Problem Set 2

Tony Lashley

April 13, 2015

1 Machinery for the Schelling Model

1.1 Write a function that calculates distances between coordinate points

```
individual <- c(x = 0,y = 0)
print(individual)

## x y
## 0 0

neighbors = matrix(1:8, ncol = 2, byrow = T)
print(neighbors)

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8

colnames(neighbors) <- c("X","Y")
dstances = matrix(ncol = 3, byrow = T)
colnames(dstances) <- c("X","Y", "Pythgorean")

f1 <- function(individual, neighbors){

  for (i in 1:nrow(neighbors)){

    neighbor_longitude = neighbors[i,1]
    ## Find your neighbor's longitude
    neighbor_latitude =  neighbors[i,2]
    ## Find your neighbor's latitude
    individual_longitude = individual[1]
    ## Find your own longitude
```

```
    individual_latitude =  individual[2]
    ## Find your own latitude

    lftrghtdstance = abs(neighbor_longitude - individual_longitude)
    ## Find east/west distance between indiv. and neighbor
    updowndstance = abs(neighbor_latitude - individual_latitude)
    ## Find north/south distance between indiv. and neighbor
    pyth = sqrt(((lftrghtdstance)^2) + ((updowndstance)^2))
    ## Find Euclidian distance
    currentdistance = c(lftrghtdstance,updowndstance,pyth)
    ## Make vector with Manhattan and Euclidian distances


    dstances <- rbind(dstances, currentdistance)
    ## Add vector as row in matrix of distances
  }

  return(dstances)
}

f1(individual, neighbors)

##                  X  Y Pythgorean
##                 NA NA         NA
## currentdistance  1  2   2.236068
## currentdistance  3  4   5.000000
## currentdistance  5  6   7.810250
## currentdistance  7  8  10.630146
```

