R in ROGER

Myeong-Hun Jeong -- Postdoc
Johnathan Rush -- Education, Outreach, and Training Coordinator

CyberGIS Center for Advanced Digital and Spatial Studies
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign

July, 14, 2016

Contents

Unit 1: What is R?

- "R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS."
  - https://www.r-project.org
  - In a general programming, you want computers to do things for you
  - In as statistical programming, you want computers to learn things for you
- Available from The Comprehensive R Archive Network
  - http://cran.r-project.org/mirrors.html
- Origin and History
  - Initially written by Ross Ihaka and Robert Gentleman at Department of Statistics of University of Auckland, New Zealand during 1990s.
  - International project since 1997
- Open source with GPL license
  - Free to anyone
  - In actively development

How to work with R

- Downloading and installing R
  - Go to the R CRAN website (http://www.r-project.org/), and click on the download R link (http://cran.r-project.org/mirrors.html)
- Downloading and installing RStudio
  - To write an R script, one can use R Console, R commander, or any text editor (EMACS, VIM, or sublime)
  - However, the assistance of RStudio, an **integrated development environment** (**IDE**) for R, can make development a lot easier.
  - Access RStudio's official site by using the following URL: http://www.rstudio.com/products/RStudio/.
  - Rstudio IDE cheat sheet http://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf

How to work with R

- Materials
  - Go to the GitHub for CyberGIS R
  - https://github.com/mhjeong74/CyberGISR
  - If you have RStudio in your computer, you can download the materials via RStudio.
  - File> New Project > Version Control > Git > Repository URL (put the address above) > Create Project

How to work with R

- R as a calculator

```
> 4 + 1 - 2 #add and subtract
[1] 3
```

- Assignments

```
> x <- 8  #The object (variable) x holds the value 8
> x
[1] 8
```

- Value comparisons

```
> 2 == 2 #Equality
> 2 != 2 #inequality
```

- Executing functions

```
> print(x)  #print() is a function. It prints its argument, x.
[1] 8
>ls()        #lists the objects in memory
[1] “x”
>rm(x)      #remove x from memory
>ls()         #no objects in memory, therefore:
character(0)
```

How to work with R

- Basic data structures
  - Vector: An ordered collection of data of the same type

```
> x <- 0: 5
>x
[1] 0 1 2 3 4
```

  - Matrix: A rectangular table of data of the same type

```
> mdat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3)
```

  - List: An ordered collection of data of arbitrary types.

```
>person = list(name="Matt", age=30, married=F)
>person$name
[1] "Matt"
```

  - Data frames: A special kind of list used for storing dataset tables

```
>dfr1 <- data.frame( ID=1:4,
                     FirstName=c("John","Jim","Jane","Jill"),
                     Female=c(F,F,T,T),
                     Age=c> (22,33,44,55) )
>dfr1$FirstName
[1] John Jim  Jane Jill
```

How to work with R

- Flow control and loops

```
# if (condition) expr1 else expr2
>x <- 5; y <- 10
>if (x==0) y <- 0 else y <- y/x #
>Y
##[1] 2
# for (variable in sequence) expr
ASum <- 0
for (i in 1:x)
{
  ASum <- ASum + i
}
ASum  # equivalent to sum(1:x)
##[1] 15
```

- Plots
  - > plot(x <- sort(rnorm(47)), type = "l", main = "plot(x, type = \"s\")")
- Help
  - Help(plot)
- Installing packages
  - > install.packages(Stats) #This package contains functions for statistical calculations and random number generation.

Unit 2: Scaling up R computation

- R's limitation to large-scale data analysis[1]
  - It's single-threaded
    - The R language has on explicit constructs for parallelism, such as threads. An out-of-the-box R install cannot take advantage of multiple CPUs.
  - It's memory-bound
    - R requires that your entire dataset fit in memory (RAM). Four gigabytes of RAM will not hold eight gigabytes of data, no matter how much you smile when you ask.
- Solution:
  - Spreading work across multiple CPUs overcomes R's single-threaded nature.
  - Offloading work to multiple machines reaps the multi-process benefit and also addresses R's memory barrier.

