------------------------------
----------------------------------------
replace_negatives <- function(x)
{
if (x<0)
    0
else
    x
}

#============================================================================
# Subject: Create function returing predictions or fitted values in a domain.
# sample                         The sample data set
# population                    The population data set
```

```
# model                            Model fitted to data by function lm, glm, lme, or
nlme.  The model should yield predicted values by call of function predict. Use NULL
if model not required.
# original_predictions   Vector of original predictions obtained from model
# back_transformation  Function back-transforming the predictions to the original
scale.
#-------------------------------------------------------------------------------------
----------------------------------------
prediction_function <-
function(sample,population,model,original_predictions,back_transformation)
{
predictions = sapply(original_predictions, back_transformation)
fitted = sapply(predict(model,newdata=sample), back_transformation) #NOTE: fitted
would yield already back-transformed predictions at least in glm
function(domain,data=population)
    {
    if (identical(data,population))
        domain_vector(domain,population,predictions)
    else
        {
        if (identical(data,sample))
            domain_vector(domain,data,fitted)
        }
    }
}


#=============================================================================
# Subject: Domain estimator as a function of domain
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# estimator_type           The estimator function, such as default_gini or greg_rate,
with arguments sample,model,reference_set,y,weight,domain
# sample                         The sample data set
# population                   The population data set
# y                                      Name of the y variable
# model                            Model fitted to data by function lm, glm, lme, or
nlme.  The model should yield predicted values by call of function predict. Use NULL
if model not required.
# back_transformation Function back-transforming the predictions to the original
scale.
# x_list                              Not used
# xq_list                            Not used
# unknown                        Not used
# domain_variables     List of names of variables defining the domains
# weight                             Name of design weight variable
# reference_set             Data set determining the poverty thresholds (usually whole
sample)
# percentages               (Not used)
```

```
#-------------------------------------------------------------------------------------
----------------------------------------
domain_estimators <-
function(estimator_type,sample,population=sample,y,model,back_transformation,x_list=li
st(),xq_list=list(),unknown=list(),domain_variables=list(),weight,reference_set=sample
,percentages=1:99)
{
if (is.null(model))
    {
    predicted = function(domain){c()}
    }
else
    {
    predicted <-
prediction_function(sample,population,model,predict(model,newdata=population),back_tra
nsformation)
    }

function (domain)
    {
    estimator_type(sample,population,model,predicted,reference_set,y,weight,domain)
    }
}

#=========================================================================
# Subject: Creates predictor based on expanded predictions (Equation 18 in Ameli
deliverable).
# Example: First create a function of domain: estimators <-
expanded_domain_predictors(prediction_quintile_share,identity,
sample=data,population=pop,y="y",x=list("x"),factors=list(),
#
domain_variables=list("domain"),list(),weight="w",model_type="lm",reference_set=data,
percentages=1:99, all_domains)
#Estimate for domain d is then estimators(d).
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# estimator_type           The estimator function, such as default_gini or greg_rate,
with arguments sample,model,reference_set,y,weight,domain
# sample                         The sample data set
# population                   The population data set
# y                                      Name of the y variable
# model                            Model fitted to data by function lm, glm,  lme, or
nlme.  The model should yield predicted values by call of function predict.
# back_transformation Function back-transforming the predictions to the original
scale.
# x_list                              Not used
# xq_list                            Not used
# unknown                       Not used
# domain_variables     List of names of variables defining the domains
# weight                            Name of design weight variable
```

```
# reference_set             Data set determining the poverty thresholds (usually the
sample)
# percentages               Vector of percentage points (such as 1:99) used to compare
percentiles of y and predictions. Consider 1:50 for poverty gap, 1:99 for other
statistics.
#-------------------------------------------------------------------------------
----------------------------------------
expanded_domain_predictors <-
function(estimator_type,sample,population=sample,y,model,back_transformation,x_list=li
st(),xq_list=list(),unknown=list(),
    domain_variables=list(),weight,reference_set=sample, percentages=1:99)
{
all_predictions = sapply(predict(model,newdata=population), back_transformation)
predicted = function(domain){domain_vector(domain,population,all_predictions)}
domain_list <- as.list(all_domains("",domain_variables,population))

#Function of domain, uses percentile model fitted once in all domains.
domain_predictions <-
log_expanded_predictions(sample,population,all_predictions,y,weight,percentages,domain
_list)

function(domain)
    {

estimator_type(sample,population,model,domain_predictions,reference_set,y,weight,domai
n)
    }
}


#============================================================================
# Subject: Find function creating predictors, or domain_estimators if argument is NULL
#-------------------------------------------------------------------------------
----------------------------------------
# INPUT
# modifier                        value is NULL, "plain", "expanded" or "calibrated"
#-------------------------------------------------------------------------------
----------------------------------------
estimator_modifier <- function(modifier)
{
if (is.null(modifier) || (modifier=="plain"))
    {
    return (domain_estimators)
    }
else
    {
    if (modifier=="expanded")
        return (expanded_domain_predictors)
    if (modifier=="calibrated")
        return(calibrated_predictors)
    }
```

```
}
```

Code `Gini.r`

```
#Gini coefficient estimators for domains
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#============================================================================
# Subject: Integral of a line from point (x1,y1) to (x2,y2)
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# x1        x-coordinate of the first point
# y1        y-coordinate of the first point
# x2        x-coordinate of the second point
# y2        y-coordinate of the second point
#----------------------------------------------------------------------------
----------------------------------------
line_integral <- function(x1,y1,x2,y2)
{
beta = (y2-y1)/(x2-x1)
alpha = y1- beta*x1
(beta/2)*(x2*x2 - x1*x1) + alpha*(x2-x1)
}


#============================================================================
# Subject: Cumulative sums scaled to [0,1]
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# v     vector
#----------------------------------------------------------------------------
----------------------------------------
cumulative_scaled <- function(v)
{
cumsum(v)/sum(v)
}

#============================================================================
# Subject: Integral of a piecewise linear function through given points, starting from
(0,0).
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# xvec     vector of x-coordinates of points
# yvec     vector of y-coordinates of points
#----------------------------------------------------------------------------
----------------------------------------
```

```
integral <- function(xvec,yvec)
{
sum=line_integral(0,0,xvec[1],yvec[1])
for(idx in 2:length(xvec))
    {
    sum = sum + line_integral(xvec[idx-1], yvec[idx-1], xvec[idx], yvec[idx])
    }
sum
}

#=============================================================================
# Subject: Integral of a Lorenz curve
#-----------------------------------------------------------------------------
----------------------------------------
# INPUT
# x     vector of x-coordinates of points, sorted by y
# y     vector of y-coordinates of points
#-----------------------------------------------------------------------------
----------------------------------------
lorenz_integral <- function(x,y)
{
integral(cumulative_scaled(x), cumulative_scaled(y))
}

#=============================================================================
# Subject: HT estimate of the integral of a Lorenz curve
#-----------------------------------------------------------------------------
----------------------------------------
# INPUT
# w     vector of weights of units
# y      vector of equivalized incomes
#-----------------------------------------------------------------------------
----------------------------------------
ht_integral <- function(w,y)
{
w_sorted <- w[order(y)]
lorenz_integral(w_sorted, w_sorted*sort(y))
}

#=============================================================================
# Subject: Estimated integral of a Lorenz curve when weights are constant
#-----------------------------------------------------------------------------
----------------------------------------
# INPUT
# y      vector of equivalized incomes
#-----------------------------------------------------------------------------
----------------------------------------
simple_integral <- function(y)
{
lorenz_integral( array(1,length(y)), sort(y))
}
```

```
#===========================================================================
# Subject: Estimate of the Gini coefficient based on HT estimate of Lorenz curve
#---------------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# w     vector of weights of units
# y      vector of equivalized incomes
#---------------------------------------------------------------------------------------------
---------------------------------------
weighted_gini <- function(w,y)
{
1 - 2*ht_integral(w,y)
}

#===========================================================================
# Subject: Estimated Gini coefficient when weights are constant
#---------------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# y      vector of equivalized incomes
#---------------------------------------------------------------------------------------------
---------------------------------------
simple_gini <- function(y)
{
1 - 2*simple_integral(y)
}

#===========================================================================
# Subject: Default estimator of the Gini coefficient
#---------------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# sample                       The sample data set
# population                  The population data set
# model                          Model (not used)
# predicted                    Not used
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#---------------------------------------------------------------------------------------------
---------------------------------------
direct_gini <- function(sample,population,model,predicted,reference,y,weight,domain)
{
if (is_empty_domain(domain,sample))
    return(NaN)

wgt = w_sd(sample,weight,domain)
yvec =  y_d(sample,y,domain)
```

```
weighted_gini(wgt,yvec)
}


#============================================================================
# Subject: True value  of the Gini coefficient
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# population                   The population data set
# y                                    Name of the y variable
# domain                       Domain for which the value is calculated
#----------------------------------------------------------------------------------------
----------------------------------------
true_gini <- function(population,y,domain)
{
simple_gini(y_d(population,y,domain))
}

#============================================================================
# Subject: Estimator of the Gini coefficient based on predicted values
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                       The sample data set
# population                  The population data set
# model                          Model
# predicted                     Function, returns predictions in the domain
# reference                    Data set determining the poverty threshold (usually
whole sample) (Not used here)
# y                                    Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#----------------------------------------------------------------------------------------
----------------------------------------
predictor_gini <-
function(sample,population,model,predicted,reference,y,weight,domain)
{
simple_gini(predicted(domain))
}

##########################
#TESTS
##########################

test_gini <- function()
{
test_setup()
s <- test_s()
pop <- test_U()
```

```
estimators <-
domain_estimators(direct_gini,s,s,"y",NULL,NULL,list(),list(),list(),domain_variables=
list("domain"),"w",s)
assert_equal(0.7449698189134805, estimators(dom1))
assert_equal(0.23922413793103448, estimators(dom2))

y<-s[["y"]]
x<-s[["x"]]
w<-s[["w"]]
model <-  lm(y~x, weights=w)
back <- identity
estimators <-
domain_estimators(predictor_gini,s,pop,"y",model,back,list(),list(),list(),domain_vari
ables=list("domain"),"w",s)
assert_equal(c(0.24685398630828514, 0.1760968955791954), sapply(c(dom1,dom2),
estimators))

print("Gini tests pass")
}
```

Code poverty_rate_estimator.r

```
#Poverty rate estimators for domains.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#============================================================================
# Subject: inverse of logit function
#----------------------------------------------------------------------------------------
----------------------------------------
invlogit <- function(x){exp(x)/(1+exp(x))}

