Replication of “The Diffusion of Microfinance”

Kazuki Motohashi, Sakina Shibuya, Mizuhiro Suzuki

2020-10-26

Purpose of this exercise

In this document, we aim to understand the codes used in Banerjee et al. (2013), “The Diffusion of Microfinance” and try to replicate their estimation results. The data and original codes can be downloaded here. The details of their model are not discussed in this document: please refer to the Supplementary material of the paper for details, which can be downloaded with the data and the codes.

Summary of our findings

Despite the successful replication of the intermediate results (network statistics), our replication estimates for the no endorsement model are non-trivially different. We explore potential causes at the end of this document; however, this may be due to that the original Matlab code does not contain a randomization seed which makes exact replication impossible

```r
# Install and load packages ---------------
packages <- c(
  "R.matlab",
  "tidyverse",
  "igraph",
  "pracma",
  "ggplot2",
  "dqrng",
  "Matrix",
  "pander",
  "kableExtra"
)

pacman::p_load(packages, character.only = T)
```

Overview of structural estimation steps

1. For a given parameter θ and each village r, compute $d(r, \theta) = \frac{1}{S}\sum_s m_{sim,r}(s, \theta) - m_{emp,r}$, where $m_{sim,r}(s, \theta)$ is the s'th simulated moments, $m_{emp,r}$ is the empirical moments, and S is the number of simulations. Note that $d(r, \theta)$ is a vector, whose length depends on how many moments are used for estimation (in their main specification, 5 moments are used).
2. Compute $D(\theta) = \frac{1}{R}\sum_r d(r, \theta)$.
3. Calculate $Q(\theta) = D(\theta)'\widehat{W}D(\theta)$, where $\widehat{W}$ is the weight matrix.
4. After calculating $Q(\theta)$ for all θs in the grid of parameter values, find $\theta^* = \arg\min Q(\theta)$.

There are a couple of things worth noting here:

- When conducting bootstrap, in the second step we compute $D(b, \theta) = \frac{1}{R}\omega_r^b \sum_r d(r, \theta)$, where $\omega_r^b = e_{br}/\bar{e}_r$ with $e_{br} \sim \exp(1)$ and $\bar{e}_r = \frac{1}{R}\sum_r e_{br}$, and then we find $\theta^{*b} = \arg\min D(b, \theta)'\widehat{W}D(b, \theta)$. Note that for

bootstrap, the authors do not resample the observations but generate random weights to calculate the objective function.
- The authors take the two-step approach: in the first step, they use an identity matrix for $\widehat{W}$ and estimate $\widehat{\theta}$. In the second step, they calculate $\widehat{W} = \left(\frac{1}{R} \sum_r d(r, \widehat{\theta}) d(r, \widehat{\theta})' \right)^{-1}$.
- The supplementary material details how the authors select the grid of parameters.

Codes used in the paper

In this document, we focus on the codes used for the analysis without the endorsement effect. The R script for the replication of the results with endorsement effect will be uploaded, hopefully soon. The original analyses are conducted using the following Matlab codes:

- `Main_models_1_3.m`: The main code for estimation. This code calls other functions defined in the scripts below;
- `divergence_model.m`: Function to compute the deviation of the empirical moments from the simulated ones;
- `diffusion_model.m`: Function to simulate diffusion of microfinance;
- `moments.m`: Function to calculate moments.

In the original codes, two other files, `breadthdistRAL.m` and `breadth.m`, are used to calculate statistics on the networks such as distances between households. Since we use an R package called `igraph` to derive such network statistics, we will not use the functions defined in these Matlab files.

Global setting

First, we define global settings to run the code below.

```r
# Global setting -------------------
user <- "Hiro"
if (user == "Hiro"){
  project_path <- "/Users/mizuhirosuzuki/Documents/GitHub/MFdiffusion_replication"
}

# random seed
dqRNGkind("Xoroshiro128+")
dqset.seed(453779)
# Which moment conditions to use
case <- 1
# Whether first step or second step (0 = first step, 1 = second step)
twoStepOptimal <- 1
# Number of simulations
S <- 75
# Time span for one period
timeVector <- 'trimesters'
# Model type (1 -> qN = qP, 3 -> qN \ne qP)
modelType <- 3

# Village indices used for analyses ----------------
vills <- c(1:4, 6, 9, 12, 15, 19:21, 23:25, 29, 31:33, 36,
           39, 42, 43, 45:48, 50:52, 55, 57, 59:60, 62,
           64:65, 67:68, 70:73, 75)
num_vills <- length(vills)

# Number of moment conditions
```

```
if (case == 1){
  m <- 5
} else if (case == 2){
  m <- 3
} else if (case == 3){
  m <- 3
} else if (case == 4){
  m <- 3
}

# Select time vector and number of repetitions per trial --------------
# Months
TMonths <- c(31, 35, 15, 35, 13, 2, 32, 5,
             31, 35, 31, 29, 19, 22, 25, 25,
             23, 23, 24, 25, 26, 24, 17, 16,
             17, 13, 19, 20, 20, 19, 22, 14,
             12, 15, 10, 19, 18, 18, 19, 19, 19, 17, 17)

if (timeVector == 'months'){
    t_period <- TMonths + 1
} else if (timeVector == 'quarters'){
    t_period <- ceiling(TMonths / 3) + 1 # Quarters have 3 months in them
} else if (timeVector == 'trimesters'){
    t_period <- ceiling(TMonths / 4) + 1 # Trimesters have 4 months in them
}

# Select parameter grid --------------------

if (modelType == 1){
  qN <- c(seq(0, 0.01, 0.001), seq(0.05, 1, 0.05))
} else if (modelType == 3){
  qN <- c(seq(0, 0.01, 0.001), seq(0.05, 1, 0.05))
  qP <- c(seq(0, 0.1, 0.005), seq(0.15, 1, 0.05))
}
```

There are a couple of things worth noting:

- `modelType` specifies whether different values are used for q_N (probability that an MF non-participant transmits information) and q_P (probability that an MF participant transmits information). In this document we focus on the case where q_N and q_P can differ, which is the case where `modelType = 3`.
- `case` specifies which set of moment conditions are used for estimation. The main set of moments used in the paper is the case where `case = 1`, which uses the following 5 moments:
  - The share of households with no participating neighbors that participate;
  - The share of households in the neighborhood of a participating leader that participate;
  - The share of households in the neighborhood of a nonparticipating leader that participate;
  - The covariance of household participation with the share of its neighbors that participate;
  - The covariance of household participation with the share of its second-degree neighbors that participate.

