**Lab 1: Introduction to R**

**Objectives**

* Get familiar with R
* Conduct exploratory data analysis in R
* Explore mapping functions in R
* Conduct non-spatial regression analysis in R

**Part I: Introduction to R**

R is a command-line language, which means you type in what you want it to do at a prompt and R immediately processes your request and returns the requested value.

* Reference books:

**Applied Spatial Data Analysis with R**

http://www.springerlink.com/content/978-0-387-78171-6

**Data Manipulation with R**

http://www.springerlink.com/content/978-0-387-74731-6

* An introduction to R by R development team is provided through:

http://cran.r-project.org/doc/manuals/R-intro.pdf

1. **Access R**

* You can download R from the site 'http://www.r-project.org'. Select a CRAN mirror close to you (e.g., Ohio) to download and install R. Refer to FAQ if necessary. Windows version of R can be downloaded from

http://cran.r-project.org/bin/windows/base/

Once you open your R software, you will see a prompt sign ">" in the command line. You can input or paste your command just after this sign. When you paste command, you can use your mouse to right click the command line area, and select "Paste commands only", then it will not paste the ">".

1. **Add-on packages**

All R functions and datasets are stored in packages. There are some standard packages installed with R software. To see packages installed at your site, type command

> library()

You need to load the libraries before you can use them. In Lab 1 we will need two libraries: rgdal, maps, and mapproj Before you can load them, you need to install it.

Type the following code to install three libraries necessary for Lab 1.

> install.packages("rgdal", repos = "http://cran.r-project.org")

> install.packages("maps", repos = "http://cran.r-project.org")

> install.packages("mapproj", repos = "http://cran.r-project.org")

Type the following code to load three libraries necessary for Lab 1.

> library(rgdal) # shapefile operation

> library(maps) # map operation

> library(mapproj) # map projection

Note that “#” is used to indicate comments after. Comments will not be executed by R.

1. **Set working directory**

Suppose the lab materials have been downloaded and unzipped to C:\WorkSpace\Lab1. In the folder C:\WorkSpace\Lab1, there is a folder named “data”. That is where the working data are stored. You can set your working directory to the data folder, i.e., C:\WorkSpace\Lab1\data\.

> setwd("c:/WorkSpace/Lab1/data") # set working directory to C:\Workspace\Lab1\data

> getwd() # get the current work directory.

Note:

(1) If your data have been put in other places other than "C:\WorkSpace\Lab1\data\", please change the code accordingly;

(2) The symbol used in setwd() is "/", NOT "\".

1. **Objects in R**

In R, there are different types of objects with specific data structures. To store a result to an object, use '=' or '<-' symbols. Note that object name is CASE SENSITIVE. You can use class() to find out the object type.

**(1) Numeric vector**

The simplest data structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers. You can use an assignment statement using the function c() to concatenate numeric data.

> x = c(1.1,3.2,2.1,4.9,5.5,7.0,7.8,9.0,2.3,6.9)

> y = c(3,3.5,6,1.5,5.5,3.2,1,4.5,1,7)

> class(x)

You have now defined three vector variables, 'x', 'y', each with the same number of elements. To find out how many elements they have, simply type the following:

> length(x)

You can also define a new variable to store the size of an existing variable, e.g.,

> n=length(x)

> n # To see the value of 'n'.

> z=10+3\*x+rnorm(n,0,5) # create variable z based on x

Notice that the variable name comes first, followed by an equal sign and the name of the function with a set of arguments in parentheses. So, when we defined 'x' and 'y', the function we used was the 'c()' function, which is used for turning its arguments into a list. The arguments were a series of numbers, in this case, pertaining to elements of the variables 'x' and 'y'.

The function 'rnorm()' takes a random sample of size 'n' from a Normal Probability Distribution with mean 0 and standard deviation 5. This is akin to adding error to a known linear function of the covariate 'x'.

