```r
conteo <- function(datos_j,N_j){
#
#   This function counts the number of individuals in \eqn{\mathcal{S}_j} and
#   \eqn{widetilde{\mathcal{S}}_j}, with respect to the total number of individuals
#   in a given planned domain \eqn{j}.
#
#   Input:
#       "datos_j"   -   This object should contain two columns labelled: "n_i" and "domplan".
#                       "n_i"     - Number of individuals in the ith group of individuals, and
#                       "domplan" - Column vector with categories for the planned domains
#       "N_j"       -   This object should include two columns labelled: "N_j" and "domplan".
#                       "N_j"     - Number of individuals in the jth planned domain, and
#                       "domplan" - Column vector with categories for the planned domains
#
#   Output:
#       M           -   Number of groups in "S_j"
#       CardS       -   Number of individuals in "S_j" (if "n_i"=1 for any "i", then Cards=M=nrow(datos_j))
#       CardNoS     -   Number of individuals out of sample "Stilde_j"
#

#    Counting the number of individuals in the sample
CardS <- as.matrix(sum(datos_j[,"n_i"]))
colnames(CardS) <- c("CardS")

#    Counting the number of individuals outside the sample
CardNoS <- as.matrix(N_j - CardS)
colnames(CardNoS) <- c("CardNoS")

#   Number of groups in the sample
M <- as.matrix(nrow(datos_j))
colnames(M) <- c("M")

#   Output
salida <- list(M,CardS,CardNoS)
return(salida)

#
#  --  END of conteo --
}

domnoplan <- function(datos,datos_ant,domplan_N,alphaDP,colid_D,alpha_D,inter,part,nSim){
#
#   Generates Monte Carlo samples of the predictive distribution of totals of a finite population
#   segmented in planned and unplanned domains, along with simulations of the predictive distribution
#   for the composition of the population between the unplanned domains.
#
#   Input:
#       datos       -   (Mxp)-dimensional array with positive entries for S_j
#       datos_ant   -   (Mxp)-dimensional reference array for calibration of G_0
#       domplan_N   -   Matrix array with counts of individuals in each planned domain
#       alphaDP     -   J-dimensional array with positive entries for the parameters of the Dirichlet process for F_j (with J being the number of planned domains)
#       colid_D     -   D-dimensional matrix array with the columns in ´datos´ that correspond to the indicator variables of the planned domains (those indicator variables represent a partion of 'datos')
#       alpha_D     -   (JxD)-dimensional array with positive entries for the parameters of the multinomial-Dirichlet component for the composition across unplanned domains.
#                       Note: Each one of the J rows is a vector of composition for P_j divided across the D unplanned domains
#       inter       -   Tuning parameter for model comparison and selection (related to calibration of G_0)
#       part        -   Number of partitions for predictive cross-validation (related to calibration of G_0)
#       nSim        -   Number of Monte Carlo simulated replicates of the predictive distribution
#
#   Details:
#           - datos : Represents the data sample of the target population, unplanned domains labelled.
#                       It should contain the following columns:
#                           "domplan" - Planned domains categories.
#                           "y_i"     - Actual measurements of individual positive and the sample group.
#                           "n_i"     - Number of members in the group (if the unit of observation in the sample are individuals, then "n_i" must be equal to 1)
#
#           - datos_ant : Represents the data reference used to calibrate G_{j0}
#                       The data must be labelled by domains planned. It should contain the following columns:
#                           "domplan" - Planned categories of domains
#                           "y_i"     - Positive real and individual measurements of each group in the sample (when the units of observation are the groups, "y_i" should be per capita measurement)
#
#           - domplan_N : Represents counts (or reference population) of the target population, divided by the planned domains.
#                       Tagged data must be labelled by domains planned. It should contain the following columns:
#                           "domplan" - Planned domains categories
#                           "N_j"     - Number of individuals in each population planned domain
#
#   Output:
#       total_domnoplan_sim     -   Matrix array of dimension "J x (3 + 2*D) x nSim" with predictions for relevant quantities of planned and unplanned domains
#                                   Column 1 - Indicator of the planned domains
#                                   Column 2 - T_j (totals of the planned domains)
#                                   Column 3 - N_j (composition of the planned domains)
#                                   Column 4 to (4+D-1) - T^d_j (totals of unplanned domains, such that T_j = sum  T^d_j (over d))
#                                   Column (4+D) to (3 + 2*D) - N^d_j (composition of
unplanned domains, such that N_j = sum  N^d_j (over d))
#

#      Consulted domains
cualdomplan <- as.matrix(domplan_N[,"domplan"])
J <- length(cualdomplan)

#   Size unplanned domains
D <- length(colid_D)

#       Repository totals in "unplanned domains" with three categories
total_domnoplan_sim <- array(NaN,c(J,3+2*D,nSim))

#----------------------------
#   validation
#----------------------------

#   A.  Parameter alphaDP
if(any(alphaDP <= (0*alphaDP))){
    stop("Error in the specification of 'alphaDP'!!!")}

#   B.  Parameters alpha_D and colid_D
if(ncol(alpha_D) != ncol(colid_D)){
    stop("The dimenciones of 'alpha_D' and 'colid_D' are different!!!")}

#   C.  parameter alpha_D
if(any(alpha_D <= (0*alpha_D))){
    stop("Error in the specification of 'alpha_D'!!!")}

#   D.  Parameter nSim
if(nSim <= 0){
    stop("Error in the specification of 'nSim'!!!")}

#   E.  Parameter inter
if(inter <= 0){
    stop("Error in the specification of 'inter'!!!")}

#   F. Total domains unplanned
if(sum(datos[,colid_D]) != nrow(datos)) {
   stop("Error in total of unplanned domains")}

if(any(rowSums(datos[,colid_D]) != 1)) {
   stop("Error in total of unplanned domains")}

#   ----------------------------------------------
#   Scanning planned domains
#   ----------------------------------------------
j <- 1
for(j in 1:J){
    #   Extraction of indexes in "datos_ant"
    P_j_ant <- which(datos_ant[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "datos_ant"
    datos_j_ant <- datos_ant[P_j_ant, ]

```

```r
    #   Extraction of indexes in "data" (i.e. sample S_j)
    P_j <- which(datos[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    datos_j <- datos[P_j, ]

    #   Population size in \mathcal{P}_{j}
    P_j <- which(domplan_N[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    N_j <- domplan_N[P_j,"N_j"]

    #   domain name the planned arrangement and constant data impute "n_j"
    sim <- 1
    for(sim in 1:nSim){
        total_domnoplan_sim[j,1,sim] <- cualdomplan[j]
        total_domnoplan_sim[j,3,sim] <- N_j
        }

    #   ``conteo''
    conteo_sal <- conteo(datos_j,N_j)

    #   Estimating the size of the population out of the sample (in "Stil_j")
    N_S_j <- conteo_sal[[2]]

    #   Estimating the size of the population out of the sample (in "Stil_j")
    N_Stil_j <- conteo_sal[[3]]

    #   Simulating of the composition of "N_Stil_j" between domains unplanned
    domnoplan_composicion_sim <- domnoplan_composicion(datos_j,N_j,N_S_j,N_Stil_j,
    colid_D,alpha_D[j,],nSim)

    #   Simulating data"N_j" distributed in "domnoplan1"
    N_S_domnoplan_j <- domnoplan_composicion_sim[[1]]
    N_Stil_domnoplan_j_sim <- domnoplan_composicion_sim[[2]]
    total_domnoplan_sim[j,c((3+D+1):(3+2*D)),] <- matrix(t(N_S_domnoplan_j),D,nSim) +
    N_Stil_domnoplan_j_sim

    #   ``unicstar''
    unicstar_sal <- unicstar(datos_j)
    ystar <- unicstar_sal[[2]][,"ystar"]

    #   ``pesos''
    pesos_sal <- pesosDP(unicstar_sal,alphaDP[j])
    rho <- pesos_sal[[1]]
    phi <- pesos_sal[[2]]

    #   ``g0_licitacion''
    g0_licitacion_sal <- g0_licitacion(datos_j_ant,inter,part)
    #g0_licitacion_sal <- g0_licitacion(datos_j_ant,inter)

    #   Simulating the composition of "T_Stil_j" between domains unplanned
    domnoplan_totalcomp_sim <- domnoplan_totalcomp(datos_j,rho,ystar,phi,
    g0_licitacion_sal,N_Stil_domnoplan_j_sim,nSim,colid_D)

    #   Simulating data grouped "n_j" distributed "domnoplan1"
    T_S_domnoplan_j <- domnoplan_totalcomp_sim[[1]]
    T_Stil_domnoplan_j <- domnoplan_totalcomp_sim[[2]]
    T_Stil_domnoplan_j_sim <- domnoplan_totalcomp_sim[[3]]
    total_domnoplan_sim[j,c((3+1):(3+D)),] <- matrix(t(T_S_domnoplan_j),D,nSim) + T_Stil_domnoplan_j_sim

    #   Totals for planned domains
    total_domnoplan_sim[j,c(2),] <- matrix(sum(T_S_domnoplan_j),1,nSim) + T_Stil_domnoplan_j

    }

#----------------------------
#   Output
#----------------------------
return(total_domnoplan_sim)

#
#   --  End of  "domnoplan.R"--
}

domnoplan_composicion <- function(datos_j,N_j,N_S_j,N_Stil_j,colid_D,alpha_D,nSim){
#
#   This function simulates Monte Carlo samples from the predictive distribution of the vector \eqn{\mathbold{N}_j} across the \eqn{D} unplanned domains in a given planned domain \eqn{j}.