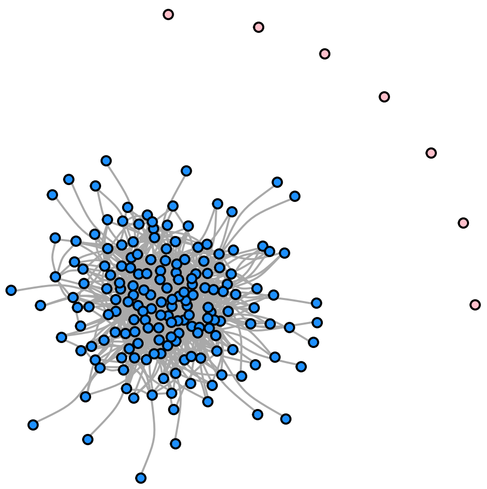
Load data

Here, we load data used in the analysis. One thing that seems not mentioned in the paper or in the Supplementary material is that, in each village, only households that belong to the largest cluster in the village are used. By the largest cluster, we mean the cluster composed of the largest number of sampled households. For example, see the network graph among households in a village 1 below. Each circle indicates a household. Blue circles are households in the biggest cluster, while pink circles are those which are not.

```
i <- 1
vill_rel <- read.csv(file.path(
  project_path,
  paste0(
    "datav4.0/Data/1. Network Data/Adjacency Matrices/adj_allVillageRelationships_HH_vilno_",
    as.character(i),
    ".csv"
    )
  ), header = FALSE)
vill_rel_graph <- graph_from_adjacency_matrix(as.matrix(vill_rel), mode = "undirected")

tempinGiant <- read_tsv(file.path(
  project_path,
  paste0(
    "datav4.0/Matlab Replication/India Networks/inGiant",
    as.character(vills[i]),
    ".csv"
    )
  ), col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double()
## )
```

```
tempinGiant <- as.logical(pull(tempinGiant))

vi <- set_vertex_attr(vill_rel_graph, "inGiant", index = V(vill_rel_graph), tempinGiant)
V(vi)[V(vi)$inGiant == TRUE]$color <- "dodgerblue"
V(vi)[V(vi)$inGiant == FALSE]$color <- "pink"

plot(
  vi,
  layout = layout_with_fr,
  vertex.label = NA,
  vertex.size = 4,
  edge.arrow.size = .1,
  edge.curved = .3)
```



Keeping this in mind, we load the data used in the estimation.

```r
# Load data -------------------------
# Adjacency matrix
X <- readMat(file.path(
  project_path,
  "datav4.0/Matlab Replication/India Networks/adjacencymatrix.mat"
  ))
X <- X$X

# Load other matrices
leaders <- vector("list", length = num_vills)
TakeUp <- vector("list", length = num_vills)
EmpRate <- vector("list", length = num_vills)
inGiant <- vector("list", length = num_vills)
hermits <- vector("list", length = num_vills)
Z <- vector("list", length = num_vills)
TakingLeaders <- vector("list", length = num_vills)
ZLeaders <- vector("list", length = num_vills)
Outcome <- c()
Covars <- tibble()
Sec <- vector("list", length = num_vills)

for (i in seq_along(vills)){
  # Giant-component vectors
  # (ie. vectors of indicators of whether belonging to giant components in each village)
  # (stored in a list)
```

```r
tempinGiant <- read_tsv(file.path(
  project_path,
  paste0(
    "datav4.0/Matlab Replication/India Networks/inGiant",
    as.character(vills[i]),
    ".csv"
    )
  ), col_names = FALSE)
tempinGiant <- as.logical(pull(tempinGiant))
inGiant[[i]] <- tempinGiant

# Leaders vectors (stored in a list)
templeaders <- read_tsv(file.path(
  project_path,
  paste0(
    "datav4.0/Matlab Replication/India Networks/HHhasALeader",
    as.character(vills[i]),
    ".csv"
    )
  ), col_names = FALSE)
templeaders_all <- as.logical(pull(templeaders[,2]))
templeaders <- as.logical(pull(templeaders[tempinGiant,2]))
leaders[[i]] <- templeaders

# Take-up vectors (stored in a list)
tempTakeUp <- read_tsv(file.path(
  project_path,
  paste0(
    "datav4.0/Matlab Replication/India Networks/MF",
    as.character(vills[i]),
    ".csv"
    )
  ), col_names = FALSE)
EmpRate[[i]] <- mean(pull(tempTakeUp[!templeaders_all,]))
tempTakeUp <- as.logical(pull(tempTakeUp[tempinGiant,]))
TakeUp[[i]] <- tempTakeUp

# Hermits (isolated HHs)
d <- rowSums(X[[i]][[1]]) # number of neighbors
hermits[[i]] <- (d == 0)

# Covariates (only used ones, since W is used as a weight matrix later)
tempZ <- read_tsv(file.path(
  project_path,
  paste0(
    "datav4.0/Matlab Replication/India Networks/hhcovariates",
    as.character(vills[i]),
    ".csv"
    )
  ), col_names = FALSE)
tempZ <- tempZ[tempinGiant,1:6]
Z[[i]] <- tempZ
```

```
  # Leader statistics
  TakingLeaders[[i]] <- tempTakeUp[templeaders]
  ZLeaders[[i]] <- tempZ[templeaders,]
  Outcome <- c(Outcome, tempTakeUp[templeaders])
  Covars <- bind_rows(Covars, tempZ[templeaders,])

  # Second neighbors
  tempSec <- (X[[i]][[1]] %*% X[[i]][[1]] > 0)
  diag(tempSec) <- 0
  Sec[[i]] <- (tempSec - X[[i]][[1]] > 0)

}
```

Logistic regression using leaders to estimate β

Before structural estimation, we estimate the parameters on how the log-odds ratio of participation changes as household characteristics change. This estimation is based on the participation decisions among the set of leaders who we know are informed of the microfinance program. Since these parameters are used to calculate participation probabilities of non-leaders, one assumption imposed here is that the parameters are the same for leaders and non-leaders. The covariates used in this regression include access to electricity, quality of latrines, number of beds, number of rooms, the number of beds per capita, and the number of rooms per capita.

One thing worth noting is that, for access to electricity and quality of latrines, the variables are recorded as qualitative information. That is, for access to electricity, the question is "Does this household have electricity" and the answers are "1 = Yes, Private", "2 = Yes, Government", and "3 = No". For quality of latrines, the question is "What type of latrine does your household have?" and the answers are "1 = Owned", "2 = Common", and "3 = None". However, in the original Matlab code, these variables are used as quantitative variables in estimation. For better regression, we should convert them to indicator variables. Here, we follow the original Matlab code and use them as numerical variables.

```
# Logistic fit to get coefficients for covariates and the constant ----------------------
leader_df <- as_tibble(Outcome) %>%
  bind_cols(Covars)
glm_res <- glm(value ~ ., data = leader_df, family = "binomial")
Betas <- glm_res$coefficients
beta_table <- summary(glm_res)$coefficients
rownames(beta_table) <- c(
  "Intercept",
  "Number of rooms",
  "Number of beds",
  "Access to electricity",
  "Quality of latrine",
  "Number of rooms per capita",
  "Number of beds per capita"
)
kable(
  beta_table,
  caption = "Logistic regression estimation results",
  digits = 2,
  booktabs = TRUE
  ) %>%
  kable_minimal(full_width = F)
```