1. McCallum, Q. E. and Weston S. (2012). Parallel R. O'Reilly Media, Inc. 2011

Scaling up R computation

- What to do if the computation is too big for a single desktop?
  - Using automatically offloading with multicore/GPU.
    - HiPLAR (High Performance Linear Algebra in R): use the latest multi-core and GPU libraries to give substantial speed-ups to existing linear algebra functions in R.
    - Advantage:
      - No code changes needed
      - User can run R solution as before without knowledge of the parallel execution.
    - Limitations:
      - Only support limited computational operations.
  - Break big computation with multiple job submission
    - Running R in non-interactive session
    - Rhadoop and SparkR packages
    - Advantage:
      - Utilize efficiency of other data intensive processing framework
      - Each job can use existing R code
    - Limitations:
      - A “data-parallel” solution that may not suitable for simulation based analysis

Scaling up R computation

- What to do if the computation is too big for a single desktop?
  - Implement code using parallel packages.
    - Parallel packages: snow, parallel, or foreach
    - Advantage:
      - Do whatever you want with them
      - Get the best performance
    - Limitations:
      - Need code development
      - In some case, the analysis workflow may need be changed.

+

=

Unit 2.1 Running R with parallel packages

- Objective:
  - Run R with a couple of parallel packages.
  - Investigate resource pressures between serial and parallel approaches.
  - Parallel spatial analysis -- Parallel spatial autocorrelation analysis (Moran's I).

Serial version of R code

- To run myProc() 10 times using apply

```
ptm <- proc.time()
#sapply converts results into a vector or array of appropriate size
result <- sapply(1:10, function(i) myProc())
proc.time() – ptm
=>  user  system elapsed
 14.929   0.232  35.179
```

- myProc function

```
#Define a simple R function
myProc <- function(size=10000000) {
#Load a large vector
vec <- rnorm(size)
#Now sleep on it
Sys.sleep(2)
#Now sum the vec values
return(sum(vec))
}
```

Running R with parallel packages

- Parallel: mclapply
  - The parallel package provides several methods for parallelizing your work.
  - 'mclapply' can only used for single-node parallelism.

```
require(parallel)
ptm <- proc.time()
result <- mclapply(1:10, function(i) myProc(), mc.cores=10)
proc.time() – ptm
=>  user  system elapsed
  0.012   0.025  4.616
```

Running R with parallel packages

- Snow
  - SNOW uses either MPI or socket based connections to achieve parallelism.
  - This means it can use cores on both the local and remote nodes.

```
require(snow)
#Create a cluster object
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#Assigns the values on the master of the variables named in list to variables of the same names in the global environments of each node.
clusterExport(cluster, list('myProc'))
ptm <- proc.time()
#Call function on the first cluster node with arguments seq[[1]] and ..., on the second node with seq[[2]] and ..., and so on. If the length of seq is greater than the number of nodes in the cluster then cluster nodes are recycled.
result <- clusterApply(cluster, 1:10, function(i) myProc())
proc.time() - ptm
=>  user  system elapsed
  0.006   0.001   4.053
#To shut down the cluster
stopCluster(cluster)
```

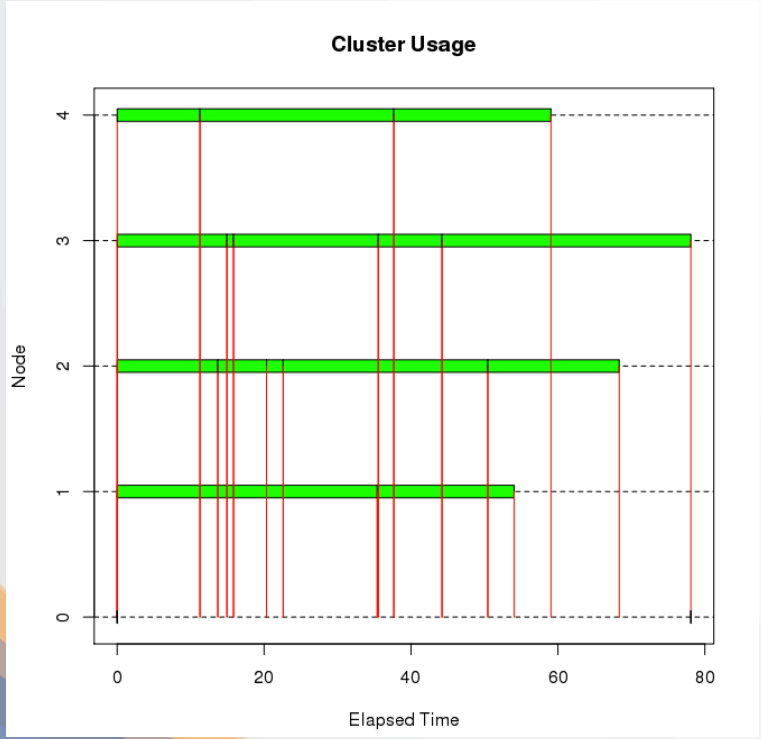
Running R with parallel packages

- Snow: Load balancing with clusterApplyLB
  - clusterApplyLB is a load balancing version of clusterApply.
  - Instead scheduling tasks in a round-robin fashion, it sends new tasks to the cluster worker as they complete their previous task.
  - clusterApply() pushes tasks to the workers, while clusterApplyLB() lets the workers pull tasks as needed.