**(2) Data frames**

Data frames are like tables, which are composed of vector variables with headings. To make a data frame from some of the variables we have already created, type:

> data.df = data.frame(cbind(x,y,z))

> names(data.df) = c("xcoord","ycoord","response") # add headings

> class(data.df) # returns the class type

> data.df # returns the data frame

Here ‘xcoord', ‘ycoord', and ‘response' correspond to ‘x', ‘y', and ‘z' above. "data.df" is the name of your data frame just created. The reason we use ".df" as the end of the name is just a custom, which makes the variable more identifiable to users.

Individual entries of data frame can be accessed with $, [ ] symbols. The following three command lines will get the same results.

> data.df$ycoord[1:3]

> data.df[1:3,2]

> data.df[1:3,"ycoord"]

**(3) Array and matrix**

An array or matrix can be considered as a multiply subscripted collection of data entries, for example

numeric. A vector can be used by R as an array or matrix only if it has a dimension vector as its dim attribute.

> dim(x) = c(1,n) # assign dim attribute to convert x to array

> class(x)

> x

> data.m = as.matrix(data.df) # use function convert data frame to matrix

> data.m2 = t(data.m) # transpose

> data.m2

We will meet other types of objects in later parts of the lab.

1. **Write and read objects**

**(1) Write and read vector**

A vector can be written to a file and read from a file. To save a vector in a file, type

> write(x, file = "myx.txt", ncolumns=1)

To read a vector from a file, scan() command can be used.

> x2 = scan("myx.txt")

**(2) Write and read data frame**

To write a data frame to a file, use write.table() function.

> write.table(data.df, file="mydf.csv", row.name=F, sep=",") # write to file

To read a table from a text file, use read.table() command.

> data2.df = read.table("mydf.csv",header=T,sep=",")

**(3) Read and write a shapefile**

Shapefile is a commonly used GIS file format that has been specified by ESRI, the publisher of ArcView and ArcGIS, which introduced it initially to support desktop mapping using ArcView. This format uses at least three files to represent the data, a file of geometries with an \*.shp extension, an index file to the geometries \*.shx, and a legacy \*.dbf DBF III file for storing attribute data. Shapefile can be read using readOGR() function in the library rgdal.

> sids.poly = readOGR(".", "sids") # read shapefile “sids.shp” from the work directory

> class(sids.poly)

> sids.df = as.data.frame(sids.poly) # define data frame from sids.poly

Shapefile "sids.shp" is the Sudden Infant Death Syndrome sample data for North Carolina counties in two time periods (1974-78 and 1979-84). Detailed information can be found in 'sids.html'. We will explore this data in later parts of the lab. The following command writes variable “sids.poly” to shapefile named as “mysids.shp”.

> writeOGR(sids.poly, ".", "mysids", driver="ESRI Shapefile")

**(4) List variables**

In order to see what variables you have now stored in your current work directory, use the function:

> ls()

**(5) Remove variables**

You may remove unwanted variables from your workspace, e.g.,

> rm(x2, data2.df,data.m2) # remove variable x2, data2.df, and data.m2.

1. **Create a plot**

In order to visualize your newly created data, display it as:

> plot(x,y,col="blue",pch="\*") # ‘col' is color, and ‘pch' specifies the plotting symbol

> text(x,y,labels=1:n,pos=1) # add labels

You can save plot as image which will be put in your workspace folder, e.g., C:\WorkSpace\ Lab1\data\.

> dev.copy(png,'myplot.png') # png file

> dev.off() # close dev function

The function ‘text()' adds a label to the plot; in this case, the label is just the number of the data entry in the data frame. To display the response data as well, consider a plot where the size of the plotted symbol corresponds to the magnitude of the data.

> plot(x,y,type="n", main="spatial process")

> symbols(x,y,circles=(z-min(z))/50,inches=FALSE,add=TRUE)

> plot(sids.poly) # plot polygon of sids

> help("plot") # help for function “plot”

**Part II: Exploratory Data Analysis**

1. **Summary statistics**

Getting summary statistics is a good start to explore our data.

> summary(sids.df$SID74)

1. **Histogram**

R provides functionalities to plot data such as histogram and scatterplot.

*>* names(sids.df)# see what variables are in sids.df

> hist(sids.df$BIR74, nclass = 10, main = "BIR74")# histogram of BIR74

> ?hist # check help for hist

As we can see from the histogram, the data are skewed (here positive skewed). We can transform the data to make it look normal.

