Adaptive Sampling simulations

Corey Williams

2024-05-09

Contents

Motivation for adaptive sampling

In many scenarios we would like to be able to adaptively increase sampling effort when certain observed values are of interest. This would be especially beneficial in the case where we are try to observe rare events that are likely to be clustered together. For example a rare plant species that has very particular growing conditions. This would be the case where it may be very rare to see the plant but if we see one we see 100 and it is desired to be able to sample all of the plants in the cluster.

Simulations

First I'll write a few functions that I think will be helpful in making this adaptive sampling demonstration. I want to recreate the example in Adaptive cluster Sampling (1990) where there are points distributed in clusters in an area. The area is split into a grid, then grid cells are chosen as the primary sampling unit.

Goals:

- choose a number of clusters
  - easy enough, this can just be rpois(1,n), this will choose a number of clusters randomly based on some mean. We could use any discrete distribution to decide this
- generate points in those clusters
  - decide how many points are in each cluster, this it is probably actually most appropriate to use a poisson distribution since it will be a count.
- at this step I also need to choose the locations of the centers, runif(ncenters,0,20)
- I can then generate points normally distributed around these centers using rnorm(ncenters*2,mu,sd)
- choose a sample of initial grid cells
  - sample(1:ncells,n)
- expand if the cell next to it is occupied
  - need a list of currently occupied cells and a way to determine adjacent cells. maybe just a vector called occupied
  - function takes vector of neighbours, checks against vector of occupied, returns neighbors in occupied

Creating functions for sampling

```
require(tidyverse)
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# install.packages("ggridges")
require(ggridges)
```

```
## Loading required package: ggridges
```

```
# install.packages("colorspace")
require(colorspace)
```

```
## Loading required package: colorspace
```

make clusters

```r
make_clusters<-function(grid_size=20, nclusters=3, avg_size=20, force_inclusion=FALSE){
  # create a list of clusters.
  centers<-split(runif(nclusters*2,0,grid_size),seq(nclusters))
  # get the three cluster sizes
  sizes<-rpois(3,avg_size)
  # get list of matrices of the centers
  center_dfs<-map2(sizes,centers,
                   ~ kronecker(matrix(rep(1,.x),ncol=1), matrix(.y,ncol=2)))
                             # I actually used the kronecker product holy crap!

  if(force_inclusion==T){
    # get coordinates of locations
    locations<-sizes %>% # use sizes
      map( ~data.frame(matrix(rnorm(.x*2),ncol=2))) %>% # make list of df of changes from centers
      map2(.y=center_dfs, ~.x+.y) %>% # add the changes on to the center
      map2(.y=seq(nclusters), ~ mutate(.x, Group=paste("Cluster",.y)))%>%
      bind_rows()
    colnames(locations)<-c("X","Y","Group")
    # force points to snap inside if they were generated outside. This means some
    # tiles will be more densely clustered at edges when a center is near an edge
    # should help maintain the average number of points though
    locations<-locations%>%
      mutate(X=ifelse(X>grid_size,grid_size,X))%>%
      mutate(X=ifelse(X<0, 0, X))%>%
      mutate(Y=ifelse(Y>grid_size,grid_size,Y))%>%
      mutate(Y=ifelse(Y<0, 0, Y))

  }else{
    # get coordinates of locations
    locations<-sizes %>% # use sizes
      map(~data.frame(matrix(rnorm(.x*2),ncol=2))) %>% # make list of df of changes from centers
      map2(.y=center_dfs,~.x+.y) %>% # add the changes on to the center
      map2(.y=seq(nclusters), ~ mutate(.x, Group=paste("Cluster",.y)))%>%
      bind_rows()
    colnames(locations)<-c("X","Y","Group")
  }
  locations
}
make_clusters(force_inclusion = TRUE)
```

```
##             X        Y     Group
## 1   7.2354176 13.87934 Cluster 1
## 2   6.9230426 14.06817 Cluster 1
## 3   6.2339740 15.14407 Cluster 1
## 4   7.5985982 15.79386 Cluster 1
## 5   7.3701211 15.12030 Cluster 1
## 6   7.7127795 14.40265 Cluster 1
## 7   6.8525749 11.52742 Cluster 1
## 8   6.8649927 14.18105 Cluster 1
## 9   7.0705686 13.56717 Cluster 1
## 10  7.0140592 15.17341 Cluster 1
## 11  5.9844756 14.53277 Cluster 1
## 12  6.6186166 14.05733 Cluster 1
```

```
## 13  6.2897073 14.61303 Cluster 1
## 14  5.7578222 14.00192 Cluster 1
## 15  6.3631624 13.21535 Cluster 1
## 16  6.0848959 16.04614 Cluster 1
## 17  5.6012191 15.02990 Cluster 1
## 18  6.0400932 13.95460 Cluster 1
## 19  6.6770795 15.68947 Cluster 1
## 20 11.7876982 15.21169 Cluster 2
## 21 12.3819285 17.39832 Cluster 2
## 22 14.5709930 16.93818 Cluster 2
## 23 12.7544678 13.56259 Cluster 2
## 24 12.4862175 16.43010 Cluster 2
## 25 13.0000807 15.65734 Cluster 2
## 26 12.5426855 15.02517 Cluster 2
## 27 12.0985125 15.77299 Cluster 2
## 28 12.7760775 16.69902 Cluster 2
## 29 15.1955766 15.39080 Cluster 2
## 30 12.7990597 14.35018 Cluster 2
## 31 12.7498606 15.99575 Cluster 2
## 32 10.3260436 16.16294 Cluster 2
## 33 12.2814138 15.65073 Cluster 2
## 34 11.4654643 14.89374 Cluster 2
## 35 13.7560922 15.31345 Cluster 2
## 36  0.0000000 16.35330 Cluster 3
## 37  0.7710799 15.90485 Cluster 3
## 38  1.8053378 14.17472 Cluster 3
## 39  0.6412762 16.14340 Cluster 3
## 40  1.9882179 16.59248 Cluster 3
## 41  2.0393869 14.84005 Cluster 3
## 42  1.6469785 17.76067 Cluster 3
## 43  1.3575303 15.64542 Cluster 3
## 44  3.5787199 16.89870 Cluster 3
## 45  0.6898551 16.96012 Cluster 3
## 46  0.5725554 15.81204 Cluster 3
## 47  0.8556792 16.57469 Cluster 3
## 48  1.4361975 15.94732 Cluster 3
## 49  0.5203196 16.22623 Cluster 3
## 50  2.7739015 15.76755 Cluster 3
## 51  1.7550820 16.70159 Cluster 3
## 52  2.7831596 15.78786 Cluster 3
## 53  0.4073001 17.01651 Cluster 3
## 54  2.8209497 15.66995 Cluster 3
## 55  0.4443133 16.79267 Cluster 3
## 56  3.4829689 16.16179 Cluster 3
## 57  2.0163230 15.24089 Cluster 3
```

plotting

```
plot_clusters<-function(cluster_df, grid_size=20,samp=NULL){
  # cluster_df is the set of points on the grid
  # samp is the set of tiles that were sampled
  # create a plot of the clusters
```

```
  p<-ggplot(cluster_df, aes(x=X, y=Y, color=Group))+
    geom_point()+
    scale_x_continuous(breaks=seq(grid_size))+
    scale_y_continuous(breaks=seq(grid_size))+
    coord_cartesian(xlim=c(0,grid_size), ylim=c(0,grid_size))
  if(!is.null(samp)){
    # this will highlight tiles that are sampled
    # samp is the dataframe containing the coordinates for tiles
    p<-p+geom_rect(data=samp,
                   aes( xmin=X-1, xmax=X, ymin=Y-1, ymax=Y),
                   color="black",
                   fill=NA)
  }
  plot(p)
}
```

Check whether a tile is occupied

```
is_occupied<-function(tile=c(1,1,1),df){
  # check if a tile is occupied tile location is given as c(row,column)
  # give a dataframe with coordinates of point
  # check if there are any points in xrange & yrange at the same time
  #sum((df[,1]<tile[2] & df[,1]>tile[2]-1) * (df[,1]<tile[2] & df[,1]>tile[2]-1))>0

  sum((df[,1]<tile[1]&df[,1]>tile[1]-1) & (df[,2]<tile[2] & df[,2]>tile[2]-1))>0
}
```

find the average in a single tile this will get used in the hansen hurwitz estimator as well. Just including it here so we can have y_k in the simulated data.

```
tile_sum<-function(tile=c(X=1,Y=1,K=1), df){
  # This function returns the y_k for tile k
  # tile is the tile to find the response of
  # df is the dataframe of points on the grid
  sum((df[,1]<tile[1]&df[,1]>tile[1]-1) & (df[,2]<tile[2] & df[,2]>tile[2]-1))
}
```

get the neighbours of a tile

```
get_neighbours<- function(tile=c(X=1,Y=1,k=1), hard_border=TRUE,grid_size=20){
  # gets a list of neighbouring tiles
  neighbours<-list(c(tile[1]-1, tile[2],   tile[3]),
                   c(tile[1],   tile[2]-1, tile[3]),
                   c(tile[1]+1, tile[2],   tile[3]),
                   c(tile[1],   tile[2]+1, tile[3]))
  if(hard_border){
    # do we want to include neighbours outside of the grid?
    # returns the neighbours that are only within the border
    neighbours<-neighbours[map_lgl(neighbours, ~prod(c(.x>0,.x<=grid_size)))]
```

```
  }
  # returns list of neighbours for a given tile
  neighbours
}
```

choose tiles for sample

```
get_tiles<-function(grid_size=20,n1=10,...){
  # this function returns a sample of n1 tiles using a square grid_size grid
  # get a tile
  tiles<-sample(1:grid_size^2,n1,...)
  # convert tile numbers into X and Y
  samp<-data.frame(X=tiles%%grid_size, Y=(tiles-1)%/%grid_size+1)
  samp$X[samp$X==0]<-grid_size
  samp$k<-1:n1
  samp
}
```