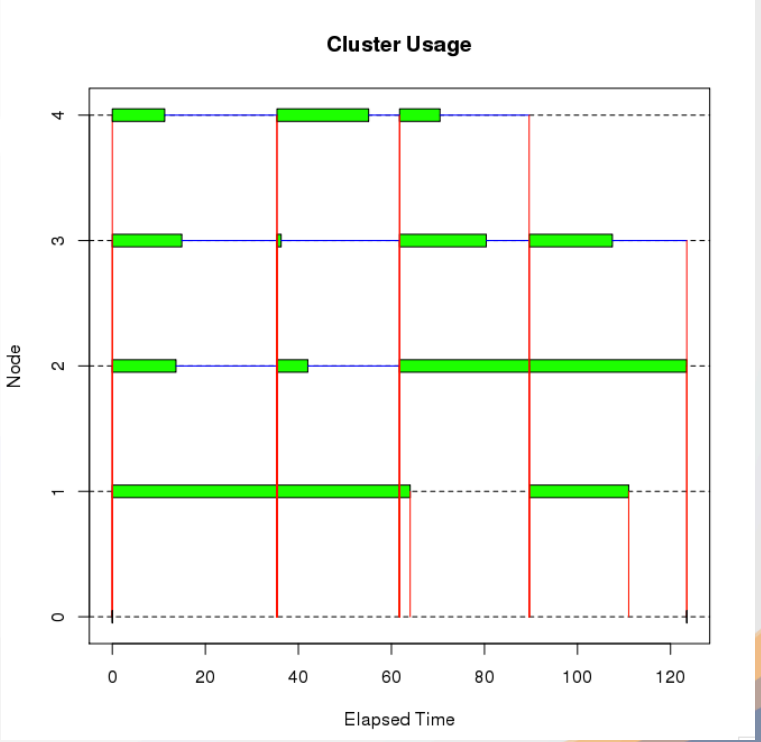
```
require(snow)
#Create a cluster object
hostnames <- rep('localhost', 4)
cluster <- makeSOCKcluster(hostnames)
#15 observations, mean = 15, sd = 15
sleeptime <- abs(rnorm(15,15,15))
#snow.time collects and returns timing information for cluster usage
tm <- snow.time(clusterApplyLB(cluster,sleeptime,Sys.sleep))
plot(tm)
stopCluster(cluster)
```

Running R with parallel packages

- Snow: Load balancing with clusterApplyLB
  - clusterApplyLB() is much efficient than clusterApply() in this example: 78 seconds for clusterApplyLB(), and 120 seconds for clusterApply()



clusterApplyLB()



clusterApply()

Running R with parallel packages

- foreach + snow
  - 'foreach' is a package that makes parallelization easier.
  - 'foreach' uses other parallel functions under the hood, but hides some of the complexity.

```
require(foreach)
require(doSNOW)
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#register the SNOW parallel backend with the foreach package.
registerDoSNOW(cluster)
ptm <- proc.time()
#'c' is useful for concatenating the results into a vector.
result <- foreach(i=1:10, .combine=c) %dopar% {
myProc()
}
proc.time() - ptm
⇒  user  system elapsed
0.023   0.010   4.983
stopCluster(cluster)
```

Running R with parallel packages

- Executing snow programs on a cluster with Rmpi

```
require(Rmpi)
require(snow)
# Initialize SNOW using MPI communication.
cluster <- makeCluster(8, type="MPI")

# Compute row sums in parallel using all processes, then a grand sum at the end on the master process
parallelSum <- function(m, n)
{
    A <- matrix(rnorm(m*n), nrow = m, ncol = n)
    row.sums <- parApply(cluster, A, 1, sum)
    print(sum(row.sums))
}

parallelSum(500, 500)

stopCluster(cluster)
mpi.exit()
```