#============================================================================
# Subject: Poverty threshold
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# reference                   Data set determining the poverty threshold (usually
whole sample)
# y                                   Name of the y variable
# weight                        Name of design weight variable
#----------------------------------------------------------------------------------------
----------------------------------------
poverty_line <- function(reference, y, weight)
    {
    cdf = new("EstimatedDistributionFunction",data=reference,y=y,w=weight)
    0.6 * percentile(cdf,0.5)
    }

#============================================================================
# Subject: Poverty indicator (1 for poor people with equivalized income smaller than
at-risk-of-poverty threshold)
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                      The sample data set
# y                                   Name of the y variable
# weight                        Name of design weight variable
# reference_set            Data set determining the poverty threshold (usually whole
sample)
#----------------------------------------------------------------------------------------
----------------------------------------
create_poverty_indicator <- function(sample,y,weight,reference_set)
{
indicator = sample[[y]]<=poverty_line(reference_set,y,weight)
as(indicator,"numeric")
}
```

```
#===========================================================================
# Subject: Default estimator of the poverty rate
#---------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# sample                       The sample data set
# population                  The population data set
# model                          Model (not used)
# predicted                     (not used)
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#---------------------------------------------------------------------------------------
---------------------------------------
direct_rate <- function(sample, population,model,predicted,reference,y,weight,domain)
{
if (is_empty_domain(domain,sample))
    return(NaN)
sum(y_d(sample,y,domain) * w_sd(sample,weight,domain)) /
pop_nhat_d(sample,weight,domain)
}



#===========================================================================
# Subject: GREG estimator of the poverty rate
#---------------------------------------------------------------------------------------
---------------------------------------
# INPUT
# sample                       The sample data set
# population                  The population data set
# model                          Model
# predicted                     Function, returns predictions in the domain
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#---------------------------------------------------------------------------------------
---------------------------------------
greg_rate <- function(sample,population,model,predicted,reference,y,weight,domain)
{
size_correction <- function(sample,domain) {1} # alternative could be
pop_n_d(domain,population) / pop_nhat_d(sample,weight,domain)
ressum = sum(weighted_residuals(predicted,domain,sample,y,weight))
replace_negatives( (sum(predicted(domain)) + size_correction(sample,domain)*ressum) /
pop_n_d(domain,population))
```

```
}

#============================================================================
# Subject: Synthetic estimator of the poverty rate
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                      The sample data set
# population                 The population data set
# model                         Model
# predicted                    Function, returns predictions in the domain
# reference                   Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                        Name of design weight variable
# domain                      Domain for which the estimate is calculated
#----------------------------------------------------------------------------
----------------------------------------
predictor_rate <- function(sample,
population,model,predicted,reference,y,weight,domain)
{
sum(predicted(domain)) / pop_n_d(domain,population)
}

#============================================================================
# Subject: EBP estimator of the poverty rate
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                      The sample data set
# population                 The population data set
# model                         Model (typically logistic mixed model)
# predicted                    Vector of predictions from the model
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                          Name of design weight variable
# domain                       Domain for which the estimate is calculated
#----------------------------------------------------------------------------
----------------------------------------
ebp_rate <- function(s, population,model,predicted,reference,y,weight,domain)
{
fit_sum <- domain_sum(domain,s,fitted(model))
replace_negatives( (sum(predicted(domain)) - fit_sum + sum(y_d(s,y,domain))) /
pop_n_d(domain,population))
}


##########################
#TESTS
##########################
```

```
test_indicator <- function()
{
y <- c(1,2,2,3,4, 1,4,5,5,6)
w <- c(2,3,2,3,2, 2,3,3,2,2)
data <- data.frame(y,w)
expected = c(1,1,1,0,0,1,0,0,0,0)

ind <- create_poverty_indicator(sample=data,y="y",weight="w",reference_set=data)
stopifnot(identical(as.vector(ind), expected))
print("PovertyIndicator tests pass")
}


test_poverty_rate <- function()
{
test_setup()
data = test_s()
pop = test_U()
data[["ind"]] = ind = create_poverty_indicator(data,"y","w",data)
estimators <-
domain_estimators(direct_rate,data,data,"ind",NULL,NULL,list(),list(),list(),domain_va
riables=list("domain"),"w",data)
assert_equal(c(0.35714285714285715, 0.4375), sapply(c(dom1,dom2), estimators))

x<-data[["x"]]
mod <- glm(ind~x,data=data,family=binomial) # yields warning about 0/1 predictions
back <- invlogit
estimators <-
domain_estimators(greg_rate,data,pop,"ind",mod,back,list(),list(),list(),domain_variab
les=list("domain"),"w",data)
assert_equal(c(0.2, 0.2), sapply(c(dom1,dom2), estimators))

estimators <-
domain_estimators(ebp_rate,data,pop,"ind",mod,back,list(),list(),list(),domain_variabl
es=list("domain"),"w",data)
assert_equal(c(0.2, 0.2), sapply(c(dom1,dom2), estimators))


# Another data, where glm yields estimated probabilities between 0 and 1.
y = c(1,2,2,3,4, 1,4,5,5,6)
x = c(1,2,3,2,5, 1,4,3,5,6)
w= c(2,3,2,3,2, 2,3,3,2,2)
domain = c(1,1,1,1,1, 2,2,2,2,2)
data2 = data.frame(y,w,x,domain)
data2[["ind"]] = ind = create_poverty_indicator(data2,"y","w",data2)
mod2 <- glm(ind~x,data=data2,family=binomial)
back <- invlogit
dom1 = new("Domain",region="domain",
variables=list("domain"),
```

```
values = list(1)
)

dom2 = new("Domain",region="domain",
variables=list("domain"),
values = list(2)
)

estimators <-
domain_estimators(greg_rate,data2,pop,"ind",mod2,back,list(),list(),list(),domain_vari
ables=list("domain"),"w",data2)
assert_equal(c(0.3546431647924232, 0),  sapply(c(dom1,dom2), estimators))

estimators <-
domain_estimators(ebp_rate,data2,pop,"ind",mod2,back,list(),list(),list(),domain_varia
bles=list("domain"),"w",data2)
assert_equal(c(0.35907751509319, 0.12468298812130071),  sapply(c(dom1,dom2),
estimators))


}

test_poverty_rate_estimator <- function()
{
test_indicator()
test_poverty_rate()
print("Poverty rate estimator tests pass")
}
```

Code poverty_gap_estimator.r

```
#Poverty gap estimators for domains
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#If there are no poor in a domain, direct estimator uses whole sample and estimators
based on predictions use predictions
#for whole population.

#============================================================================
# Subject: Default estimator of the poverty gap
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                        The sample data set
# population                   The population data set
# model                          Model (not used)
# predicted                     Function of domain (not used)
# reference                     Data set determining the poverty threshold (usually
whole sample)
# y                                     Name of the y variable
# weight                          Name of design weight variable
# domain                        Domain for which the estimate is calculated
#-------------------------------------------------------------------------------------
----------------------------------------
direct_gap <- function(sample,population,model,predicted,reference,y,weight,domain)
{
if (is_empty_domain(domain,sample))
    return(NaN)
sd = data_s_d(domain,sample)
t = poverty_line(reference,y,weight)
poor_data = sd[ sd[[y]] <= t,]
if (dim(poor_data)[2]==0)
    {
    poor_data = sample[ sample[[y]] <= t ]
    }
cdf = new("EstimatedDistributionFunction",data=poor_data,y=y,w=weight)
md = percentile(cdf,0.5)
(t - md) / t
}

#============================================================================
# Subject: Poverty gap estimated from predictions.
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# t                                Poverty threshold
```

```
# predicted                Vector of predictions in domain of the population
# all_predicted         Function returning predictions in the whole population; used
when there are no poor in the domain.
#-------------------------------------------------------------------------------------
----------------------------------------
prediction_gap_in_domain <- function(t,predicted,all_predictions)
{
if (min(predicted) > t)
    {
    predictions=all_predictions()
    print("all predictions")
    print(class(predictions))
    print(predictions)
    print("poor pred.")
    print(predictions[predictions <= t])
    }
else
    {
    predictions=predicted
    }
predictions_for_poor = predictions[predictions <= t]
print("poor pred.")
print(predictions_for_poor)
if (length(predictions_for_poor)==0)
    {
    0
    }
else
    {
    (t - median(predictions_for_poor)) / t
    }
}


#=============================================================================
# Subject: Estimator of the poverty gap based on predictions
#-------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                      The sample data set
# population                 The population data set
# model                         Model
# predicted                    Function of domain, returning domain's predictions from
the model
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                    Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#-------------------------------------------------------------------------------------
----------------------------------------
```

```
predictor_gap <- function(sample,population,model,predicted,reference,y,weight,domain)
{
backup <- function(){Reduce("c",sapply(all_similar_domains(domain,population),
predicted))}
prediction_gap_in_domain(poverty_line(reference,y,weight), predicted(domain), backup)
}