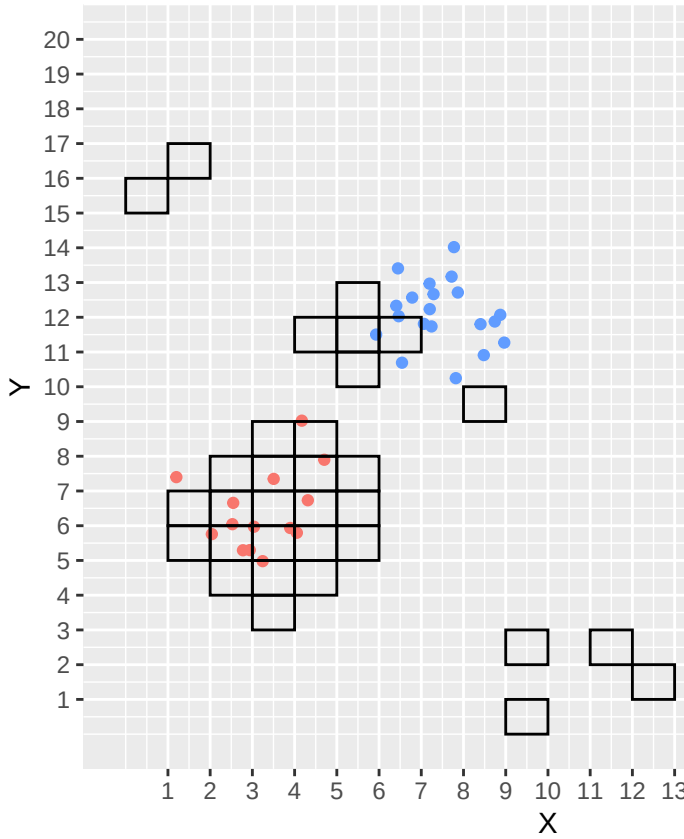
Performing adaptive cluster sampling

One sample from a population of points

```
simulate_one<-function(nclusters=3, grid_size=20,n1=10,force_inclusion=FALSE, hard_border=TRUE,... ){
  # generate clusters
  points<-make_clusters(force_inclusion = force_inclusion)
  # choose the starting grid cells
  sample_tiles<-get_tiles(n1=10,...)
  # save a copy to check against for updates
  temp<-sample_tiles
  # check whether or not they are occupied based on the clusters
  occupied<-apply(sample_tiles,1,function(x) is_occupied(x, points))
  # find the neighbours of the occupied points
  neighbours<-apply(sample_tiles[occupied,], 1, function(x) get_neighbours(x,hard_border = TRUE))%>% #
    bind_rows() %>% # turn list of neighbours into tibble
    as.data.frame() # into dataframe
  # update sample tiles to include neighbours
  sample_tiles<-rbind(sample_tiles,neighbours) %>%
    unique()

  # keep looping until sample_tiles does not grow
  while(dim(temp)[1]!=dim(sample_tiles)[1]){
    # save a copy to check against for updates
    temp<-sample_tiles
    # check whether or not they are occupied based on the clusters
    occupied<-apply(sample_tiles,1,function(x) is_occupied(x,points))
    # find the neighbours of the occupied points
    neighbours<-apply(sample_tiles[occupied,], 1, function(x) get_neighbours(x,hard_border = TRUE))%>%
      bind_rows() %>% # turn list of neighbours into tibble
      as.data.frame() # into dataframe
```

```
    # update sample tiles to include neighbours
    sample_tiles<-sample_tiles %>%
      bind_rows(neighbours) %>%
      unique()
  }

  # get the values of the response for each unit in the sample
  sample_tiles$y_k<-apply(sample_tiles,1,function(x) tile_sum(x, points))
  # get the number of units in each network m_k and add occupied to sample_tiles
  sample_tiles<-sample_tiles%>%
    cbind(occupied)%>%
    group_by(k)%>%
    mutate(m_k=sum(occupied))
  # minimum for m_k is 1 not 0, set all 0s to 1
  sample_tiles$m_k[sample_tiles$m_k==0]<-1

  return(list(sample_tiles=sample_tiles,
              points=points,
              nclusters=nclusters,
              grid_size=grid_size,
              n1=n1))
}

set.seed(pi)
sim_data<-simulate_one()
```

```
set.seed(pi)
sample_one<-simulate_one(n1=10,nclusters=3, grid_size=20,force_inclusion=TRUE)
plot_clusters(sample_one$points,samp=sample_one$sample_tiles)
```



Demonstrating the function works to generate a sample

Multiple samples from the same population of points

Goals: - take m samples from a population - return output in the same format as `simulate_one` but since there are m samples there will be a list of lists

```
# m numer of samples from the same population
simulate_m<-function(m=10,nclusters=3, grid_size=20,n1=10,force_inclusion=FALSE, hard_border=TRUE,...
  results<-list()
  # generate clusters
  points<-make_clusters(force_inclusion = force_inclusion)
  for(i in 1:m){ # loop for the m samples
    # this is just the code from simulate_one() see that for more comments
    sample_tiles<-get_tiles(n1=10,...)
    temp<-sample_tiles
    occupied<-apply(sample_tiles,1,function(x) is_occupied(x,points))

    neighbours<-apply(sample_tiles[occupied,], 1, function(x) get_neighbours(x,hard_border = hard_borde
      bind_rows() %>%
      as.data.frame()

    sample_tiles<-rbind(sample_tiles,neighbours) %>%
      unique()
     # keep looping until sample_tiles does not grow
    while(dim(temp)[1]!=dim(sample_tiles)[1]){
      # save a copy to check against for updates
```

```
      temp<-sample_tiles
      # check whether or not they are occupied based on the clusters
      occupied<-apply(sample_tiles,1,function(x) is_occupied(x,points))
      # find the neighbours of the occupied points
      neighbours<-apply(sample_tiles[occupied,], 1, function(x)get_neighbours(x,hard_border = hard_borde
        bind_rows() %>% # turn list of neighbours into tibble
        as.data.frame() # into dataframe
      # update sample tiles to include neighbours
      sample_tiles<-sample_tiles %>%
        bind_rows(neighbours) %>%
        unique()
    }

    # get the values of the response for each unit in the sample
    sample_tiles$y_k<-apply(sample_tiles,1,function(x) tile_sum(x, points))
    # get the number of units in each network m_k and add occupied to sample_tiles
    sample_tiles<-sample_tiles%>%
      cbind(occupied)%>%
      group_by(k)%>%
      mutate(m_k=sum(occupied))
    # minimum for m_k is 1 not 0, set all 0s to 1
    sample_tiles$m_k[sample_tiles$m_k==0]<-1

    results[[i]]<-list(sample_tiles=sample_tiles,
                                  points=points,
                                  nclusters=nclusters,
                                  grid_size=grid_size,
                                  n1=n1)
  }
  return(results)
}

set.seed(pi)
sim_data_m<-simulate_m()
```

When trying to write this simulation the largest issue I ran in to was determining how to treat points that lie outside of the grid. Should they be included in the sample? should they be ignored? should I change the generating mechanism to force points to be bounded by the edges? There are lots of things you need to specify in regards to how the points are generated. This seems like it could also be an issue that comes up in practical situations as well, for example the case where you have a defined area you are allowed to collect samples from but the thing you are measuring can occur up to and outside of that area. It seems like we would be underestimating the average number in the greater population if those are excluded from the cluster since it means we are underestimating the cluster size.