#
#   Input:
#       datos_j                     -   Data matrix with features and number of individuals in the sample 'S_j'
#       N_j                         -   Number of individuals in 'P_j' (jth planned domain)
#       N_S_j                       -   Number of individuals in the sample 'S_j'
#       N_Stil_j                    -   Number of individuals out the sample 'S_j'
#       colid_D                     -   D-dimensional matrix array with the columns in ´datos´ that correspond to the indicator variables of the planned domains (those indicator variables represent a partion of 'datos')
#       alpha_D                     -   D-dimensional array with positive entries for the parameters of the multinomial-Dirichlet component for the composition across unplanned domains (NOTE: this one makes reference to a single planned domain)
#       nSim                        -   Number of Monte Carlo simulated replicates of the predictive distribution
#
#   Output:
#       N_S_domnoplan               -   Composition of the number of individuals in sample 'S_j' ('N_S_j') across the 'D' unplanned domains
#       domnoplan_composicion_sim   -   (1 x D x nSim) matrix with Monte Carlo samples of the predictive distribution of the composition of 'Stil_j'
#

#   Size of unplanned domains
D <- length(colid_D)
dim_D_j <- dim(datos_j)

```

```
#   Sample Repository
domnoplan_composicion_sim <- array(0,c(D,nSim))
colnames(domnoplan_composicion_sim) <- c(1:nSim)

#   Computing sample counts
d <- 1
for(d in 1:D){
    #   Creating the table with members of the group number for each unplanned domain
    N_S_domnoplan <- datos_j[,"n_i"] * datos_j[,colid_D[d]]
    datos_j <- cbind(datos_j,N_S_domnoplan)
    }

#   Composition of unplanned domains in the sample "S_j"
N_S_domnoplan <- colSums(datos_j[,c((dim_D_j[2]+1):(dim_D_j[2]+D))])

if(N_Stil_j >0){
    #   Parameters of the Dirichlet distribution for the proportions of "domnoplan"
    alpha_D_new <- alpha_D + N_S_domnoplan

    #   Predictive simulation of multinomial-Dirichlet model
    sim <- 1
    for(sim in 1:nSim){
        #   Sample of the composition "p_domplan_j"
        p_domnoplan_j <- randDirichlet(alpha_D_new,1)

        #   Sample of the composition "N_Stil_domplan_j"
        N_Stil_domnoplan_j <- t(rmultinom(1, N_Stil_j, p_domnoplan_j))

        #   Repository
        domnoplan_composicion_sim[, sim] <- N_Stil_domnoplan_j
        }
}

#   Output
salida <- list(N_S_domnoplan,domnoplan_composicion_sim)
return(salida)

#
#   --  End of domnoplan_composicion.R  --
}

domnoplan_g0<- function(datos,domplan_N,alphaDP,colid_D,alpha_D,nSim,g0_licitacion_sal){
#
#   Generates Monte Carlo samples of the predictive distribution of totals of a finite
population
#   segmented in planned and unplanned domains, using a predefined set of
\eqn{(G_{j0})_{j=1}^{J}},
#   along with simulations of the predictive distribution for the composition of the
population
#   between the unplanned domains.
#
#   Input:
#       datos       -   (Mxp)-dimensional array with positive entries for S_j
#       domplan_N   -   Matrix array with counts of individuals in each planned domain
#       alphaDP     -   J-dimensional array with positive entries for the parameters of the Dirichlet process for F_j (with J being the number of planned domains)
#       colid_D     -   D-dimensional matrix array with the columns in ´datos´ that correspond to the indicator variables of the planned domains (those indicator variables represent a partion of 'datos')
#       alpha_D     -   (JxD)-dimensional array with positive entries for the parameters of the multinomial-Dirichlet component for the composition across unplanned domains.
#                       Note: Each one of the J rows is a vector of composition for P_j segmented across D unplanned domains
#       nSim        -   Number of Monte Carlo simulated replicates of the predictive distribution
# g0_licitacion_sal -   (Jx1) object list, each entry is another object list itself associated with each G_{j0} for the J planned domains. The first element for arch G_{j0} should be the name of the chosen ´distribution´(see details below for alternatives), the second element should be a vector object with the parameters associated with ´distribution´, and the third element should be its associated expectation
#
#
#   Details:
#           - datos : Represents the data sample of the target population, unplanned domains labelled.
#                       It should contain the following columns:
#                           "domplan" - Categories fot planned domains.
#                           "y_i"     - Actual individual measurements/outcomes (for the moment, they must be positive) for the group of observation.
#                           "n_i"     - Number of individuals in the group (if the unit of observation in the sample are individuals, then "n_i" must be equal to 1)
#
#           - domplan_N : Represents counts (or reference population) of the target population, divided by the planned domains.
#                       Tagged data must be labelled by domains planned. It should contain the following columns:
#                           "domplan" - Categories for planned domains.
#                           "N_j"     - Number of individuals in each planned domain.
#
#           - g0_licitacion_sal: Chose one and only one of the distribution:
#                                         i)    Gamma
#                                        ii)    Weibull
#                                       iii)    Lognormal
#                                        iv)    Inverse-Gaussian
#
#                       Parameterization for distribution:
#
#                        i) Gamma    with parameters theta = c(alpha>0 , beta>0) and density function
#
#                               f(x)= x^{alpha-1} exp{-x/beta}
#
#                               where alpha is the shape parameter, and beta is the scale parameter.
#
#                       ii) Weibull  with parameters theta = c(alpha>0 , beta>0) and
```

```
density function
#
#                               f(x)=(x/beta)^{alpha-1} exp{-(x/beta)^alpha}
#
#                               where alpha is the shape parameter, and beta is the
scale parameter.
#
#                      iii) Lognormal with parameters theta = c(alpha>0 , beta>0) and
density function
#
#                               f(x)= exp{-(log(x)-alpha)^2/(2*beta^2)}
#
#                               where alpha is the mean, and beta is the standard
deviation of the logarithm.
#
#                       iv) Inverse-Gaussian  with parameters theta = c(alpha>0 ,
beta>0) and density function
#
#                               f(x)= .....
#
#                               where alpha is the shape parameter, and beta is the
scale parameter.
