Generalized RDS Simulation on R

Angela Yoon

6/10/2021

```r
rds_simulation <- function(G, lambda, n, start_index, num_coupon) {

  ### build zero adjacency matrix that will be updated to output G_R
  size <- nrow(G)
  G_R <- matrix(rep(0,size^2),size)

  ### build zero matrix to keep track of waiting time
  T <- matrix(rep(NA,size^2),size)

  ### create empty coupon matrix C
  C <- matrix(rep(0,n*size),size)

  ### create empty vector to keep track of order of recruitment and waiting times
  order <- c(start_index)
  w <- c(0)

  ### initialize i
  i <- start_index

  ### iterate until we recruit n-1 more people
  for (x in 1:(n-1)) {

    ### from original network, find all vertices adjacent to our most recently recruited vertex
    for (j in 1:size) {
      if (G[i,j] == 1) {
        ### "activate" corresponding edges by assigning a waiting time
        T[i,j] <- rexp(1, lambda)
      }
    }

    ### update C
    for (k in 1:size) {
      if (sum(T[k,],na.rm=TRUE)>0) {
        C[k,x] =1
      }
    }

    ### find the minimum waiting time from T to find out the next recruitment
    waiting_time <- min(T,na.rm = TRUE)
    new <- arrayInd(which.min(T),dim(T))

    ### append minimum waiting time to vector w
    w <- append(w,waiting_time)

    ### identify the indexes of the recruiter and recruitee of this new recruitment
    recruiter <- new[1,1]
    recruitee <- new[1,2]

    ### update G_R and order to reflect this new recruitment
    G_R[recruiter,recruitee] <- 1
    order <- append(order,recruitee)

    ### update T by subtracting minimum waiting time from all other waiting times
    ### also deactivate all edges from any other vertex to recruitee
    ### also eliminate all waiting times from most recent recruiter if recruiter used all coupons
    for(row in 1:nrow(T)) {
      for(col in 1:ncol(T)) {
          if(!is.na(T[row, col])) {
            T[row,col] <- T[row,col]-waiting_time
          }
      }
    }
    T[,recruitee] <- rep(NA,size)
    if (sum(G_R[recruiter,])==2) {
      T[recruiter,] <- rep(NA,size)
    }

    ### update G by eliminating all edges to most recent recruiter and recruitee
    ### note: is there a need to do both? simplify if possible
    G[,recruiter] <- rep(0,size)
    G[,recruitee] <- rep(0,size)

    ### update i
    i <- new[1,2]
  }

  return(list(G_R, order, w, C))
}
```

```
### calculate s
get_s <- function(G_R, order, C, G) {

  ### reindexing A
  A <- matrix(rep(0,ncol(C)^2),ncol(C))

  for (row in 1:nrow(G_R)) {
    if (row %in% order) {
      new_row = match(row,order)
      for (col in 1:ncol(G_R)) {
        if (G_R[row,col]==1) {
          new_col = match(col,order)
          A[new_row,new_col] = 1
        }
      }
    }
  }
```

```r
  ### reindexing C
  C_new <- matrix(rep(0,ncol(C)^2),ncol(C))

  for (i in 1:nrow(C)) {
    if (i %in% order) {
      j = match(i,order)
      C_new[j,]=C[i,]
    }
  }

  ### get vector u
  u <- c(rep(0,ncol(C)))
  for (i in 1:length(u)) {
    connected_edge <- sum(A[i,])
    total_edge <- sum(G[order[i],])
    u[i] <- total_edge-connected_edge
  }

  ### get lt(AC)
  low_tri_template <- lower.tri(A, diag=TRUE)
  AC <- A %*% C_new
  for(row in 1:nrow(AC)) {
      for(col in 1:ncol(AC)) {
          if(low_tri_template[i,j]) {
            AC[row,col] <- AC[row,col]
          } else {
            AC[row,col] <- 0
          }
      }
  }

  ### get s
  one_vec <- rep(1,nrow(AC))
  s <- t(AC) %*% one_vec + t(C_new) %*% u
  return(s)
}
```

```r
### for now, only think of cases where |M|=1
num_seeds <- 1

### repeat simulations to get vector of lambda_hats
runs <- 1000
lambda_hat <- rep(0,runs)

### initiate values
G <- matrix(c(0,1,1,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0,1,1,0
isSymmetric(G)
```