Running R with parallel packages

- foreach + snow for multi-nodes
  - Set up environmental setting (Before setting the environmental variable, you must quit R)
    vim ~/.bashrc
    - Put the information to load module R and then save the file
      module load R
  - Assign the number of nodes.
    qsub -I -l nodes=2 #The -I option tells qsub you want to run an interactive job on the compute nodes.
  - Start R

```
require(foreach)
require(doSNOW)
#Get backend hostnames
nodelist <- Sys.getenv("PBS_NODEFILE")
hostnames <- scan(nodelist, what="", sep="\n")
#Set reps to match core count'
num.cores <- 10
hostnames <- rep(hostnames, each=num.cores)
hostnames
cluster <- makeSOCKcluster(hostnames)
registerDoSNOW(cluster)
ptm <- proc.time()
result <- foreach(i=1:100, .combine=c) %dopar% {
myProc()
}
proc.time() - ptm
stopCluster(cluster)
```

Running R with parallel packages

- Bootstrap calculations.
  - Serial implementation.

```
random.data <- matrix(rnorm(1000000), ncol = 1000)
bmed <- function(d, n) median(d[n])
library(boot)
sapply(1:100, function(n) {sd(boot(random.data[, n], bmed, R = 10000)$t)})
=> user  system elapsed
103.305   0.828 224.822
```

  - Parallel implementation

```
require(foreach)
require(doSNOW)
cluster = makeCluster(10, type = "SOCK")
clusterExport(cluster, c("random.data", "bmed"))
results = foreach(n = 1:100, .combine = c) %dopar% {
    library(boot); sd(boot(random.data[, n], bmed, R = 10000)$t)
}
stopCluster(cluster)
 ⇒ user  system elapsed
0.340   0.069  21.037
```

Lab 1: Running parallel R

- Objective:
  - Learn how to run parallel R
- Successful outcome:
  - Investigate resource pressures between serial and parallel approaches.
- Before you begin:
  - Environmental variable setting
    - vi ./bashrc -- open bashrc file
    - module load R -- put this line and exit the file after saving.
  - Module load
    - module load binutils/2.25 openblas/0.2.14 R/3.2.1-openmpi geos
  - Install parallel R packages

Running parallel R

- Perform the following steps:
- Step 1: Install packages for this course.
  - Start R
  - $ R
  - Install packages for this course.
    - > install.packages("snow")
    - > install.packages("Rmpi")
    - > install.packages("foreach")
    - > install.packages("doSNOW")
    - > install.packages("parallel")
    - > install.packages("boot")
    - > install.packages("maptools")
    - > install.packages("spdep")
    - You select a CRAN mirror for installing packages (e.g., USA(IN)).
    - Or > install.packages(c("snow", "Rmpi", "foreach", "doSNOW", "parallel", "boot", "maptools", "spdep"))

Running parallel R

- Step 2: Define a simple R function.

```
myProc <- function(size=10000000) {
  #Load a large vector
  vec <- rnorm(size)
   #Now sleep on it
  Sys.sleep(2)
  #Now sum the vec values
  total  <- 0
  for(v in vec) {
    total  <- total + v
  }
}
```

Running parallel R

- Step 3: To run myProc() 10 times using apply.

```
ptm <- proc.time()
#sapply converts results into a vector or array of appropriate size
result <- sapply(1:10, function(i) myProc())
proc.time() – ptm
=>  user  system elapsed
 14.929   0.232  35.179
```

Running parallel R

- Step 3: To run myProc() 10 times using a parallel package.

```
require(parallel)
ptm <- proc.time()
result <- mclapply(1:10, function(i) myProc(), mc.cores=10)
proc.time() – ptm
=>  user  system elapsed
 0.012   0.025  4.616
```

Running parallel R

- Step 4: To run myProc() 10 times using a SNOW package.

```
require(snow)
#Create a cluster object
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#Assigns the values on the master of the variables named in list to variables of the same names in the global environments of each node.
clusterExport(cluster, list('myProc'))
ptm <- proc.time()
#Call function on the first cluster node with arguments seq[[1]] and ..., on the second node with seq[[2]] and ..., and so on. If the length of seq is greater than the number of nodes in the cluster then cluster nodes are recycled.
result <- clusterApply(cluster, 1:10, function(i) myProc())
proc.time() - ptm
=>  user  system elapsed
  0.006   0.001   4.053
#To shut down the cluster
stopCluster(cluster)
```

Running parallel R

- Step 5: To run myProc() 10 times using foreach + SNOW packages.

```
require(foreach)
require(doSNOW)
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#register the SNOW parallel backend with the foreach package.
registerDoSNOW(cluster)
ptm <- proc.time()
#'c' is useful for concatenating the results into a vector.
result <- foreach(i=1:10, .combine=c) %dopar% {
myProc()
}
proc.time() - ptm
⇒ user  system elapsed
0.023   0.010   4.983
stopCluster(cluster)
```