#
#   Output:
#       total_domnoplan_sim     -   Matrix array of dimension "J x (3 + 2*D) x nSim"
with predictions for relevant quantities of planned and unplanned domains
#                                   Column 1 - Indicator of the planned domains
#                                   Column 2 - T_j (totals of the planned domains)
#                                   Column 3 - N_j (composition of the planned domains)
#                                   Column 4 to (4+D-1) - T^d_j (totals of unplanned
domains, such that T_j = sum  T^d_j (over d))
#                                   Column (4+D) to (3 + 2*D) - N^d_j (composition of
unplanned domains, such that N_j = sum  N^d_j (over d))
#

#      Consulted domains
cualdomplan <- as.matrix(domplan_N[,"domplan"])
J <- length(cualdomplan)

#   Size unplanned domains
D <- length(colid_D)

#       Repository totals in "unplanned domains" with three categories
total_domnoplan_sim <- array(NaN,c(J,3+2*D,nSim))

#----------------------------
#   validation
#----------------------------

#   A.  Parameter alphaDP
if(any(alphaDP <= (0*alphaDP))){
    stop("Error in the specification of 'alphaDP'!!!")}

#   B.  Parameters alpha_D and colid_D
```

```r
if(ncol(alpha_D) != ncol(colid_D)){
    stop("The dimenciones of 'alpha_D' and 'colid_D' are different!!!")}

#   C.  parameter alpha_D
if(any(alpha_D <= (0*alpha_D))){
    stop("Error in the specification of 'alpha_D'!!!")}

#   D.  Parameter nSim
if(nSim <= 0){
    stop("Error in the specification of 'nSim'!!!")}

#   F. Total domains unplanned
if(sum(datos[,colid_D]) != nrow(datos)) {
   stop("Error in total of unplanned domains")}

if(any(rowSums(datos[,colid_D]) != 1)) {
   stop("Error in total of unplanned domains")}

#   -----------------------------------------------
#   Scanning planned domains
#   -----------------------------------------------
j <- 1
for(j in 1:J){

    #   Extraction of indexes in "data" (i.e. sample S_j)
    P_j <- which(datos[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    datos_j <- datos[P_j, ]

    #   Population size in \mathcal{P}_{j}
    P_j <- which(domplan_N[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    N_j <- domplan_N[P_j,"N_j"]

    #   domain name the planned arrangement and constant data impute "n_j"
    sim <- 1
    for(sim in 1:nSim){
        total_domnoplan_sim[j,1,sim] <- cualdomplan[j]
        total_domnoplan_sim[j,3,sim] <- N_j
        }

    #   ``conteo''
    conteo_sal <- conteo(datos_j,N_j)

    #   Estimating the size of the population out of the sample (in "Stil_j")
    N_S_j <- conteo_sal[[2]]

    #   Estimating the size of the population out of the sample (in "Stil_j")
    N_Stil_j <- conteo_sal[[3]]

    #   Simulating of the composition of "N_Stil_j" between domains unplanned
    domnoplan_composicion_sim <- domnoplan_composicion(datos_j,N_j,N_S_j,N_Stil_j,
    colid_D,alpha_D[j,],nSim)

    #   Simulating data"N_j" distributed in "domnoplan1"
    N_S_domnoplan_j <- domnoplan_composicion_sim[[1]]
    N_Stil_domnoplan_j_sim <- domnoplan_composicion_sim[[2]]
    total_domnoplan_sim[j,c((3+D+1):(3+2*D)),] <- matrix(t(N_S_domnoplan_j),D,nSim) +
    N_Stil_domnoplan_j_sim

    #   ``unicstar''
    unicstar_sal <- unicstar(datos_j)
    ystar <- unicstar_sal[[2]][,"ystar"]

    #   ``pesos''
    pesos_sal <- pesosDP(unicstar_sal,alphaDP[j])
    rho <- pesos_sal[[1]]
    phi <- pesos_sal[[2]]


    #-----------------------------------
    #   Validation ``g0_licitacion_sal''
    #-----------------------------------
    #    i) Gamma
    #   ii) Weibull
    #  iii) Log-normal
    #   iv) Inverse-Gaussian

    dis_sel <- list("Gamma","Weibull","Lognormal","Inverse-Gaussian")

    #   Validation of distribution

    if(g0_licitacion_sal[[j]][[1]] == dis_sel[[1]] ||g0_licitacion_sal[[j]][[1]] ==
    dis_sel[[2]] ||g0_licitacion_sal[[j]][[1]] == dis_sel[[3]] ||g0_licitacion_sal[[j
    ]][[1]] == dis_sel[[4]]){

    }else{
        stop("Error in the specification of 'distribution'!!!")}

    #   Validation of parameters
    if(g0_licitacion_sal[[j]][[2]][[1]]>0 & g0_licitacion_sal[[j]][[2]][[2]]>0){
    }else{
        stop("Error in the specification of 'parameters'!!!")}

    #   Simulating the composition of "T_Stil_j" between domains unplanned
    domnoplan_totalcomp_sim <- domnoplan_totalcomp(datos_j,rho,ystar,phi,
    g0_licitacion_sal[[j]],N_Stil_domnoplan_j_sim,nSim,colid_D)

    #   Simulating data grouped "n_j" distributed "domnoplan1"
    T_S_domnoplan_j <- domnoplan_totalcomp_sim[[1]]
    T_Stil_domnoplan_j <- domnoplan_totalcomp_sim[[2]]
    T_Stil_domnoplan_j_sim <- domnoplan_totalcomp_sim[[3]]
    total_domnoplan_sim[j,c((3+1):(3+D)),] <- matrix(t(T_S_domnoplan_j),D,nSim) +
    T_Stil_domnoplan_j_sim

    #   Totals for planned domains
    total_domnoplan_sim[j,c(2),] <- matrix(sum(T_S_domnoplan_j),1,nSim) +
    T_Stil_domnoplan_j

    }

#----------------------------
#   Output
#----------------------------
return(total_domnoplan_sim)

#
#   --  End of  "domnoplan_g0.R"--
}

domnoplan_totalcomp <- function(datos_j,rho,ystar,phi,g0_licitacion_sal,
N_Stil_domnoplan_j_sim,nSim,colid_D){
#
#   This function simulates Monte Carlo samples from the predictive distribution of the
vector \eqn{\mathbold{T}_j} across the \eqn{D} unplanned domains in a given planned
domain \eqn{j}.
