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)
```

```r
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
    R = c(1,x,y)
```

```
  ## Create vector with point coordinates, labeling point as red
  LocationTable <- rbind(LocationTable, R)
  ## 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)

  G = c(0,x,y)

  LocationTable <- rbind(LocationTable, G)
}

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

  LocationTable <- rbind(LocationTable, B)
}

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

Testpool <- c(rep(0, nrow(LocationTable)))
## Create column for individual's testpool

Threshold <- c(rep(0, nrow(LocationTable)))
## Create column for individual's threshold

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


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])

for (row in 1:nrow(LocationTable)){

  own_race <- LocationTable[row,2]

  if(own_race == 1){
  ##If the point is red...

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

      LocationTable[row,6] <- testpool
      LocationTable[row,7] <- threshold

    }

    if(own_race == 0){
    ##If the point is green...

      testpool <- testpoolG
      threshold <- thresholdG

      LocationTable[row,6] <- testpool
      LocationTable[row,7] <- threshold

    }

    if(own_race == -1){
    ##If the point is blue...

      testpool <- testpoolB
      threshold <- thresholdB

      LocationTable[row,6] <- testpool
      LocationTable[row,7] <- threshold
```

```
    }
  }

  print(LocationTable)

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

  LoopUnhappyLocationTable <- LocationTable

  justXYtable = LocationTable[,3:4]
  #Make seperate table with race value column removed for
  #nearest neighbor function

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

  print(neighborList)

  ##Initialize value for total number of neighbors evaluate

  cycles <- 0


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

    NumUnhappy <- nrow(LoopUnhappyLocationTable)

    cycles <- cycles + 1
    happy_counter<- sum(LocationTable[,5])

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

      for (column in 1:(LocationTable[row,6])){
      ## For each closest neighbor of the given point

        neighborList <- neighborList[,1:LocationTable[row,6]]
        ##Get rid of extraneous neighbors who are ranked lower than
        ## k closest

        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) < (LocationTable[row,6])){

            good_neighbors <- 0
            bad_neighbors <- 0

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

              if ((good_neighbors + bad_neighbors) == (LocationTable[row,6])){
                LocationTable[row,5] = 1
                LoopUnhappyLocationTable = LoopUnhappyLocationTable[-row,]
              }
            }

            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 > (LocationTable[row,7])){
                ##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
                  LoopUnhappyLocationTable[row,3] <- new_x
                  LoopUnhappyLocationTable[row,4] <- new_y