1.2 Write a function that simulates Schelling's Segregation model

```
library(RANN)
library(FNN)
library(ggplot2)
library(reshape2)

library(foreach)
library(doParallel)

## Loading required package:  iterators
## Loading required package:  parallel

library(parallel)
```

```
require(foreach)
require(doParallel)
require(parallel)
require(ggplot2)

numCores <- detectCores()
cl <- makeCluster(numCores)
registerDoParallel(cl)

testRacialPreferenceTable <- matrix(1:15, ncol = 5, nrow = 3)
testRacialPreferenceTable[1,] <- c("R",1, 20, 5, 2)
testRacialPreferenceTable[2,] <- c("G", 0, 10, 5, 2)
testRacialPreferenceTable[3,] <- c("B", -1, 10, 5, 2)
colnames(testRacialPreferenceTable) <- c("Color", "Value", "Pop.", "Test Pool Size", "Racial
print(testRacialPreferenceTable)

##      Color Value Pop. Test Pool Size Racial Threshold
## [1,] "R"   "1"   "20" "5"            "2"
## [2,] "G"   "0"   "10" "5"            "2"
## [3,] "B"   "-1"  "10" "5"            "2"

nR <- as.numeric(testRacialPreferenceTable[1,"Pop."])
nG <- as.numeric(testRacialPreferenceTable[2,"Pop."])
nB <- as.numeric(testRacialPreferenceTable[3,"Pop."])

n <- sum(nR + nG + nB)
## Find total population from summing each racial population

inputs <- testRacialPreferenceTable

stop.val <- .95
happy_counter <- 0

Schelling <- function(racialPreferenceTable = testRacialPreferenceTable){
  set.seed(20016)
  library(ggplot2)
  LocationTable <- matrix(ncol = 3)
  ## Initalizing table for initial neighborhood coordinates

  for (i in 1:nR){
    x <- runif(1, min=0, max=1)
    ## Generate random X coordinate between 0 and 1 for point
    y <- runif(1, min=0, max=1)
    ## Generate random Y coordinate between 0 and 1 for point
    currentpointR = c(1,x,y)
```

```
  ## Create vector with point coordinates, labeling point as red
  LocationTable <- rbind(LocationTable, currentpointR)
  ## Add red point to table of all neighborhood coordinates
}

for (i in 1:nG){
  x <- runif(1, min=0, max=1)
  y <- runif(1, min=0, max=1)

  currentpointG = c(0,x,y)

  LocationTable <- rbind(LocationTable, currentpointG)
}

for (i in 1:nB){
  x <- runif(1, min=0, max=1)
  y <- runif(1, min=0, max=1)
  currentpointB = c(-1,x,y)

  LocationTable <- rbind(LocationTable, currentpointB)
}

LocationTable <- LocationTable[-1,]
Count <- c(1:nrow(LocationTable))
## Create column counting number of points or people

Happy <- c(rep(0, nrow(LocationTable)))
## Create column to keep track of if person is happy

LocationTable <- cbind(Count, LocationTable, Happy)
## Add columns to Location Table

print(LocationTable)

p <- qplot(x = LocationTable[,3], y = LocationTable [,4], col = ifelse(LocationTable[,2]

print(p)


testpoolR <- as.numeric(racialPreferenceTable[1,4])
## Pull m value for given race
thresholdR <- as.numeric(racialPreferenceTable[1,5])
##Pull j value for given race

testpoolG <- as.numeric(racialPreferenceTable[2,4])
```

```
  thresholdG <- as.numeric(racialPreferenceTable[2,5])

  testpoolB <- as.numeric(racialPreferenceTable[3,4])
  thresholdB <- as.numeric(racialPreferenceTable[3,5])

  maxtestnumb <- max(testpoolR, testpoolG, testpoolB)
  #Finding max testpool value so we can create neighborlist outside loop
  testpool <- maxtestnumb
  ## Initializing testpool variable as max testpool number outside loop

  minthresholdnumb <- min(thresholdR, thresholdG, thresholdB)
  threshold <- minthresholdnumb

  justXYtable = LocationTable[,-1]
  #Make seperate table with race value column removed

  neighborList <- get.knn(data = justXYtable, k = maxtestnumb)$nn.index
  ## Create matrix of m closest neighbors for each point

  print(neighborList)

  bad_neighbors <- 0
  good_neighbors <- 0
  total_neighbors <- bad_neighbors + good_neighbors
  ##Initialize value for number of neighbors individual likes and doesn't

  cycles <- 0
  row <- 0

while ((happy_counter/n) < stop.val){

    happy_counter<- sum(LocationTable[,5])

    for (row in sample(1:n)){
    ##For a point in the location table...

      own_race <- LocationTable[row,1]

      if(LocationTable[row,1] == 1){
      ##If the point is red...

        testpool <- testpoolR
        ## Pull m value for individual given race
        threshold <- thresholdR
        ##Pull j value for indvidual given race
```

```
    own_race <- 1
  }

  if(LocationTable[row,1] == 0){
  ##If the point is green...

    testpool <- testpoolG
    threshold <- thresholdG
    own_race <- 0
  }

  if(LocationTable[row,1] == -1){
  ##If the point is blue...

    testpool <- testpoolB
    threshold <- thresholdB
    own_race <- -1
  }

  for (column in sample(1:testpool)){
  ## For each closest neighbor of the given point

    a_neighbor <- neighborList[row,column]
    ## Find numerical value of neighboor in Location matrix

    a_neighbors_race <- LocationTable[a_neighbor,1]
    ## Find neighbor's race

      while ((bad_neighbors + good_neighbors) < testpool){

        if (a_neighbors_race != own_race){

          bad_neighbors <- bad_neighbors + 1
          ## If a neighbor's race is different from individual's,
          ## increase number of bad neighbors

            if (bad_neighbors > threshold){
            ##If the number of bad neighbors exceeds threshold...

              new_x <- runif(1, min=0, max=1)
              new_y <- runif(1, min=0, max=1)
              LocationTable[row,3] <- new_x
              LocationTable[row,4] <- new_y
              cycles <- cycles + 1
```

```
            }
          }

          if (a_neighbors_race == own_race){
          good_neighbors <- goodneighbors + 1

            if ((good_neighbors + bad_neighbors) == testpool){
              LocationTable[row,5] = 1
            }

          }
        }
      }
    }
  }
}

# results <- foreach(i=testRacialPreferenceTable) %dopar% {
#    Schelling(i)
#  }
```

2 Machinery for the Schelling Model