#
#   Input:
#       datos_j                     -   Data matrix with features and number of
individuals in the sample 'S_j'
#       rho                         -   (Ux2)-dimensional vector with weights
associated with the sample ties 'ystar'
#       ystar                       -   Sample ties 'y^*_i' in the sample 'S_j'
#       phi                         -   Probability weight associated with "G_{j0}"
(the continuous part of Ghat_j)
#       g0_licitacion_sal           -   Object list with the continuous component in
'Ghat'
#       N_Stil_domnoplan_j_sim      -   (1 x D x nSim) matrix with Monte Carlo samples
of the predictive distribution of the composition of 'Stil_j'
#       nSim                        -   Number of Monte Carlo simulated replicates of
the predictive distribution
#       colid_D                     -   D-dimensional matrix array with the columns in
´datos´ that correspond to the indicator variables of the planned domains (those
indicator variables represent a partion of 'datos')
#
#   Output:
#   Object list with three entries:
#       T_S_domnoplan               -   Composition of 'T_j' for the planned domain 'j'
in sample 'S_j'
#       T_Stil_domnoplan_j          -   (1xDxnSim)-dimensional array with simulated
samples of the convolution for 'T_Stil_j'
#       domnoplan_totalcomp_sim     -   (1xDxnSim)-dimensional array with samples from
the predictive distribution of the composition of 'Stil_j'
#

#   Number of unplanned domains
D <- length(colid_D)
dim_D_j <- dim(datos_j)

#   Repository: Unplanned domains "T_Stil_domnoplan_d_j"
domnoplan_totalcomp_sim <- array(NaN,c(D,nSim))

```

```r
#   Repository: Non-sampled total the domain j - "T_Stil_domnoplan_j"
T_Stil_domnoplan_j <- array(NaN,c(1,nSim))
rownames(T_Stil_domnoplan_j) <- c("T_Stil_domnoplan_j")
colnames(T_Stil_domnoplan_j) <- c(1:nSim)

# Computing sample totals
d <- 1
for(d in 1:D){
    #   vector with number of members of the group for each unplanned domains
    T_S_domnoplan <- datos_j[,"n_i"] * datos_j[,"y_i"] * datos_j[,colid_D[d]]
    datos_j <- cbind(datos_j,T_S_domnoplan)
    }

#   Composition of the total across domains on the sample unplanned "S_j"
T_S_domnoplan <- colSums(datos_j[,c((dim_D_j[2]+1):(dim_D_j[2]+D))])

#   Simulations
sim <- 1
d <- 1
for(sim in 1:nSim){
    for(d in 1:D){
        #   Extract the auxiliary size N_Stil_domnoplan1_d
        N_d_aux <- N_Stil_domnoplan_j_sim[d,sim]

        if(N_d_aux > 0){
            #   simulating a sample of size "N_d_aux" of "ghat"
            y_d_aux <- ghat_simulacion(rho,ystar,phi,g0_licitacion_sal,N_d_aux)

            #   Simulation subtotal "T_Stil_domnoplan1_d"
            domnoplan_totalcomp_sim[d,sim] <- sum(y_d_aux)
        }else{
            #   Sum of the subtotal "T_Stil_domnoplan1_d" zero
            domnoplan_totalcomp_sim[d,sim] <- 0
        }
    }
    T_Stil_domnoplan_j[sim] <- sum(domnoplan_totalcomp_sim[,sim])
}

#   Output
salida <- list(T_S_domnoplan,T_Stil_domnoplan_j,domnoplan_totalcomp_sim)
return(salida)

#
#   --  END of domnoplan_totalcomp.R    --
}

domplan <- function(datos,datos_ant,domplan_N,alphaDP,inter,part,nSim){
#
#   Generates Monte Carlo samples of the predictive distribution of totals of a finite
population segmented in planned domains.
#
#   Input:
#       datos       -   p-dimensional vector with positive entries
#       datos_ant   -   Reference data for calibration of G_0
#       domplan_N   -   Free dimension matrix array represents counts
#       alphaDP     -   J-dimensional positive parameter for the Dirichlet process
(with J being the number of planned domains)
#       inter       -   Tuning parameter for model comparison and selection
#       part        -   Number of partitions for predictive cross-validation
#       nSim        -   Number of simulated replicates
#
#   Details:
#           - datos : Represents the data sample of the target population, unplanned
domains labelled.
#                       It should contain the following columns:
#                           "domplan" - Planned domains categories.
#                           "y_i"     - Actual measurements of individual positive and
the sample group.
#                           "n_i"     - Number of members in the group (if the unit of
observation in the sample are individuals, then "n_i" must be equal to 1)
#
#           - datos_ant : Represents the data of reference used to calibrate G_{0}.
#                       The data must be labelled by domains planned. It should contain
the following columns:
#                           "domplan" - Planned categories of domains
#                           "y_i"     - Positive real and individual measurements of
each group in the sample (when the units of observation are groups, "y_i" should be a
per capita measurement)
#
#            - domplan_N : Represents counts (or population frame) of the target
population, divided by the planned domains.
#                       Tagged data must be labelled by domains planned. It should
contain the following columns:
#                           "domplan" - Planned domains categories
#                           "N_j"     - Number of individuals in each population
planned domain
#
#   Output:
#       total_domplan_sim   - "(J x 3 x nSim)" Matrix array dimension with the
predictions of the planned domains
#                                   Column 1 - Indicator of the planned domains
#                                   Column 2 - T_j (totals of the planned domains)
#                                   Column 3 - N_j (composition of the planned domains)
#

#      Definition consulted domains
cualdomplan <- as.matrix(domplan_N[,"domplan"])
J <- length(cualdomplan)

#       Repository totals in "unplanned domains" with three categories
total_domplan_sim <- array(NaN,c(J,3,nSim))

#----------------------------
#   Validation
#----------------------------

#   A.  Parameter alphaDP
if(any(alphaDP <= (0*alphaDP))){
    stop("Error in the specification of 'alphaDP'!!!")}

#   B.  Parameter nSim
if(nSim <= 0){
    stop("Error in the specification of 'nSim'!!!")}

#   C.  Parameter inter
if(inter <= 0){
    stop("Error in the specification of 'inter'!!!")}

#   -----------------------------------------------
#   Scanning planned domains
#   -----------------------------------------------
j <- 1
for(j in 1:J){
    #   Extraction of indexes in "datos_ant"
    P_j_ant <- which(datos_ant[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "datos_ant"
    datos_j_ant <- datos_ant[P_j_ant, ]

    #   Extraction of indexes in "data" (ie sample S_j)
    P_j <- which(datos[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    datos_j <- datos[P_j, ]

    #   Population size of \mathcal{P}_{j}
    P_j <- which(domplan_N[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    N_j <- domplan_N[P_j,"N_j"]

    #   Name the planned domain grooming and constant data impute "n_j"
    for(sim in 1:nSim){
        total_domplan_sim[j,1,sim] <- cualdomplan[j]
        total_domplan_sim[j,3,sim] <- N_j
        }

    #   ``conteo''
    conteo_sal <- conteo(datos_j,N_j)

    #   Estimating the size of the population outside of the sample (in "Stil_j")
    N_S_j <- conteo_sal[[2]]

    #   Estimating the size of the population outside of the sample (in "Stil_j")
    N_Stil_j <- conteo_sal[[3]]

    if(N_Stil_j >0){
        #   ``unicstar''
        unicstar_sal <- unicstar(datos_j)
        ystar <- unicstar_sal[[2]][,"ystar"]

        #   ``pesos''
        pesos_sal <- pesosDP(unicstar_sal,alphaDP[j])
        rho <- pesos_sal[[1]]
        phi <- pesos_sal[[2]]
```

```r

            #   ``g0_licitacion''
            g0_licitacion_sal <- g0_licitacion(datos_j_ant,inter,part)
            #g0_licitacion_sal <- g0_licitacion(datos_j_ant,inter)

            #   Simulation the T_Stil_j
            domplan_total_sim <- domplan_total(datos_j,rho,ystar,phi,g0_licitacion_sal,
            N_Stil_j,nSim)

            #   Simulating data grouped "n_j" distributed "domnoplan1"
            T_S_domplan_j <- domplan_total_sim[[1]]
            T_Stil_domplan_j <- domplan_total_sim[[2]]
            total_domplan_sim[j,2,] <- matrix(t(T_S_domplan_j),1,nSim) + T_Stil_domplan_j
        }else if(N_Stil_j ==0){
            #   Calculating the total "T_S_j" compositional in S_j (within the sample)
            T_S_j_gpo <- datos_j[,"n_i"] * datos_j[,"y_i"]

            #   Composition of the total between domains on the sample unplanned "S_j"
            T_S_domplan_j <- sum(T_S_j_gpo)

            #   Simulating data store "n_j" distributed "domnoplan1"
            total_domplan_sim[j,2,] <- matrix(t(T_S_domplan_j),1,nSim)
            }

        }

    #----------------------------
    #   Output
    #----------------------------
    return(total_domplan_sim)

    #
    #   --  END of  "domplan.R"--
    }

    domplan_g0<- function(datos,domplan_N,alphaDP,nSim,g0_licitacion_sal){
    #
    #   Generates Monte Carlo samples of the predictive distribution of totals of a finite
    population segmented in planned domains, using a predefined set of
    \eqn{(G_{j0})_{j=1}^{J}}.