############################
#TESTS
############################


test_poverty_gap_estimator <- function()
{
data = test_s()
pop = test_U()

estimators <-
domain_estimators(direct_gap,data,pop,"y",NULL,NULL,list(),list(),list(),domain_variab
les=list("domain"),"w",reference=data)
expected1 = 0.6666666666666666
expected2 = 0.3333333333333333
est1 = estimators(dom1)
assert_equal(expected1, est1)
est2 = estimators(dom2)
assert_equal(expected2, est2)


y<- c(1,2,2,3,4, 1,4,5,5,6)
x<-  c(1,2,3,2,5, 1,4,3,5,6)
w<- c(2,3,2,3,2, 2,3,3,2,2)
domain<-c(1,1,1,1,1, 2,2,2,2,2)
s<-data.frame(y,x,w,domain)
model <-  lm(y~x, weights=w)
back <- identity
predicted = sapply(predict(model,newdata=pop),back)
print(predicted)
estimators <-
expanded_domain_predictors(predictor_gap,s,pop,"y",model,back,list(),list(),list(),dom
ain_variables=list("domain"),"w",reference=s,1:99)
assert_equal(c(0.510648116027541,0.23378992653710895), sapply(c(dom1,dom2),
estimators))

y<-data[["y"]]
x<-data[["x"]]
w<-data[["w"]]
model2 <-  lm(y~x, weights=w)
estimators <-
domain_estimators(predictor_gap,data,pop,"y",model2,back,list(),list(),list(),domain_v
ariables=list("domain"),"w",reference=data)
```

```
expected_syn1 = 0.1440507834663234
expected_syn2 = 0.07751191142773001
est1 = estimators(dom1)
assert_equal(expected_syn1, est1)
est2 = estimators(dom2)
assert_equal(expected_syn2, est2)


print("Poverty gap tests pass")

}
```

Code quintile_share.r

```
#Quintile share estimators for domains.
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#============================================================================
# Subject: Cutoff point of cumulative sums, largest index for which cumulative sum is
below or at cutoff point
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# v                 vector
# pct             percentage point (in [0,100])
#----------------------------------------------------------------------------------------
----------------------------------------
cutoff <- function(v,pct)
{
sum=0
target = (pct/100)*sum(v)
found=dim
idx=1
for(elem in v)
    {
    sum = sum + elem
    if (sum>target)
        {
        found=max(1,idx-1)
        break
        }
    idx=idx+1
    }
found
}


#============================================================================
# Subject: Cutoff point of cumulative sums from above, smallest  index for which
cumulative sum is below or at cutoff point,
# when cumulative sum is calculated starting from the largest index
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# v                 vector
# pct             percentage point (in [0,100])
#----------------------------------------------------------------------------------------
----------------------------------------
cutoff_above <- function(v,pct)
```

```
{
sum=0
found=0
target = (pct/100)*sum(v)
dim=length(v)
idx=dim
for(elem in rev(v))
    {
    sum = sum + elem
    if (sum>target)
        {
        found=min(dim,idx+1)
        break
        }
    idx=idx-1
    }
found
}

#==========================================================================
# Subject: Default estimator of the quintile share
#--------------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                     The sample data set
# population                The population data set
# model                        Model (not used)
# predicted                   Function, returns predictions in the domain (not used)
# reference                   Data set determining the poverty threshold (usually
whole sample)
# y                                  Name of the y variable
# weight                         Name of design weight variable
# domain                       Domain for which the estimate is calculated
#--------------------------------------------------------------------------------------------
----------------------------------------
direct_quintile_share <-
function(sample,population,model,predicted,reference,y,weight,domain)
{
if (is_empty_domain(domain,sample))
    return(NaN)

wgt = w_sd(sample,weight,domain)
yvec =  y_d(sample,y,domain)

w_sorted_by_y = wgt[order(yvec)]
y_sorted = sort(yvec)
q20 = cutoff(w_sorted_by_y,20)
q80 = cutoff_above(w_sorted_by_y,20)
dim = length(w_sorted_by_y)
s20 = sum(y_sorted[1:q20]*w_sorted_by_y[1:q20]) / sum(w_sorted_by_y[1:q20])
s80 = sum(y_sorted[q80:dim]*w_sorted_by_y[q80:dim]) / sum(w_sorted_by_y[q80:dim])
```

```
qs<-s20/s80
qs
}

#============================================================================
# Subject: Cutoff point of cumulative sums of constant weights
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# dim           Dimension of the weight vector
# pct             percentage point (in [0,100])
#----------------------------------------------------------------------------
----------------------------------------
cutoff_integer <- function(dim,pct)
{
max(1, as.integer( (pct/100)*dim))
}

#============================================================================
# Subject: Cutoff point from above of cumulative sums of constant weights,
# smallest  index for which cumulative sum is below or at cutoff point, calculated
from the largest index
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# dim           Dimension of the weight vector
# pct             percentage point (in [0,100])
#----------------------------------------------------------------------------
----------------------------------------
cutoff_integer_above <- function(dim,pct)
{
dim - cutoff_integer(dim,pct) + 1
}

#============================================================================
# Subject: Quintile share calculated from predictions
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# predicted     Vector of predictions in a domain
#----------------------------------------------------------------------------
----------------------------------------
prediction_quintile_share_in_domain <- function(predicted)
{
pred_sorted = sort(predicted)
dim = length(pred_sorted)
q20 = cutoff_integer(dim,20)
q80 = cutoff_integer_above(dim,20)
s20 = sum(pred_sorted[1:q20]) / q20
s80 = sum(pred_sorted[q80:dim]) / length(q80:dim)
s20/s80
```

```
}


#============================================================================
# Subject: Estimator of the quintile share based on predictions
#----------------------------------------------------------------------------------------
----------------------------------------
# INPUT
# sample                      The sample data set
# population                 The population data set
# model                         Model
# predicted                    Function, returns predictions in the domain
# reference                    Data set determining the poverty threshold (usually
whole sample)
# y                                  Name of the y variable
# weight                        Name of design weight variable
# domain                       Domain for which the estimate is calculated
#----------------------------------------------------------------------------------------
----------------------------------------
predictor_quintile_share <-
function(sample,population,model,predicted,reference,y,weight,domain)
{
prediction_quintile_share_in_domain(predicted(domain))
}


##################
#TESTS
##################


test_cutoff <- function()
{
w <- c(1, 2, 3, 4)
assert_equal(1, cutoff(w,1))
assert_equal(1, cutoff(w,20))
assert_equal(2, cutoff(w,30))
assert_equal(2, cutoff(w,59))
assert_equal(3, cutoff(w,60))
assert_equal(3, cutoff(w,61))
assert_equal(3, cutoff(w,99))

w2 <- c(1, 2, 3, 6, 3, 2,1)
assert_equal(2, cutoff(w2,20))


w <- c(4,3,2,1)
assert_equal(4, cutoff_above(w,1))
assert_equal(4, cutoff_above(w,20))
assert_equal(3, cutoff_above(w,30))
assert_equal(3, cutoff_above(w,59))
```

```
assert_equal(2, cutoff_above(w,60))
assert_equal(2, cutoff_above(w,61))
assert_equal(2, cutoff_above(w,99))
w2 <- c(1, 2, 3, 6, 3, 2,1)
assert_equal(6, cutoff_above(w2,20))


assert_equal(1, cutoff_integer(20,1))
assert_equal(1, cutoff_integer(20,9))
assert_equal(2, cutoff_integer(20,10))
assert_equal(2, cutoff_integer(20,11))
assert_equal(3, cutoff_integer(20,19))
assert_equal(4, cutoff_integer(20,20))

assert_equal(1, cutoff_integer_above(1,20))
assert_equal(2, cutoff_integer_above(2,20))
assert_equal(3, cutoff_integer_above(3,20))
assert_equal(4, cutoff_integer_above(4,20))
assert_equal(5, cutoff_integer_above(5,20))
assert_equal(6, cutoff_integer_above(6,20))
assert_equal(8, cutoff_integer_above(8,20))
assert_equal(9, cutoff_integer_above(9,20))
assert_equal(9, cutoff_integer_above(10,20))
assert_equal(10, cutoff_integer_above(11,20))

print("Cutoff tests pass")
}

test_quintile_share <- function()
{
data = test_s()
pop = test_U()

estimators <-
domain_estimators(direct_quintile_share,data,data,"y",NULL,NULL,list(),list(),list(),d
omain_variables=list("domain"),"w",data)
assert_equal(c(0,1/3), sapply(c(dom1,dom2),estimators))

x<-data[["x"]]
y<-data[["y"]]
w<-data[["w"]]
model <-  lm(y~x, weights=w)
back <- identity
estimators <-
domain_estimators(predictor_quintile_share,data,pop,"y",model,back,list(),list(),list(
),domain_variables=list("domain"),"w",data)
assert_equal(c(0.232351729725087,0.22053830413736536), sapply(c(dom1,dom2),
estimators))

y <- c(1,2,2,3,4, 1,4,5,5,6)
x <-  c(1,2,3,2,5, 1,4,3,5,6)
```

```
w <- c(2,3,2,3,2, 2,3,3,2,2)
domain<-c(1,1,1,1,1, 2,2,2,2,2)
s<-data.frame(y,x,w,domain)
model <-  lm(y~x, weights=w)
back <- identity
estimators <-
expanded_domain_predictors(predictor_quintile_share,s,pop,"y",model,back,list(),list()
,list(),domain_variables=list("domain"),"w",s,1:99)
assert_equal(c(0.2936645873962118,0.31338628666682744),
sapply(c(dom1,dom2),estimators))

print("Quintile share test pass")
}
```

Code calibrated_predictions.r

```
#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi


#Frequencies of predictions are estimated using known marginal population totals or
known marginal class frequencies
#of auxiliary variables in each domain.
#Reference: Section 3.4 in Deliverable D2.2.

#To use the calibration technique, use function domain_estimate_data (interface.r).

# The population data set should satisfy the following conditions:
# - Numeric variables determining the domain must be present in the unit-level
population.
# - Auxiliary variables used in calibration have one non-zero observation, equal to
the domain sum, in each domain
# For example, if domains are determined by "d"  and "x" is an auxiliary variable,
then required population contains variables
# d=c(1,1,1,2,2,2), x=c(0,0,4,0,0,5) assuming that population domain sums of x are 4
and 5 in domains 1 and 2, respectively.

#If the marginal frequencies of the classes of a qualitative variable are not known in
the population, they can be estimated by
#GREG assisted by a multinomial logistic model. Corresponding GREG estimator for a
quantitative variable is not implemented.