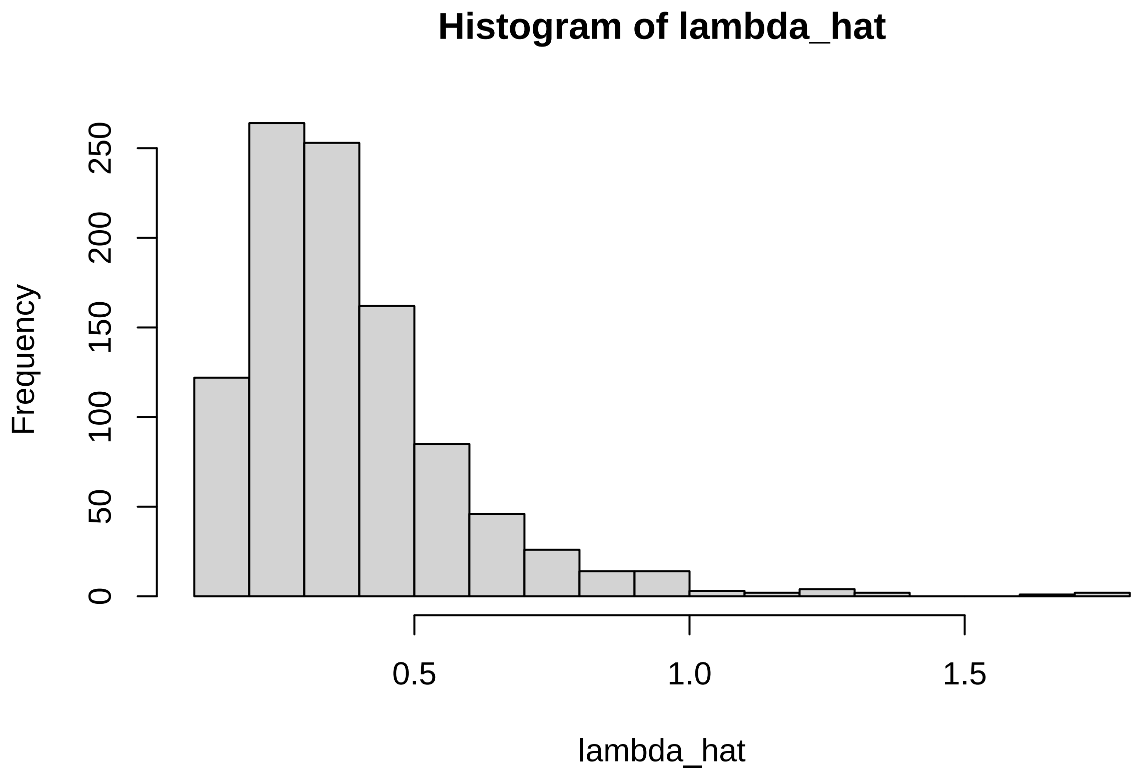
```
## [1] TRUE
```

```r
lambda <- 0.5
n <- 8
start_index <- 1
num_coupon <- 2
```

```r
for (i in 1:runs) {
  results <- rds_simulation(G, lambda, n, start_index, num_coupon)
  G_R <- results[[1]]
  order <- results[[2]]
  w <- results[[3]]
  C <- results[[4]]

  s <- get_s(G_R, order, C, G)
  lam_hat <- (n-num_seeds)/(t(s) %*% w)
  lambda_hat[i] <- lam_hat
}
```

```r
### visualize the distribution of lambda_hat
hist(lambda_hat,breaks=20)
```



```r
### find the expected value of lambda_hat
mean(lambda_hat)
```

```
## [1] 0.3853575
```