    #
    #   Input:
    #       datos               -   (Mxp)-dimensional array with positive entries for {S}_j
    #       domplan_N           -   Matrix array with counts of individuals in each planned
    domain
    #       alphaDP             -   J-dimensional array with positive entries for the
    parameters of the Dirichlet process for F_j (with J being the number of planned domains)
    #       nSim                -   Number of Monte Carlo simulated replicates of the
    predictive distribution
    #   g0_licitacion_sal       -   (Jx1) object list, each entry is another object list
    itself associated with each G_{j0} for the J planned domains. The first element for
    arch G_{j0} should be the name of the chosen distribution(see details below for
    alternatives), the second element should be a vector object with the parameters
    associated with distribution, and the third element should be its associated expectation.
```

```
#
#   Details:
#           - datos : Represents the data sample of the target population, unplanned domains labelled.
#                       It should contain the following columns:
#                           "domplan" - Categories fot planned domains.
#                           "y_i"     - Actual individual measurements/outcomes (for the moment, they must be positive) for the group of observation.
#                           "n_i"     - Number of individuals in the group (if the unit of observation in the sample are individuals, then "n_i" must be equal to 1)
#
#           - domplan_N : Represents counts (or reference population) of the target population, divided by the planned domains.
#                       Tagged data must be labelled by domains planned. It should contain the following columns:
#                           "domplan" - Categories for planned domains.
#                           "N_j"     - Number of individuals in each planned domain.
#
#           - g0_licitacion_sal: Chose one and only one of the distribution:
#                                         i)    Gamma
#                                        ii)    Weibull
#                                       iii)    Lognormal
#                                        iv)    Inverse-Gaussian
#
#                       Parameterization for distribution:
#
#                        i) Gamma    with parameters theta = c(alpha>0 , beta>0) and density function
#
#                               f(x)= x^{alpha-1} exp{-x/beta}
#
#                               where alpha is the shape parameter, and beta is the scale parameter.
#
#                       ii) Weibull  with parameters theta = c(alpha>0 , beta>0) and density function
#
#                               f(x)=(x/beta)^{alpha-1} exp{-(x/beta)^alpha}
#
#                               where alpha is the shape parameter, and beta is the scale parameter.
#
#                      iii) Lognormal with parameters theta = c(alpha>0 , beta>0) and density function
#
#                               f(x)= exp{-(log(x)-alpha)^2/(2*beta^2)}
#
#                               where alpha is the mean, and beta is the standard deviation of the logarithm.
#
#                       iv) Inverse-Gaussian  with parameters theta = c(alpha>0 , beta>0) and density function
#
#                               f(x)= .....
```

```r
#
#                               where alpha is the shape parameter, and beta is the scale parameter.
#
#
#   Output:
#       total_domplan_sim   - "(J x 3 x nSim)" Matrix array dimension with the predictions of the planned domains
#                                   Column 1 - Indicator of the planned domains
#                                   Column 2 - T_j (totals of the planned domains)
#                                   Column 3 - N_j (composition of the planned domains)
#

#      Definition consulted domains
cualdomplan <- as.matrix(domplan_N[,"domplan"])
J <- length(cualdomplan)
#       Repository totals in "unplanned domains" with three categories
total_domplan_sim <- array(NaN,c(J,3,nSim))

#----------------------------
#   Validation
#----------------------------

#   A.  Parameter alphaDP
if(any(alphaDP <= (0*alphaDP))){
    stop("Error in the specification of 'alphaDP'!!!")}

#   B.  Parameter nSim
if(nSim <= 0){
    stop("Error in the specification of 'nSim'!!!")}

#   -----------------------------------------------
#   Scanning planned domains
#   -----------------------------------------------
j <- 1
for(j in 1:J){

    #   Extraction of indexes in "data" (ie sample S_j)
    P_j <- which(datos[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    datos_j <- datos[P_j, ]

    #   Population size of \mathcal{P}_{j}
    P_j <- which(domplan_N[,"domplan"]==cualdomplan[j])
    #   Extraction of the data in "data"
    N_j <- domplan_N[P_j,"N_j"]

    #   Name the planned domain grooming and constant data impute "n_j"
    for(sim in 1:nSim){
        total_domplan_sim[j,1,sim] <- cualdomplan[j]
        total_domplan_sim[j,3,sim] <- N_j
        }

    #   ``conteo''
```

```
        conteo_sal <- conteo(datos_j,N_j)

        #   Estimating the size of the population outside of the sample (in "Stil_j")
        N_S_j <- conteo_sal[[2]]

        #   Estimating the size of the population outside of the sample (in "Stil_j")
        N_Stil_j <- conteo_sal[[3]]

        if(N_Stil_j >0){
            #   ``unicstar''
            unicstar_sal <- unicstar(datos_j)
            ystar <- unicstar_sal[[2]][,"ystar"]

            #   ``pesos''
            pesos_sal <- pesosDP(unicstar_sal,alphaDP[j])
            rho <- pesos_sal[[1]]
            phi <- pesos_sal[[2]]


        #-----------------------------------
        #   Validation ``g0_licitacion_sal''
        #-----------------------------------
        #    i) Gamma
        #   ii) Weibull
        #  iii) Log-normal
        #   iv) Inverse-Gaussian

        dis_sel <- list("Gamma","Weibull","Lognormal","Inverse-Gaussian")

        #   Validation of distribution

        if(g0_licitacion_sal[[j]][[1]] == dis_sel[[1]] ||g0_licitacion_sal[[j]][[1]] ==
        dis_sel[[2]] ||g0_licitacion_sal[[j]][[1]] == dis_sel[[3]] ||g0_licitacion_sal[[j
        ]][[1]] == dis_sel[[4]]){
        }else{
            stop("Error in the specification of 'distribution'!!!")}

        #   Validation of parameters
        if(g0_licitacion_sal[[j]][[2]][[1]]>0 & g0_licitacion_sal[[j]][[2]][[2]]>0){
        }else{
            stop("Error in the specification of 'parameters'!!!")}

            #   Simulation the T_Stil_j
            domplan_total_sim <- domplan_total(datos_j,rho,ystar,phi,g0_licitacion_sal[[j]],
            N_Stil_j,nSim)

            #   Simulating data grouped "n_j" distributed "domnoplan1"
            T_S_domplan_j <- domplan_total_sim[[1]]
            T_Stil_domplan_j <- domplan_total_sim[[2]]
            total_domplan_sim[j,2,] <- matrix(t(T_S_domplan_j),1,nSim) + T_Stil_domplan_j
        }else if(N_Stil_j ==0){
            #   Calculating the total "T_S_j" compositional in S_j (within the sample)
            T_S_j_gpo <- datos_j[,"n_i"] * datos_j[,"y_i"]

        #   Composition of the total between domains on the sample unplanned "S_j"
        T_S_domplan_j <- sum(T_S_j_gpo)

        #   Simulating data store "n_j" distributed "domnoplan1"
        total_domplan_sim[j,2,] <- matrix(t(T_S_domplan_j),1,nSim)
        }

    }

#----------------------------
#   Output
#----------------------------
return(total_domplan_sim)

#
#   --  END of  "domplan_g0.R"--
}

domplan_total <- function(datos_j,rho,ystar,phi,g0_licitacion_sal,N_Stil_j,nSim){
#
#   Simulates Monte Carlo samples of size '\eqn{nSim}' of the final distribution for the total for the given planned domain \eqn{\mathcal{P}_j}.