> bir74.log = log(sids.df$BIR74) # log transformation

> bir74.sqrt = sqrt(sids.df$BIR74) # square root transformation

> par(mfrow=c(1,2),pty = "s") # 1 x 2 pictures on one plot with square plotting region

> hist(bir74.log, col = "light blue")

> hist(bir74.sqrt, col = "light blue")

> par(mfrow=c(1,1)) # reset plot area

You can use your mouse to drag the plot area and change the display size to get a better view.

1. **Boxplot and qq-plot**

We can explore the data further by boxplot and qq-plot.

> par(mfrow=c(2,2), pty="s") # set plot area to plot four figures together

> boxplot(sids.df$BIR74, main="BIR74")

> boxplot(bir74.log, main="BIR74.LOG")

> qqnorm(sids.df$BIR74)

> qqline(sids.df$BIR74)

> qqnorm(bir74.log)

> qqline(bir74.log)

> par(mfrow=c(1,1)) # reset plot area

Notice that the transformed data look normal. Again, you can use your mouse to drag the plot area and change the display size to get a better view of your figure.

1. **Bivariate/multivariate relationships**

We can check the relationship between two or more variables. First, we can plot a scatter plot of our variables.

> plot(sids.df$BIR74, sids.df$SID74, xlab="BIR74", ylab="SID74")

> plot(sids.df[,c(9,10,12,13)], pch = 20) # scatter plot matrix

Note that there is a strong linear relationship between BIR74 and BIR79 as their scatter plot follows a good linear shape. There are some positive correlation between SID74 and BIR74, as well as SID79 and BIR79.

**Part III: Mapping Functions**

1. **Mapping in R**

Producing maps is essential to spatial statistics. Geographic Information System (GIS) software packages are very adept at manipulating and displaying geographic data (i.e., maps), however, at the present time they are lacking in statistical functionality (though they are constantly improving). For the sake of convenience, it is useful to be able to employ basic mapping techniques within statistical software; so that time spent transitioning between two software packages is minimal.

Fortunately, there are several add-on packages available for R that is capable of producing high-quality maps. The map function in R is an interface to geographical databases. The mapproject function provides numerous map projections to be used when displaying maps. You can find out more about these packages from their user manuals:

http://cran.r-project.org/web/packages/maps/index.html

http://cran.r-project.org/web/packages/mapproj/index.html

Simple maps can be made, of varying political resolutions, by using the map function. Consider the following examples for plotting maps of the USA.

> graphics.off() # close all previous plots

> windows() # start a new plot

> map('usa')

> map('state')

> map('county')

The map function has a very large number of optional parameters that can be specified. See

> help(map)

For example, we can use an alternative map projection while plotting the states:

> map('state',proj='albers',param=c(30,40))

The command above uses an Albers equal-area projection. The param=c(30,40) option specifies the two latitudes at which the projection is “true”. You can see other projection options by checking the help: help(mapproject). Try other projections to see how they differ.

1. **Focusing on specific regions**

It is possible to map certain regions:

> map('state','ohio')

> map('state',c('ohio','indiana','kentucky'))

> map('county','ohio')

#Now, try using different projections:

> map('county',proj='albers',param=c(37,41),'ohio')

> map('county',proj='cylequalarea',param=c(39),'ohio')

> map('county',proj='mercator','ohio')

We can, of course, plot other parts of the world with the maps database:

> map('world')

> map('world',proj='albers',param=c(37,41))

Overlay the locations of cities (with a minimum population of say, 200,000) onto a map of China:

> map('world','china',proj='azequalarea')

> map.cities(country='China',projection = TRUE, minpop=200000)

It is also easy to add text to various maps of interest:

> map('state',proj='albers',param=c(37,41))

> text(mapproject(state.center), state.abb, cex=.5)

Note that state.center contains the longitude and latitude of the central points of each state (which must be projected onto the current mapprojection); state.abb contains the state abbreviations, which is the text to be plotted; cex=.5 gives the size of the text.

1. **Create thematic/choropleth maps**

Using the North Carolina Sudden Infant Deaths datasets, we can construct a map illustrating counties that differ by SIDS cases.

