Replication of "The Diffusion of Microfinance"

Kazuki Motohashi, Sakina Shibuya, and Mizuhiro Suzuki

2020-10-18

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: refer to the Supplementary material of the paper for details, which can be downloaded with the data and the codes.

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

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 $\omega_r^b = e_{br}/\overline{e}_r$ with $e_{br} \sim \exp(1)$ and $\overline{e}_r = \frac{1}{R}\sum_r e_{br}$, 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 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 updated on this GitHub repo (** add link

after making this public **). 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.

```
# 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 particpate.

Load data

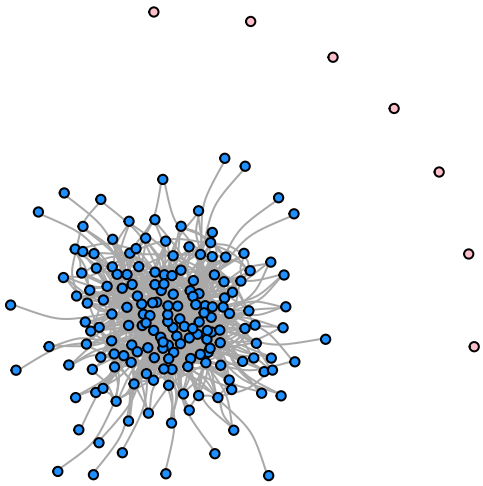
Here, we load data used in the analysis. One thing that seems not mentioned in the paper or in the Supplementary material is that, in each village, only households that belong to the largest cluster in the village are used. By the largest cluster, we mean the cluster composed of the largest number of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

```
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.

```
# 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
  # (contained in a list)
  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 (contained 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 (contained 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 quality of 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. ** I coundn't find an explanation of which variable indicates what. Maybe I should try harder to find it. **

```
# 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$coefficients
kableExtra::kable(summary(glm_res)$coefficients, digits = 2)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.21	0.32	-3.76	0.00
X1	0.01	0.09	0.08	0.93
X2	-0.28	0.14	-1.99	0.05
X3	0.16	0.12	1.27	0.20
X4	0.18	0.08	2.23	0.03
X5	-1.02	0.39	-2.61	0.01
X6	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.

```
# 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)
  # Some how 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

}
```

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, for each village we simulate take-ups of microfinance by households, which is used when calculating simulated moments.

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

```r
# 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.
  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.

```
# 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 $d(r, \theta)$).

```
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, now we are ready to estimate 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: on a PC of one of the us, it takes almost 10 hours, 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 $\widehat{W}$.

```
# 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))

    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 $Q(b, \theta)$.

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

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

  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)
        )
```

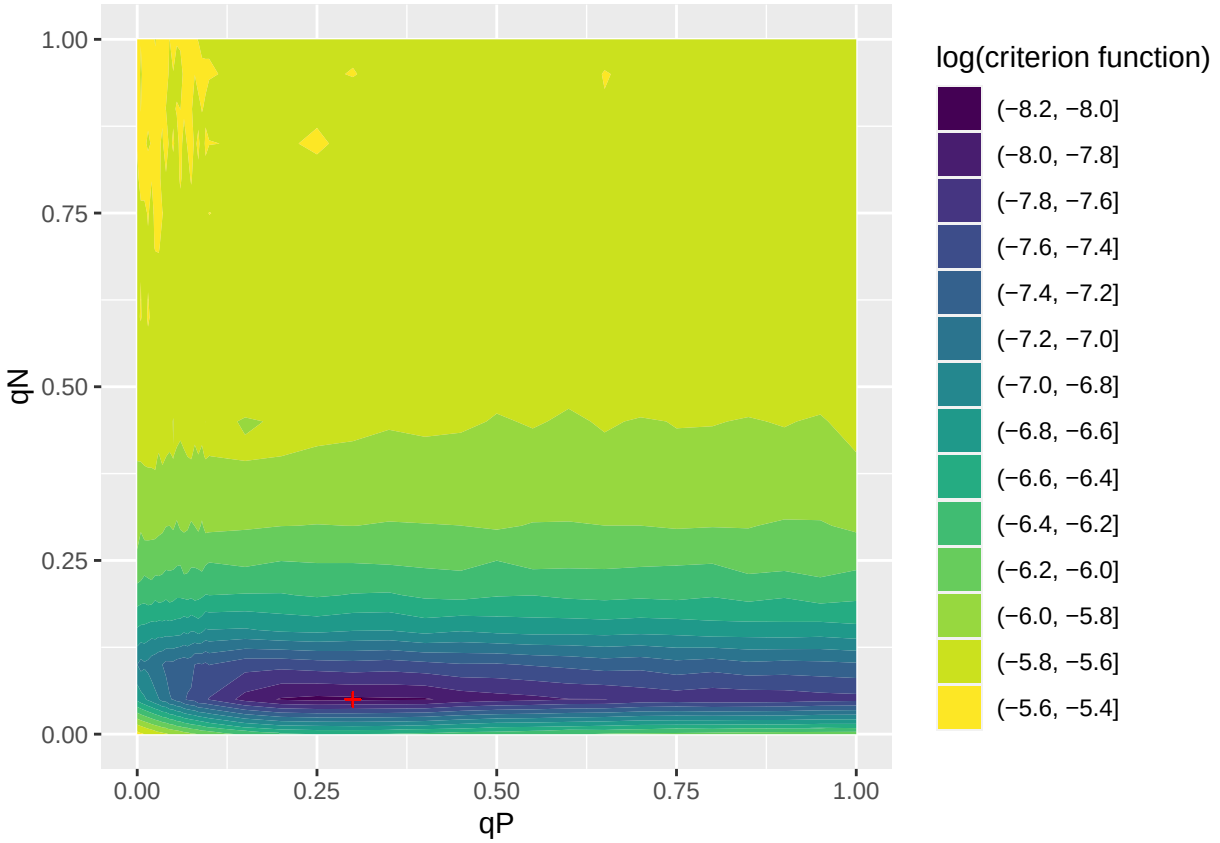
```
      for (i in seq(length(qN))){
        for (j in seq(length(qP))){
          # Compute the moment function
          momFunc[i,j,b,] <- wt[b,] %*% D[,,i,j] / num_vills
          # Criterion function
          Qa[i,j,b] <- t(momFunc[i,j,b,]) %*% W %*% momFunc[i,j,b,]
        }
      }
      min_ind <- 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]]
    }
  }