#
#   Input:
#       datos_j                     -   Data matrix with features and number of individuals in the sample 'S_j'
#       rho                         -   (Ux2)-dimensional vector with weights associated with the sample ties 'ystar'
#       ystar                       -   Sample ties 'y^*_i' in the sample 'S_j'
#       phi                         -   Probability weight associated with 'G_{j0}' (the continuous part of Ghat_j)
#       g0_licitacion_sal           -   Object list with the continuous component in 'Ghat'
#       N_Stil_sal                  -   Composition of individuals out of 'S_j'
#       nSim                        -   Number of Monte Carlo simulated replicates of the predictive distribution
#
#   Output:
#   Object list with two entries:
#       T_S_j                       -   Composition of 'T_j' across planned domains in the sample 'S_j'
#       domplan_total_sim           -   (1 x nSim) matrix with samples from the predictive distribution of the total 'T_Stil_j'
#

#   Repository: Total non-sampling domain j -- "T_Stil_domnoplan_j"
T_Stil_domnplan_j <- array(NaN,c(1,nSim))
rownames(T_Stil_domnplan_j) <- c("T_Stil_domnplan_j")
colnames(T_Stil_domnplan_j) <- c(1:nSim)

#   Computing the total "T_S_j" compositional in "S_j" (within the sample)
#   Vector with the totals of each group
T_S_j_gpo <- datos_j[,"n_i"] * datos_j[,"y_i"]
datos_j <- cbind(datos_j,T_S_j_gpo)
```

```r

#   Composition of the total between domains on the sample unplanned "S_j"
T_S_j <- sum(datos_j[,"T_S_j_gpo"])

#   Simulations
sim <- 1
for(sim in 1:nSim){
    #   Computing the sample of size "N_Stil_j" of "ghat"
    y_i_aux <- ghat_simulacion(rho,ystar,phi,g0_licitacion_sal,N_Stil_j)

    #   Simulation subtotal "T_Stil_j"
    T_Stil_domnplan_j[sim] <- sum(y_i_aux)
    }

#   Output
salida <- list(T_S_j,T_Stil_domnplan_j)
return(salida)

#
#   --  END of  domplan_total.R --
}

g0_licitacion <- function(datos_ant,inter,part){
#   This function computes the prior elicitation of \eqn{G_{j0}} using reference data
for the planned domain \eqn{j}.
#   \eqn{G_{j0}} is used by the SSM as the baseline function. The distribution is
elicited using a predictive cross-validation
#   procedure for model comparison and selection among the following alternatives:
Gamma, Weibull, Lognormal and Inverse-Gaussian.
#   The function also computes the expectation of the chose distribution.
#
#   input:
#             datos_ant    -    Reference data for calibration of G_0
#             inter        -    Tuning parameter for model comparison and selection
#
#
#   output:  List object:
#             'distribution' -   String object for 'distribution'
#                                    i) Gamma
#                                   ii) Weibull
#                                  iii) Log-normal
#                                   iv) Inverse-Gaussian
#
#             theta      -    Vector of parameters associated with 'distribution'
#             mu         -    Expected value of 'distribution'
#

y <- datos_ant[,"y_i"]
n <- length(y)
n1 <- n/20
# Vector with assessments of the 20 groups with missing observations
  g0LogN20 <- rep(NA,20)
  g0Gam20  <- rep(NA,20)
  g0Wei20  <- rep(NA,20)
```

```r
    g0IGau20 <- rep(NA,20)
  # Vector with assessments of each observation using the estimated parameters with the
  observations that were omitted
    g0LogNN <- rep(NA,n)
    g0GamN  <- rep(NA,n)
    g0WeiN  <- rep(NA,n)
    g0IGauN <- rep(NA,n)
  # initial parameters
    sigmaLN0c <- rnorm(1,0,1)
    muLN0c    <- rnorm(1,0,1)
    nu1G0c    <- rnorm(1,0,1)
    nu2G0c    <- rnorm(1,0,1)
    nu1W0c    <- rnorm(1,0,1)
    nu2W0c    <- rnorm(1,0,1)
    nu1IG0c   <- rnorm(1,0,1)
    nu2IG0c   <- rnorm(1,0,1)
  # Randomization group
    y <- sample(y)
  # Vector with 20 parameters and parameter groups of the entire group
    muLN0  <- rep(NA,21)
    sigLN0 <- rep(NA,21)
    shG0   <- rep(NA,21)
    scG0   <- rep(NA,21)
    shW0   <- rep(NA,21)
    scW0   <- rep(NA,21)
    shIG0  <- rep(NA,21)
    scIG0  <- rep(NA,21)

  i <- 1
  for (i in 1:20)
  {
  # Ranges to form 20 groups
      i5 <- round((i-1)*n1)+1
      i6 <- round(i*n1)
      i7 <- i6+1
      if (i6 <= n)
      {
      for (j in (i5:i6))
        {
        # Missing observations according to previous ranks
          y[j] <- NA
        }
      }else{for (j in i7:n){y[j] <- NA}}
      y <- na.omit(y)

    # Calculations prior to the estimation of Weibull
    parameters
      nOmiOrd <- length(sort(y))
      x00 <- c(1:nOmiOrd)
      x0  <- log(log(1/(1-(x00/(nOmiOrd+1)))))
      x1  <- (1/nOmiOrd)*sum(x0)
      y1  <- mean(log(sort(y)))

    # ESTIMATES IN EACH GROUP:
```

```r

    # Mu de X ~ Normal, where
    X=log(Y)
      muLN0[i]  <- sum(log(y))/length(y)
    # Sigma of X ~ Normal, where X=log(Y)
      sigLN0[i] <- sqrt(sum((log(y)-muLN0[i])^2)/length(y))
    # Form parameter of Gamma
      shG0[i]   <- 0.5/(log(mean(y))-mean(log(y)))
    # Scale parameter of Gamma
      scG0[i]   <- mean(y)/shG0[i]
    # Form parameter of Weibull
      shW0[i]   <- ((nOmiOrd*(sum((log(sort(y)))*(x0))))-(sum(x0)*sum(log(sort(y)))))/((
      nOmiOrd*sum(log(sort(y))^2))-((sum(log(sort(y))))^2))
    # Scale parameter of Weibull
      scW0[i]   <- exp(y1-(x1/shW0[i]))
    # Parameter "mu" of the Inverse Gaussian (not a location parameter)
      shIG0[i]  <- mean(y)
    # Parameter "sigma" of the Inverse Gaussian (not a scale parameter)
      scIG0[i]  <- (sum((1/y)-(1/shIG0[i])))/length(y)

      y <- datos_ant[,"y_i"]

      g0LogNPaso <- rep(NA,n)
      g0GamPaso  <- rep(NA,n)
      g0WeiPaso  <- rep(NA,n)
      g0IGauPaso <- rep(NA,n)

      for (j in (i5:i6))
      {
        # Conditional densities
          g0LogNPaso[j] <- (pnorm(log(y[j]+inter),mean=muLN0[i],sd=sigLN0[i])-pnorm(log(
          max(y[j]-inter,0)),mean=muLN0[i],sd=sigLN0[i]))/(2*inter)
          g0GamPaso[j]  <- (pgamma(y[j]+inter,shape=shG0[i],scale=scG0[i])-pgamma(max(y[j
          ]-inter,0),shape=shG0[i],scale=scG0[i]))/(2*inter)
          g0WeiPaso[j]  <- (pweibull(y[j]+inter,shape=shW0[i],scale=scW0[i])-pweibull(max(
          y[j]-inter,0),shape=shW0[i],scale=scW0[i]))/(2*inter)
          g0IGauPaso[j] <- (pinvgauss(y[j]+inter,shIG0[i],scIG0[i])-pinvgauss(max(y[j]-
          inter,0.000001),shIG0[i],scIG0[i]))/(2*inter)
        # Vectors with assessments of the "n" observations
          g0LogNN[j]  <- g0LogNPaso[j]
          g0GamN[j]   <- g0GamPaso[j]