                }
            }
          }
        }
    }

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

    if (cycles %% 5 == 0) {print(p)}
 }

return(cycles)
```

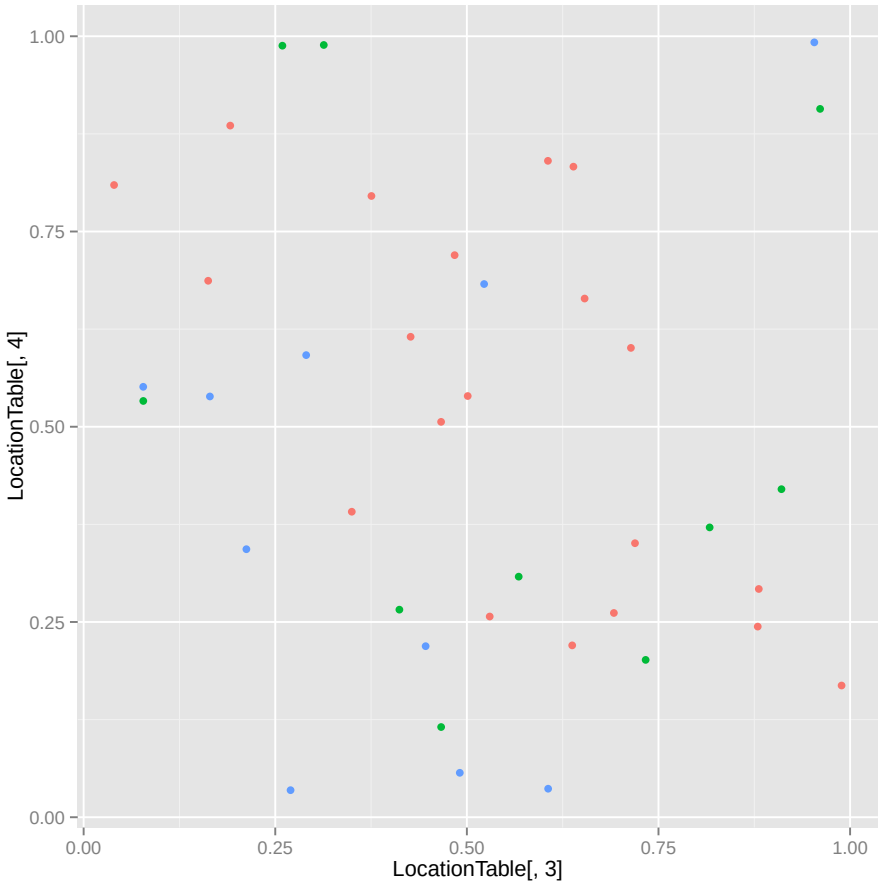
```
}

Schelling(testRacialPreferenceTable)
```

```
##   Count                          Happy Testpool Threshold
## R     1  1 0.88087670 0.29227436     0        5         2
## R     2  1 0.50111460 0.53932951     0        5         2
## R     3  1 0.34984375 0.39117480     0        5         2
## R     4  1 0.71935381 0.35087338     0        5         2
## R     5  1 0.46636470 0.50625997     0        5         2
## R     6  1 0.37548038 0.79547449     0        5         2
## R     7  1 0.48403496 0.71971009     0        5         2
## R     8  1 0.16247257 0.68690637     0        5         2
## R     9  1 0.19127365 0.88560590     0        5         2
## R    10  1 0.71405276 0.60097207     0        5         2
## R    11  1 0.60577097 0.84048701     0        5         2
## R    12  1 0.63742605 0.22005018     0        5         2
## R    13  1 0.03969718 0.80952938     0        5         2
## R    14  1 0.69181073 0.26153457     0        5         2
## R    15  1 0.65363839 0.66415248     0        5         2
## R    16  1 0.63901260 0.83302126     0        5         2
## R    17  1 0.52979992 0.25700429     0        5         2
## R    18  1 0.98889149 0.16865283     0        5         2
## R    19  1 0.42661108 0.61516778     0        5         2
## R    20  1 0.87949560 0.24409957     0        5         2
## G    21  0 0.31329813 0.98882309     0        5         2
## G    22  0 0.41195102 0.26577442     0        5         2
## G    23  0 0.25931412 0.98791598     0        5         2
## G    24  0 0.73325532 0.20144791     0        5         2
## G    25  0 0.07781230 0.53302146     0        5         2
## G    26  0 0.96087329 0.90699890     0        5         2
## G    27  0 0.56762140 0.30803760     0        5         2
## G    28  0 0.81676286 0.37110086     0        5         2
## G    29  0 0.46643300 0.11548545     0        5         2
## G    30  0 0.91045513 0.42004432     0        5         2
## B    31 -1 0.95330606 0.99217334     0        5         2
## B    32 -1 0.21233086 0.34319235     0        5         2
## B    33 -1 0.60603245 0.03649611     0        5         2
## B    34 -1 0.49077000 0.05688147     0        5         2
## B    35 -1 0.07772721 0.55111953     0        5         2
## B    36 -1 0.52250407 0.68272916     0        5         2
## B    37 -1 0.44620386 0.21901630     0        5         2
## B    38 -1 0.26996361 0.03462394     0        5         2
## B    39 -1 0.16472249 0.53875963     0        5         2
## B    40 -1 0.29034521 0.59171952     0        5         2
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   20   28   30   18    4
##  [2,]    5   19   36    7   15
##  [3,]   22   32    5   37   40
##  [4,]   14   28   24   12   27
##  [5,]    2   19    3   36   40
##  [6,]    7   36   19   21    9
##  [7,]   36   19    6   11   15
##  [8,]   39   40   35   13   25
##  [9,]   23   21   13    8    6
## [10,]   15   36    2   16    4
## [11,]   16    7   36   15    6
## [12,]   14   24   27   17    4
## [13,]    9    8   35   25   23
## [14,]   12   24    4   27   17
## [15,]   10   36   16    7   11
## [16,]   11   15   36    7   10
## [17,]   27   37   12   22   29
## [18,]   20    1   24   30   28
## [19,]    2    5   36    7   40
## [20,]    1   18   28   24   30
## [21,]   23    9    6    7   13
## [22,]   37   17    3   29   27
## [23,]   21    9    6   13    8
## [24,]   14   12    4   20    1
## [25,]   35   39    8   40   32
## [26,]   31   16   11   15   10
## [27,]   17   12   14   37    4
## [28,]    4    1   30   20   14
## [29,]   34   37   17   22   33
## [30,]   28    1   20    4   18
## [31,]   26   16   11   15   10
## [32,]    3   39   22   25   35
## [33,]   34   29   12   24   17
## [34,]   29   33   37   17   12
## [35,]   25   39    8   40   32
## [36,]    7   19   15    2   11
## [37,]   22   17   29   27   34
## [38,]   29   34   37   22   32
## [39,]   25   35   40    8   32
## [40,]   39   19    8    5    3

## Error:  object 'bad_neighbors' not found
```




```
# print(system.time(Schelling(testRacialPreferenceTable)))
```

2 Machinery for the Schelling Model