  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, ie. point estimates.

```
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:

```
param_boot_df <- as_tibble(param_boot)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
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")

knitr::kable(param_mat, caption = "Estimation Results")
```

The point estimates are slightly different from the ones in the paper, which could be because of the simulation. Note that since the original Matlab codes do not specify the random seed, we cannot exactly replicate their results.

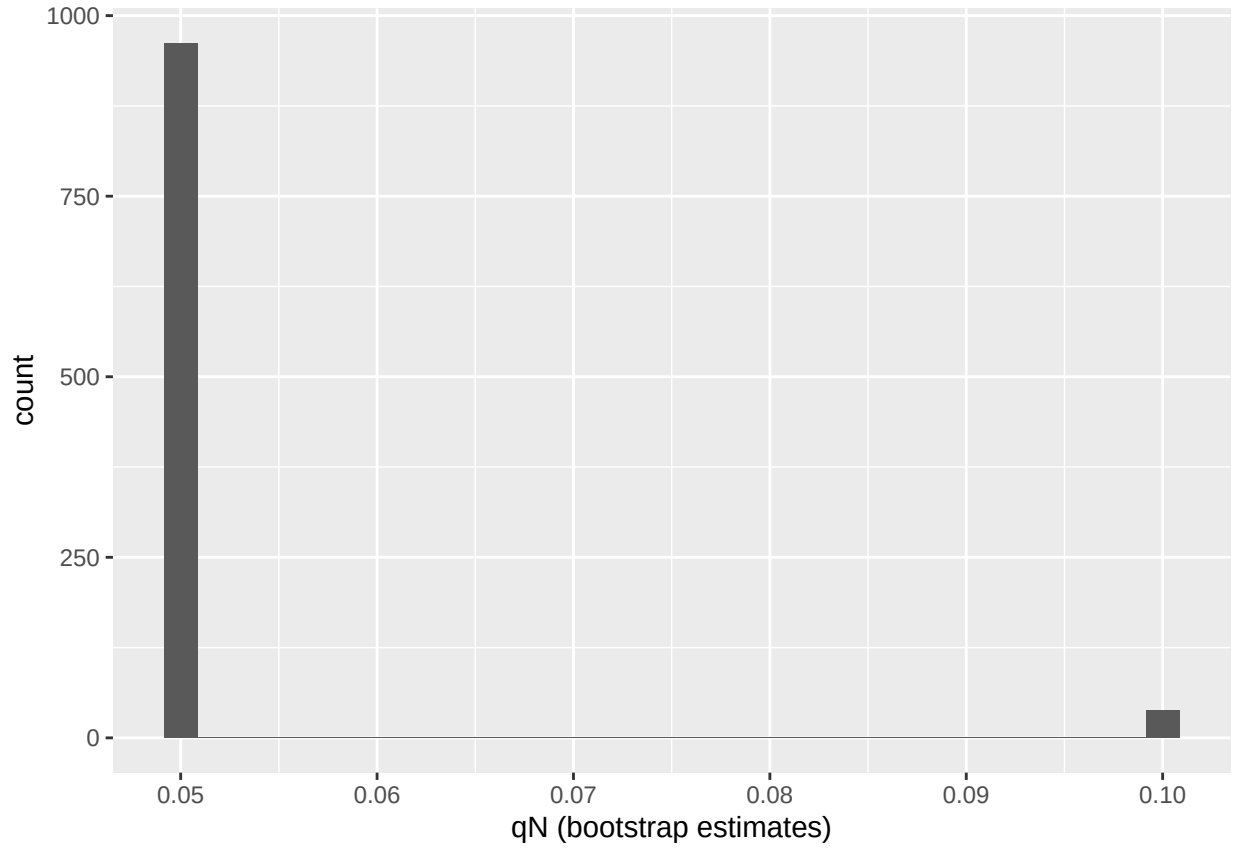
The distributions of bootstrap estimates are shown below.

Table 1: Estimation Results

	qN	qP
Estimate	0.05	0.300
SE	0.01	0.115

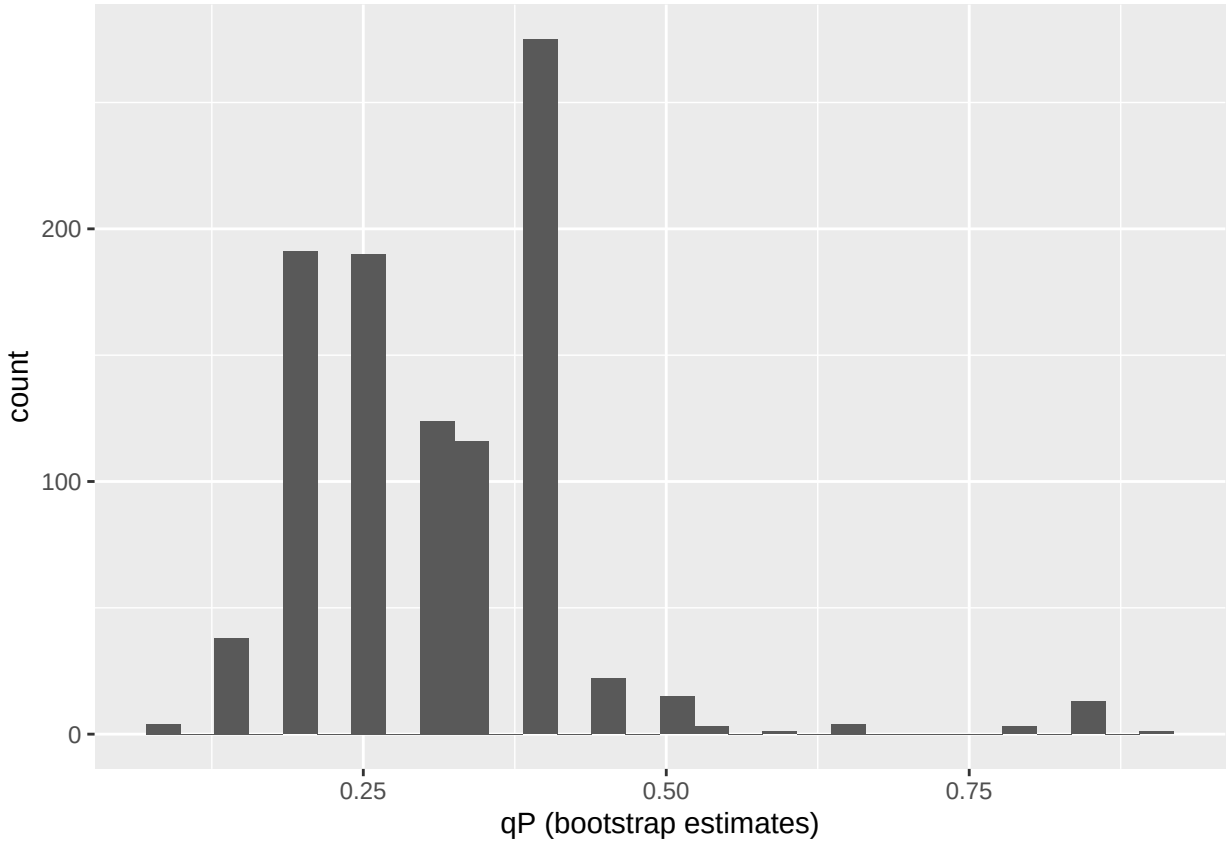
```
ggplot(param_boot_df) +
  geom_histogram(aes(x = qN)) +
  xlab("qN (bootstrap estimates)")
```

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




```
ggplot(param_boot_df) +
  geom_histogram(aes(x = qP)) +
  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, instead of randomly resampling observations and estimating the parameters with the resampled dataset, they generate random weights many times and use them in construcing an objective function for estimation. While this substantially reduces the computation time since thye can avoid to calculate the differences between empirical moments and simulated moments for each resampled data, we have not heard of this "bootstrap" method, and any comments on this methods are appreciated.
- As mentioned above, the original Matlab code does not include the random seed, which prohibits us from reproducing the original results.

Minor comments

- When calculating network statistics, the variable about whether having an infected neighboring leader or not (`neighborOfInfected`) is 0 when a HH has both a neighboring infected leader and a neighboring non-infected leader, but it seems it 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 they create the grid of qN: 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 and hence the standard error of the estimate.