Scientific Programming Assignment 1

Changmin Yu

October 2017

1 Words

1.1

In the first part of Question 1, I import the `usr-share-dict-words.txt` from my home directory using R function *readLine*() instead of using *read.table*() since after comparison, the output of *readLine*() is much faster in applying operations to all elements in the data frame especially when the data size is very large.
After converting all words to upper case using function *toupper*() and using *unique*() to obtain a list of unique words, there are 97723 words in the remaining list, a reduction from initial size 99171.

1.2

In part 2, I use function *grepl*() to find the words contain an apostrophe in the list. There are in total 25751 words contain an apostrophe in the list.

1.3

In part 3, I use a combination of function *grepl*() and *iconv*() to perform the search for words contain non-ASCII characters. There are in total 159 words contain non-ASCII in the current list after removing words contain an apostrophe in part 2.

1.4

In part 4, I use the function *substr*() to obtain the last-2 and last-4 characters in each word and then loop through the whole list to obtain two lists containing words end with 'OG' and 'OGUE' respectively. And eventually, I found 10 such pairs:

```
[1] "ANALOG"     "ANALOGUE"   "CATALOG"    "CATALOGUE"  "DEMAGOG"    "DEMAGOGUE"
 [7] "DIALOG"     "DIALOGUE"   "EPILOG"     "EPILOGUE"   "MONOLOG"    "MONOLOGUE"
[13] "PEDAGOG"    "PEDAGOGUE"  "PROLOG"     "PROLOGUE"   "SYNAGOG"    "SYNAGOGUE"
[19] "TRAVELOG"   "TRAVELOGUE"
```

1.5

In part 5, after importing the file `scrabble.txt` using *read.table*(), I used a for loop to store the information into a 26×2 matrix, note that when storing the information of the letters into the matrix, letters become numerical values in their alphabetical order, i.e., A is corresponding to 1, B is corresponding to 2 and so on. Then I reorder the matrix in terms of its first column (the column contain the information of the letters) and hence the second column of the matrix after reordering is our desired vector *scores* which stores the scrabble score of the *i*th letter of the alphabet.
The vector *scores* is shown below;

```
[1]  1  3  3  2  1  4  2  4  1  8  5  1  3  1  1  3 10  1  1  1  1  4  4  8  4 10
```

1.6

In part 6, I use a for loop through the list to calculate the individual score of each word in the *database*, within each iteration, I use function *which*() to obtain the scrabble score of each letter in the word and then sum up to get the scrabble score for each word. Then I sue *plot*(*density*)) to plot the distribution of the scores, the density plot is shown below:

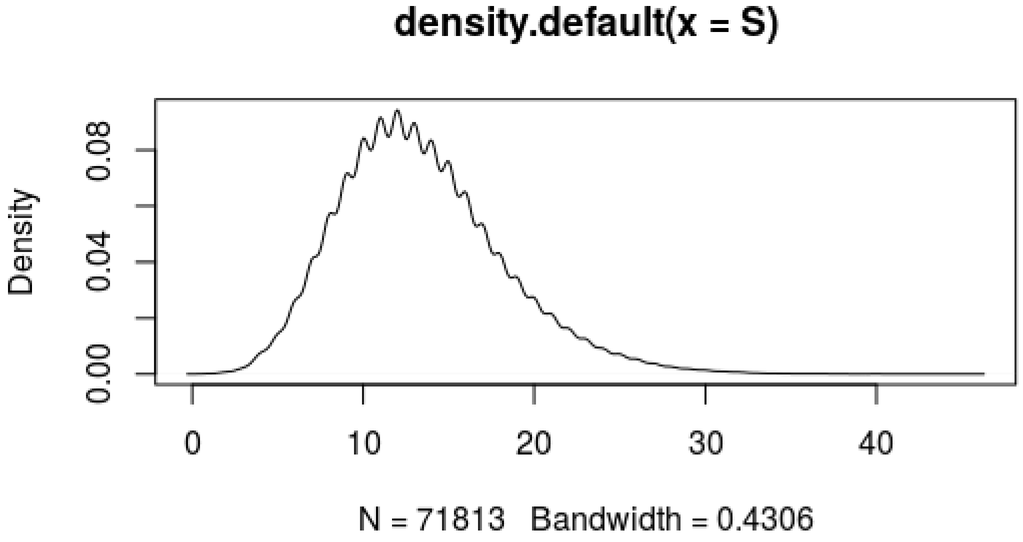

Figure 1: Density plot of scrabble scores of words

And after ranking the scrabble scores of the words in the list, the highest scoring-word is PIZZAZZ, and its scrabble score is 45.

1.7

In part 7, in order to efficiently obtain the reverse complement of each words in the list, I wrote a function called *reverse.complement*, which takes an input of a character (which will return FALSE if the input is not an character) and gives an output of its reverse complement based on the rule given in the question.
Then, in finding the words that both itself and its reverse complement are in the database, I used a specific operator in R, *%in%*, which works in the same way as *in* words in python, and after looping through the list, I obtain the list of such words, theh list is as following:

```
> l.rev
  [1] "A"       "Z"       "AL"      "OZ"      "ARIZ"    "AS"      "HZ"      "B"
  [9] "Y"       "BIRD"    "WIRY"    "C"       "X"       "CI"      "RX"      "CL"
 [17] "OX"      "D"       "W"       "E"       "V"       "F"       "U"       "FLO"
 [25] "LOU"     "FM"      "NU"      "G"       "T"       "GILL"    "OORT"    "H"
 [33] "S"       "HE"      "VS"      "HF"      "US"      "HG"      "TS"      "HO"
 [41] "LS"      "HORN"    "MILS"    "I"       "R"       "IN"      "MR"      "IO"
 [49] "LR"      "IR"      "J"       "Q"       "K"       "P"       "L"       "O"
 [57] "LN"      "MO"      "LT"      "GO"      "LYRA"    "ZIBO"    "M"       "N"
 [65] "MILO"    "LORN"    "MN"      "ND"      "WM"      "NOV"     "ELM"     "OK"
 [73] "PL"      "ORLY"    "BOIL"    "POLK"    "RH"      "SI"      "RN"      "MI"
 [81] "UR"      "IF"      "YB"      "ZN"      "MA"      "ZOLA"    "ASP"     "KHZ"
 [89] "BEVY"    "BID"     "WRY"     "BIG"     "TRY"     "BOORISH" "SHRILLY" "BOORS"
 [97] "HILLY"   "BY"      "GIRT"    "GRID"    "WRIT"    "GRIT"    "HI"      "RS"
[105] "HOOFS"   "HULLS"   "HOVELS"  "LO"      "MILD"    "WORN"    "MILK"    "PORN"
```

```
[113] "SH"       "TRIG"     "VELD"     "WOVE"     "VOLE"     "WILD"     "WORD"     "WIZARD"
```

1.8

In the last part of Question 1, I write up two functions to search for words satisfy the requirements stated. My first function is *check.once*(), this function is to check the condition that each letter is used only once (except for L since there are two Ls in the list). The second function is *match.*1(), which is used to check that every letter in each word is in the list given and that the word contains A. Using the two function together in a for loop and we have all the required words. The list is shown as following:

```
 [1] "AINU"      "FULANI"    "LILA"      "LINA"      "LULA"      "LUNA"
 [7] "PAUL"      "PAULI"     "PIAF"      "YALU"      "YUAN"      "ALLY"
[13] "FAIL"      "FAIN"      "FALL"      "FAUN"      "FINAL"     "FINALLY"
[19] "FLAIL"     "FLAN"      "FLAP"      "FLAY"      "INLAY"     "LAIN"
[25] "NAIL"      "PAIL"      "PAILFUL"   "PAIN"      "PAINFUL"   "PAINFULLY"
[31] "PALL"      "PILAF"     "PILAU"     "PLAIN"     "PLAINLY"   "PLAN"
[37] "PLAY"      "PLAYFUL"   "ULNA"
```

As we can see from the list shown above, the only nine-letter word is PAINFULLY.