Table 1: Logistic regression estimation results

	Estimate	Std. Error	z value	Pr(>|z|)
Intercept	-1.21	0.32	-3.76	0.00
Number of rooms	0.01	0.09	0.08	0.93
Number of beds	-0.28	0.14	-1.99	0.05
Access to electricity	0.16	0.12	1.27	0.20
Quality of latrine	0.18	0.08	2.23	0.03
Number of rooms per capita	-1.02	0.39	-2.61	0.01
Number of beds per capita	1.15	0.66	1.75	0.08

Calculate network statistics

Here we calculate the statistics of networks in each village. In particular, we calculate distances between each household and leaders in a village. As mentioned above, although the authors use functions defined in `breadthdistRAL.m` and `breadth.m` to calculate them, here we use an R package called `igraph` to derive these statistics.

```r
# Calculate network statistics (netstats) for each village ------------
# Calculated statistics of each village are stored in the list "netstats"
netstats <- list()

for (i in seq_along(vills)){

  # Create an undirected graph from an adjacency matrix
  X_graph <- graph_from_adjacency_matrix(X[[i]][[1]], mode = "undirected")
  # Calculate distances between households
  D <- distances(X_graph)
  # Somehow the original Matlab code defines
  # that the distance from a household to itself is 2,
  # so we follow this way here
  diag(D) <- 2
  # Here we are interested only in the distance between a household and leaders,
  # so let distances from non-leaders be 0
  D[, !leaders[[i]]] = 0

  # Distance from the closest leader
  minDistFromLeaders <- apply(D[, which(leaders[[i]])], 1, min)
  # Average distance from leaders
  avgDistFromLeaders <- apply(D[, which(leaders[[i]])], 1, mean)

  # Set of households who take up MF
  infected <- TakeUp[[i]]

  # Distance from the closest leader who takes up
  if (dot(as.numeric(infected), as.numeric(leaders[[i]])) > 0){
    minDistInfectedLeaders <- apply(as.matrix(D[, which(leaders[[i]] & infected)]), 1, min)
  } else {
    minDistInfectedLeaders <- rep(0, nrow(leaders[[i]]))
  }

  # Distance from the closest leader who does not take up
  if (dot(1 - as.numeric(infected), as.numeric(leaders[[i]])) > 0){
    minDistNonInfectedLeaders <- apply(D[, which(leaders[[i]] & !infected)], 1, min)
  } else {
    minDistNonInfectedLeaders <- rep(0, nrow(leaders[[i]]))
  }

  # Indicator if neighboring leader is infected:
  # minimum distance to infected leaders is 1 &
  # minimum distance to non-infected leaders is 0 or > 1
  neighborOfInfected <- (
    ((minDistInfectedLeaders == 1) - (minDistNonInfectedLeaders == 1)) > 0
    )

  # Indicator if neighboring leader is not infected:
  # minimum distance to infected leaders is 0 or > 1 &
  # minimum distance to non-infected leaders is 1
  neighborOfNonInfected <- (
    ((minDistInfectedLeaders == 1) - (minDistNonInfectedLeaders == 1)) < 0
    )

  # Degree of network (number of connected households)
  network_degree <- rowSums(X[[i]][[1]])

  netstats_j <- list(
    minDistFromLeaders = minDistFromLeaders,
    avgDistFromLeaders = avgDistFromLeaders,
    minDistInfectedLeaders = minDistInfectedLeaders,
    minDistNonInfectedLeaders = minDistNonInfectedLeaders,
    neighborOfInfected = neighborOfInfected,
    neighborOfNonInfected = neighborOfNonInfected,
    network_degree = network_degree
  )

  netstats[[i]] <- netstats_j

}
```

Here, as a sanity check, we compare the network statistics derived here with the one derived by using the original Matlab code. Since we use the same networks to calculate statistics, they should be identical.

```
netstats_mat <- readMat(file.path(
  project_path,
  "datav4.0/Matlab Replication/netstats.mat"
  ))

netstats_compare <- cbind(
  c(
    mean(map_dbl(netstats, function(x) mean(x$minDistFromLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$avgDistFromLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$minDistInfectedLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$minDistNonInfectedLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$neighborOfInfected))),
    mean(map_dbl(netstats, function(x) mean(x$neighborOfNonInfected))),
    mean(map_dbl(netstats, function(x) mean(x$network_degree)))
  ),
  c(
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$minDistFromLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$avgDistFromLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$minDistInfectedLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$minDistNonInfectedLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$neighborOfInfected))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$neighborOfNonInfected))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_mat[[1]][,,x]$degree))
      )
    )
  )

colnames(netstats_compare) <- c("R", "Matlab")
rownames(netstats_compare) <- c(
  "minDistFromLeaders",
  "avgDistFromLeaders",
  "minDistInfectedLeaders",
  "minDistNonInfectedLeaders",
  "neighborOfInfected",
  "neighborOfNonInfected",
  "degree"
)

kable(netstats_compare, digits = 3, booktabs = TRUE) %>%
  kable_minimal(full_width = F)
```

	R	Matlab
minDistFromLeaders	1.342	1.342
avgDistFromLeaders	2.611	2.611
minDistInfectedLeaders	1.923	1.862
minDistNonInfectedLeaders	1.426	1.426
neighborOfInfected	0.075	0.090
neighborOfNonInfected	0.405	0.377
degree	9.656	9.656

Notice that there are differences for several network statistics. The reason is the lines 14 and 19 in the file `moments.m`:

`minDistInfectedLeaders = min(D(:,logical(infected.*leaders))')';`

`minDistNonInfectedLeaders = min(D(:,logical((1-infected).*leaders))')';`

These derive minimum distances from take-up and non take-up leaders, respectively. When there are more than one take-up leaders, then `D(:,logical(infected.*leaders))'` is a matrix whose number of columns is the number of villagers and number of rows is the number of take-up leaders. Then, the operator `min` takes the minimum of each column, and its transpose is a column vector. The problem is that, if there is only one take-up leader, then `D(:,logical(infected.*leaders))'` becomes a vector and `min` operator takes the minimum of the elements in the vector, which is a scalar. To avoid this problem, we need to rewrite them as follows:

`minDistInfectedLeaders = min(D(:,logical(infected.*leaders), [], 1)')';`

`minDistNonInfectedLeaders = min(D(:,logical((1-infected).*leaders), [], 1)')';`.