Finding the HT and HH Estimators

Vocabulary

- neighborhood: the collection of units that are immediately included in the sample if a given unit is included. This relationship is symmetric and is typically (but not necessarily) geographic.
- Cluster: the collection of all the units that are observed under the design as a result of initial selection of unit i

- Network: selection of any unit within the network would lead to the inclusion in the sample of every other unit in the network.
- edge unit: any unit not satisfying the condition but in the neighborhood of one that does

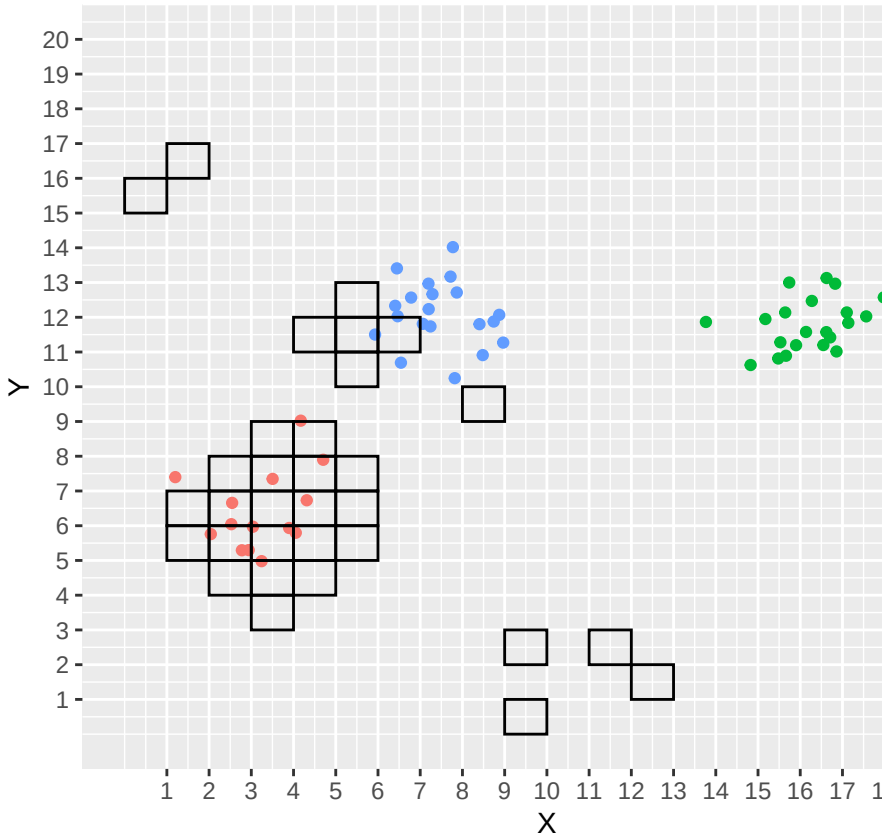
Hansen-Hurwitz

- Let Ψ_k be the network that includes unit k and m_k be the number of units in that network. Any unit not satisfying the criterion is size 1.
- Let $\bar{y}_k^* = (m_k)^{-1} \sum_{j \in \Psi_k} y_j$ represent the average number of observations in the network that includes the kth unit of the initial sample
- The modified Hansen-Hurwitz estimators given as $t_{HH^*} = n_1^{-1} \sum_{k=1}^{n_1} \bar{y}_k^*$

Steps

- Find average number of response in each network
  - file the average number of the response in a single tile.
- compute estimator using the averages

```
modified_HH<-function(sim_data,plot=F,n1=10,...){
  # Find the hansen hurwitz estimator of a sample called sim_data
  if(plot==T){ # plot sample
    plot_clusters(sim_data$points,samp=sim_data$sample_tiles)
  }

  # sample_one$points and sample_one$sample_tiles
  # get the response values for each tile sampled
  y_k<-sim_data$sample_tiles$y_k
  # find the sum of each network that a unit belongs to this returns
  # k and the mean
  temp<-data.frame(y_k,group=sim_data$sample_tiles$k) %>%
    group_by(group)%>%
    filter(y_k>0) %>%
    summarize(network_means=mean(y_k))

  means<-data.frame(init_sample=1:n1,network_means=rep(0,n1))
  means[temp$group,2]<-temp$network_means
  mean(means$network_means)
}

modified_HH(sim_data,plot=T)
```



Applying the function to a single sample

```
## [1] 0.25
```

Hovritz-Thompson

The classic Horvitz-Thompson estimator is given by dividing each y-value by the associated inclusion probability and is an unbiased estimator of the population mean. This is not viable in adaptive cluster sampling as the inclusion probabilities for all units are not known. None-the-less we can still create an unbiased estimator by modifying the Horvitz-Thompson estimator. First we define

$$\alpha_k^* = 1 - \binom{N - m_k}{n_1} \Big/ \binom{N}{n_1}$$

Where m_k is the number of units in the network that includes unit k, N is the number of units in the population, and n_1 is the number of units in the initial sample. Next let

$$J_k = \begin{cases} 0 & \text{If the condition is not satisfied} \\ 1 & \text{Otherwise} \end{cases}$$

Then the modified estimator is given by

$$t_{HT^*} = N^{-1} \sum_{k=1}^{v} y_k J_k / \alpha_k^*$$

where v is the number of distinct units in the sample

Steps

- Find α_k^* for each unit in the sample
- Find $y_k J_k / \alpha_k^*$ for each unit
- Find t_{HT^*}

α_k^*

$$\alpha_k^* = 1 - \binom{N - m_k}{n_1} / \binom{N}{n_1}$$

```
alpha_k<-function(sim_data){
  # get the response values for each tile sampled
  y_k<-sim_data$sample_tiles$y_k
  n1<-sim_data$n1
  N<-sim_data$grid_size^2
  1-choose(N-sim_data$sample_tiles$m_k, sim_data$n1)/choose(N,sim_data$n1)
}
alpha_k(sim_data)
```

```
##  [1] 0.0250000 0.0250000 0.0250000 0.0250000 0.0250000 0.1848315 0.0250000
##  [8] 0.0250000 0.0250000 0.0250000 0.0250000 0.0250000 0.0250000 0.0250000
## [15] 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315
## [22] 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315
## [29] 0.1848315 0.1848315 0.1848315 0.1848315 0.1848315
```

Finding the estimator

$$t_{HT^*} = N^{-1} \sum_{k=1}^{v} y_k J_k / \alpha_k^*$$

```
modified_TH<-function(sim_data,plot=F){
  if(plot==T){
    plot_clusters(sim_data$points,samp=sim_data$sample_tiles)
  }
  a_k<-alpha_k(sim_data)
  N<-sim_data$grid_size^2
  y_k<-sim_data$sample_tiles$y_k
  J_k<-sim_data$sample_tiles$occupied
  1/N*sum(y_k*J_k/a_k)
}

modified_TH(sim_data)
```

```
## [1] 0.26231
```

Running simulations

Initial results, Hard border, inclusion not forced

This was the first simulation performed where 1000 populations were sampled once, this limits what we are able to see though as the variation of the sampling method for a given population may also be of interest.

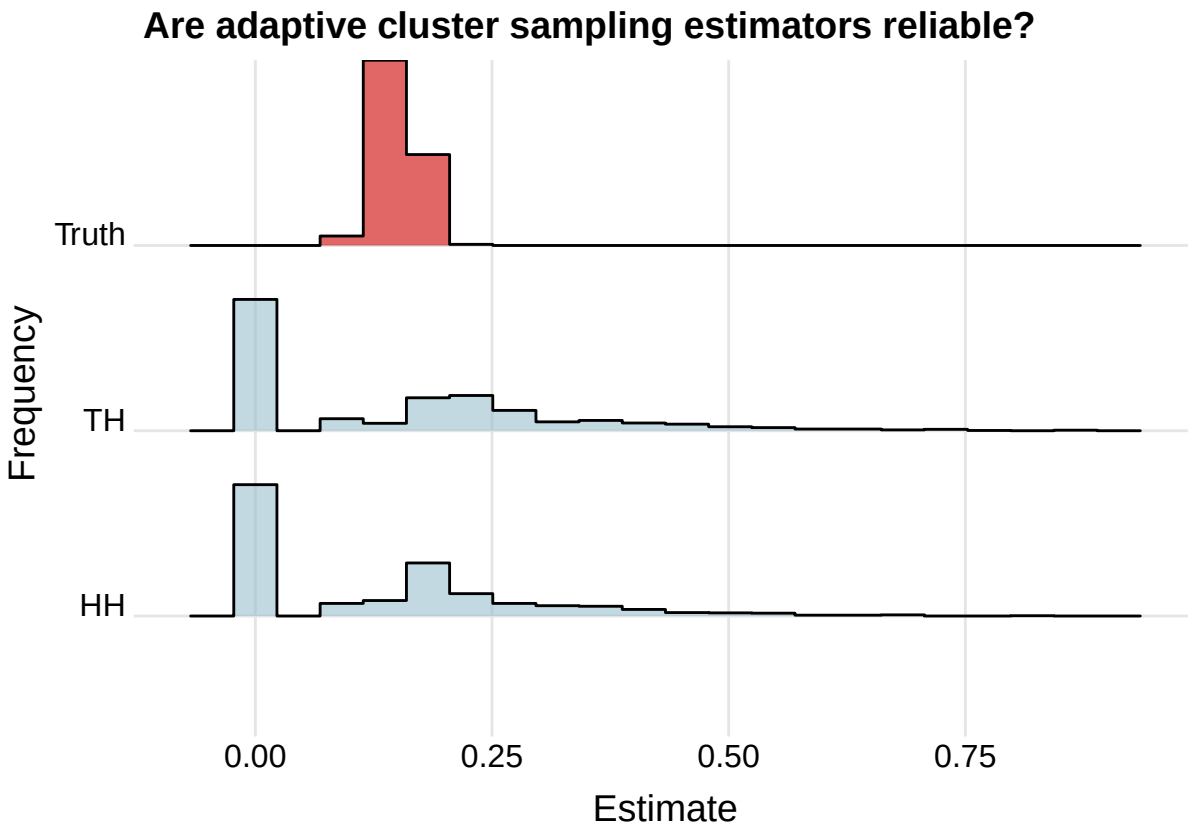
Moving forward with simulations rather than sampling thousands of populations we will only generate 9 populations but sample from them 1000 times each.

```
n_sim<-1000
results<-data.frame(HH=rep(NA,n_sim),
                    TH=rep(NA,n_sim),
                    Truth=rep(NA,n_sim))
for(i in 1:n_sim){
  sim_data<-simulate_one()
  results$HH[i]<-modified_HH(sim_data)
  results$TH[i]<-modified_TH(sim_data)
  results$Truth[i]<-length(sim_data$points$X)/(sim_data$grid_size^2)
}
write.csv(results,file="sampling 1000 populations.csv")
```

```
results<-read_csv("sampling 1000 populations.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
results_long<-results %>%
  pivot_longer(cols = HH:Truth,
               names_to = "Estimator",
               values_to = "Value")

ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
  geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
  theme_ridges()+
  theme(legend.position="none",
        axis.title.y = element_text( hjust = 0.5),
        axis.title.x = element_text( hjust = 0.5))+
  xlab("Estimate") +
  ylab("Frequency")+
  ggtitle("Are adaptive cluster sampling estimators reliable?")+
  scale_fill_manual(values=c("lightblue3","lightblue3","red3"))
```