#Functional style of programming is adopted. When a function f is returned from a
function g, the body of function f may refer
#to local variables in function g and to the arguments of function g. The function f
"remembers" these local variables
#(by so-called "closure"), thus avoiding recalculation of the local variables.


library(nnet) #Required for imputing unknown marginal frequencies by GREG assisted by
a multinomial logistic model
#==========================================================================
#Creates a "parameter object" used to avoid long parameter lists. The return value is
used as argument "setup" in other functions.
#Elements are obtained by $: for example, sample is obtained from created setup by
setup$sample.
#Arguments:
#   y: name of the y variable
#   population: data.frame
#   sample: data.frame
#    x_list is the list of names of quantitative x-variables, such as list("age") or
list().
```

```
#    xq_list is the list of names of qualitative x-variables, such as
list("socstrat","lfs") or list().
#    unknown is the list of names of x-variables whose domain marginals are to be
estimated.
#   name_of_domains: name of the domain variable.
#   name_of_weights: name of the design weight variable in sample
#   back_transformation: function back-transforming predictions

create_setup <-
function(y,population,sample,x_list,xq_list,unknown,name_of_domains,name_of_weights,ba
ck_transformation)
    {
    list(y=y,population=population,sample=sample,x=x_list,xq=xq_list,unknown=unknown,

name_of_domains=name_of_domains,name_of_weights=name_of_weights,back_transformation=ba
ck_transformation)
    }


#=============================================================================
#Vector of domains in a data set.
domains <- function(setup, data)
    {
    data[[setup$name_of_domains]]
    }

#=============================================================================
#Vector of domains in the sample.
sample_domains <- function(setup)
    {
    domains(setup,setup$sample)
    }
#=============================================================================
#Vector of domains in the population.
population_domains <- function(setup)
    {
    domains(setup,setup$population)
    }

#=============================================================================
# Sample size
# If the argument domain if missing, this returns the sample size; if present, returns
the size of the domain in the sample
n <- function(setup,domain)
    {
    if (missing(domain))
        {
        dim(setup$sample)[1]
        }
    else
        {
```

```
        sum(sample_domains(setup)==domain)
        }
}

#=========================================================================
# Population size
# If the argument domain if missing, this returns the population size; if present,
returns the size of the domain in the population
N <- function(setup,domain)
    {
    if (missing(domain))
        {
        dim(setup$population)[1]
        }
    else
        {
        sum(population_domains(setup)==domain)
        }
}

#=========================================================================
#Vector of design weights in the sample.
weights <- function(setup)
    {
    setup$sample[[setup$name_of_weights]]
    }

#=========================================================================
#Vector of variable's values in a population domain
#Arguments
# name: name of the variable
values_in_population_domain <- function(setup,name,domain)
    {
    setup$population[[name]][domains(setup,setup$population)==domain]
    }

#=========================================================================
#Vector of variable's values in a sample domain
#Arguments
# name: name of the variable
values_in_sample_domain <- function(setup,name,domain)
    {
    setup$sample[[name]][sample_domains(setup)==domain]
    }

#=========================================================================
#Subvector of a vector in a sample domain
#Arguments
# vec: the vector
subvector_in_sample_domain <- function(setup,vec,domain)
    {
```

```
    vec[sample_domains(setup)==domain]
    }

#=============================================================================
#Vector of design weights in a sample domain.
domain_weights <- function(setup,domain)
    {
    subvector_in_sample_domain(setup,weights(setup),domain)
    }

#=============================================================================
#Subvector of a vector in a population domain
#Arguments
# vec: the vector
subvector_in_population_domain <- function(setup,vec,domain)
    {
    vec[population_domains(setup)==domain]
    }

#=============================================================================
#Elements of a list or vector, excluding the first element
rest <- function(coll)
    {
    len = length(coll)
    if (len >= 2)
        coll[2:len]
    else
        c()
    }


#=============================================================================
# Unique sorted values of a variable in whole sample; first value is excluded
rest_unique <- function(setup,variable)
    {
    rest(sort(unique(setup$sample[[variable]])))
    }

#=============================================================================
# Indicator variable in sample
indicator_in_sample<- function(setup,variable,value)
    {
    as(setup$sample[[variable]]==value,"numeric")
    }

#=============================================================================
# Indicator variables in sample, as columns of a matrix
indicators_in_sample<- function(setup,variable)
    {
    sapply(rest_unique(setup,variable),
function(value){indicator_in_sample(setup,variable,value)})
```

```
    }

#=========================================================================
# Indicator variable in population
indicator_in_population <- function(setup,variable,value)
    {
    as(setup$population[[variable]]==value,"numeric")
    }

#=========================================================================
# Indicator variables in population; only for values appearing in sample
indicators_in_population <- function(setup,variable)
    {
    sapply(rest_unique(setup,variable),
function(value){indicator_in_population(setup,variable,value)})
    }


#=========================================================================
#Returns TRUE if  list or vector contains given element
contains <- function(coll,value)
    {
    any(coll==value)
    }


#=========================================================================
#Applies given function (applied) with arguments (name,value), where value ranges over
the unique values of the variable in a sample domain,
# excluding the first.
#Arguments:
#   name: name of the qualitative variable
#   applied: a function with arguments (name,value)
#Returns:
# Suppose the variable name has m unique values in the domain: v[1],v[2],...,v[m].
# Then this function returns a list with elements applied(name, v[2]), applied(name,
v[3]),...,applied(name, v[m]).
apply_over_values_of_qualitative <- function(setup,name,domain,applied)
    {
    values <- rest(sort(unique(values_in_sample_domain(setup,name,domain))))
    lapply(values, function(v){applied(name,v)})
    }

#=========================================================================
#Returns true if column at given index already appears in the matrix.
equals_previous_column <- function(index,matrix)
    {
    for(idx in 1:index-1)
        {
        if (identical(matrix[,idx], matrix[,index]))
            return(TRUE);
```

```
        }
    return(FALSE);
    }

#=========================================================================
#Subset of list (or vector) coll corresponding to unique columns in a given matrix.
#Arguments:
#   matrix has length(coll) columns
#   coll is a list or vector of scalar elements.

for_unique_columns <- function(matrix,coll)
    {
    coll[Filter(function(idx){!equals_previous_column(idx,matrix)},1:length(coll))]
    }

#=========================================================================
#Returns the column matrix of x variables (indicators included) in a domain
domain_x_with_indicators <- function(setup,domain)
    {
    columns_of_qualitative <-
function(x){Reduce("cbind",apply_over_values_of_qualitative(setup,x,domain,function(na
me,value){indicator_in_sample(setup,name,value)[sample_domains(setup)==domain]}))}
    cbind(
        array(1,n(setup,domain)),
        sapply(setup$x,function(name){values_in_sample_domain(setup,name,domain)}),
        Reduce("cbind",lapply(setup$xq, columns_of_qualitative)))
    }

#=========================================================================
#Returns a vector of elements obtained by applying a given function to the names of
quantitative variables and
#names of indicators constructed from qualitative variables
#The name of the constant is number 1.
#The name of an indicator for qualitative variable "var" with var=value is a pair
c("var",value).
#Identical columns in the column matrix of x variables are avoided.
#Arguments:
#   applied is a function of a variable name. It has arguments name and value, where
value is allowed to be missing.
#Returns:
#   This function returns a list c(applied(1), applied(<name of first quantitative
variable>),...,applied(<name of last quantitative variable>),
#   applied(<name of first indicator>),..., applied(<name of last indicator>)).
#   Example: if setup$x=list("x","x2"), setup$xq=list("socstrat") and if socstrat has
values 1,2,3 in the sample domain, then
#   this function returns a list with elements applied(1), applied("x"),
applied("x2"), applied("socstrat",2), and applied("socstrat",3).

apply_over_x_variables <- function(setup,applied,domain)
    {
```

```
    values <-   c(applied(1),sapply(setup$x,applied),Reduce("c",lapply(setup$xq,
function(name){apply_over_values_of_qualitative(setup,name,domain,applied)})))
    for_unique_columns(domain_x_with_indicators(setup,domain),values)
    }

#=============================================================================
# Returns a list of elements in vector a not in list b.
# Example: if a=list("age","socstrat","lfs") and b=list("socstrat"), this function
returns list("age","lfs").
difference <- function(a,b)
    {
    Filter(function(x){!any(b==x)},a)
    }


#=============================================================================
#Value of a named variable in a vector of x-values, ordered by names c(1,x,xq).
#Arguments:
#   x: list of names of quantitative variables
#   xq: list of names of qualitative variables
#Returns:
#   Element of x_vector at position corresponding to the position of name in
c(1,x,xq).
value_in_vector <- function(name,x_vector,x,xq)
    {
    x_vector[Position(function(n){n==name}, c(1,x,xq))]
    }


#=============================================================================
#Expansion of a x-vector so that values of indicator variables for qualitative
variables are included. Indicators are not created for values
# that do not appear in the sample domain.
#The length of the return value may vary from a domain to another (depending on which
values of qualitative variables are missing from each domain).
#Arguments:
#   x_vector: vector of values of x-variables, in the order determined by
c(1,setup$x,setup$xq).
#Returns:
#   List of elements: 1, <values of quantitative variables>, <values of indicators for
qualitative variables>.
#   Example: if setup$x=list("x"), setup$xq=list("x2") and x2 has values 1, 2 and 3 in
the domain, then
#   x_expanded(setup,domain,c(1,2,2)) = c(1,2,1,0), where the third element (1)
corresponds to indicator x2=2 and
#   fourth element (0) corresponds to indicator x2=3; also, the return value for
x_vector=c(1,2,1) is c(1,2,0,0),
#   and the return value for x_vector=c(1,2,1) is c(1,2,0,1).

x_expanded <- function(setup,domain,x_vector)
    {
```

```
    expansion <- function(name,value)
        {
        value_of_x = value_in_vector(name,x_vector,setup$x,setup$xq)
        if (missing(value)) #Quantitative variable
            {
            value_of_x
            }
        else #Values of indicators are created for qualitative variable
            {
            if (value_of_x == value)
                1
            else
                0
            }
         }
    as.numeric(apply_over_x_variables(setup,expansion,domain))
    }