Correcting these gives identical network statistics:

```r
netstats_correct_mat <- readMat(file.path(
  project_path,
  "datav4.0/Matlab Replication/netstats_correct.mat"
  ))

netstats_compare <- cbind(
  c(
    mean(map_dbl(netstats, function(x) mean(x$minDistFromLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$avgDistFromLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$minDistInfectedLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$minDistNonInfectedLeaders))),
    mean(map_dbl(netstats, function(x) mean(x$neighborOfInfected))),
    mean(map_dbl(netstats, function(x) mean(x$neighborOfNonInfected))),
    mean(map_dbl(netstats, function(x) mean(x$network_degree)))
  ),
  c(
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$minDistFromLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$avgDistFromLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$minDistInfectedLeaders))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$minDistNonInfectedLeaders)
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$neighborOfInfected))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$neighborOfNonInfected))
      ),
    mean(
      sapply(seq(num_vills), function(x) mean(netstats_correct_mat[[1]][,,x]$degree))
      )
    )
  )
```

```r
colnames(netstats_compare) <- c("R", "Matlab (corrected)")
rownames(netstats_compare) <- c(
  "minDistFromLeaders",
  "avgDistFromLeaders",
  "minDistInfectedLeaders",
  "minDistNonInfectedLeaders",
  "neighborOfInfected",
  "neighborOfNonInfected",
  "degree"
)

kable(netstats_compare, digits = 3, booktabs = TRUE) %>%
  kable_minimal(full_width = F)
```

	R	Matlab (corrected)
minDistFromLeaders	1.342	1.342
avgDistFromLeaders	2.611	2.611
minDistInfectedLeaders	1.923	1.923
minDistNonInfectedLeaders	1.426	1.426
neighborOfInfected	0.075	0.075
neighborOfNonInfected	0.405	0.405
degree	9.656	9.656

Define a function to calculate moments (function `moments`)

Here we define a function to calculate moments, given the information on who take up microfinance. This function is used to calculate simulated moments and empirical moments. When calculating empirical moments, we use the data of take-ups by each household. On the other hand, we simulate take-ups of microfinance by households for each village, which is then used to calculate simulated moments.

```r
# Define a function to calculate moments (moments) ------------
moments <- function(X, leaders, netstats, infected, Sec, j, case){

  # Number of households in the village
  N <- nrow(X)

  network_degree <- netstats['network_degree'][[1]]
  neighborOfInfected <- netstats['neighborOfInfected'][[1]]
  neighborOfNonInfected <- netstats['neighborOfNonInfected'][[1]]

  if (case == 1){
    # 1. Fraction of nodes that have no taking neighbors but are takers themselves
    infectedNeighbors <- rowSums(outer(rep(1, N), infected) * X) # Number of infected neighbors
    if (sum(infectedNeighbors == 0 & network_degree > 0) > 0){
      stats_1 <- sum((infectedNeighbors == 0 & infected == 1 & network_degree > 0)) /
        sum(infectedNeighbors == 0 & network_degree > 0)
    } else if (sum(infectedNeighbors == 0 & network_degree > 0) == 0){
      stats_1 <- 0
    }

    # 2. Fraction of individuals that are infected
    #   in the neighborhood of infected leaders stats_1 == 0
    if (sum(neighborOfInfected) > 0){
      stats_2 <- sum(infected * neighborOfInfected) / sum(neighborOfInfected)
    } else {
      stats_2 <- 0
    }

    # 3. Fraction of individuals that are infected
    #   in the neighborhood of non-infected leaders
    if (sum(neighborOfInfected) > 0){
      stats_3 <- sum(infected * neighborOfNonInfected) / sum(neighborOfNonInfected)
    } else {
      stats_3 <- 0
    }

    # 4. Covariance of individuals taking with share of neighbors taking
    NonHermits = (network_degree > 0)
    ShareofTakingNeighbors = infectedNeighbors[NonHermits] / network_degree[NonHermits]
    NonHermitTakers = infected[NonHermits]
    stats_4 <- sum(NonHermitTakers * ShareofTakingNeighbors) / sum(NonHermits)

    # 5. Covariance of individuals taking with share of second neighbors taking
    infectedSecond = rowSums(Sec * outer(infected, rep(1, N)))
    ShareofSecond = infectedSecond[NonHermits] / network_degree[NonHermits]
    stats_5 <- sum(NonHermitTakers * ShareofSecond) / sum(NonHermits)

    return(c(stats_1, stats_2, stats_3, stats_4, stats_5))

  } else if (case == 2){
    # 1. Fraction of nodes that have no taking neighbors but are takers themselves
    infectedNeighbors <- rowSums(outer(rep(1, N), infected) * X) # Number of infected neighbors

    if (sum(infectedNeighbors == 0 & network_degree > 0) > 0){
      stats_1 <- sum((infectedNeighbors == 0 & infected == 1 & network_degree > 0)) /
        sum(infectedNeighbors == 0 & network_degree > 0)
    } else if (sum(infectedNeighbors == 0 & network_degree > 0) == 0){
      stats_1 <- 0
    }

    # 2. Covariance of individuals taking with share of neighbors taking
    NonHermits = (network_degree > 0)
    ShareofTakingNeighbors = infectedNeighbors[NonHermits] / network_degree[NonHermits]
    NonHermitTakers = infected[NonHermits]
    stats_2 <- sum(NonHermitTakers * ShareofTakingNeighbors) / sum(NonHermits)

    # 3. Covariance of individuals taking with share of second neighbors taking
    infectedSecond = rowSums(Sec * outer(infected, rep(1, N)))
    ShareofSecond = infectedSecond[NonHermits] / network_degree[NonHermits]
    stats_3 <- sum(NonHermitTakers * ShareofSecond) / sum(NonHermits)

    return(c(stats_1, stats_2, stats_3))

  } else if (case == 3){
    # same as case 2, but purged of leader injection points.
```

```r
    leaderTrue = (leaders > 0) # a variable that denotes whether a node is either a leader

    # 1. Fraction of nodes that have no taking neighbors but are takers themselves
    infectedNeighbors <- rowSums((outer(rep(1, N), infected)) * X) # Number of infected neighbors

    if (sum(infectedNeighbors == 0 & network_degree > 0) > 0){
      stats_1 <- sum(
        (infectedNeighbors == 0 & (leaderTrue == 0) & infected == 1 & network_degree > 0)
        ) / sum(infectedNeighbors == 0 & network_degree > 0)
    } else if (sum(infectedNeighbors == 0 & (leaderTrue == 0) & network_degree > 0) == 0){
      stats_1 <- 0
    }

    # 2. Covariance of individuals taking with share of neighbors taking
    NonHermits = (network_degree > 0)
    NonHermitsNonLeaders = (NonHermits & (1 - leaderTrue)) # not isolates, not leaders

    ShareofTakingNeighbors = infectedNeighbors[NonHermitsNonLeaders] /
      network_degree[NonHermitsNonLeaders]
    NonHermitTakers = infected[NonHermitsNonLeaders]
    stats_2 <- sum(NonHermitTakers * ShareofTakingNeighbors) / sum(NonHermitsNonLeaders)

    # 3. Covariance of individuals taking with share of second neighbors taking
    infectedSecond = rowSums(Sec * outer(infected, rep(1, N)))
    ShareofSecond = infectedSecond[NonHermitsNonLeaders] /
      network_degree[NonHermitsNonLeaders]
    stats_3 <- sum(NonHermitTakers * ShareofSecond) / sum(NonHermitsNonLeaders)

    return(c(stats_1, stats_2, stats_3))

  } else if (case == 4){
    # same as case 3, but purged of ALL leader nodes.
    leaderTrue = (leaders > 0) # a variable that denotes whether a node is either a leader

    # 1. Fraction of nodes that have no taking neighbors but are takers themselves
    infectedNeighbors <- rowSums(
      outer(rep(1, N), infected) * X %*% (1 - leaderTrue)
      ) # Number of infected neighbors

    if (sum(infectedNeighbors == 0 & network_degree > 0) > 0){
      stats_1 <- sum(
        (infectedNeighbors == 0 & (leaderTrue == 0) & infected == 1 & network_degree > 0)
        ) / sum(infectedNeighbors == 0 & network_degree > 0)
    } else if (sum(infectedNeighbors == 0 & (leaderTrue == 0) & network_degree > 0) == 0){
      stats_1 <- 0
    }

    # 2. Covariance of individuals taking with share of neighbors taking
    NonHermits = (network_degree > 0)
    NonHermitsNonLeaders = (NonHermits & (1 - leaderTrue)) # not isolates, not leaders

    ShareofTakingNeighbors = infectedNeighbors[NonHermitsNonLeaders] /
      network_degree[NonHermitsNonLeaders]
    NonHermitTakers = infected[NonHermitsNonLeaders]
    stats_2 <- sum(NonHermitTakers * ShareofTakingNeighbors) / sum(NonHermitsNonLeaders)

    # 3. Covariance of individuals taking with share of second neighbors taking
    infectedSecond = rowSums(Sec * outer(infected, rep(1, N)) %*% (1 - leaderTrue))
    ShareofSecond = infectedSecond[NonHermitsNonLeaders] /
      network_degree[NonHermitsNonLeaders]
    stats_3 <- sum(NonHermitTakers * ShareofSecond) / sum(NonHermitsNonLeaders)

    return(c(stats_1, stats_2, stats_3))

  }

}
```