Inclusion not forced

Hard border when finding neighbours

```
set.seed(pi)
n_samples<-1000
n_pops<-9
# Create blank dataframes for the estimates and the populations of points
results<-data.frame(HH=rep(NA,n_pops*n_samples),
                    TH=rep(NA,n_pops*n_samples),
                    Truth=rep(NA,n_pops*n_samples),
                    Population=rep(NA,n_pops*n_samples))

population_points<-data.frame(X=c(),Y=c(),Group=c(),Population=c())

for(i in 1:n_pops){
  sim_data<-simulate_m(m=n_samples,hard_border = TRUE, force_inclusion = FALSE)
  # save the population points to plot later
  population_points<-population_points%>%
    rbind( cbind(sim_data[[i]]$points,Population=paste("pop",i)) )
  for(j in 1:n_samples){
    # saving estimators into results
    results$HH[(i-1)*n_samples+j]<-modified_HH(sim_data[[j]])
    results$TH[(i-1)*n_samples+j]<-modified_TH(sim_data[[j]])
```

```
    results$Truth[(i-1)*n_samples+j]<-length(sim_data[[j]]$points$X)/(sim_data[[j]]$grid_size^2)
    results$Population[(i-1)*n_samples+j]<-paste("Population",i)
  }
}

# save end results of simulation
write.csv(results,file="estimates-inclusion_not_forced-hard_border.csv")
write.csv(population_points,file="pop_points-inclusion_not_forced-hard_border.csv")
```

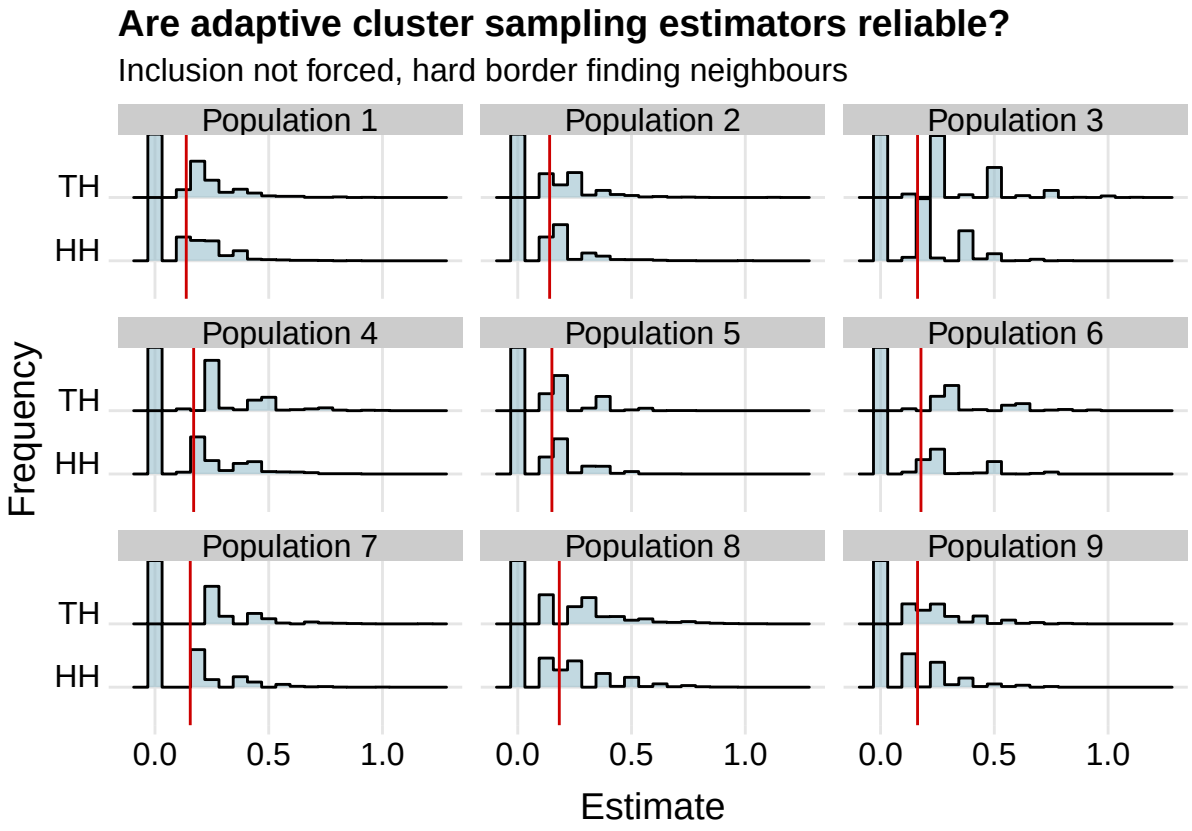
```
# read in results from simulation above
results<-read_csv("estimates-inclusion_not_forced-hard_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

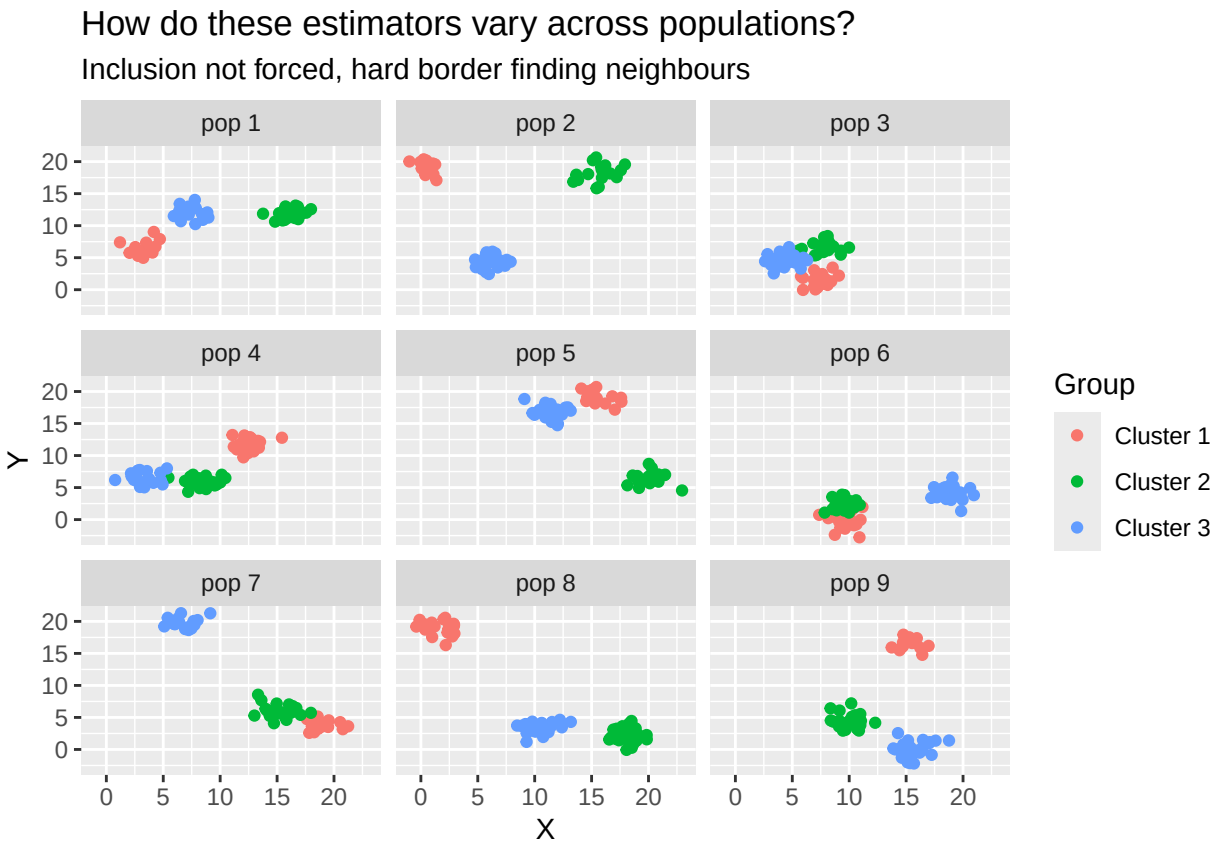
```
results_long<-results%>%
  pivot_longer(cols = HH:TH,
               names_to = "Estimator",
               values_to = "Value")

population_points<-read_csv("pop_points-inclusion_not_forced-hard_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
# plot the distributions of the estimators across the 9 populations
ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
  geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
  theme_ridges()+
  theme(legend.position="none",
        axis.title.y = element_text( hjust = 0.5),
        axis.title.x = element_text( hjust = 0.5))+
  xlab("Estimate") +
  ylab("Frequency")+
  ggtitle("Are adaptive cluster sampling estimators reliable?",
          subtitle="Inclusion not forced, hard border finding neighbours")+
  scale_fill_manual(values=c("lightblue3","lightblue3"))+
  facet_wrap(~Population)+
  geom_vline(aes(xintercept=Truth),color="red3")
```