Running parallel R

- Step 6: Bootstrap calculations.
  - Serial implementation.

```
random.data <- matrix(rnorm(1000000), ncol = 1000)
bmed <- function(d, n) median(d[n])
library(boot)
sapply(1:100, function(n) {sd(boot(random.data[, n], bmed, R = 10000)$t)})
=> user  system elapsed
103.305   0.828 224.822
```

  - Parallel implementation

```
library(doSNOW)
cluster = makeCluster(10, type = "SOCK")
clusterExport(cluster, c("random.data", "bmed"))
results = foreach(n = 1:100, .combine = c) %dopar% {
    library(boot); sd(boot(random.data[, n], bmed, R = 10000)$t)
}
stopCluster(cluster)
 ⇒ user  system elapsed
0.340   0.069  21.037
```

Lab 2: Parallel spatial analysis

- Objective:
  - Learn how to calculate parallel spatial autocorrelation (Moran's I)
- Location of files:
  - /gpfs_scratch/geog479/lab9
- Successful outcome:
  - Your will calculate a spatial autocorrelation index, based on serial and parallel approaches.
- Before you begin:
  - SSH into the ROGER system.
  - Make a data directory and copy shape files into the data directory
    - cp /gpfs_scratch/geog479/lab9/*.* dataset

Lab: Parallel spatial analysis

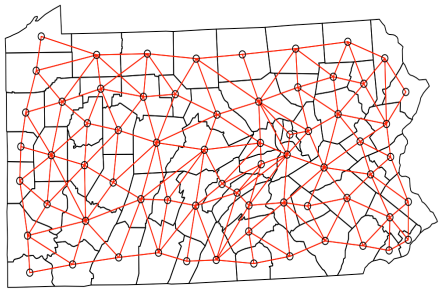
- A large and growing number of libraries for handling spatial data in R have been developed.
- spdep package
  - A collection of functions to create spatial weights matrix objects from polygon contiguities, from point patterns by distance and tessellations, for summarizing these objects, and for permitting their use in spatial data analysis.
  - https://cran.r-project.org/web/packages/spdep/index.html
- Spatial package
  - Functions for kriging and point pattern analysis.
  - https://cran.r-project.org/web/packages/spatial/index.html
- GISTools package
  - Mapping and spatial data manipulation tools.
  - https://cran.r-project.org/web/packages/GISTools/index.html

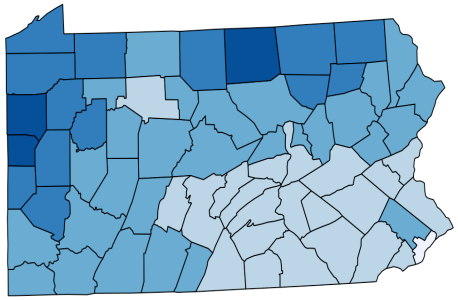
Parallel spatial analysis

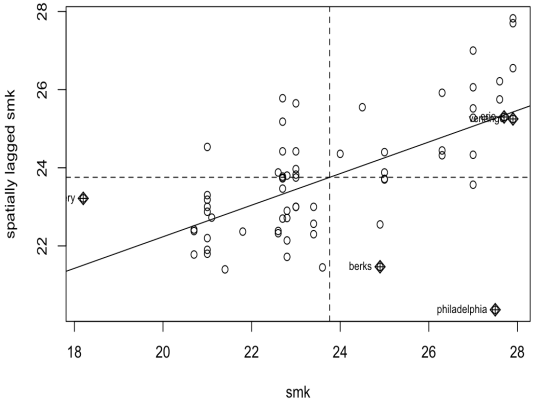
- maptools package
  - Set of tools for manipulating and reading geographic data.
  - https://cran.r-project.org/web/packages/maptools/index.html
- sp package
  - A package that provide classes and methods for spatial data;
  - https://cran.r-project.org/web/packages/sp/index.html
- There are more available spatial packages. However, these packages are not available for parallel computations.