          g0WeiN[j]   <- g0WeiPaso[j]
          g0IGauN[j]  <- g0IGauPaso[j]
        # Vectors with assessments of the observations of the group "i" (20 groups)
          g0LogNsinNA  <- na.omit(g0LogNPaso)
          g0GamsinNA   <- na.omit(g0GamPaso)
          g0WeisinNA   <- na.omit(g0WeiPaso)
          g0IGausinNA  <- na.omit(g0IGauPaso)
      }
    # Vectors with the sum of the logarithms of the evaluations of the observations of
    the group "i"
      g0LogN20[i] <- sum(log(g0LogNsinNA))
      g0Gam20[i]  <- sum(log(g0GamsinNA))
      g0Wei20[i]  <- sum(log(g0WeisinNA))
      g0IGau20[i] <- sum(log(g0IGausinNA))
  }
  # Average of the 20 sums of logarithms
  criLN <- mean(g0LogN20)
  criG  <- mean(g0Gam20)
  criW  <- mean(g0Wei20)
  criIG <- mean(g0IGau20)

  # ESTIMATE FOR THE WHOLE SAMPLE :

  # Previous calculations for Weibull parameter
  estimation
  x00  <- c(1:n)
  x0   <- log(log(1/(1-(x00/(n+1)))))
  x1   <- (1/n)*sum(x0)
  y1   <- mean(log(y))

  # Mu of Y ~ Lognormal (mu_{y})
  muLN0[21]  <- sum(log(y))/length(y)
  # Sigma of Y ~ Lognormal (sigma_{y})
  sigLN0[21] <- sqrt(sum((log(y)-muLN0[21])^2)/length(y))
  # Form parameter of Gamma
  shG0[21]   <- 0.5/(log(mean(y))-mean(log(y)))
  # Scale parameter Gamma
  scG0[21]   <- mean(y)/shG0[21]
  #Form parameter of Weibull
  shW0[21]   <- ((n*(sum((log(sort(y)))*(x0))))-(sum(x0)*sum(log(sort(y)))))/((n*sum(log(
  sort(y))^2))-((sum(log(sort(y))))^2))
  # Scale parameter of Weibull
  scW0[21]   <- exp(y1-(x1/shW0[21]))
  # Parameter "mu" of the Inverse Gaussian (not a location parameter)
  shIG0[21]  <- mean(y)
  # Parameter "sigma" of the Inverse Gaussian (not a scale parameter)
  scIG0[21]  <- (sum((1/y)-(1/shIG0[21])))/length(y)

  # COMPARISON CRITERIA:

  # Including all criteria
  #MaxCri <- max(criLN, criG, criW, criIG)
  # We removed the Gamma distribution criteria
  MaxCri <- max(criLN, criW, criIG)
  # Selected Distribution
  DenSel <- NA
  # Parameters of the selected distribution
  ParSel <- 0
  # Expectation of selected distribution
  MuSel <- 0
  if((MaxCri==criLN)) {DenSel="Lognormal"
  ParSel <- c(muLN0[21],sigLN0[21])
  MuSel <- exp(muLN0[21]+(0.5*(sigLN0[21]^2)))}else{if((MaxCri==criW)) {DenSel="Weibull"
  ParSel <- c(shW0[21],scW0[21])
  MuSel <- shG0[21]*scG0[21]}else{if(MaxCri==criG) {DenSel="Gamma"
  ParSel <- c(shG0[21],scG0[21])
  MuSel <- shG0[21]*scG0[21]}else{if(MaxCri==criIG) {DenSel="Inverse-Gaussian"
```

```r
ParSel <- c(shIG0[21],scIG0[21])
MuSel <- shIG0[21]}else{DenSel="Adjustment is not available"}}}}

g0_licitacion_sal <- list(DenSel,ParSel,MuSel)

#   Output:
salida <- g0_licitacion_sal
return(salida)

#   END of g0_licitacion.R
}

g0_simulacion <- function(g0_licitacion_sal,nSim){
#
#   This function computes Monte Carlo samples from a given continuous \eqn{G_{j0}}.
#
#   Input:
#       g0_licitacion_sal   -   Object list with three elements (produced with 'g0_licitacion'):
#                               a)  String object for 'distribution'
#                                     i) Gamma
#                                    ii) Weibull
#                                   iii) Log-normal
#                                   iv)  Inverse-Gaussian
#
#                               b) theta            -   Vector of parameters associated with 'distribution'
#                               c) mu               -   Expected value of 'distribution'
#       nSim                -   Number of Monte Carlo simulations
#
#   Output:
#       g0_simulacion_sal   -   (nSim x 1) dimensional array with simulated data.
#

#   Simulations case-particularl
if(g0_licitacion_sal[[1]]=='Lognormal'){
    g0_simulacion_sal <- as.matrix(rlnorm(nSim, meanlog =  g0_licitacion_sal[[2]][1], sdlog =  g0_licitacion_sal[[2]][2]))
    colnames(g0_simulacion_sal) <- c("ind_sim")
}else if(g0_licitacion_sal[[1]]=='Gamma'){
    g0_simulacion_sal <- as.matrix(rgamma(nSim, shape = g0_licitacion_sal[[2]][1], scale = g0_licitacion_sal[[2]][2]))
    colnames(g0_simulacion_sal) <- c("ind_sim")
}else if(g0_licitacion_sal[[1]]=='Weibull'){
    g0_simulacion_sal <- as.matrix(rweibull(nSim, shape = g0_licitacion_sal[[2]][1], scale = g0_licitacion_sal[[2]][2]))
    colnames(g0_simulacion_sal) <- c("ind_sim")
}else if (g0_licitacion_sal[[1]]=='Inverse-Gaussian'){
    g0_simulacion_sal <- as.matrix(rinvgauss(nSim, g0_licitacion_sal[[2]][1], g0_licitacion_sal[[2]][2]))
    colnames(g0_simulacion_sal) <- c("ind_sim")
}

```

```
#   Output
salida <- g0_simulacion_sal
return(salida)

#
#   --  END of g0_simulacion.R --
}

#
#   Generalized inverse Gaussian distribution
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
#

pginvgauss <- function(q, m, s, f){
#   Generalized inverse Gaussian distribution - Distribution
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
if(any(q<=0)){stop("q must contain positive values")}
if(any(m<=0)){stop("m must be positive")}
if(any(s<=0)){stop("s must be positive")}
len <- max(length(q),length(m),length(s))
if(length(q)!=len){
    if(length(q)==1)q <- rep(q,len)
    else stop("length of q incorrect")}
if(length(m)!=len){
    if(length(m)!=1)stop("m has incorrect length")
    else m <- rep(m,len)}
if(length(s)!=len){
    if(length(s)!=1)stop("s has incorrect length")
    else s <- rep(s,len)}
if(length(f)!=len){
    if(length(f)!=1)stop("f has incorrect length")
    else f <- rep(f,len)}
z <- .C("pginvgauss",
    as.double(q),
    as.double(m),
    as.double(s),
    as.double(f),
    len=as.integer(len),
    eps=as.double(1.0e-6),
    pts=as.integer(5),
    max=as.integer(16),
    err=integer(1),
    res=double(len),
    DUP=FALSE,
    PACKAGE="rmutil")
if(z$err==1){warning("Unable to allocate memory for integration")}
```

```r
if(z$err==2){warning("Division by zero in integration")}
else if(z$err==3){warning("No convergence in integration")}
return(z$res)
}

dginvgauss <- function(y, m, s, f, log=FALSE){
#   Generalized inverse Gaussian distribution - Density
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
if(any(y<=0)){stop("y must contain positive values")}
if(any(m<=0)){stop("m must be positive")}
if(any(s<=0)){stop("s must be positive")}
tmp <- (f-1)*log(y)-(1/y+y/m^2)/(2*s)-f*log(m)-log(2*besselK(1/(s*m),abs(f)))
if(!log)tmp <- exp(tmp)
return(tmp)
}

rginvgauss <- function(n=1, m, s, f){
#   Generalized inverse Gaussian distribution - Simulation
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
tmp <- qginvgauss(runif(n),m=m,s=s,f=f)
return(tmp)
}