```
# plot the 9 populations
ggplot(population_points,aes(x=X,y=Y,color=Group))+
  geom_point()+
  facet_wrap(~Population)+
  ggtitle("How do these estimators vary across populations?",
          subtitle="Inclusion not forced, hard border finding neighbours")
```



Soft border when finding neighbours

```
set.seed(pi)
n_samples<-1000
n_pops<-9
# estimator results
results<-data.frame(HH=rep(NA,n_pops*n_samples),
                    TH=rep(NA,n_pops*n_samples),
                    Truth=rep(NA,n_pops*n_samples),
                    Population=rep(NA,n_pops*n_samples))
# saving the population sampled from
population_points<-data.frame(X=c(),Y=c(),Group=c(),Population=c())

for(i in 1:n_pops){ # generating the populations and samples
  sim_data<-simulate_m(m=n_samples,hard_border = FALSE, force_inclusion = FALSE)
  # save the population points to plot later
  population_points<-population_points%>%
    rbind( cbind(sim_data[[i]]$points,Population=paste("pop",i)) )
  for(j in 1:n_samples){
    # save estimates for each sample
    results$HH[(i-1)*n_samples+j]<-modified_HH(sim_data[[j]])
    results$TH[(i-1)*n_samples+j]<-modified_TH(sim_data[[j]])
    results$Truth[(i-1)*n_samples+j]<-length(sim_data[[j]]$points$X)/(sim_data[[j]]$grid_size^2)
    results$Population[(i-1)*n_samples+j]<-paste("Population",i)
```

```
  }
}
# save estimates of the samples and the populations sampled from
write.csv(results,file="estimates-inclusion_not_forced-soft_border.csv")
write.csv(population_points,file="pop_points-inclusion_not_forced-soft_border.csv")
```

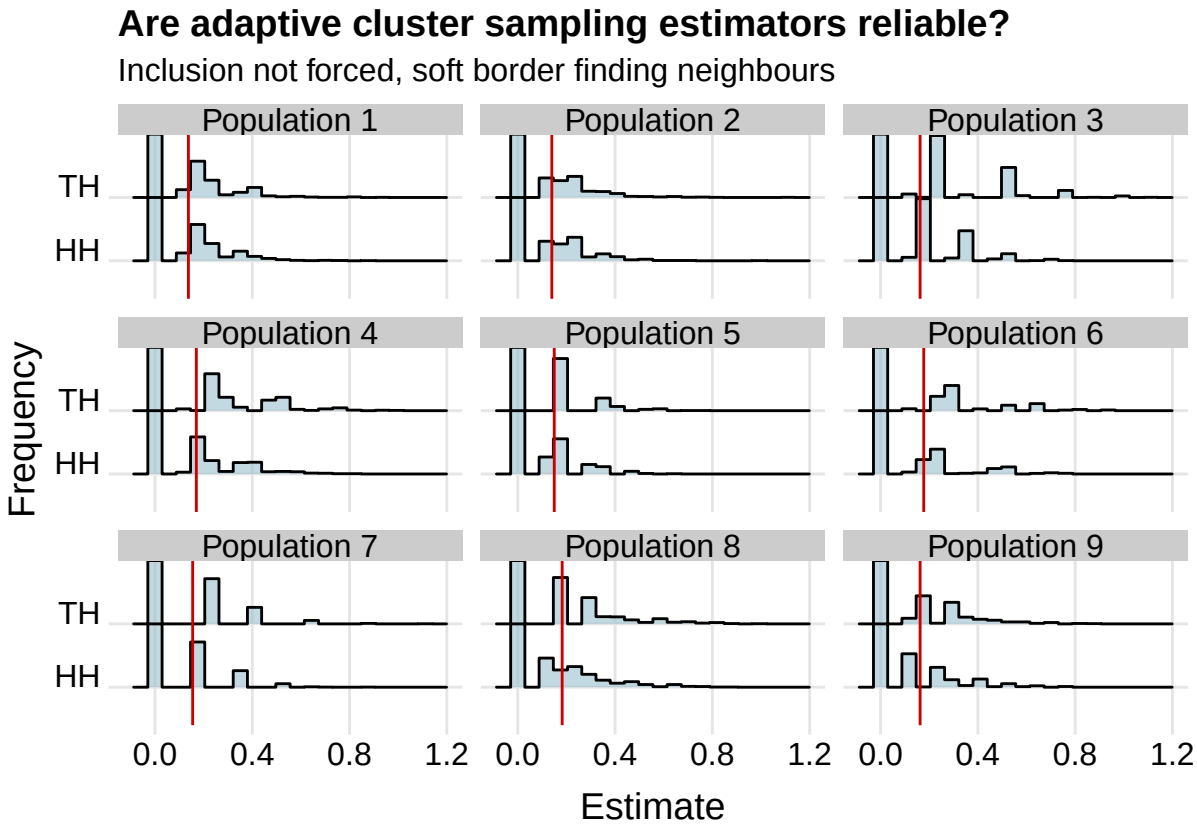
```
results<-read_csv("estimates-inclusion_not_forced-soft_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

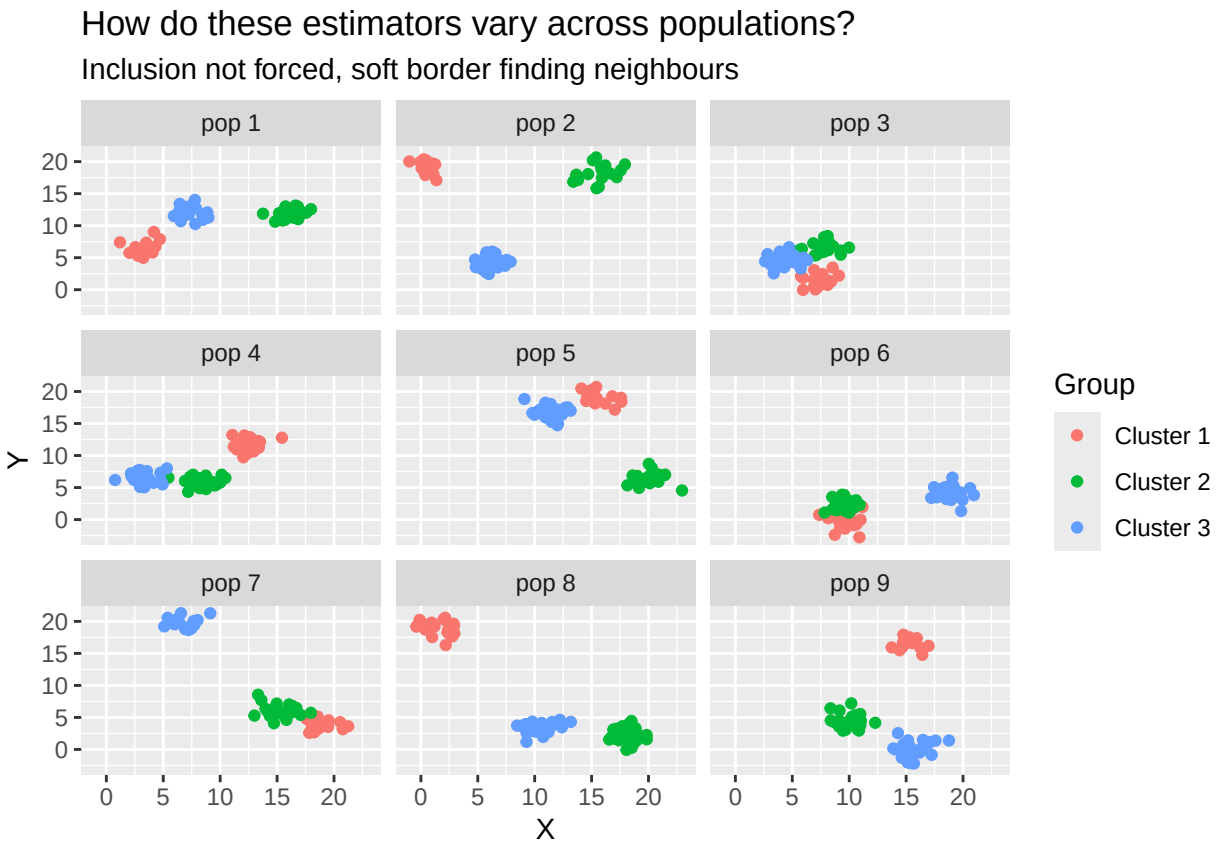
```
results_long<-results%>%
  pivot_longer(cols = HH:TH,
               names_to = "Estimator",
               values_to = "Value")

population_points<-read_csv("pop_points-inclusion_not_forced-soft_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
  geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
  theme_ridges()+
  theme(legend.position="none",
        axis.title.y = element_text( hjust = 0.5),
        axis.title.x = element_text( hjust = 0.5))+
  xlab("Estimate") +
  ylab("Frequency")+
  ggtitle("Are adaptive cluster sampling estimators reliable?",
          subtitle="Inclusion not forced, soft border finding neighbours")+
  scale_fill_manual(values=c("lightblue3","lightblue3"))+
  facet_wrap(~Population)+
  geom_vline(aes(xintercept=Truth),color="red3")
```




```
ggplot(population_points,aes(x=X,y=Y,color=Group))+
  geom_point()+
  facet_wrap(~Population)+
  ggtitle("How do these estimators vary across populations?",
          subtitle="Inclusion not forced, soft border finding neighbours")
```