Define a function to simulate diffusion of MF (function `diffusion_model`)

Here we define a function to simulate how microfinance diffuses in a network. Among other things, this function depends on β estimated above and the parameters that we try to estimate: the probabilities of information transmitted from a participant and a non-participant.

```r
# Define a function to simulate diffusion of MF (diffusion_model) ---------------------

diffusion_model <- function(parms, Z, Betas, X, leaders, j, t_period, EmpRate){

  qN <- parms[1] # Probability non-taker transmits information
  qP <- parms[2] # Probability that a just-informed-taker transmits information
  N <- nrow(X) # Number of households

  infected <- rep(FALSE, N) # Nobody has been infected yet.
  infectedbefore <- rep(FALSE, N) # Nobody has been infected yet.
  contagiousbefore <- rep(FALSE, N) # People who were contagious before
  contagious <- leaders # Newly informed/contagious.
  dynamicInfection <- rep(0, t_period) # Will be a vector that tracks the infection rate
                                       # for the number of periods it takes place

  x <- matrix(dqrunif(N * t_period), N, t_period)
  t <- 1
  for (t in seq(t_period)){
    qNt <- qN
    qPt <- qP

    # Step 1: Take-up decision based on newly informed
    LOGITprob <- 1 / (1 + exp(- cbind(rep(1, N), as.matrix(Z)) %*% Betas))
    infected <- ((!contagiousbefore & contagious & as.vector(x[,t] < LOGITprob)) | infected)
    s1 <- sum(infected)
    s2 <- sum(infectedbefore)
    infectedbefore <- (infectedbefore | infected)
    contagiousbefore <- (contagious | contagiousbefore)
    C <- sum(contagious)

    # Step 2: Information flows
    transmitPROB <- (contagious & infected) * qPt + (contagious & !infected) * qNt
    contagionlikelihood <- X[contagious,] * outer(transmitPROB[contagious], rep(1, N))
    # Step 3
    contagious <- (
      (colSums(contagionlikelihood > matrix(dqrunif(C * N), C, N)) > 0) | contagiousbefore
      )
    dynamicInfection[t] <- sum(infectedbefore) / N

  }

  return(list(infectedbefore, dynamicInfection, contagious))

}
```

Define a function to compute the deviation of the empirical moments from the simulated moments (function `divergence_model`)

We define a function to compute $d(r, \theta)$ for a village r and a given parameter θ. Note that this is a vector whose length is the number of moments used for estimation. These vectors will be stored to a matrix D, whose dimension is (# of villages, # of moments) (ie. each row of D is $d(r, \theta)$).

```r
divergence_model <- function(
  X, Z, Betas, leaders, TakeUp, Sec, theta, m, S, t_period, EmpRate, case
  ){

  # Number of villages
  G <- length(X)

  # Computation of the vector of divergences across all the moments
  EmpiricalMoments <- matrix(0, G, m)
  MeanSimulatedMoments <- matrix(0, G, m)
  D <- matrix(0, G, m)
  TimeSim <- matrix(0, G, S)

  for (g in seq(G)){
    # Compute moments - G x m object
    EmpiricalMoments[g,] <- moments(
      X[[g]][[1]], leaders[[g]], netstats[[g]], TakeUp[[g]], Sec[[g]], g, case
      )

    # Compute simulated moments
    SimulatedMoments <- matrix(0, S, m)
    for (s in seq(S)){
      infectedSIM <- diffusion_model(
        theta, Z[[g]], Betas, X[[g]][[1]], leaders[[g]], g, t_period[g], EmpRate[[g]]
        )
      SimulatedMoments[s,] <- moments(
        X[[g]][[1]], leaders[[g]], netstats[[g]], as.vector(infectedSIM[[1]]), Sec[[g]], g, case
        )
    }

    # Compute the mean simulated moment - a G x m object
    MeanSimulatedMoments[g,] <- colMeans(SimulatedMoments)
    D[g,] <- MeanSimulatedMoments[g,] - EmpiricalMoments[g,]
  }
  return(D)

}
```

Run the model to obtain moment deviations for each village-simulation-parameter pair

Given the functions defined so far, we are now ready to estimate the parameters. First, we calculate the objective function $d(r, \theta)$ for each θ in the grid. This is the most time-consuming part of the estimation process: using one of our PCs, it takes almost 10 hours on R, even without the endorsement effect. Using parallel computation can speed up this process.

```
# Running the model
if (modelType == 1){ # Case where qN = qP
  D <- array(rep(0, num_vills * m * length(qN)), dim = c(num_vills, m, length(qN)))

  for (i in seq(length(qN))){
    print(i)
    theta <- c(qN[i], qN[i])
    D[,,i] <- divergence_model(
      X, Z, Betas, leaders, TakeUp, Sec, theta, m, S, t_period, EmpRate, case
      )
  }
} else if (modelType == 3){ # Case where qN \ne qP
  D <- array(
    rep(0, num_vills * m * length(qN) * length(qP)),
    dim = c(num_vills, m, length(qN), length(qP))
    )
  for (i in seq(length(qN))){
    print(i)
    for (j in seq(length(qP))){
      theta <- c(qN[i], qP[j])
      D[,,i,j] <- divergence_model(
        X, Z, Betas, leaders, TakeUp, Sec, theta, m, S, t_period, EmpRate, case
        )
    }
  }
}

# Save the output ------------------
file_name <- paste0(
  'data_model_', as.character(modelType), '_mom_',
  as.character(case), '_', timeVector, '.RData'
  )
save(D, file = file.path('../Rdata', file_name))
```