#=========================================================================
#Column matrix composed of x-variables (indicators are not created) in a domain.
domain_x <- function(setup,domain)
    {
    cbind(array(1,n(setup,domain)),
sapply(c(setup$x,setup$xq),function(name){values_in_sample_domain(setup,name,domain)})
)
    }

#=========================================================================
#Returns a function that returns the estimated (HT) frequency of a given x-vector.
#Returns:
#   Function; for example:  n <- n_hat(setup,domain); then n(xv) is the sum of weights
over observations with x=xv.
n_hat <- function(setup,domain)
    {
        x_d <- domain_x(setup,domain)
        weights_in_domain <- weights(setup)[sample_domains(setup)==domain]
        nd <- dim(x_d)[1]
        function(x_k)
            {
            equals_x_k <- function(k)
                {
                all(x_d[k,]==x_k)
                }
            sum(weights_in_domain[sapply(1:nd,equals_x_k)])
            }
    }


#=========================================================================
```

```
#Returns function that applies a function to the list of all distinct x-vectors in
domain, with arguments (x, n_hat(setup,domain)(x))
#for each x in the set of distinct x-vectors.
#Argument of returned function:
#   applied: a function with arguments x-vector, estimated frequency of the x-vector.
#Returns:
#   Function with a function argument. Example: if the unique x-vectors in the domain
are x[1], x[2],...,x[p], and n<-n_hat(setup,domain),
#   then apply_over_distinct_x(setup,domain)(applied) is a list with elements
applied(x[1],n(x[1])), applied(x[2],n(x[2])),...,applied(x[p],n(x[p])).

apply_over_distinct_x <- function(setup,domain)
    {
    distinct_x <- unique(domain_x(setup,domain))
    rows = dim(distinct_x)[1]
    n <- n_hat(setup,domain)
    ns <- sapply(1:rows,function(r){n(distinct_x[r,])}) #vector of n_hat-estimates
    function(applied)
        {
        lapply(1:rows, function(r){applied(distinct_x[r,],ns[r])})
        }
    }


#=========================================================================
#Returns function that applies a function to the list of all distinct expanded x-
vectors in domain, with arguments (x, n_hat(setup,domain)(x), <fitted value for x>)
#for each x in the set of distinct expanded x-vectors.
#Argument of returned function:
#   applied: a function with three arguments: expanded x-vector, estimated frequency
of the x-vector, and fitted value for the x-vector.
#Returns:
#  Function with a function argument. Example: if the unique expanded x-vectors in the
domain are x[1], x[2],...,x[p],  n<-n_hat(setup,domain),
# and fit(x) is the fitted value for x,   then
apply_over_expanded_distinct_x(setup,model,domain)(applied) is a list
# with elements applied(x[1],n(x[1]),fit(x[1])),
applied(x[2],n(x[2]),fit(x[2])),...,applied(x[p],n(x[p]),fit(x[p])).

apply_over_expanded_distinct_x <- function(setup,model,domain)
    {
    fitted_in_domain <-
subvector_in_sample_domain(setup,sapply(predict(model,newdata=setup$sample),setup$back
_transformation),domain)
    distinct_x <- unique(cbind(domain_x(setup,domain), fitted_in_domain)) #fitted
values are in the last column to aid mapping from x-vectors to fitted values.
    rows = dim(distinct_x)[1]
    cols = dim(distinct_x)[2]
    fitted_for_x <- distinct_x[,cols]
    distinct_x <- distinct_x[1:rows,1:cols-1] #Drop the fitted values from distinct_x
    n <- n_hat(setup,domain)
```

```
    ns <- sapply(1:rows,function(r){n(distinct_x[r,])}) #vector of n-estimates

    expanded_distinct_x <- Reduce("rbind",
lapply(1:rows,function(r){as.numeric(x_expanded(setup,domain,distinct_x[r,]))}))
    function(applied)
        {
        lapply(1:rows, function(r){applied(expanded_distinct_x[r,],ns[r],
fitted_for_x[r])})
        }
    }


#=============================================================================
# Row sums of a matrix
row_sums <- function(mat)
    {
    sapply(1:dim(mat)[1],function(k){sum(mat[k,])})
    }


#=============================================================================
#Class probabilities of a multinomial logistic model fitted to given qualitative
variable
#using known auxiliary variables as x-variables.
# Arguments:
# variable name of the variable not known in population
# Returns: A list of two functions, first returns vector of fitted values, that is,
estimated probabilities, for given value of the variable in a domain,
# the second function returns the population predictions for given value of the
variable in a domain.

predictions_of_multinomial_logistic_model <- function(setup,variable)
    {
        not_used <- c(variable,setup$unknown)
        used_x <- difference(setup$x,not_used)
        used_xq <- difference(setup$xq,not_used)
        x_matrix <- cbind(
                                    array(1,n(setup)),
                                    sapply(used_x,function(x){setup$sample[[x]]}),
                                    Reduce("cbind",lapply(used_xq,
function(x){indicators_in_sample(setup,x)})))
        colnames(x_matrix) <- paste("x",1:dim(x_matrix)[2], sep="")
        y <- setup$sample[[variable]]
        d <- data.frame(cbind(y, x_matrix))
        formula <- paste("y ~ ", paste(rest(colnames(x_matrix)), collapse="+"))
        mod <- multinom(formula,d)

        population_x_matrix <- cbind(
                                    array(1,N(setup)),

sapply(used_x,function(x){setup$population[[x]]}),
```

```
                                    Reduce("cbind",lapply(used_xq,
function(x){indicators_in_population(setup,x)})))

        distinct_values <-  rest_unique(setup,variable)
        index_of_value <- function(value){Position(function(x){x==value},
distinct_values)}
        coefficients_of_class <- function(value)
            {
            if (!contains(distinct_values,value))
                {
                array(0,dim(x_matrix)[2])
                }
            else
                {
                if (is.null(dim(coefficients(mod))))
                    {
                    coefficients(mod)
                    }
                else
                    {
                    coefficients(mod)[index_of_value(value),]
                    }
                }
            }

        log_odds_s <-  function(value,domain)
            {
            x_matrix[(sample_domains(setup)==domain),] %*%
coefficients_of_class(value)
            }

        log_odds_U <- function(value,domain)
            {
            population_x_matrix[(population_domains(setup)==domain),] %*%
coefficients_of_class(value)
            }

        probabilities <- function(f, value,domain) # f = log_odds_s or log_odds_U
            {
            log_odds <- f(value,domain)
            z <-  row_sums(cbind(array(1,length(log_odds)),
sapply(distinct_values,function(v){exp(f(v,domain))})))
            exp(log_odds)/z
            }

        list(
        function(value,domain)
            {
            probabilities(log_odds_s, value,domain)
            },
        function(value,domain)
```

```
            {
            probabilities(log_odds_U, value,domain)
            })
    }

#=========================================================================
#GREG estimate of the marginal frequency of a value of a qualitative variable.
#Arguments:
# model is the return value of predictions_of_multinomial_logistic_model

multinomial_logistic_greg <- function(setup,model,variable,value,domain)
    {
        y <-
subvector_in_sample_domain(setup,indicator_in_sample(setup,variable,value),domain)
        fitted <- model[[1]](value,domain)
        predictions <- model[[2]](value,domain)
        weighted_residuals <- domain_weights(setup,domain)*(y - fitted)
        sum(predictions) + sum(weighted_residuals)
    }

#=========================================================================
#Population totals of x-variables constructed from constant (1), quantitative
variables and indicators for qualitative variables
# (indicator must be have some non-zero values in the domain)
#Returns:
#   A numerical vector of population totals in the order of c(1,setup$x, <indicators
for qualitative variables>).
population_marginals <- function(setup,domain)
    {
       if (length(setup$unknown) > 0)
           {
           models_for_unknown <-
lapply(setup$unknown,function(x){predictions_of_multinomial_logistic_model(setup,x)})
           index_of_model <- function(name){Position(function(n){n==name},
setup$unknown)}
           }

        population_sum <- function(name,value)
            {
            if (identical(1,name))
                {
                return(sum(domains(setup,setup$population)==domain))
                }
           if (missing(value))
                {
                return(sum(values_in_population_domain(setup,name,domain)))
                }
           else #qualitative variable
                {
                if (contains(setup$unknown,name))
                    {
```

```
                    model <- models_for_unknown[[index_of_model(name)]]
                    greg <- multinomial_logistic_greg(setup,model,name,value,domain)
                    if (greg < 0)
                        {
                        return(0)
                        }
                     else
                        {
                        return(greg)
                        }
                    }
                else
                    {
                    return(sum(values_in_population_domain(setup,name,domain) ==
value))
                    }
                }
            }
        as.numeric(apply_over_x_variables(setup,population_sum,domain))
    }


#============================================================================
#Vector b in the estimation algorithm (Section 3.4 in Deliverable D2.2).
#Argument applyf is the function returned by apply_over_expanded_distinct_x for the
domain
b <- function(applyf)
    {
    Reduce("+",applyf(function(x,n,f) {n * x }))
    }

#============================================================================
#Matrix A in the algorithm (Section 3.4 in Deliverable D2.2).
#Argument applyf is the function returned by apply_over_expanded_distinct_x for the
domain
A <- function(applyf)
    {
    Reduce("+", applyf(function(x,n,f) {n * (x %o% x)}))
    }

#============================================================================
#Lagrange multiplier estimated using chi-square distance measure.
#Returns:
#   Result of iteration, or array of zeroes in the case of numerical failure; then the
estimated frequencies will be equal to HT estimates.
calibrated_lambda <- function(setup,model,domain)
    {
    totals <- population_marginals(setup,domain)
    applyf <- apply_over_expanded_distinct_x(setup,model,domain)
    result <- try(solve(A(applyf), (totals - b(applyf))))
    print("A")
```

```
    print(A(applyf))
    print("b")
    print(b(applyf))
    if (identical("try-error",result))
        {
        array(0,length(totals))
        }
    else
        {
       if (all(sapply(result,is.finite)))
            {
            result
            }
       else
            {
           array(0,length(totals))
            }
        }
    }
```