Inclusion forced

Hard border when finding neighbours

```
set.seed(pi)
n_samples<-1000
n_pops<-9
# Create blank dataframes for the estimates and the populations of points
results<-data.frame(HH=rep(NA,n_pops*n_samples),
                    TH=rep(NA,n_pops*n_samples),
                    Truth=rep(NA,n_pops*n_samples),
                    Population=rep(NA,n_pops*n_samples))

population_points<-data.frame(X=c(),Y=c(),Group=c(),Population=c())

for(i in 1:n_pops){
  sim_data<-simulate_m(m=n_samples,hard_border = TRUE, force_inclusion = TRUE)
  # save the population points to plot later
  population_points<-population_points%>%
    rbind( cbind(sim_data[[i]]$points,Population=paste("pop",i)) )
  for(j in 1:n_samples){
    # saving estimators into results
    results$HH[(i-1)*n_samples+j]<-modified_HH(sim_data[[j]])
    results$TH[(i-1)*n_samples+j]<-modified_TH(sim_data[[j]])
```

```
    results$Truth[(i-1)*n_samples+j]<-length(sim_data[[j]]$points$X)/(sim_data[[j]]$grid_size^2)
    results$Population[(i-1)*n_samples+j]<-paste("Population",i)
  }
}

# save end results of simulation
write.csv(results,file="estimates-inclusion_forced-hard_border.csv")
write.csv(population_points,file="pop_points-inclusion_forced-hard_border.csv")
```

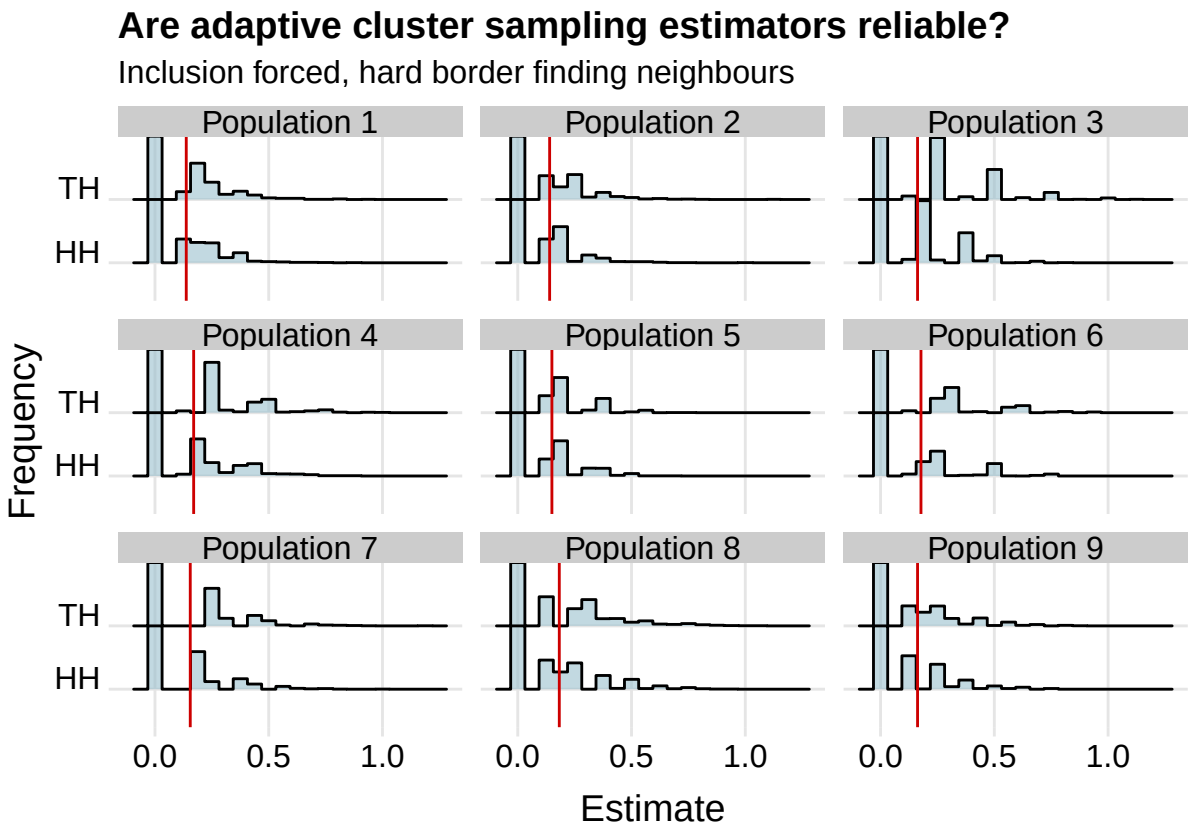
```
# read in results from simulation above
results<-read_csv("estimates-inclusion_forced-hard_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
results_long<-results%>%
  pivot_longer(cols = HH:TH,
               names_to = "Estimator",
               values_to = "Value")

population_points<-read_csv("pop_points-inclusion_forced-hard_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
# plot the distributions of the estimators across the 9 populations
ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
  geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
  theme_ridges()+
  theme(legend.position="none",
        axis.title.y = element_text( hjust = 0.5),
        axis.title.x = element_text( hjust = 0.5))+
  xlab("Estimate") +
  ylab("Frequency")+
  ggtitle("Are adaptive cluster sampling estimators reliable?",
          subtitle="Inclusion forced, hard border finding neighbours")+
  scale_fill_manual(values=c("lightblue3","lightblue3"))+
  facet_wrap(~Population)+
  geom_vline(aes(xintercept=Truth),color="red3")
```




```
# plot the 9 populations
ggplot(population_points,aes(x=X,y=Y,color=Group))+
  geom_point()+
  facet_wrap(~Population)+
  xlim(0,20)+ylim(0,20)+
  ggtitle("How do these estimators vary across populations?",
          subtitle="Inclusion forced, hard border finding neighbours")
```

Soft border when finding neighbours

```
set.seed(pi)
n_samples<-1000
n_pops<-9
# Create blank dataframes for the estimates and the populations of points
results<-data.frame(HH=rep(NA,n_pops*n_samples),
                    TH=rep(NA,n_pops*n_samples),
                    Truth=rep(NA,n_pops*n_samples),
                    Population=rep(NA,n_pops*n_samples))

population_points<-data.frame(X=c(),Y=c(),Group=c(),Population=c())

for(i in 1:n_pops){
  sim_data<-simulate_m(m=n_samples,hard_border = FALSE, force_inclusion = TRUE)
  # save the population points to plot later
  population_points<-population_points%>%
    rbind( cbind(sim_data[[i]]$points,Population=paste("pop",i)) )
  for(j in 1:n_samples){
    # saving estimators into results
    results$HH[(i-1)*n_samples+j]<-modified_HH(sim_data[[j]])
    results$TH[(i-1)*n_samples+j]<-modified_TH(sim_data[[j]])
    results$Truth[(i-1)*n_samples+j]<-length(sim_data[[j]]$points$X)/(sim_data[[j]]$grid_size^2)
    results$Population[(i-1)*n_samples+j]<-paste("Population",i)
```

```
  }
}

# save end results of simulation
write.csv(results,file="estimates-inclusion_forced-soft_border.csv")
write.csv(population_points,file="pop_points-inclusion_forced-soft_border.csv")
```