Run the aggregator to calculate objective functions for each parameter pair and obtain parameters that minimize the objective function

Next, we calculate the objective function $Q(\theta)$ for each θ in the grid. If using the optimal weight for this step, we calculate the weight matrix $\widehat{W}$ based on the estimate in the first step. Otherwise, we use the identity matrix for $\widehat{W}$.

```r
# Two step optimal weights
if (twoStepOptimal == 1){
  if (modelType == 1){
    # Load the first-step estimates
    file_name <- paste0(
      'param_est_', as.character(modelType), '_mom_',
      as.character(case), '_', timeVector, '_0_0.RData'
      )
    load(file = file.path('../Rdata', file_name))

    for (i in seq(43)) {
      aa[[i]] <- array(runif(10 * 160 * 75), c(10, 160, 75))
    }

    qN_info <- param_est[1,1]
    qP_info <- param_est[1,1]
    theta <- c(qN_info, qP_info)
    Dtemp <- divergence_model(
      X, Z, Betas, leaders, TakeUp, Sec, theta, m, S, t_period, EmpRate, case
      )
    A <- (t(Dtemp) %*% Dtemp) / num_vills
    W <- inv(A)
  } else if (modelType == 3){
    # Load the first-step estimates
    file_name <- paste0(
      'param_est_', as.character(modelType), '_mom_',
      as.character(case), '_', timeVector, '_0_0.RData'
      )
    load(file = file.path('../Rdata', file_name))

    qN_info <- param_est[1,1]
    qP_info <- param_est[1,2]
    theta <- c(qN_info, qP_info)
    Dtemp <- divergence_model(
      X, Z, Betas, leaders, TakeUp, Sec, theta, m, S, t_period, EmpRate, case
      )
    A <- (t(Dtemp) %*% Dtemp) / num_vills
    W <- inv(A)
  }
} else if (twoStepOptimal == 0){
  W <- eye(m)
}
```

Then, we calculate $Q(\theta)$ and estimate $\theta^* = \arg\min Q(\theta)$. For bootstrap, we randomly generate weights for each village and calculate $Q(b, \theta)$.

```r
# weights for bootstrap
estimate_params <- function(bootstrap){

  if (bootstrap == 0){
    B <- 1
  } else if (bootstrap == 1){
    B <- 1000
  }
```

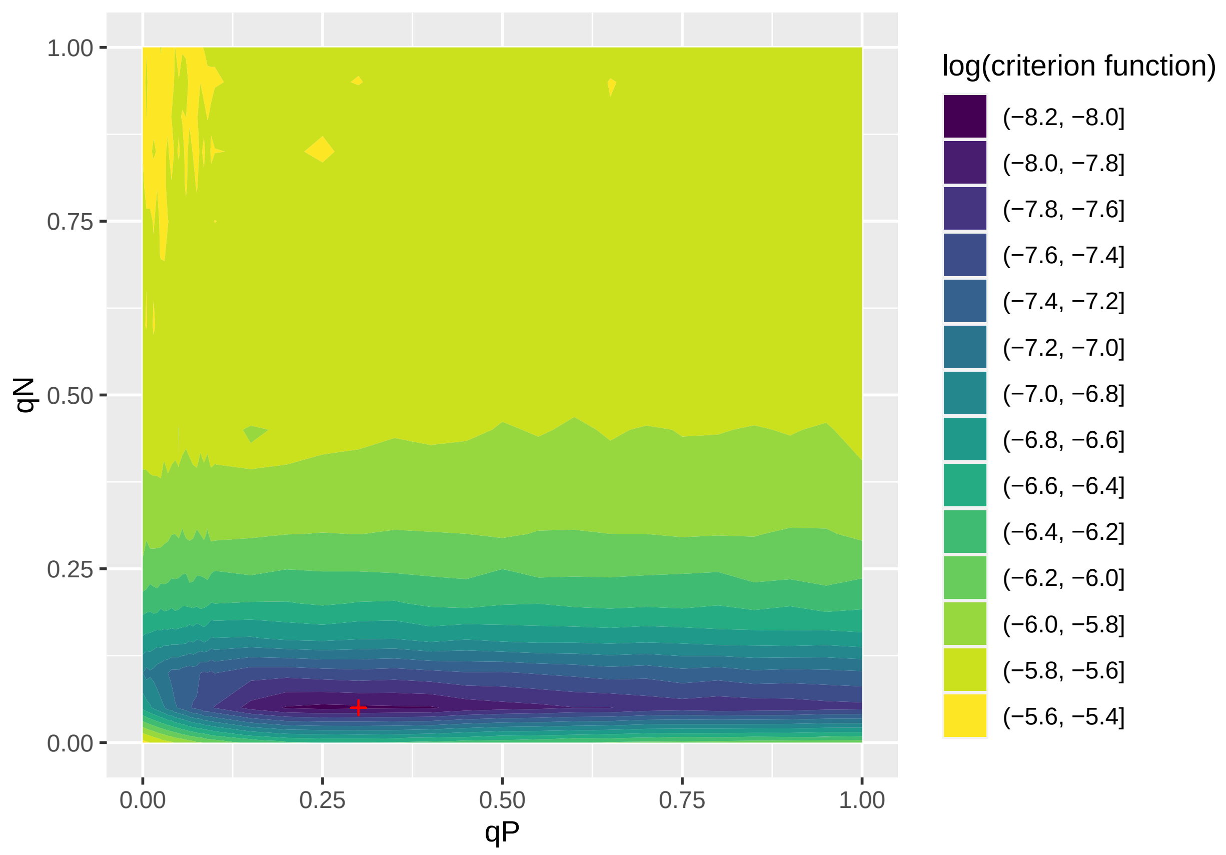
```r
if (modelType == 1){
  param_est <- zeros(B, 1)
} else if (modelType == 3){
  param_est <- zeros(B, 2)
}

wt <- zeros(B, num_vills)
for (b in seq(B)){

  # Generate weights b
  if (bootstrap == 1){
    wt[b,] <- rexp(num_vills)
    wt[b,] <- wt[b,] / mean(wt[b,])
  } else if (bootstrap == 0){
    wt[b,] <- rep(1 / num_vills, num_vills)
  }

  # For each model, generate the criterion function value for this
  # bootstrap run

  # Info model
  if (modelType == 1){
    momFunc <- array(
      rep(0, length(qN) * B * m), dim = c(length(qN), B, m)
      )
    Qa <- array(rep(0, length(qN) * B), dim = c(length(qN), B))
    for (i in seq(length(qN))){
      # Compute the moment function
      momFunc[i,b,] <- wt[b,] %*% D[,,i] / num_vills
      # Criterion function
      Qa[i,b] <- t(momFunc[i,b,]) %*% W %*% momFunc[i,b,]
    }
    param_est[b] <- qN[which.min(Qa[,b])]
  } else if (modelType == 3){
    momFunc <- array(
      rep(0, length(qN) * length(qP) * B * m), dim = c(length(qN), length(qP), B, m)
      )
    Qa <- array(
      rep(0, length(qN) * length(qP) * B), dim = c(length(qN), length(qP), B)
      )
    for (i in seq(length(qN))){
      for (j in seq(length(qP))){
        # Compute the moment function
        momFunc[i,j,b,] <- wt[b,] %*% D[,,i,j] / num_vills
        # Criterion function
        Qa[i,j,b] <- t(momFunc[i,j,b,]) %*% W %*% momFunc[i,j,b,]
      }
    }
    min_ind <- which(Qa[,,b] == min(Qa[,,b]), arr.ind = TRUE)
    param_est[b,1] <- qN[min_ind[1]]
    param_est[b,2] <- qP[min_ind[2]]
  }
}
```

```r
  return(param_est)
}

# Estimation
param_est <- estimate_params(bootstrap = 0)
file_name <- paste0(
  'param_est_', as.character(modelType), '_mom_', as.character(case), '_',
  timeVector, '_', as.character(twoStepOptimal),
  '_', as.character(0), '.RData'
  )
save(param_est, file = file.path('../Rdata', file_name))

# Bootstrap
param_boot <- estimate_params(bootstrap = 1)
file_name <- paste0(
  'param_est_', as.character(modelType), '_mom_', as.character(case), '_',
  timeVector, '_', as.character(twoStepOptimal),
  '_', as.character(1), '.RData'
  )
save(param_boot, file = file.path('../Rdata', file_name))
```