2 Examination marking

2.1

In Question 2, I first import the student grade files and crib file into R workspace using *dataFiles*() and *read.table*(). Now I will produce the table step by step, by generating vectors *correct*, *alpha.grades* and *rank* respectively, I do not need to compute the *num.students* since in this case it is trivially 12.
Amongst the remaining three terms, the most important one to compute is the vector *correct*, after which we could easily obtain *alpha.grades* and *rank*. To compute *correct*, I first transform column 1 and column 2 of each student's grade data and the data of actual correct answer into matrix form using *data.matrix*(), then for each individual student, I loop through the 30 questions the student answers in each iteration and by matching to the *crib* vector to compute the number of correct answers the student gets in each iteration.
After we have obtained *correct* vector, import the file `grade.txt` and transform the scores in *correct* vector to get the percentage scores of the students and match them with the data in `grade.txt` and hence compute the vector *alpha.grades*.
Finally, using either the *correct* vector or the vector contains the percentage scores of students to rank them using function *rank*(). One thing worth noticing is that there are two students having the exactly identical final grade, hence we must include the parameter `ties.method='min'` in the *rank*() function to rank the two students with equal grades as their co-higher rank.
The final table generated using *data.frame*() is as following:

```
> print(results)
   student score grade rank
1        1    19     B    6
2        2    24     A    2
3        3    14     D   10
4        4    17     C    8
5        5    20     B    5
6        6    23     A    3
7        7    29     A    1
8        8     7     F   12
9        9    22     A    4
10      10    17     C    8
```

```
11      11     9     F   11
12      12    18     B    7
```

2.2

Now, to check that whether a selected pair of students have similar results, I write a function called *check.sim*(a,b) which takes two inputs: the student IDs (1,...,12) for the selected two students and returns an output TRUE if cheating was detected and FALSE if not. Note that although before when I imported the data files of the students' grades into R workspace, the order of the students are actually not correctly stored, when I created the vector *correct*, I have successfully converted the vector into its correct order, hence we should not worry about this issue when applying this function. My R code for *check.sim*(a,b) is as shown below.

```
check.sim <- function(a, b){
  if (abs(correct[a]-correct[b])<=2){
    return(FALSE)
  }
  else{
    if (length(intersect(dataFiles[a]$V1[2:31], dataFiles[b]$V1[2:31]))>=15){
      return(TRUE)
    }
    else{
      return(FALSE)
    }
  }
}
```

Note that here *dataFiles* is the list of student grades data imported.
After applying the function to the student grades data, one of student 2 and student 6 are the most likely pair of students who committed the plagiarism.

3 DMB question 10

3.1

In Question 3, we will be dealing with the simulation project in Part 10 in the DMB practicals.
In the first part, we are asked to start with 5 individuals in each patch at time t=1, and loop through each iteration t=1 to t=50. In each iteration, we first need to compute the vector of geometric growth within patches based on the information of λ_js for each patch and the number of individuals in each patch in the previous time state. Then, by the given dispersal law between neighbouring patches, we could firstly allocate memory for an empty 50×20 matrix and then set each row j equal to the distribution of number of individuals in each patch at time t=j, where j=1,...,50. After we have obtained this matrix, we could then plot the logarithm sums of individuals at time t=1,...,50 against the time. The sums of individuals at time t=1,...,50 is shown below:

```
> s
 [1]    100.0000    105.0000    112.4550    122.6705    136.0337    153.0261
 [7]    174.2387    200.3915    232.3561    271.1849    318.1443    374.7562
[13]    442.8462    524.6034    622.6498    740.1253    880.7879   1049.1351
[19]   1250.5476   1491.4626   1779.5806   2124.1121   2536.0744   3028.6449
[25]   3617.5852   4321.7473   5163.6805   6170.3560   7374.0353   8813.3081
[31]  10534.3347  12592.3320  15053.3522  17996.4101  21516.0292  25725.2880
[37]  30759.4648  36780.3995  43981.7120  52595.0480  62897.5510  75220.8054
[43]  89961.5369 107594.4183 128687.3932 153920.0146 184105.3902 220216.4452
[49] 263417.3523 315101.1451
```