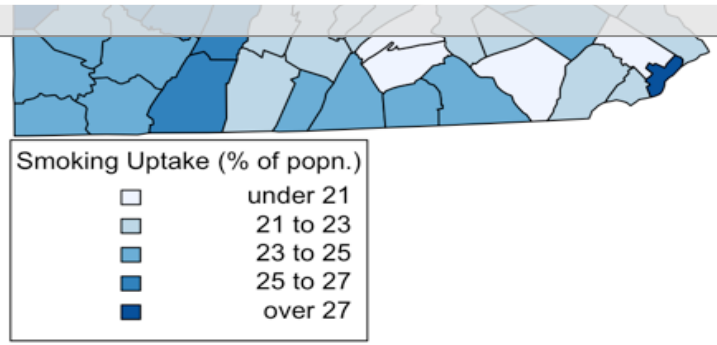
Example : Parallel spatial autocorrelation

- Spatial autocorrelation
  - The extent to which points that are "close together" in space have similar values, on



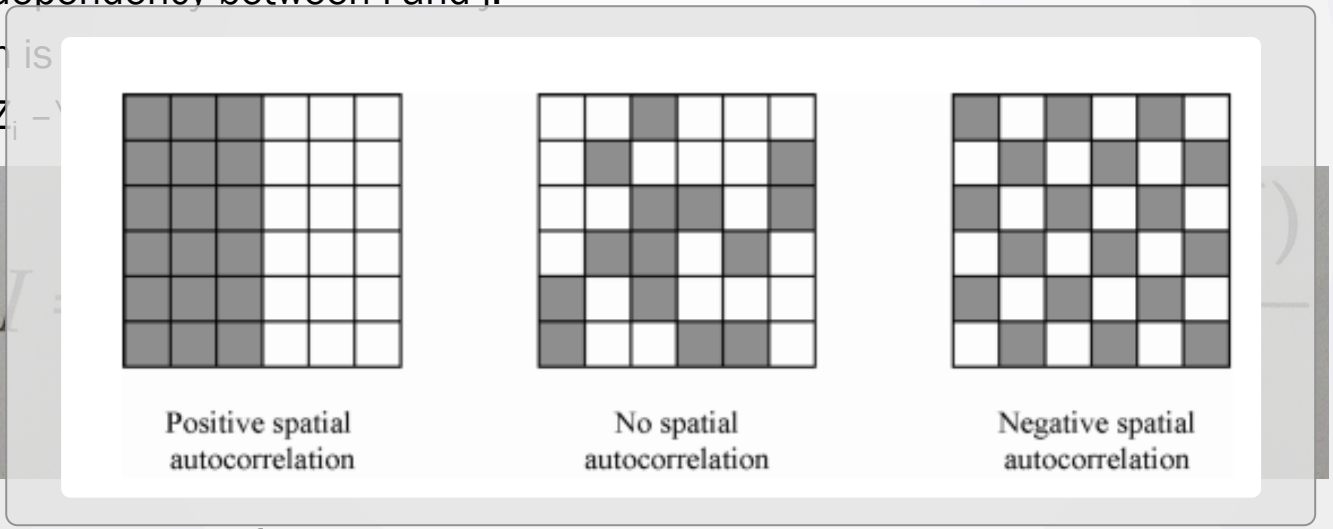






Example : Parallel spatial autocorrelation

- Moran's I is an index of autocorrelation[1].
  - Where w_ij is the (i,j)th element of a weights matrix W, specifying the degree of dependency between i and j.
  - n is
  - z_i –

