Supplement 5.1. R code for GRS sampling

Trent McDonald, Western EcoSystems Technology, Inc., Cheyenne, Wyoming, USA.

This supplement contains two R functions for implementing the General Random Sample (GRS) algorithm outlined in Chapter 7: one to draw a General Random Sample (GRS), the other to approximate second-order inclusion probabilities for a GRS. Inputs, outputs, and example calls are contained in the header comments of both functions. The functions are identical to those presented in Appendix 5.1 of Chapter 5.

General Random Sample Function

```
F.grs <- function(id, n, x=NULL, sort.by=NULL, seed=NULL){
#
#   F.grs - R function for drawing a General Random Sample.
#
#   Inputs:
#   id= either a scalar or vector.  If id is a scalar, it is
#       assumed to be population size (N), and an equal
#       probability sample of size n is drawn from a population
#       of size N. The ID's in this case are 1:N. If id is a
#       vector, the values in ID are used as ID values for
#       units in the population. Population size is the length
#       of id in this case.
#   n = scalar, the desired sample size.
#   x = a vector of weights to which inclusion probabilities are
proportional.
#       Length of x must be same as length of id.  Or,
#       x=NULL if id is a scalar.
#   sort.by = a vector parallel to id and x used to sort units
#       in the population. If sort.by is null, no sorting is
#       performed.
#   seed = an integer scalar to use for the initial random seed
#       If missing, set.seed is not called.
#
#   Output:
#   A data frame with dimension nx2 containing the general random
#   sample. The data frame contains the variables $id = ID's of
#   units in the sample and $pi = inclusion probability for units
#   in the sample.
#
#   Examples:
#   s <- F.grs(100,20,sort.by=runif(100)) # draw a simple random
#                                         # sample of 20 from 100
```

```
#   s <- F.grs(100,20)  # draw simple systematic sample of 20 from 100
#   s <- F.grs(1:100,20,1:100)  # draw systematic sample of units
#                               # with ID's 1 to 100 and probabilities
#                               # proportional to ID values
#   s <- F.grs(1:100,20,1:100,runif(100))  # draw randomized sample
#                                          # of with probability
#                                          # proportional to ID values

    if( length(id) <= 1){
        id <- 1:id
        x <- rep(1,length(id))
    }
    if( length(x) <= 0 & is.null(x)){
        x <- rep(1, length(id))
    }
    if( !is.null(sort.by) ){
        ind <- order(sort.by)
        id <- id[ind]
        x <- x[ind]
    }
    if( !is.null(seed) ){
        set.seed( seed )
    }


    p <- n * x / sum(x)

    while( any( p > 1) ){
        gt.1 <- p > 1
        p[gt.1] <- 1
        p[!gt.1] <- (n - sum(gt.1)) * x[!gt.1] / sum(x[!gt.1])
    }

    x <- cumsum(p)
    m <- seq(runif(1), n, by=1)
    ind <- findInterval(m,x)+1
    s <- id[ ind ]
    p.s <- p[ ind ]

    ans <- data.frame( id = s, inclusion.prob = p.s )
    ans
}
```

Function to Generate 2nd-order Inclusion Probabilities

```
F.2nd.order <- function(id, n, x=NULL, randomize=FALSE, seed=NULL,
reps=1000){
#
#   F.2nd.order - R function to approximate 2nd order inclusion
#   probabilities of a General Random Sample.
#
#   Inputs:
#   id= either a scalar or vector.  If id is a scalar, it is
#       assumed to be population size (N), and a equal
#       probability sample of size n is drawn from a population
```

```
#       of size N. The ID's in this case are 1:N. If id is a
#       vector, the values in ID are used as ID values for
#       units in the population. Population size is the length
#       of id in this case.
#   n = scalar, the desired sample size.
#   x = a vector of weights to which inclusion probabilities are
        proportional. Length of x must be same as length of id.  Or,
#       x=NULL if id is a scalar.
#   randomize = a boolean value specifying whether or not
#       to randomize the population before drawing the
#       sample.  TRUE = randomize, FALSE = do not randomize.
#   seed = an integer scalar to use for the initial random seed
#   reps = scalar specifying the number of iterations to
#       do in the simulation.
#
#   Output:
#   A symmetric matrix of size NxN containing 2nd order inclusion
#   probabilities.  Cell (i,j) is the approximate probability
#   that unit i and unit j both appear in the sample.
#
#   Note: Inputs id, n, and x are exactly the same as the
#   inputs to F.grs. To implement a design ordered by another
#   variable, sort the vectors id and x before calling
#   this routine.
#   Examples:
#   pi2 <- F.2nd.order(100,20,randomize=T)# 2nd order inclusion
#                                         # probs for simple
#                                         # random sample.
#   pi2 <- F.2nd.order(100,20)  # 2nd order probs for systematic
#   pi2 <- F.2nd.order(1:100,20,1:100) # 2nd order probs for
#                               # unequal prob systematic
#   pi2 <- F.2nd.order(1:100,20,1:100,T)  # 2nd order probs
#                               # for randomized unequal
#                               # prob design.

    if( length(id) <= 1){
        id <- 1:id
        x <- rep(1,length(id))
    }
    N <- length(id)
    pi2 <- matrix(0,nrow=N,ncol=N)
    id.ord <- sort(id)

    for( i in 1:reps ){
        if( randomize ){
            s <- F.grs(id, n, x, sort.by=runif(N))
        } else {
            s <- F.grs(id, n, x)
        }
        ind <- id.ord %in% s$id
        ind <- matrix(ind, ncol=1) %*% matrix(ind, nrow=1)
        pi2 <- pi2 + ind
    }

    pi2 <- pi2 / reps
    pi2
}
```