The graph of the logarithm of the total population size over time is as following:

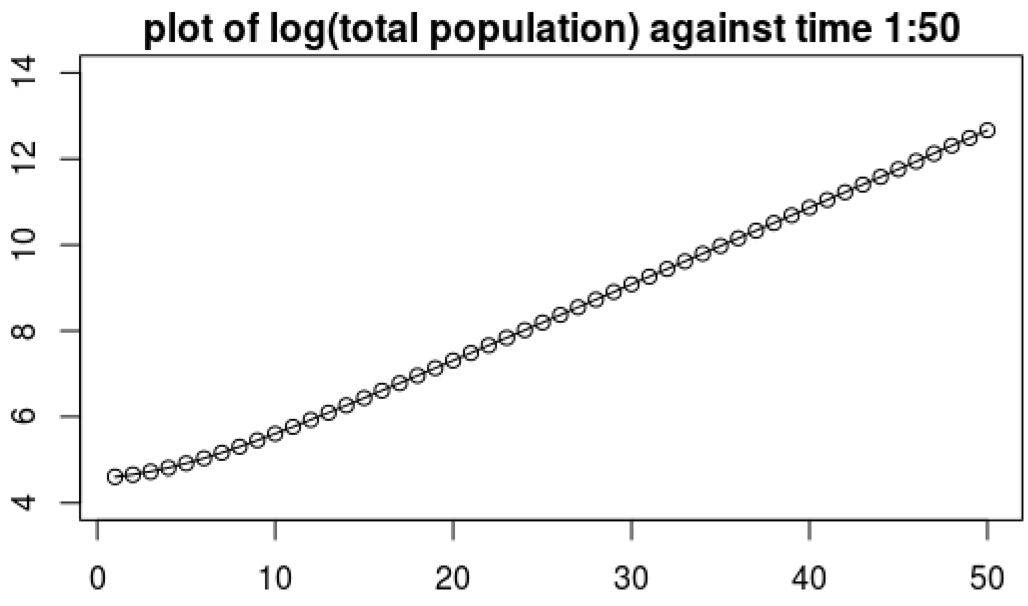

Figure 2: Plot of logarithm of total population against time

3.2

In the next part, the task is to write a program so that d and L are parameters rather than fixed constants, hence I write a function called $simulation.project(d,L)$ which takes inputs of d as half of the 'dispersal rate' and L as the number of batches. The growth rates λ_js are fixed global variable in this case and treated as given constants in this function.
The R code for the function is as following:

```
simulation.project <- function(L, d){
  # this function is as required in the second part of
  # the question
  v <- rep(5, L)
  l.lambda <- 0.9
  r.lambda <- 1.2
  V <- matrix(0, 50, L)
  V[1,] <- v
  s <- rep(0, 50)
  s[1] <- 5*L

  for (t in 1:49){
    # compute the geometric growth within patches
    M <- c(l.lambda*V[t, 1:as.integer(L/2)], r.lambda*V[t, (as.integer(L/2)+1):L])
    V[t+1, 2:L-1] <- (1-2*d)*M[2:(L-1)] + d*M[1:(L-1)] + d*M[3:L]
    V[t+1, 1] <- (1-d)*M[1] + d*M[2]
    V[t+1, 20] <- (1-d)*M[L] + d*M[L-1]
    s[t+1] <- sum(V[t+1,])
  }

  plot(1:50, log(s), type='o', xlab='days',
       ylab='log(total population size)')
  return(s)
}
```