```
#=========================================================================
#Vector of predictions in a domain, obtained by calibration.
#For each distinct x-vector (x) in the sample, the corresponding fitted value is
cloned n(x) times, where n(x) is the rounded estimated frequency of x in population.
# Negative estimates of n(x) are replaced by 0.
#Arguments:
#   setup: created by create_setup
#   model:  R model. It must be possible to call predict(model,newdata=setup$sample).
#   domain: the value of the domain variable
#Returns:
#   A vector of cloned fitted values. The original estimated frequencies are scaled so
that their sum equals the domain size in population
# (rounding errors may cause some deviations).

calibrated_domain_predictions <- function(setup,model,domain)
    {
    fit <-
as.numeric(apply_over_expanded_distinct_x(setup,model,domain)(function(x,n,f){f}))
#Fitted values for distinct x-vectors in sample
    replace_neg <- function(x){if (x<0) 0 else x}
    lambda <- calibrated_lambda(setup,model,domain)
    print("lambda")
    print(lambda)
    calibrated_frequencies <-
apply_over_expanded_distinct_x(setup,model,domain)(function(x,n,f){replace_neg(n * (1
+ sum(x*lambda)))})
    print("calibrated fr")
    print(calibrated_frequencies)
```

```
    fr_sum <- sum(as.numeric(calibrated_frequencies))
    domain_size <- N(setup,domain)
    scaled_frequencies <-
sapply(calibrated_frequencies,function(n){n*(domain_size/fr_sum)}) #Scaled frequencies
sum upto population domain size
    as.numeric(Reduce("c", lapply(1:length(fit),
function(r){array(fit[r],round(scaled_frequencies[r]))})))
    }

#=========================================================================
# Frequency-calibrated predictor. Numeric variables determining the domain must be
present in the unit-level population.
# For each domain, the domain sums of auxiliary variables can be given in one
observation; then other
# domain observations contain zeroes. It is also possible to give unit-level
information about auxiliary
# variables. In all cases, this code calculates the domain sum of values of an
auxiliary variable by a sum over
# all observations belonging to the domain. This function assumes that variable
"domain__" does not appear in data sets.
# Returns a function of domain that returns the domain estimate.
# INPUT
# estimator_type           The estimator function, such as default_gini or greg_rate
# sample                        The sample data set
# population                   The population data set
# y                                    Name of the y variable
# model                           Model fitted to data by function lm, glm,  lme, or
nlme.  The model should yield predicted values by call of function predict.
# back_transformation Function back-transforming the predictions to the original
scale.
# x_list                             The list of names of quantitative x-variables,
such as list("age") or list().
# xq_list                           The list of names of qualitative x-variables, such
as list("socstrat","lfs") or list().
# unknown                       The list of names of x-variables not known in
population, such as list("socstrat") or list().
# domain_variables     List of names of variables defining the domains
# weight                           Name of design weight variable
# reference_set             Data set determining the poverty thresholds (usually whole
sample)
# percentages               Vector of percentage points (such as 1:99) used to compare
percentiles of y and predictions. Consider 1:50 for poverty gap, 1:99 for other
statistics.
#-------------------------------------------------------------------------------
----------------------------------------

calibrated_predictors <-
function(estimator_type,sample,population=sample,y,model,back_transformation,x_list,xq
_list,unknown,
    domain_variables,weight,reference_set=sample, percentages=1:99)
{
```

```
sample <- add_domain_variable(sample,"domain__",domain_variables)
population <- add_domain_variable(population,"domain__",domain_variables)
population_domains <- all_domains("",domain_variables,population) #NOTE: must be in
same order as indices in domain__.

setup <-
create_setup(y,population,sample,x_list,xq_list,unknown,"domain__",weight,back_transfo
rmation)
unique_domains <- unique(population[["domain__"]])

for (d in unique_domains)
    {
    print("predictions")
    print(calibrated_domain_predictions(setup,model,d))
    }

all_predictions <- Reduce("c", sapply(unique_domains,
function(d){calibrated_domain_predictions(setup,model,d)}))
domain__ <- Reduce("c", sapply(unique_domains,
function(d){array(d,length(calibrated_domain_predictions(setup,model,d)))}))
pred_data <- data.frame(domain__)
domain_list <- as.list(all_domains("",list("domain__"),population)) #Domain objects,
necessary for function log_expanded_predictions

expanded_predictions <-
log_expanded_predictions(sample,pred_data,all_predictions,y,weight,percentages,domain_
list) #Function of elements in domain_list

function(d)
    {
    domain_object <- domain_from_list(" ",
list("domain__"),list(as.numeric(Position(function(x){is_same_domain(x,d)},
population_domains))))
    estimate <-
estimator_type(sample,pred_data,model,expanded_predictions,reference_set,y,weight,doma
in_object)
    estimate
    }
}


############################################################
test_calibration <- function()
{

#helper function
nearly_equal <- function(vec1,vec2,tol)
    {
    diff = abs(as.numeric(vec1)-as.numeric(vec2))
    length(as.numeric(vec1))>0 & max(diff)<=tol
    }
```

```
#helper function
assert_equal <- function(vec1,vec2)
    {
    if (is.numeric(vec1) & is.numeric(vec2))
        {
        equals = nearly_equal(vec1,vec2,1e-6)
        }
    else
        {
        equals = identical(vec1,vec2)
        }
    if (!equals)
        {
        print("not equal:")
        print(vec1)
        print(vec2)
        stopifnot(equals)
        }
    }

country <- c(1,1,1,1,1,1,1,1,1)
     nuts <- c(1,1,1,1,2,2,3,3,3)
pop_nuts <- nuts
           x <- c(1,2,3,1,3,1,1,4,2)
          x2 <-c(3,1,1,2,2,4,1,3,3)
          x2b <- x2

population <- data.frame(country,nuts,x,x2,x2b)



country <- c(1,1,1,1,1,1)
nuts <- c(1,1,1,2,2,2)
y <- c(1,1,1,1,2,1)
x <- c(1,1,2,1,2,2)
x2 <- c(1,1,1,1,2,1)
x3 <- c(1,2,2,1,2,1)
w <- c(1.5,2.5,3.5,1.5,2.5,3.5)

sample <- data.frame(country,nuts,y,x,x2,w)

setup <-
create_setup("y",population,sample,list("x"),list("x2"),list(),"nuts","w",identity)

assert_equal(nuts,domains(setup, sample))
assert_equal(pop_nuts,domains(setup,population))
assert_equal(nuts,sample_domains(setup))
assert_equal(w,weights(setup))
assert_equal(c(1,2,3,1), values_in_population_domain(setup,"x",1))
assert_equal(c(2,4), values_in_population_domain(setup,"x2",2))
```

```
assert_equal(c(1,1,2), values_in_sample_domain(setup,"x",1))
assert_equal(c(1,2,1), values_in_sample_domain(setup,"x2",2))
assert_equal(c(1,2,3),subvector_in_sample_domain(setup,c(1,2,3,4,5,6),1))
assert_equal(c(4,5,6),subvector_in_sample_domain(setup,c(1,2,3,4,5,6),2))
assert_equal(c(2,3,4),rest(c(1,2,3,4)))
assert_equal(c(),rest(c(1)))

assert_equal(4,n_hat(setup,1)(c(1,1,1)))
assert_equal(3.5,n_hat(setup,1)(c(1,2,1)))
assert_equal(1.5,n_hat(setup,2)(c(1,1,1)))
assert_equal(2.5,n_hat(setup,2)(c(1,2,2)))
assert_equal(3.5,n_hat(setup,2)(c(1,2,1)))

model <- lm(y~x,weights=c(1.5,2.5,3.5,1.5,2.5,3.5))
assert_equal(c(4,7),population_marginals(setup,1))
assert_equal(c(2,4,1),population_marginals(setup,2))

assert_equal(c(-1.3571428571428583, 0.6071428571428579),
calibrated_lambda(setup,model,1))
assert_equal(c(1.0, 1.263157894736842, 1.263157894736842,
1.263157894736842),calibrated_domain_predictions(setup,model,1))

assert_equal(c(-1.2857142857142851, 0.2857142857142854 , 0.11428571428571423),
calibrated_lambda(setup,model,2))
assert_equal(c(1.263157894736842, 1.263157894736842),
calibrated_domain_predictions(setup,model,2))

###############
#Test with x-variable unknown in population

x2b <- c(1,1,1,1,2,3)
sample <- data.frame(country,nuts,y,x,x2b,x3,w)
setup <-
create_setup("y",population,sample,list("x"),list("x2b","x3"),list("x3"),"nuts","w",id
entity) #x3 not known in population
assert_equal(6,n(setup))
assert_equal(3,n(setup,1))
assert_equal(9,N(setup))
assert_equal(4,N(setup,1))
assert_equal(2,N(setup,2))
assert_equal(3,N(setup,3))
assert_equal(c(1.5,2.5,3.5),domain_weights(setup,1))
assert_equal(c(1.5,2.5,3.5),domain_weights(setup,2))
assert_equal(TRUE,contains(c(1,2,3),1))
assert_equal(FALSE,contains(c(1,2,3),1.5))
assert_equal(FALSE,contains(c(1,2,3),0))
assert_equal(FALSE,contains(c(1,2,3),4))
assert_equal(c(2,3),rest_unique(setup,"x2b"))
assert_equal(c(1,1,0,1,0,0),indicator_in_sample(setup,"x",1))
assert_equal(c(0,0,1,0,1,1),indicator_in_sample(setup,"x",2))
assert_equal(cbind(c(0,0,0,0,1,0),c(0,0,0,0,0,1)),indicators_in_sample(setup,"x2b"))
```

```
assert_equal(c(1,2,3),row_sums(cbind(c(0,1,2),c(1,0,1),c(0,1,0))))
fit <- predictions_of_multinomial_logistic_model(setup,"x3")[[1]]
assert_equal(c(0.3333742,0.3333742,0.9999049),fit(2,1))
assert_equal(c(1,1,1),fit(2,1) + fit(1,1))
predicted <- predictions_of_multinomial_logistic_model(setup,"x3")[[2]]
assert_equal(c(1,0.3333742),predicted(2,2))
assert_equal(c(1,1),predicted(1,2) + predicted(2,2))
assert_equal(c(0.3333742,0.9998627,1.65e-5),predicted(2,3))
assert_equal(c(1,1,1),predicted(1,3) + predicted(2,3))
model <- lm(y~x+x2+x3,weights=c(1.5,2.5,3.5,1.5,2.5,3.5))
assert_equal(c(-1.5015326,  0.3905583,  0.5775588), calibrated_lambda(setup,model,1))