Shape of objective functions

Before diving into the estimation results, we show the shape of objective functions by parameter values. The figure below is the value of objective functions, where we use the log scale for better visibility. The red cross in the figure indicates the parameter values minimizing the objective function, i.e. point estimates.

```r
if (modelType == 1){

  # Criterion functions
  wt <- rep(1 / num_vills, num_vills)
  momFunc <- array(rep(0, length(qN) * m), dim = c(length(qN), m))
  Qa <- rep(0, length(qN))
  for (i in seq(length(qN))){
    # Compute the moment function
    momFunc[i,] <- wt %*% D[,,i] / num_vills
    # Criterion function
    Qa[i] <- t(momFunc[i,]) %*% W %*% momFunc[i,]
  }

  plot_df <- as_tibble(cbind(qN, Qa))

  filename <- paste0(
    "../Figures/", 'data_model_', as.character(modelType), '_mom_',
    as.character(case), '_', timeVector, '_',
    as.character(twoStepOptimal), '.pdf'
    )
  pdf(file = filename)
  p <- ggplot(plot_df, aes(x = qN, y = log(Qa))) +
    geom_line() +
    geom_point() +
    scale_fill_viridis_d(name = 'log(criterion function)')
  print(p)
  dev.off()
  p
} else if (modelType == 3){

  # Criterion functions
  wt <- rep(1 / num_vills, num_vills)
  momFunc <- array(
    rep(0, length(qN) * length(qP) * m), dim = c(length(qN), length(qP), m)
    )
  Qa <- matrix(
    rep(0, length(qN) * length(qP)), nrow = length(qN), ncol = length(qP)
    )
  for (i in seq(length(qN))){
    for (j in seq(length(qP))){
      # Compute the moment function
      momFunc[i,j,] <- wt %*% D[,,i,j] / num_vills
      # Criterion function
      Qa[i,j] <- t(momFunc[i,j,]) %*% W %*% momFunc[i,j,]
    }
  }

  plot_df <- expand_grid(qP, qN) %>%
    mutate(Qa = c(Qa))

  filename <- paste0(
    "../Figures/", 'data_model_', as.character(modelType), '_mom_',
    as.character(case), '_', timeVector, '_',
    as.character(twoStepOptimal), '.pdf'
    )
  pdf(file = filename)
  p <- ggplot(plot_df, aes(x = qP, y = qN, z = log(Qa))) +
    geom_contour_filled() +
    geom_point(aes(x = param_est[1,2], y = param_est[1,1]), color = 'red', shape = 3) +
    scale_fill_viridis_d(name = 'log(criterion function)')
  print(p)
  dev.off()
  p
}
```



Estimation results

Here are the estimation results:

```r
param_boot_df <- as_tibble(param_boot)
colnames(param_boot_df) <- c("qN", "qP")

param_mat <- rbind(
  param_est,
  c(std(param_boot[,1]), std(param_boot[,2]))
  ) %>%
  round(3)
colnames(param_mat) <- c("qN", "qP")
rownames(param_mat) <- c("Estimate", "SE")

kable(
  param_mat,
  caption = "Structural estimation Results",
  booktabs = TRUE
  ) %>%
  kable_minimal(full_width = F)
```

The point estimates are slightly different from the ones in the paper: according to Table 1 of the paper, the estimates of qN and qP are 0.05 and 0.35, respectively. This could be partly because of the simulation procedure. Note that since the original Matlab codes do not specify the random seed, we cannot exactly replicate their results.