3.3

In this part, we are asked to explore the effects of spatial arrangement of good versus bad habitat patches have on the population growth rate, and by good and bad habitat patches I mean that a good patch j associated with $\lambda_j > 1$ and a bad one j is associated with $\lambda_j < 1$. One thing worth noticing is that while changing the spatial arrangement of the habitats, one should not change its proportion of numbers of good versus bad patches since this will contaminate the results.
To carry out such investigation, three general modes will be examined respectively: all good sites are at one end of the model, all good sites are in the middle of the model, and good sites are distributed randomly (evenly) within the model. To do such, I first need to write an updated version of the function $simulation.project()$ which takes an additional input of spatial arrangement, which is a vector storing the positions of the good sites within the model. And then carry out similar operations to compute the vector containing the total population from t=1 to t=50.
Then, to explore the effects of these spatial arrangements onto the total population growth rates, I write a function $population.growth(a)$ which takes an input of a vector and then returns a vector (of length 1 less than the input vector) which contains the information of growth rate from t=i to t=i+1 for i=1,...,50. Then I compute such vectors for the 3 general patterns mentioned above (two for the random arrangements) and plot them against the time respectively. The plots are shown as following:

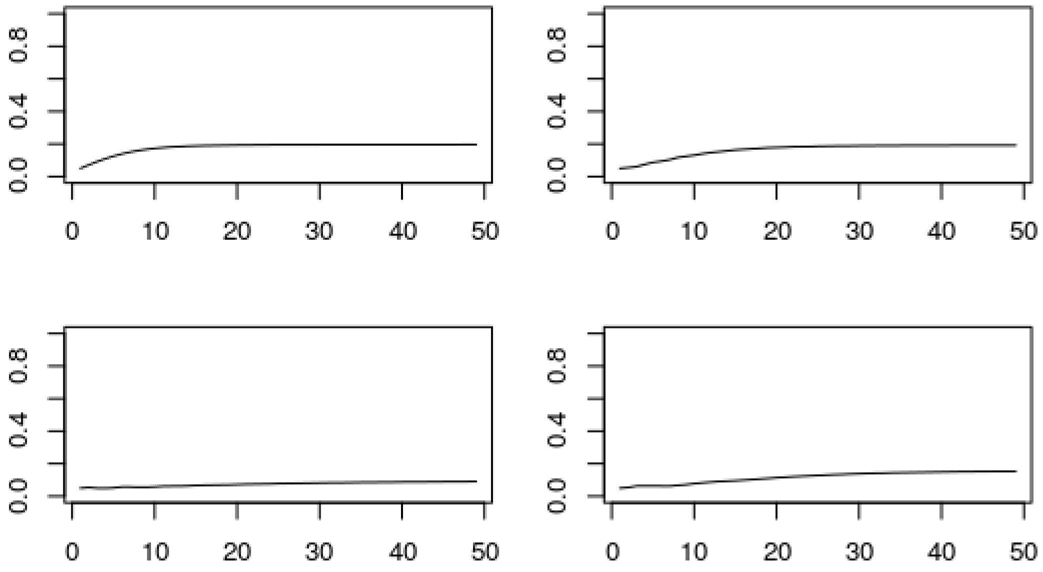

Figure 3: Plot of population growth rate against time

The four plots are respectively representing growth rates against time for spatial arrangements: good ones are in the left end, good ones are in the middle and bottom two are good ones are distributed randomly (evenly) in the model.
As we can see from the four plots and the final outputs of the growth rate vectors under the four conditions, we could safely draw the conclusion that: it does not affect the total population growth rates if the good ones are clustered in one end or in the middle, the population growth rate in each iteration converges to approximately 19% to 20%, but if the good patches are spread out evenly or randomly (in some sense) across the entire habitat, the total population growth rates converges to different values depending on the specific arrangements ranging from 8% to 15%.