##########

x3b <- c(1,2,3,1,2,1) #The unknown variable has three values in domain 1.
sample <- data.frame(country,nuts,y,x,x2b,x3b,w)
setup <-
create_setup("y",population,sample,list("x"),list("x2b","x3b"),list("x3b"),"nuts","w",
identity) #x3b not known in population
fit <- predictions_of_multinomial_logistic_model(setup,"x3b")[[1]]
assert_equal(c(6.666170e-01,6.666170e-01,9.019243e-05),fit(1,1))
assert_equal(c(3.333401e-01,3.333401e-01,1.182743e-06),fit(2,1))
assert_equal(c(4.288247e-05, 4.288247e-05, 9.999086e-01),fit(3,1))
assert_equal(2.166636,multinomial_logistic_greg(setup,predictions_of_multinomial_logis
tic_model(setup,"x3b"),"x3b",2,1))
assert_equal(2.000057,multinomial_logistic_greg(setup,predictions_of_multinomial_logis
tic_model(setup,"x3b"),"x3b",3,1))
assert_equal(c(4.000000, 7.000000, 2.166636, 2.000057), population_marginals(setup,1))

##################
# x2b = x3b in domain 2
x2b <- c(1,1,1,1,2,3,2)
x3b <- c(1,2,3,1,2,3,2)
country <- c(1,1,1,1,1,1,1)
nuts <- c(1,1,1,2,2,2,2)
y <- c(1,1,1,1,2,1,2)
x <- c(1,1,2,1,2,2,3)
w <- c(1,1,1,1,1,1,1)
s <- data.frame(country,nuts,y,x,x2b,x3b,w)
model <- lm(y~x+x2b+x3b)
setup <-
create_setup("y",population,s,list("x"),list("x2b","x3b"),list("x3b"),"nuts","w",ident
ity) #x3b not known in population

overs <- apply_over_x_variables(setup, function(x,v){if (missing(v)) x else c(x,v)},1)
assert_equal(4,length(overs))
assert_equal(1, overs[[1]])
assert_equal("x", overs[[2]])
assert_equal(as.character(c("x3b","2")), overs[[3]])
assert_equal(as.character(c("x3b","3")), overs[[4]])
# x2b does not appear because it is constant (1)
```

```
overs2 <- apply_over_x_variables(setup, function(x,v){if (missing(v)) x else
c(x,v)},2)
assert_equal(4,length(overs2))
assert_equal(1, overs2[[1]])
assert_equal("x", overs2[[2]])
assert_equal(as.character(c("x2b","2")), overs2[[3]])
assert_equal(as.character(c("x2b","3")), overs2[[4]])
# indicators for x3b do not appear because they are identical with corresponding x2b
indicators

assert_equal(c(0,0,0,0),calibrated_lambda(setup,model,1))
assert_equal(c(-1,1,-2,-2),calibrated_lambda(setup,model,2))


x2b <- c(1,2,2,1,2,3,2)
x3b <- c(1,2,3,1,2,3,2)
s <- data.frame(country,nuts,y,x,x2b,x3b,w)
setup <-
create_setup("y",population,s,list("x"),list("x2b","x3b"),list("x3b"),"nuts","w",ident
ity)

overs <- apply_over_x_variables(setup, function(x,v){if (missing(v)) x else c(x,v)},1)
assert_equal(5,length(overs))
assert_equal(1, overs[[1]])
assert_equal("x", overs[[2]])
assert_equal(as.character(c("x2b","2")), overs[[3]])
assert_equal(as.character(c("x3b","2")), overs[[4]])
assert_equal(as.character(c("x3b","3")), overs[[5]])

y <-  c(1,2,2,3,4, 1,4,5,5,6)
x <- c( 1,2,3,2,5, 1,4,3,5,6)
x2 <- c(1,1,2,2,2, 1,2,1,1,2)
w <-  c( 2,3,2,3,2, 2,3,3,2,2)
domain <- c(1,1,1,1,1, 2,2,2,2,2)
s <- data.frame(y,x,x2,w,domain)
pop <- test_U()
pop[["x2"]] = c(2,1,2,1,2,1,2,2,1,1,1,2,1,2,2,1,2,1,1,2,2,2,1,2,2)
model <- lm(y~x,weights=w)
assert_equal(c(0.35612292764193426,0.4749468613475171),
sapply(c(dom1,dom2),calibrated_predictors(predictor_quintile_share,s,pop,"y",model,ide
ntity,list("x"),list(),list(),list("domain"),"w",s,1:99)))

fx <- factor(x2)
model <- lm(y~fx,weights=w)
assert_equal(c(0.21309565038568, 0.1020643620248225),
sapply(c(dom1,dom2),calibrated_predictors(predictor_gini,s,pop,"y",model,identity,list
(),list("x2"),list(),list("domain"),"w",s,1:99)))

print("Calibration tests pass")
}
```

Code `interface.r`

```
#Interface for generation of domain estimates as a data frame.

#Ameli WP2, Domain estimators Version 2.0
#author: Ari Veijanen ari.veijanen@pp1.inet.fi

#============================================================================
# Subject: Returns TRUE if given string contains a substring. Uses grep.
# INPUT
# str       String
# sub     Regular expression searched in str
#-----------------------------------------------------------------------------------------
---------------------------------------
contains_string <- function(str,sub)
    {
    if (length(grep(sub, str)) == 1)
        TRUE
    else
        FALSE
    }


#============================================================================
# Subject: Returns a function that finds which of alternatives appears in a string
finder <- function(alternatives)
    {
    function(str)
        {
        Find(function(part){contains_string(str,part)},alternatives)
        }
    }

#============================================================================
# Subject: Get names of estimator functions and modifications (expanded, calibrated)
of predictors from a string.
# Estimator is recognized by words appearing in the description. The Laeken indicator
is identified by "poverty rate", "gini", "quintile share", or "poverty gap".
# Type of estimator is "direct" for the default estimator, "predictor" for predictor,
"ebp" for EBP, and "greg" for GREG.
# A predictor can be modified by "calibrated" or "expanded".
# These words can be in any order, as long as no extra words appear between the words
in  "poverty rate", "quintile share" and "poverty gap".
# A composite estimator is recognized by "+" between the unbiased component and the
predictor component; the unbiased component is given first.
# Examples: "direct poverty rate" yields the direct (default) poverty rate estimator,
# "expanded gini predictor" yields an expanded version of predictor for estimation of
the Gini coefficient, and
```

```
# "direct gini + expanded gini predictor" yields a composite estimator of Gini
coefficient incorporating an expanded predictor.
# INPUT
# description       String containing the definition of an estimator.
# RETURNS
# If not composite, returns list(<type: "expanded" or "calibrated" or "plain">, <name
of estimator function>) (2 elements).
# If composite, returns list("plain", <name of unbiased estimator>, <type of
predictor: "expanded" or "calibrated" or "plain">, <name of predictor function>) (4
elements).
#-------------------------------------------------------------------------------------
----------------------------------------
get_estimator <- function(description)
    {
    identify_indicator <- finder(list("poverty rate","poverty gap","gini","quintile
share"))
    identify_estimator <- finder(list("direct","predictor","greg","ebp"))
    identify_modifier <- finder(list("expanded","calibrated"))

    indicator = identify_indicator(description)
    if (indicator=="quintile share")
        indicator="quintile_share"
    if (indicator=="poverty rate")
        indicator="rate"
    if (indicator=="poverty gap")
        indicator="gap"

    is_composite = contains_string(description,"\\+")
    if (is_composite)
        {
        parts = strsplit(description,"\\+")[[1]]
        c(get_estimator(parts[[1]]),get_estimator(parts[[2]]))
        }
    else
        {
        type = identify_estimator(description)
        modifier = identify_modifier(description)
        if (is.null(modifier))
            modifier="plain"
        estimator <- paste(type,indicator,sep="_")
        list(modifier,estimator)
        }
    }


#=========================================================================
# Subject: Get the domain estimator handling missing values of direct domain
estimator.
# Estimator is recognized as in function get_estimator.
# INPUT
```

```
# estimator       String containing the definition of the full estimator, contains
also the name of the Laeken indicator
# handler              Type of the estimator handling missing values: "expanded
predictor" or "calibrated predictor"; Laeken indicator not included
# Rest of the arguments are as in function domain_estimate_data
# RETURNS
# A function of domain that returns a domain estimate in the domain of the sample
#-------------------------------------------------------------------------------
----------------------------------------
get_missing_handler <-
function(estimator,handler,sample,population,y,model,back_transformation,x_list=list()
,xq_list=list(),unknown=list(),domain_variables=list(),weight,reference_set=sample,
percentages=1:99)
    {
    identify_indicator <- finder(list("poverty rate","poverty gap","gini","quintile
share"))
    identify_estimator <- finder(list("direct","predictor","greg","ebp"))
    identify_modifier <- finder(list("expanded","calibrated"))

    indicator = identify_indicator(estimator)
    if (indicator=="quintile share")
        indicator="quintile_share"
    if (indicator=="poverty rate")
        indicator="rate"
    if (indicator=="poverty gap")
        indicator="gap"

        type = identify_estimator(handler)
        modifier = identify_modifier(handler)
        estimator <- get(paste(type,indicator,sep="_"))
        f <- estimator_modifier(modifier)

f(estimator,sample=sample,population=population,y,model,back_transformation,x_list,xq_
list,unknown,domain_variables,weight,reference_set,percentages)
    }