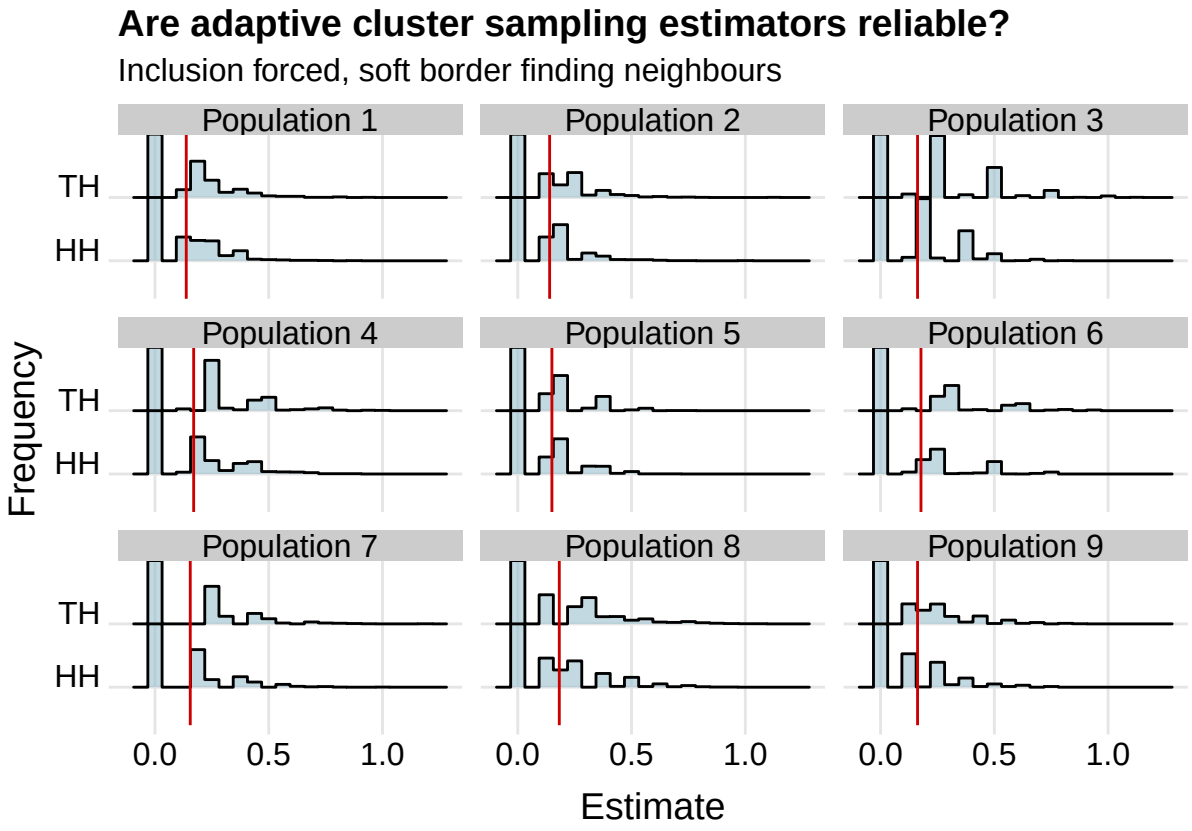
```
# read in results from simulation above
results<-read_csv("estimates-inclusion_forced-soft_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
results_long<-results%>%
  pivot_longer(cols = HH:TH,
               names_to = "Estimator",
               values_to = "Value")

population_points<-read_csv("pop_points-inclusion_forced-soft_border.csv",show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
# plot the distributions of the estimators across the 9 populations
ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
  geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
  theme_ridges()+
  theme(legend.position="none",
        axis.title.y = element_text( hjust = 0.5),
        axis.title.x = element_text( hjust = 0.5))+
  xlab("Estimate") +
  ylab("Frequency")+
  ggtitle("Are adaptive cluster sampling estimators reliable?",
          subtitle="Inclusion forced, soft border finding neighbours")+
  scale_fill_manual(values=c("lightblue3","lightblue3"))+
  facet_wrap(~Population)+
  geom_vline(aes(xintercept=Truth),color="red3")
```

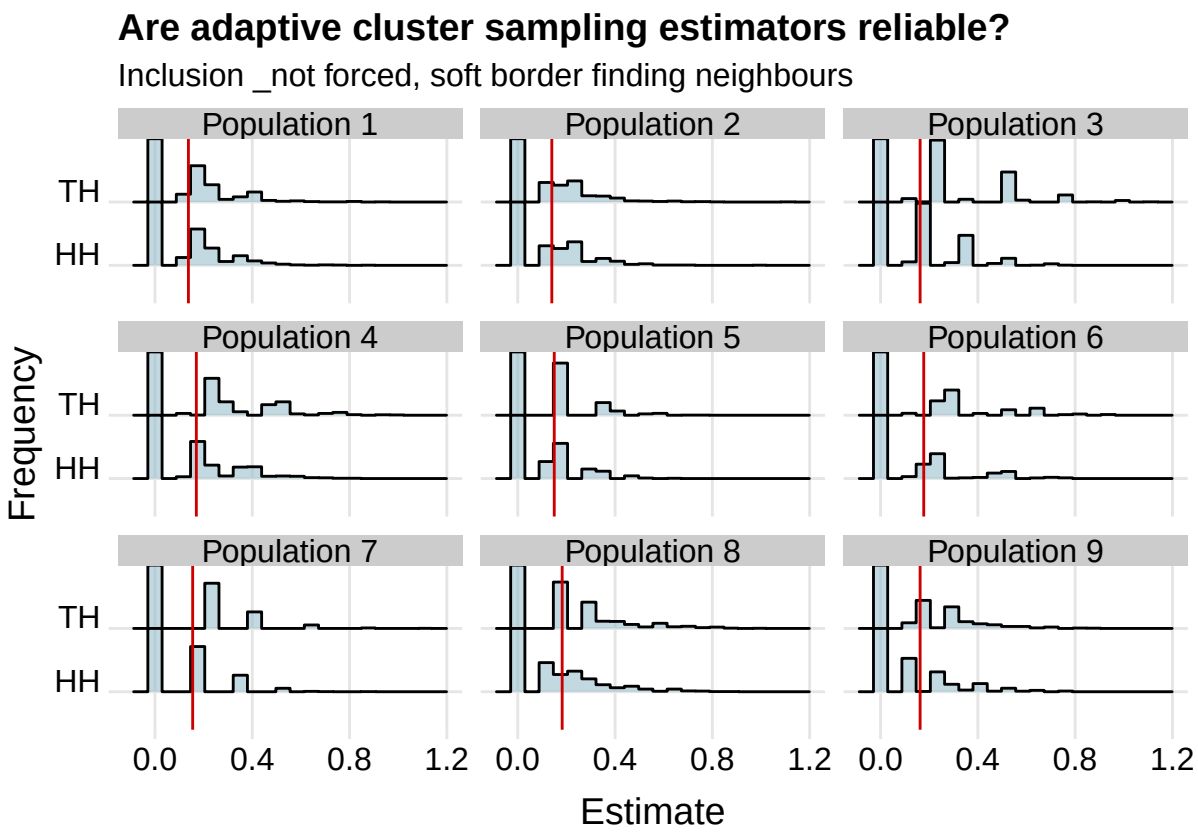



```
# plot the 9 populations
ggplot(population_points,aes(x=X,y=Y,color=Group))+
  geom_point()+
  facet_wrap(~Population)+
  xlim(0,20)+ylim(0,20)+
  ggtitle("How do these estimators vary across populations?",
          subtitle="Inclusion forced, soft border finding neighbours")
```

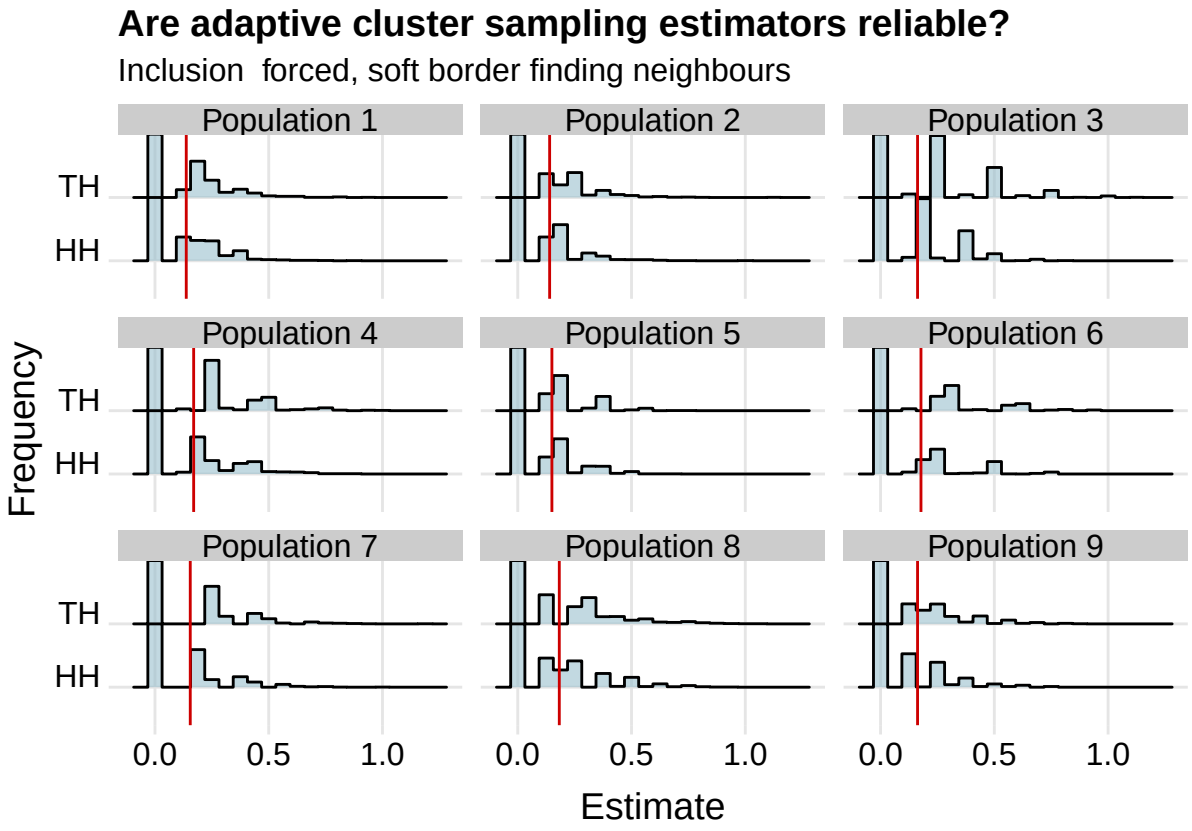

```
for(i in c("soft","hard")){
  for(j in c("_not","")){
    # read in results from simulation above
    results<-read_csv(paste("estimates-inclusion",j,"_forced-",i,"_border.csv",sep=""),
                      show_col_types = FALSE,
                      col_select = HH:Population)
    results_long<-results%>%
      pivot_longer(cols = HH:TH,
                   names_to = "Estimator",
                   values_to = "Value")
       # plot the distributions of the estimators across the 9 populations
    p<-ggplot(results_long,aes(x=Value, y=Estimator, fill=Estimator))+
      geom_density_ridges(alpha=0.6, stat="binline",bins=20,scale=1)+
      theme_ridges()+
      theme(legend.position="none",
            axis.title.y = element_text( hjust = 0.5),
            axis.title.x = element_text( hjust = 0.5))+
      xlab("Estimate") +
      ylab("Frequency")+
      ggtitle("Are adaptive cluster sampling estimators reliable?",
              subtitle=paste("Inclusion",j,"forced,", i,"border finding neighbours"))+
      scale_fill_manual(values=c("lightblue3","lightblue3"))+
      facet_wrap(~Population)+
      geom_vline(aes(xintercept=Truth),color="red3")
    plot(p)
  }
```

```
}
```