3.4

In the last part of this project, we are asked to write a function which takes two inputs: d for growth rates and M(t) for pre-dispersal population and returns a vector contains the number of individuals in each patch in the next time step t+1. This part is rather trivial and the R code for doing this is as following:

```
reflecting <- function(M, d){
  m <- length(M)
  N.1 <- (1-d)*M[1]+d*M[2]
  N.L <- (1-d)*M[m]+d*M[m-1]
  N <- (1-2*d)*M[2:m-1]+d*M[1:m-2]+d*M[3:m]
  N <- c(N.1, N, N.L)
  return(N)
}
```

Note that in this question (and in a few parts in previous questions) I have taken great advantage to use linear algebra operations to replace loops, which is more efficient comparing to the alternative. As in this case, instead of using for-loops to obtain the correct pre-dispersal population, I used only vector addition and scalar multiplication, which makes the algorithm faster and more concise for interpretation.

A R Code for Question 1

```
# 1.1
df <- readLines(
  '/home/cy302/SPA1/usr-share-dict-words.txt')
df <- toupper(df)
unique.df <- unique(df)
n.unique <- length(unique.df)
# there are 97723L (ignore the L) unique words

# 1.2
# in this part, I am going to use the grepl function
# to check for existence of the apostrophe in words
n.apos <- 0
l.apos <- c()
for (i in seq.int(n.unique)){
  ind <- grepl('\'', unique.df[i])
  if (ind == TRUE){
    n.apos <- n.apos + 1
    l.apos <- c(l.apos, i)
  }
}
unique.df <- unique.df[-l.apos]
print(n.apos)
# there are 25751 words contain an apostrophe

# 1.3
n.non.asc <- 0
l.non.asc <- c()
for (i in seq.int(length(unique.df))){
  ind <- grepl("NOT_ASCII",
               iconv(unique.df[i],"latin1","ASCII",
                     sub="NOT_ASCII"))
  if (ind==TRUE){
    n.non.asc <- n.non.asc + 1
    l.non.asc <- c(l.non.asc, i)
  }
}
unique.df <- unique.df[-l.non.asc]
print(n.non.asc)
# define database for the following questions
database <- unique.df

# 1.4
n <- length(database)
l.og <- c()
l.ogue <- c()
for (i in seq.int(n)){
  len.a <- nchar(database[i])
  if (substr(database[i],len.a-1,len.a)=='OG'){
    l.og <- c(l.og, database[i])
  }
  else if (substr(database[i], len.a-3, len.a)=='OGUE'){
    l.ogue <- c(l.ogue, database[i])
  }
}
l <- c()
for (j in 1:length(l.og)){
  len.og.j <- nchar(l.og[j])
  f <- substr(l.og[j], 1, len.og.j-2)
  for (k in 1:length(l.ogue)){
    len.ogue.k <- nchar(l.ogue[k])
    g <- substr(l.ogue[k], 1, len.ogue.k-4)
    if (f == g){
      l <- c(l, l.og[j], l.ogue[k])
    }
  }
}

# 1.5
scrabble <- read.table('/home/cy302/SPA1/scrabble.txt', header=FALSE)
m <- matrix(0, 26, 2)
for (i in 1:26){
  m[i, 1] <- scrabble[[1]][i]
  # this step directly transforms characters to numbers in terms of the
  # alphabetical order
  m[i, 2] <- scrabble[[4]][i]
}
m <- m[order(m[, 1], decreasing=FALSE), ]
scores <- m[,2]
# this is the scores vector we are looking for

#1.6
n <- length(database)
S <- rep(0, n)
for (i in seq.int(n)){
  s <- 0
  a <- database[i]
  for (j in 1:nchar(a)){
    s <- s+scores[which(LETTERS==substr(a, j, j))]
  }
  S[i] <- s
}
plot(density(S))
l.max.ind <- which.max(S)
l.max <- database[l.max.ind]
sc.max <- S[l.max.ind]
# the highest-scoring word is PIZZAZZ, and the highest score is 45, the 49913th
# element in the database

# 1.7
reverse.complement <- function(a){
  if (typeof(a) != 'character'){
    return(FALSE)
  }
  else{
    n <- nchar(a)
    a <- strsplit(a, '')[[1]]
    a <- rev(a)
    a <- paste(a, collapse = '')
    for (i in (1:n)){
      a.i <- substr(a, i, i)
      ind.old <- which(LETTERS==a.i)
      ind.new <- 27-ind.old
      substr(a, i, i) <- LETTERS[ind.new]
    }
    return(a)
  }
}
l.rev <- c()
for (i in seq.int(n)){
  a.old <- database[i]
  a.new <- reverse.complement(a.old)
  ind <- a.new %in% database
  if (ind){
    l.rev <- c(l.rev, a.old, a.new)
  }
}
l.rev <- unique(l.rev)

#1.8
l.9 <- c('F', 'A', 'L', 'U', 'Y', 'P', 'L', 'N', 'I')
check.once <- function(a){
  if (typeof(a)!='character'){
    return(FALSE)
  }
  else{
    ind <- c()
    char.a <- unlist(strsplit(a, ''))
    unique.char.a <- unique(char.a)
    for (i in 1:length(unique.char.a)){
      l <- length(char.a[char.a==unique.char.a[i]])
      if (l > 1 && unique.char.a[i]!='L'){
        ind <- c(ind, FALSE)
      }
      else if (l >2 && unique.char.a[i]=='L'){
        ind <- c(ind, FALSE)
      }
      else{
        ind <- c(ind, TRUE)
      }
    }
    return(all(ind))
  }
}
match.1 <- function(a, b){
  ind <- c()
  if (nchar(a) >= 4 && grepl('A', a)){
    for (i in 1:nchar(a)){
      if (substr(a, i, i) %in% b){
        ind <- c(ind, TRUE)
      }
      else{
        ind <- c(ind, FALSE)
      }
    }
    ind <- all(ind)
    return(ind)
  }
  else{
    return(FALSE)
  }
}
l.letters <- c()
for (i in seq.int(n)){
  a <- database[i]
  if (match.1(a, l.9) && check.once(a)){
    l.letters <- c(l.letters, a)
  }
}
```