#============================================================================
# Subject: Name of a method used in function domain_estimate_data
#-------------------------------------------------------------------------------
----------------------------------------
# INPUT
# modifier                      "expanded" or "calibrated"
# method                       The estimator function, such as ht_rate
#-------------------------------------------------------------------------------
----------------------------------------
name_of_method <- function(modifier,method)
{
if (modifier=="" || modifier=="plain")
    return(method)
else
    return(paste(modifier,method,sep="_"))
```

```
}

#============================================================================
# Subject: Name of a method used in function domain_estimate_data
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# modifier                          values "plain", "expanded" or "calibrated"
# method1                        The unbiased estimator function, such as ht_rate
# method2                        The predictor, such as syn_rate
#----------------------------------------------------------------------------
----------------------------------------
name_of_composite <- function(modifier,method1,method2)
{
return(paste("composite", method1,"with",name_of_method(modifier,method2),sep="_"))
}



#============================================================================
# Subject: Create a data set with domains as rows and domain estimates as columns.
# All domains in the crosstabulation of given variables (argument domain_variables)
are included.
#----------------------------------------------------------------------------
----------------------------------------
# INPUT
# estimator_descriptions  A list of estimator descriptions, containing strings such as
"direct poverty gap", "direct poverty gap + expanded poverty gap predictor",
#                                                 Estimator is recognized by words
appearing in the description. The Laeken indicator is identified by "poverty rate",
"gini", "quintile share", or "poverty gap".
#                                                 Type of estimator is "direct" for the
default estimator, "predictor" for predictor, "ebp" for EBP, and "greg" for GREG.
#                                                 A predictor can be modified by
"calibrated" or "expanded".
#                                                 These words can be in any order, as
long as no extra words appear between the words in  "poverty rate", "quintile share"
and "poverty gap".
#                                                 A composite estimator is recognized by
"+" between the unbiased component and the predictor component; the unbiased component
is given first.
#                                                 Examples: "direct poverty rate" yields
the direct (default) poverty rate estimator,
#                                                 "expanded gini predictor" yields an
expanded version of predictor for estimation of the Gini coefficient, and
#                                                 "direct gini + expanded gini
predictor" yields a composite estimator of Gini coefficient incorporating an expanded
predictor.
# handler                                 Type of the estimator handling missing values
of estimators: "expanded predictor" or "calibrated predictor"; Laeken indicator not
included
```

```
# sample                                The sample data set
# population                           The population data set
# y                                             Name of the y variable
# model                                   Model fitted to data by function lm, glm,
lme, or nlme.  The model should yield predicted values by call of function predict.
# back_transformation       Function back-transforming the predictions to the original
scale.
# x_list                                    The list of names of quantitative x-
variables, such as list("age") or list(). Required only for n-calibrated predictors.
# xq_list                                  The list of names of qualitative x-
variables, such as list("socstrat","lfs") or list(). Required only for n-calibrated
predictors.
# unknown                              The list of names of x-variables whose marginal
domain totals are not known in population, such as list("socstrat") or list().
Required only for n-calibrated predictors.
# domain_variables           List of names of variables defining the domains
# weight                                   Name of design weight variable
# reference_set                   Data set determining the poverty threshold (usually
whole sample)
# percentages                     Vector of percentage points (such as 1:99) used in
expanded predictors to compare percentiles of y and predictions. Consider 1:50 for
poverty gap, 1:99 for other statistics.
#-------------------------------------------------------------------------------
----------------------------------------
domain_estimate_data <-
function(estimator_descriptions,handler,sample,population=sample,y,model,back_transfor
mation,x_list=list(),xq_list=list(),unknown=list(),domain_variables=list(),weight,refe
rence_set=sample, percentages=1:99)
{
domains <-  all_domains("",domain_variables,population)
data = data_of_distinct_domains("",domain_variables,population)
original_names <- names(data)
for (description in estimator_descriptions)
    {
    interpreted = get_estimator(description)
    missing_handler <-
get_missing_handler(description,handler,sample,population,y,model,back_transformation,
x_list,xq_list,unknown,domain_variables,weight,reference_set, percentages)

    if (length(interpreted)==4) #Composite
        {
        unbiased = get(interpreted[[2]])
        modifier = interpreted[[3]]
        f <- estimator_modifier(modifier)
        predictor <-
f(get(interpreted[[4]]),sample=sample,population=population,y,model,back_transformatio
n,x_list,xq_list,unknown,domain_variables,weight,reference_set,percentages)
        estimators <-
composite_estimators(unbiased,predictor,sample=sample,population=population,y,model,ba
ck_transformation,x_list,xq_list,unknown,domain_variables,weight,reference_set,percent
ages,missing_handler)
```

```
        name <- name_of_composite(modifier,interpreted[[2]],interpreted[[4]])
        }
    else
        {
        modifier = interpreted[[1]]
        estimator = get(interpreted[[2]])
        f <- estimator_modifier(modifier)
        estimators <-
f(estimator,sample=sample,population=population,y,model,back_transformation,x_list,xq_
list,unknown,domain_variables,weight,reference_set,percentages)
        name <- name_of_method(modifier,interpreted[[2]])
        }
    domain_estimate <- function(domain)
        {
        est <- estimators(domain)
        if (is.nan(est))
            missing_handler(domain)
        else
            est
        }
    estimates = sapply(domains,domain_estimate)
    data[[name]] = estimates
    }

data
}


############################
#TESTS
############################

test_interface <- function()
{
y = c(1,2,2,3,4, 1,4,5,5,6)
x = c(1,1,2,2,5, 1,3,4,5,6)
w= c(2,3,2,3,2, 2,3,3,2,2)
domain = c(1,1,1,1,1, 2,2,2,2,2)
data = data.frame(y,w,x,domain)
y_ind = create_poverty_indicator(data,"y","w",data)
data[["ind"]] = y_ind


x = c(1,1,3,4,3,5, 1,2,4,2,4,5)
domain = c(1,1,1,2,2,2,1,1,1,2,2,2)
pop = data.frame(x,domain)

dom1 = new("Domain",region="domain",
variables=list("domain"),
values = list(1)
)
```

```
dom2 = new("Domain",region="domain",
variables=list("domain"),
values = list(2)
)

x = data[["x"]]
model <- lm(y_ind ~ x)
back_transformation=identity

estimator_data <- domain_estimate_data(list("direct poverty rate + greg poverty
rate"),
"greg",sample=data,population=pop,y="y",model,back_transformation,domain_variables=lis
t("domain"),weight="w",reference_set=data)
print("Composite: ht-rate, greg-rate")
print(estimator_data)
stopifnot(identical(c("domain", "composite_direct_rate_with_greg_rate"),
names(estimator_data)))
assert_nearly_equal(c(2.462841,
4.350582),estimator_data[["composite_direct_rate_with_greg_rate"]], 0.05)


x = data[["x"]]
logy <- logp(1)(data[["y"]])
model <- lm(logy ~ x)
back_transformation=expm(1)
qs <-
expanded_domain_predictors(predictor_quintile_share,data,pop,"y",model,expm(1),list(),
list(),list(),domain_variables=list("domain"),weight="w",reference_set=data,1:99)

estimator_data <- domain_estimate_data(list("direct quintile share", "expanded
quintile share predictor"), "predictor",
sample=data,population=pop,y="y",model,back_transformation,domain_variables=list("doma
in"),weight="w",reference_set=data,1:99)
print("Separate: Quintile share by default and expanded predictor")
print(estimator_data)
stopifnot(identical(c("domain", "direct_quintile_share",
"expanded_predictor_quintile_share"), names(estimator_data)))
assert_equal(c(1,2),estimator_data[["domain"]])
assert_equal(c(0.2500000, 0.1666667),estimator_data[["direct_quintile_share"]])
assert_equal(c(qs(dom1),qs(dom2)),estimator_data[["expanded_predictor_quintile_share"]
])


model <- lm(y ~ x,weights=w)
back_transformation=identity
pop <- test_U()
pred_est <-
calibrated_predictors(predictor_quintile_share,data,pop,"y",model,back_transformation,
list(),list("x"),list(),domain_variables=list("domain"),weight="w",reference_set=data,
1:99)
```

```
estimator_data <- domain_estimate_data(list("direct quintile share", "calibrated
quintile share predictor", "direct quintile share + expanded quintile share
predictor"), "expanded predictor",
sample=data,population=pop,y="y",model,back_transformation,x_list=list(),xq_list=list(
"x"),unknown=list(),domain_variables=list("domain"),weight="w",reference_set=data,1:99
)
print("Quintile share by composite of default estimator and expanded predictor")
print(estimator_data)
stopifnot(identical(c("domain", "direct_quintile_share",
"calibrated_predictor_quintile_share",
"composite_direct_quintile_share_with_expanded_predictor_quintile_share"),
names(estimator_data)))
assert_equal(c(1,2),estimator_data[["domain"]])
assert_equal(c(0.2500000, 0.1666667),estimator_data[["direct_quintile_share"]])
assert_equal(c(pred_est(dom1), pred_est(dom2)),
estimator_data[["calibrated_predictor_quintile_share"]])
assert_nearly_equal(c(0.2943663,
0.3127468),estimator_data[["composite_direct_quintile_share_with_expanded_predictor_qu
intile_share"]], 0.05)


estimator_data <- domain_estimate_data(list("direct quintile share + calibrated
quintile share predictor"), "expanded predictor",
sample=data,population=pop,y="y",model,back_transformation,x_list=list("x"),xq_list=li
st(),unknown=list(),domain_variables=list("domain"),weight="w",reference_set=data,1:99
)
print("Quintile share by composite of default estimator and calibrated predictor")
print(estimator_data)
stopifnot(identical(c("domain",
"composite_direct_quintile_share_with_calibrated_predictor_quintile_share"),
names(estimator_data)))
assert_equal(c(1,2),estimator_data[["domain"]])
assert_nearly_equal(c(0.3251372,0.4361961),estimator_data[["composite_direct_quintile_
share_with_calibrated_predictor_quintile_share"]],0.05)

print("Interface tested")
}
```

References

Lehtonen R., Veijanen, A., Myrskylä, M. and **Valaste, M. (2011)**: *Small Area Estimation of Indicators on Poverty and Social Exclusion.*
Research Project Report WP2 – D2.2, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net