ghat_simulacion <- function(rho,ystar,phi,g0_licitacion_sal,N_Stil_j){
#
#   This function simulates Monte Carlo samples from 'Ghat_j'.
#
#   Input:
#       rho                     -   Ux2 dimensional vector with weights associated with
'ystar'
#       ystar                   -   Sample ties 'y^*_i' in 'S_j'
#       phi                     -   Probability weight associated with 'G_{j0}' (the
continuous part of 'Ghat_j')
#       g0_licitacion_sal       -   Object list with three elements (produced with
'g0_licitacion'):
#                                   a)  String object for 'distribution'
#                                           i) Gamma
#                                          ii) Weibull
#                                         iii) Log-normal
#                                          iv) Inverse-Gaussian
#                                   b) theta            -   Parameters associated with
'distribution'
#                                   c) mu               -   Expected value of
'distribution'
#       N_Stil_j                -   Number of samples to simulate (number of
individuals in 'Stil_j') out of the sample
```

```
#
#   Output:
#       ghat_simulacion_sal -   (N_Stil_j x 1) dimensional vector with simulated data
#

#   Repository
ghat_simulacion_sal <- matrix(NaN,N_Stil_j,1)
colnames(ghat_simulacion_sal) <- c("ghat_simulacion")
#   Simulation
sim <- 1
if(N_Stil_j > 0){
    for(sim in 1:N_Stil_j){
        #   Simulated component "g_0"
        #   1) Discrete "ystar"
        #   0) Continuous "g_0"
        ghat_comp_sim <- rbinom(1,1,sum(rho))

        #   Simulation "Y_jl"
        if( ghat_comp_sim == 1){
            #   Simulating the discrete component
            ghat_simulacion_sal[sim] <- gstar_simulacion(rho,ystar,1)
            }else{
            #   Simulating the discrete component
            ghat_simulacion_sal[sim] <- g0_simulacion(g0_licitacion_sal,1)
                 }
                                }
}else{warning("The sample size is incorrect.")}
#   Output
salida <- ghat_simulacion_sal
return(salida)
#   --  END of ghat_simulacion.R --
}

gstar_simulacion <- function(rho,ystar,nSim){
#
#   This function generates Monte Carlo samples from the discrete component of
\eqn{\hat{G}_j} for the Dirichlet process prior in the planned domain \eqn{j}.
#
#   Input:
#       rho                     -   Ux2 dimensional vector with weights associated with
'ystar'
#       ystar                   -   Sample ties 'y^*_i' in 'S_j'
#       phi                     -   Probability weight associated with 'G_{j0}' (the
continuous part of 'Ghat_j')
#
#   Output:
#       gstar_simulacion_sal    - (nSim x 1) dimensional vector with simulated data
#

#   Normalising the weights "rho"
rho_norm <- rho / sum(rho)

#-----------------------------------
#   Verification:
```

```
unicstar <- cbind(ystar,rho_norm)

#----------------------------------
# Generating  the final sample discrete component (with replacement)
gstar_simulacion_sal <- sample(unicstar[,"ystar"], nSim, replace = TRUE, prob = unicstar
[,"rho"])

#   output
salida <- gstar_simulacion_sal
return(salida)

#
#   --  END of gstar_simulacion.R --
}

#
#   Inverse Gaussian distribution
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
#

pinvgauss <- function(q, m, s){
#   Inverse Gaussian distribution - Probability
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
if(any(q<=0)){stop("q must contain positive values")}
if(any(m<=0)){stop("m must be positive")}
if(any(s<=0)){stop("s must be positive")}
t <- q/m
v <- sqrt(q*s)
out <- pnorm((t-1)/v)+exp(2/(m*s))*pnorm(-(t+1)/v)
return(out)
}

dinvgauss <- function(y, m, s, log=FALSE){
#   Inverse Gaussian distribution - Density
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
if(any(y<=0)){stop("y must contain positive values")}
if(any(m<=0)){stop("m must be positive")}
if(any(s<=0)){stop("s must be positive")}
tmp <- -(y-m)^2/(2*y*s*m^2)-(log(2*pi*s)+3*log(y))/2
if(!log){tmp <- exp(tmp)}
return(tmp)
}
```

```r

rinvgauss <- function(n=1, m, s){
#   Inverse Gaussian distribution - Simulation
#
#   Package:    rmutil
#   Version:    1.0
#   Title:      Utilities for Nonlinear Regression and Repeated Measurements Models
#   Author:     Jim Lindsey <jlindsey@luc.ac.be>
temp <- qinvgauss(runif(n),m=m,s=s)
return(temp)
}

pesosDP <- function(unicos_sal,alphaDP){
#
#   This function computes the weights associated with the predictive distribution
\eqn{\hat{G}_j} for a given planned domain \eqn{j},
#   using the sample ties \eqn{(y^{*}j)}, under the Dircihlet processes prior.
#
#   Input:
#       unicos_sal  -   (U x 2) matrix with sample ties and associated frequencies
(produced with 'unicos')
#       alphaDP     -   Positive scalar for the Dirichlet process
#
#   Details:
#       The matrix with sample ties 'unicos_sal' must include a column named 'm_k' for
the frequencies of the sample ties.
#
#   Output:
#       rho     -   (U x 2) dimensional vector with weights associated with 'ystar'
(the sample ties)
#       phi     -   Probability weight associated with 'G_{j0}' (the continuous part of
'Ghat_j')
#

# computing "rho" and "phi"
unicstar <- unicos_sal[[2]]
U <- unicos_sal[[1]]
rho <- NaN * unicstar[,"m"]
M <- sum(unicstar[,"m"])
for(l in 1:U){
    rho[l] <- unicstar[l,"m"]/(M+alphaDP)
    }
phi <- as.matrix(alphaDP/(M+alphaDP))
rho <- as.matrix(rho)
colnames(rho) <- c("rho")
colnames(phi) <- c("phi")

#   Output
salida <- list(rho,phi)
return(salida)

#
#  --  END of pesosDP.R  --
}
```

```r

randDirichlet <- function(alpha,n){
#
#   This function simulates samples from the Dirichlet distribution with (px1) vector
parameter \eqn{\alpha}.
#
#   Input:
#       alpha   -   p-dimensional vector with positive entries
#       n       -   Number of simulated replicates
#
#   Output:
#       randDir -   (Nxp) matrix with 'n' simulated replicates
#
#   References:
#   -   "Non-Uniform Random Variate Generation", Berlin: Springer-Verlag. Devrog, Luz
(1986)
#

#   Dimension "p"
p <- length(alpha)

#   Repository
randDir <- matrix(NaN,n,p)
pAux <- NaN * rep(1:p)

#   Simulation
for(i in 1:n)
{
    for(k in 1:p)
    {
        pAux[k] <- rgamma(1, shape = alpha[k], scale = 1)
    }
    randDir[i,] <- pAux / sum(pAux)
}

#   Output
salida <- randDir
return(salida)

#
#   --  END of randDirichlet.R --
}

unicstar <- function(datos_j){
#
#   This function identifies the sample ties \eqn{(y^{*}_k)} in \eqn{\mathcal{S}_j} and
computes their associated frequencies.
#
#   Input:
#       datos_j     -   Data matrix with features and number of individuals in the
sample 'S_j'
#
#   Output. The function 'unicstar' produces an object list with two elements:
#       U           -   Number of sample ties in 'S_j'
#       unicstar    -   Data matrix with sample ties 'y_i' and associated frequencies
#

y_i <- as.data.frame(datos_j)
y_i <- as.data.frame(datos_j[,"y_i"])
colnames(y_i) <- c("y_i")

#   Identifying sample ties
ystar <- sort(as.matrix(unique(y_i)))
unicstar <- cbind(ystar,ystar)
colnames(unicstar) <- c("ystar","m")
dimUnicos <- dim(unicstar)

#   Computing frequencies
k <- 1
for(k in 1:dimUnicos[1]){
    ids <- which(datos_j[,"y_i"]==unicstar[k,"ystar"])
    unicstar[k,"m"] <- sum(datos_j[ids,"n_i"])
}
#   Counting the number sample ties
U <- dimUnicos[1]

#   Output
salida <- list(U,unicstar)
return(salida)

#
#  --   END of unicstar.R  --
}

```