B R code for Question 2

```
# Question 2
setwd('/home/cy302/SPA1/grade/student_grades/')
dataFiles <- lapply(Sys.glob("student*.dat.txt"), read.table)
crib <- read.table('/home/cy302/SPA1/grade/crib.dat.txt')
grade <- read.table('/home/cy302/SPA1/grade/grade.txt')

correct <- c()
for (i in 1:length(dataFiles)){
  mark <- 0
  a <- data.matrix(dataFiles[[i]]$V1[2:31])
  b <- data.matrix(dataFiles[[i]]$V2[2:31])
  c <- data.matrix(crib$V1)
  for (j in 1:30){
    ind <- as.numeric(a[j])
    student.answer <- b[j]
    right.answer <- c[ind]
    if (student.answer==right.answer){
      mark <- mark+1
    }
  }
  correct <- c(correct,mark)
}
correct <- c(correct[4:12], correct[1:3])
grade.crit <- read.table('/home/cy302/SPA1/grade/grade.txt', header=TRUE)
min <- grade.crit$min
max <- grade.crit$max
grade <- as.vector(grade.crit$grade)
alpha.grade <- rep(0, 12)
for (i in 1:12){
  g <- correct[i]
  g.percentage <- floor((g/30)*100)
  counter <- 1
  for (j in 1:5){
    if (min[j]>g.percentage){
      counter <- counter + 1
    }
    else{
      break
    }
  }
  alpha.grade[i] <- grade[counter]
}
rank.student <- rank(-correct, ties.method = 'min')
num.students <- 12
results <- data.frame(student=1:num.students, score=correct,
                      grade=alpha.grade, rank=rank.student)
print(results)

check.sim <- function(a, b){
  if (abs(correct[a]-correct[b])<=2){
    return(FALSE)
  }
  else{
    if (length(intersect(dataFiles[a]$V1[2:31], dataFiles[b]$V1[2:31]))>=15){
      return(TRUE)
    }
    else{
      return(FALSE)
    }
  }
}
```

