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
```

```r
    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

```r
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,]
print(LocationTable)

p <- qplot(x = LocationTable[,2], y = LocationTable [,3], col = ifelse(LocationTable[,1] <

print(p)


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

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

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

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

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

```
}

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

  testpool <- as.numeric(racialPreferenceTable[2,4])
  threshold <- as.numeric(racialPreferenceTable[2,5])
}

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

  testpool <- as.numeric(racialPreferenceTable[3,4])
  threshold <- as.numeric(racialPreferenceTable[3,5])

}

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

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

  a_neighbor <- neighborList[row,column]
  ## Find numerical value of neighbor in Location matrix
  a_neighbors_race <- LocationTable[a_neighbor,1]
  ## Find neighbor's race

  own_race <- LocationTable[row,1]
  ##Find individual's race
  bad_neighbors <- 0
  ##Initialize value for number of neighbors individual doesn't want

  cycles <- 0

    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)
```

```r
              LocationTable[row,2] <- new_x
              LocationTable[row,3] <- new_y
              cycles <- cycles + 1

            }
          }
      }
    }

}

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

```
## Error in {:  task 1 failed - "subscript out of bounds"
```

```r
Schelling(testRacialPreferenceTable)
```

```
##                [,1]       [,2]       [,3]
## currentpointR    1 0.88087670 0.29227436
## currentpointR    1 0.50111460 0.53932951
## currentpointR    1 0.34984375 0.39117480
## currentpointR    1 0.71935381 0.35087338
## currentpointR    1 0.46636470 0.50625997
## currentpointR    1 0.37548038 0.79547449
## currentpointR    1 0.48403496 0.71971009
## currentpointR    1 0.16247257 0.68690637
## currentpointR    1 0.19127365 0.88560590
## currentpointR    1 0.71405276 0.60097207
## currentpointR    1 0.60577097 0.84048701
## currentpointR    1 0.63742605 0.22005018
## currentpointR    1 0.03969718 0.80952938
## currentpointR    1 0.69181073 0.26153457
## currentpointR    1 0.65363839 0.66415248
## currentpointR    1 0.63901260 0.83302126
## currentpointR    1 0.52979992 0.25700429
## currentpointR    1 0.98889149 0.16865283
## currentpointR    1 0.42661108 0.61516778
## currentpointR    1 0.87949560 0.24409957
## currentpointG    0 0.31329813 0.98882309
## currentpointG    0 0.41195102 0.26577442
## currentpointG    0 0.25931412 0.98791598
## currentpointG    0 0.73325532 0.20144791
## currentpointG    0 0.07781230 0.53302146
## currentpointG    0 0.96087329 0.90699890
```

```
## currentpointG     0 0.56762140 0.30803760
## currentpointG     0 0.81676286 0.37110086
## currentpointG     0 0.46643300 0.11548545
## currentpointG     0 0.91045513 0.42004432
## currentpointB    -1 0.95330606 0.99217334
## currentpointB    -1 0.21233086 0.34319235
## currentpointB    -1 0.60603245 0.03649611
## currentpointB    -1 0.49077000 0.05688147
## currentpointB    -1 0.07772721 0.55111953
## currentpointB    -1 0.52250407 0.68272916
## currentpointB    -1 0.44620386 0.21901630
## currentpointB    -1 0.26996361 0.03462394
## currentpointB    -1 0.16472249 0.53875963
## currentpointB    -1 0.29034521 0.59171952
```