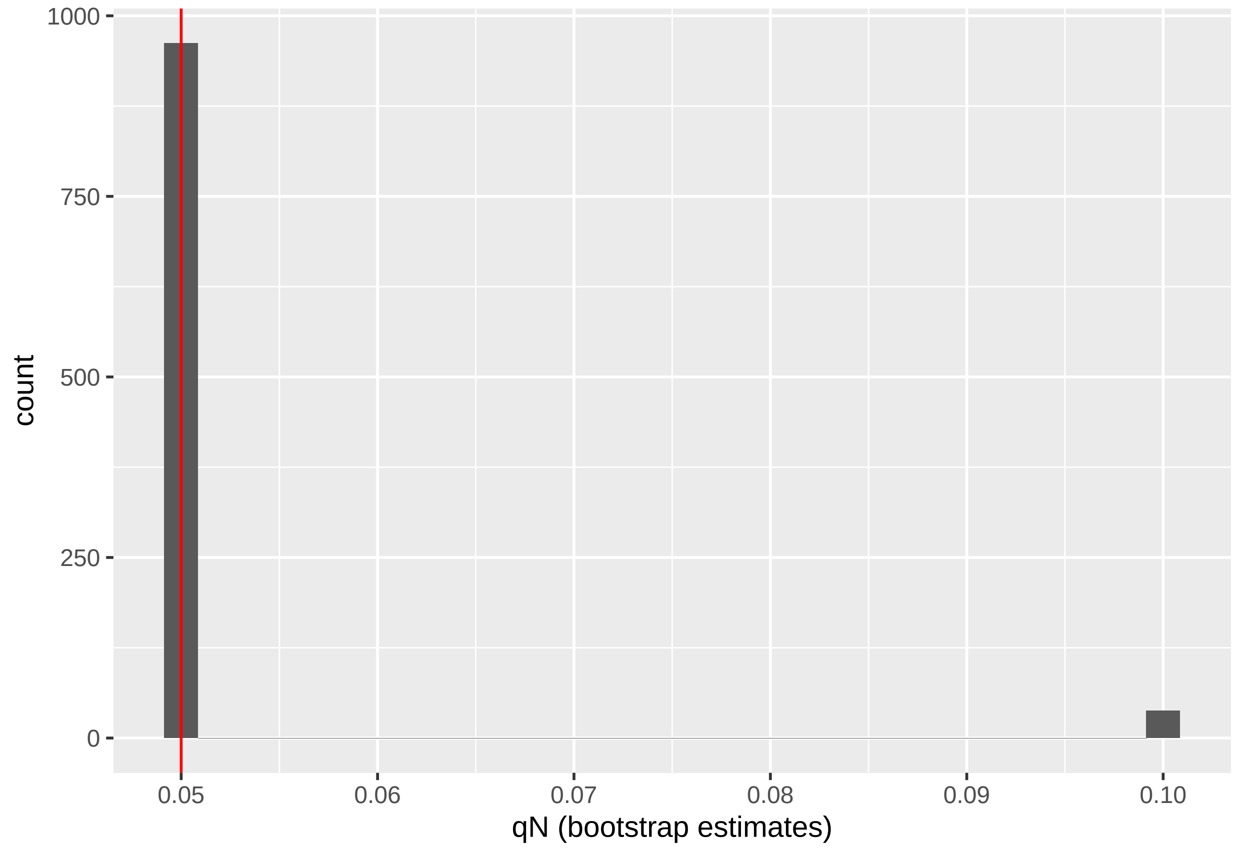
Table 2: Structural estimation Results

	qN	qP
Estimate	0.05	0.300
SE	0.01	0.115

The distributions of bootstrap estimates are shown below. The red lines indicate point estimates.

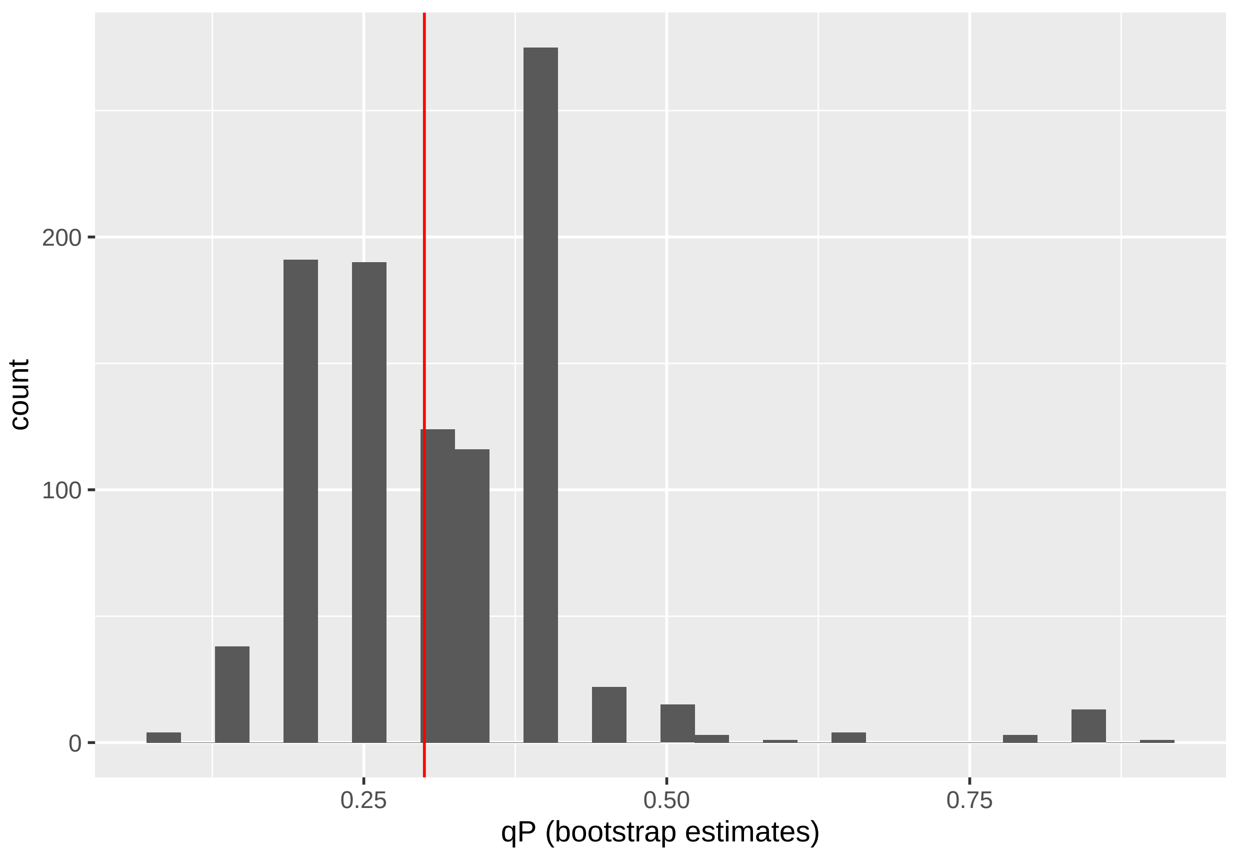
```
ggplot(param_boot_df) +
  geom_histogram(aes(x = qN)) +
  geom_vline(xintercept = param_est[1], color = "red") +
  xlab("qN (bootstrap estimates)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(param_boot_df) +
  geom_histogram(aes(x = qP)) +
  geom_vline(xintercept = param_est[2], color = "red") +
  xlab("qP (bootstrap estimates)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Comments

Major comments

- To calculate standard errors, the authors use the "bootstrap" method, in which, they generate random weights many times and use them in constructing an objective function for estimation, rather than randomly resampling observations and estimating the parameters with the resampled dataset. While this substantially reduces the computation time, since this method avoids calculating the differences between empirical moments and simulated moments for each resampled data point, it is a rather unusual "bootstrap" method. Any comments on this method are appreciated.
- As mentioned above, the original Matlab code does not include the random seed, which makes exact replication impossible.

Minor comments

- As discussed above, there is a small bug in the code to calculate network statistics, which could be why our estimates and the original estimates slightly differ.
- When calculating network statistics, the variable about whether having an infected neighboring leader (`neighborOfInfected`) is 0 when a HH has both a neighboring infected leader and a neighboring non-infected leader, but this seemingly should be 1 as that household has a neighboring infected leader. This may affect how they created the second moment.
- The distribution of the bootstrap estimates on qN is almost degenerate, which could be due to how the grid of qN is constructed: while up to $qN = 0.01$ the grid step is 0.001, after that point the grid is pretty sparse $(\ldots,\ 0.009,\ 0.01,\ 0.05,\ 0.1,\ 0.15, \ldots)$. Making the qN-grid finer around 0.05 might change the distribution of the bootstrap estimates, hence the standard error of the estimate.
- In estimation of β, qualitative variables (access to electricity and quality of latrine) are used as quantitative variables. These variables should be converted to indicator variables and included in the

regression.