C R code for Simulation Project

```
# simulation project
# part 1
L <- 20
v <- rep(5, L)
l.lambda <- 0.9
r.lambda <- 1.2
V <- matrix(0, 50, L)
V[1,] <- v
d <- 0.1
s <- rep(0, 50)
s[1] <- 5*L
for (t in 1:49){
  # compute the geometric growth within patches
  M <- c(l.lambda*V[t, 1:10], r.lambda*V[t, 11:20])
  V[t+1, 2:19] <- (1-2*d)*M[2:19] + d*M[1:18] + d*M[3:20]
  V[t+1, 1] <- (1-d)*M[1] + d*M[2]
  V[t+1, 20] <- (1-d)*M[20] + d*M[19]
  s[t+1] <- sum(V[t,])
}

plot(1:50, log(s), type='o', xlim=c(1, 50),
     ylim=c(4, 14), xlab='days',
     ylab='log(total population size)')
title('plot of log(total population) against time 1:50')

simulation.project <- function(L, d){
  # this function is as required in the second part of
  # the question
  v <- rep(5, L)
  l.lambda <- 0.9
  r.lambda <- 1.2
  V <- matrix(0, 50, L)
  V[1,] <- v
  s <- rep(0, 50)
  s[1] <- 5*L

  for (t in 1:49){
    # compute the geometric growth within patches
    M <- c(l.lambda*V[t, 1:as.integer(L/2)], r.lambda*V[t, (as.integer(L/2)+1):L])
    V[t+1, 2:L-1] <- (1-2*d)*M[2:(L-1)] + d*M[1:(L-1)] + d*M[3:L]
    V[t+1, 1] <- (1-d)*M[1] + d*M[2]
    V[t+1, 20] <- (1-d)*M[L] + d*M[L-1]
    s[t+1] <- sum(V[t+1,])
  }

  plot(1:50, log(s), type='o', xlab='days',
       ylab='log(total population size)')
  return(s)
}

reflecting <- function(M, d){
  m <- length(M)
  N.1 <- (1-d)*M[1]+d*M[2]
  N.L <- (1-d)*M[m]+d*M[m-1]
  N <- (1-2*d)*M[2:(m-1)]+d*M[1:(m-2)]+d*M[3:m]
  N <- c(N.1, N, N.L)
  return(N)
}

population.growth <- function(a){
  if (typeof(a)!='double'){
    return(FALSE)
  }
  else{
    n <- length(a)
    growth <- c()
    for (i in 1:(n-1)){
      growth[i] <- (a[i+1]-a[i])/a[i]
    }
    return(growth)
  }
}
# now we need a modified version of the simulation.project() function
simulation.project.2 <- function(d, L, spatial){
  v <- rep(5, L)
  l.lambda <- 0.9
  r.lambda <- 1.2
  V <- matrix(0, 50, L)
  V[1,] <- v
  s <- rep(0, 50)
  s[1] <- 5*L

  for (t in 1:49){
    # compute the geometric growth within patches
    M <- c(l.lambda*V[t, spatial], r.lambda*V[t, c(1:L)[-spatial]])
    V[t+1,] <- reflecting(M, d)
    s[t+1] <- sum(V[t+1,])
  }
  return(s)
}

ma <- matrix(0, 10, 4)
ma[,1] <- c(1:10)
ma[,2] <- c(5:14)
ma[,3] <- sample(c(1:20))[1:10]
ma[,4] <- sample(c(1:20))[1:10]
par(mfrow=c(2, 2))
par(mar=c(3, 3, 1.5, 0.5), mgp=c(2.5, 1.0))
for (i in 1:4){
  plot(c(1:49), population.growth(simulation.project.2(0.1, 20, ma[,i])),
       type='l', ylim=c(0.00, 1.00), xlab='t', ylab='growth rate',
       title='plot of growth rates against time')
}
```