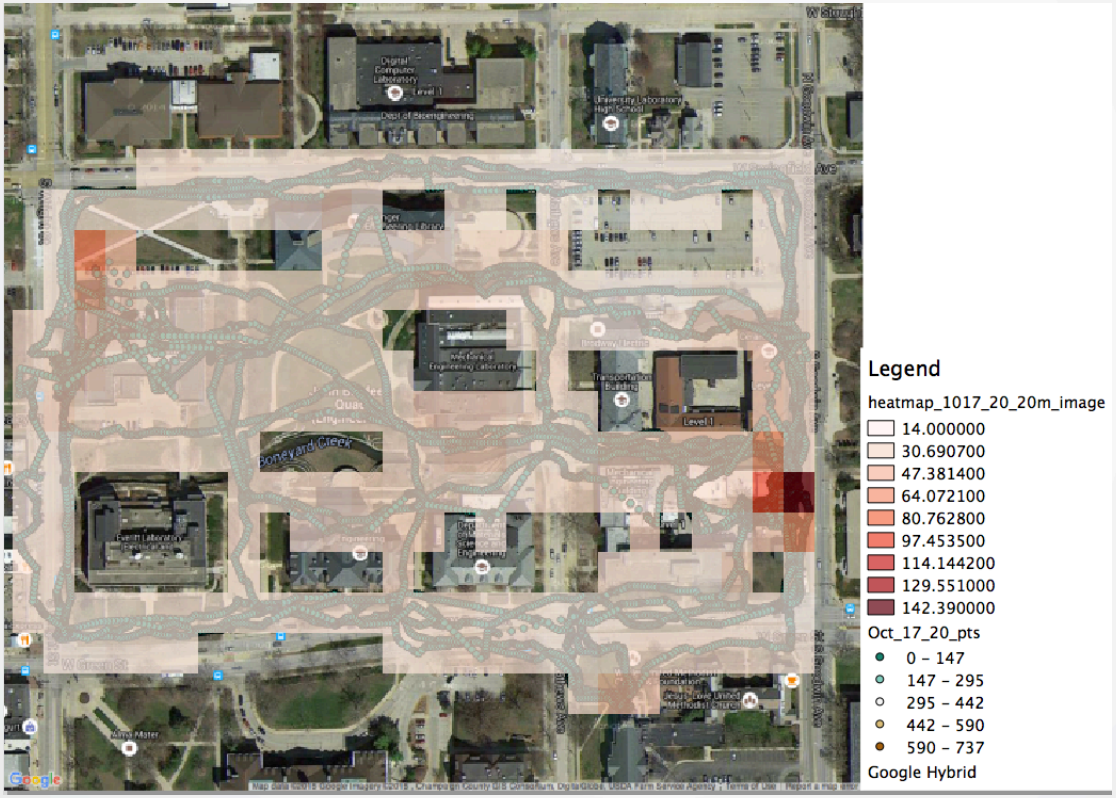


  - Large values of I suggest that there is a stronger relationship between nearby z_i values. Furthermore, I may be negative in some circumstances – suggesting that there can be a degree of inverse correlation between nearby z_i values.
  - The range of Moran's I is [-1, 1] – this interval can shrink or expands.

1. Anselin, L. (1995). Local indicators of spatial association-LISA. Geographical analysis, 27(2), 93-115.

Example : Radiation spatial autocorrelation

- Radiation measurements on UIUC campus.



- Acknowledgement of data use
  - Prof. Clair, the Department of NPRE shares her group data for the CyberGIS workshop.

Radiation spatial autocorrelation

- Step 1: Moran's I serial implementation

```
require(maptools)
require(spdep)
#an absolute filepath representing the current working directory of the R proces
getwd()
#set the working directory
setwd("~/")

#Get shapefile header information in the radiation data
getinfo.shape("dataset/Oct_17_20_proj.shp")
#Reads data from a points shapefile
radiation<-readShapePoints ("dataset/Oct_17_20_proj.shp")

#Retrieve spatial coordinates from a Spatial object
coords<-coordinates(radiation)
IDs<-row.names(as(radiation, "data.frame"))

#Neighbourhood contiguity by distance
radiation_nei<-dnearneigh(coords, d1=0, d2=20, row.names=IDs)

#Spatial weights for neighbours lists
radiation_nbq_wb<-nb2listw(radiation_nei, style="W")

#Moran's I test for spatial autocorrelation
moran.test(radiation$field_5, listw=radiation_nbq_wb)
```

Radiation spatial autocorrelation

- Step 1: Moran's I serial implementation

```
#Moran's I test for spatial autocorrelation
moran.test(radiation$field_5, listw=radiation_nbq_wb)
```

```
Moran I statistic standard deviate = 335.63, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     4.310875e-01     -1.076079e-04      1.650584e-06
```

Radiation spatial autocorrelation

- Step 2: Moran's I parallel implementation

```
gamma <- radiation$field_5
listw <- radiation_nbq_wb
nsim <- 999
require(foreach)
require(doSNOW)
n <- length(listw$neighbours)
S0 <- Szero(listw)
cluster = makeCluster(10, type = "SOCK")
registerDoSNOW(cluster)
clusterExport(cluster, c("gamma", "listw","n","S0"))
results = foreach(n = 1:nsim, .combine = c) %dopar% {
 library(spdep); moran(sample(gamma), listw, n, S0,zero.policy=NULL)$I
}
```

Radiation spatial autocorrelation

- Step 2: Parallel Moran's I calculation

```
paMoran <- function(res, x, listw, nsim,zero.policy=NULL,alternative="greater") {
  n <- length(listw$neighbours)
  S0 <- Szero(listw)

  res[nsim+1] <- moran(x, listw, n, S0, zero.policy)$I
  rankres <- rank(res)
  xrank <- rankres[length(res)]
        diff <- nsim - xrank
        diff <- ifelse(diff > 0, diff, 0)

  if (alternative == "less")
        pval <- punif((diff + 1)/(nsim + 1), lower.tail=FALSE)
        else if (alternative == "greater")
        pval <- punif((diff + 1)/(nsim + 1))

        statistic <- res[nsim+1]
        names(statistic) <- "statistic"
        parameter <- xrank
        names(parameter) <- "observed rank"
        method <- "Parallel Monte-Carlo simulation of Moran's I"
        lres <- list(statistic=statistic, parameter=parameter,
           p.value=pval, alternative=alternative, method=method,res=res)
        lres
}
```