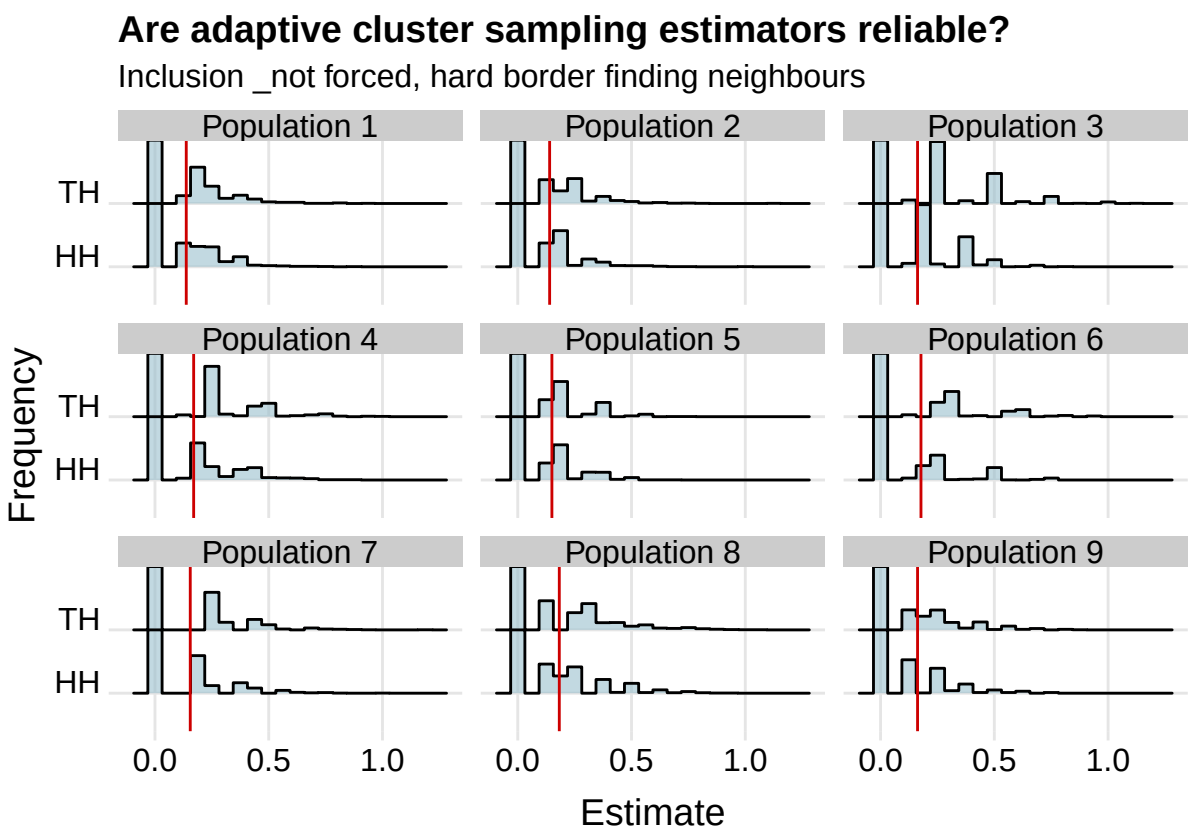
```
## New names:
## New names:
## * `` -> `...1`
```

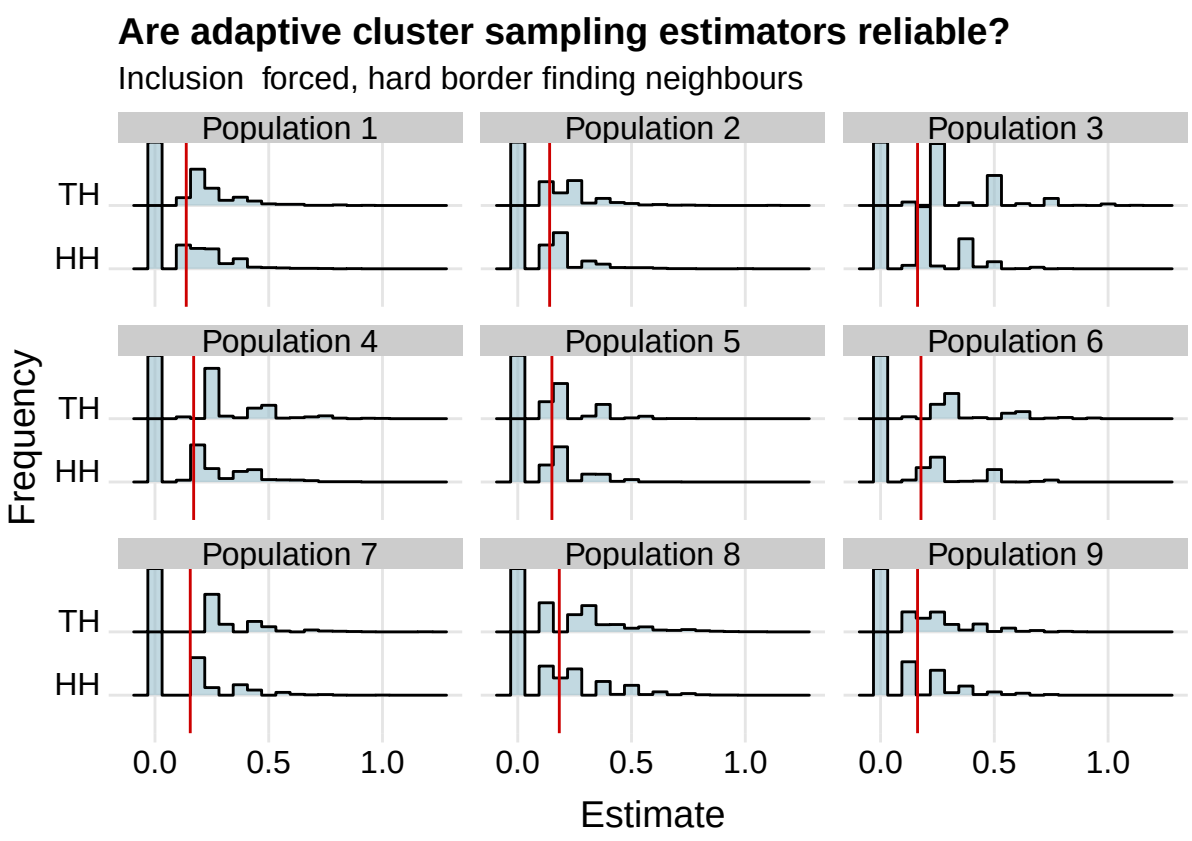



```
## New names:
## * `` -> `...1`
```


Are adaptive cluster sampling estimators reliable?
Inclusion forced, soft border finding neighbours
Population 1
Population 2
Population 3
Population 4
Population 5
Population 6
Population 7
Population 8
Population 9
TH
HH
Frequency
0.0
0.5
1.0
Estimate

```
## New names:
## * `` -> `...1`
```


Are adaptive cluster sampling estimators reliable?
Inclusion _not forced, hard border finding neighbours
Population 1
Population 2
Population 3
Population 4
Population 5
Population 6
Population 7
Population 8
Population 9
TH
HH
Frequency
0.0
0.5
1.0
Estimate


Are adaptive cluster sampling estimators reliable?
Inclusion forced, hard border finding neighbours
Population 1
Population 2
Population 3
Population 4
Population 5
Population 6
Population 7
Population 8
Population 9
TH
HH
Frequency
0.0
0.5
1.0
Estimate