Radiation spatial autocorrelation

- Step 2: Parallel Moran's I calculation

```
mtest <- paMoran(results,gamma,listw,nsim)
mtest$method      => Parallel Monte-Carlo simulation of Moran's I
mtest$statistic   => 0.43109
mtest$parameter   => observed rank 1000
mtest$p.value     =>  0.001
```

Unit 2.2 R and Hadoop

- Objective:
  - Preparing the RHadoop environment
  - Installing rmr2
  - Installing rhdfs
  - Operating HDFS with rhdfs

Preparing the RHadoop environment

- RHadoop is a collection of R packages that enables users to process and analyze big data with Hadoop.
  - https://github.com/RevolutionAnalytics/RHadoop/wiki
  - In RHadoop, there are five main packages, which are:
    - rmr: This is an interface between R and Hadoop MapReduce, which calls the Hadoop streaming MapReduce API to perform MapReduce jobs across Hadoop clusters.
    - Rhdfs: This is an interface between R and HDFS, which calls the HDFS API to access the data stored in HDFS
    - Rhbase: You can use rhbase to read/write data and manipulate tables stored within HBase.
    - Plyrmr: This is a higher-level abstraction of MapReduce, which allows users to perform common data manipulation in a plyr-like syntax.
    - Ravro: This allows users to read avro files in R, or write avro files.

Installing rmr2

- Install dependent packages before installing rmr2
  - Start R

  > install.packages(c("codetools", "Rcpp", "RJSONIO", "bitops", "digest", "functional", "stringr", "plyr", "reshape2", "rJava", "caTools"))

  >q()
- Download rmr-3.3.1
  - $ wget –no-check-certificate https://github.com/RevolutionAnalytics/rmr2/releases/download/3.3.1/rmr2_3.3.1.tar.gz
- Install rmr-3.3.1
  - $R CMD INSTALL rmr2_3.3.1.tar.gz
  - $R
  - >library(rmr2)
- Set up bashrc
  - vi .bashrc
  - export R_LIBS=/home/user_name/R
  - Exit after saving the change

Installing rhdfs

- Download rhdfs 1.0.8 from GitHub.Start
  - $ wget –no-check-certificate https://github.com/RevolutionAnalytics/rhdfs/blob/master/build/rhdfs_1.0.8.tar.gz?raw=true
- Install rhdfs under the command-line mode:
  - $ HADOOP_CMD=/usr/bin/hadoop R CMD INSTALL rhdfs_1.0.8.tar.gz?raw=true
  - $R
  - > Sys.setenv(HADOOP_CMD="/usr/bin/hadoop")
  - > Sys.setenv(HADOOP_STREAMING="/usr/hdp/2.3.2.0-2602/hadoop-mapreduce/hadoop-streaming-2.7.1.2.3.2.0-2602.jar")
  - > library(rhdfs)
  - > hdfs.init()

Operating HDFS with rhdfs

- Initialize the rhdfs package:
  - $R
  - > Sys.setenv(HADOOP_CMD="/usr/bin/hadoop")
  - > Sys.setenv(HADOOP_STREAMING="/usr/hdp/2.3.2.0-2602/hadoop-mapreduce/hadoop-streaming-2.7.1.2.3.2.0-2602.jar")
  - > library(rhdfs)
  - > hdfs.init()
- Manipulating files stored on HDFS, as follows:
  - dfs.put: Copy a file from the local filesystem to HDFS:
    - >hdfs.put('word.txt', './')
  - hdfs.ls: Read the list of directory from HDFS:
    - > hdfs.ls('./')
  - hdfs.copy: Copy a file from one HDFS directory to another:
    - > hdfs.copy('word.txt', 'wordcnt.txt')
  - hdfs.move : Move a file from one HDFS directory to another:
    - > hdfs.move('wordcnt.txt', './data/wordcnt.txt')
  - hdfs.delete: Delete an HDFS directory from R:
    - > hdfs.delete('./data/')
  - hdfs.get: Download a file from HDFS to a local filesystem:
    - > hdfs.get(word.txt', '/home/user_name/word.txt')