Recall that we have imported the data from the SIDS shapefile into R, and have created a data frame named as "sids.df". This data frame contains several columns. We are interested in {*Si*}, the number of SIDS deaths, e.g.,'SID79' field. In addition, we will use {*ni*}, the total number of live births in that same year, which is in 'BIR79'. Note that it would be misleading to plot {*S*i}, the number of SIDS deaths by county, since the overall population and birth rate also varies significantly by county. One could simply plot the rates; say {*Si*/*ni*}. However, Cressie and Chan (1989, JASA) argue that it is better to transform these rates with a Freeman-Tukey square-root transformation: . This is easily done in R:

> sids.y=sqrt(1000)\*( sqrt(sids.df$SID79/sids.df$BIR79) +

sqrt((sids.df$SID79+1)/sids.df$BIR79) )

Now, we are interested in plotting according to ranges of this variable. Specifically, [ < 2.2 ] [ 2.2-3.0 ] [ 3.0-3.5 ] [ > 3.5 ] . One way to do this is by making a new variable containing the categories:

> sids.ygrp=rep(1,length(sids.y))

> sids.ygrp[(sids.y >= 2.2 & sids.y < 3.0)]=2

> sids.ygrp[(sids.y >= 3.0 & sids.y < 3.5)]=3

> sids.ygrp[sids.y >= 3.5]=4

Now let us plot these:

> rainbow4=rainbow(4,start=0,end=.7)

> clr=rev(rainbow4)[sids.ygrp]

> plot(sids.poly,fill=T,col=clr)

> legend("topleft",

legend=c("< 2.2","2.2-3.0","3.0-3.5", "> 3.5"),

fill=rev(rainbow4) )

In general, a color scale with so few categories and so many different colors is not desirable. In light of that fact, consider the map above just as a quick introduction to the rainbow color scale. To create a more effective map, note that you can also plot with gray scale colors:

> clrg=rev(gray.colors(4))[sids.ygrp]

> plot(sids.poly,fill=T,col=clrg)

> legend(("topleft",legend=c("< 2.2","2.2-3.0","3.0-3.5", "> 3.5"),

fill=rev(gray.colors(4)))

**Part IV: Non-Spatial Regression Analysis**

1. **Bivariate regression**

Basic statistical analyses are easy to perform using R, for example, simple linear regression. Fit a linear model to the data (i.e., estimate the regression coefficients 'a' and 'b', for the linear relationship: *y*=*a*+*bx*) and store the output in a new variable.

> bs79.lm=lm(sids.df$SID79~sids.df$BIR79)

> summary(bs79.lm)

The function 'summary()'displays the model output (e.g., parameter values and levels of significance). As we can see, it is significant at 0.01 level. The adjusted-square is 0.7995. According to the result, the regression model is



To investigate the results visually, get the residuals and fitted values. Then generate plots with the values.

> resids = resid(bs79.lm)

> fits = fitted(bs79.lm)

> graphics.off() # close previous plot if it is still open

> windows()

> par(mfrow=c(2,2), pty = 's')

# SIDS79 vs. BIR79 with fitted line

> plot(sids.df$BIR79, sids.df$SID79, xlab="BIR79", ylab="SID79")

> lines(sids.df$BIR79,fits)

# BIR79 vs. residuals

> plot(sids.df$BIR79, resids,ylab="Residuals")

> abline(h=0)

# Histogram and Q-Q plot

> hist(resids, xlab="Residuals")

> qqnorm(resids, ylab="Residuals")

> qqline(resids)

> par(mfrow=c(1,1))

1. **Multiple regression**

Recall that we have created a data frame (named "data.df") at the beginning of the lab. To fit another regression model, you can use the data frame and both x and y coordinates as covariates:

> zxy.lm=lm(response ~ xcoord + ycoord , data=data.df)

> summary(zxy.lm)

Ask yourself what the results of the linear-model fit tell you about the relationship between the response and the coordinates? Note, you can look at the estimated mean response (zxy.lm$fit) and residuals (zxy.lm$residuals).

1. **Prediction**

Now, suppose you want to predict the process at unobserved locations, given the estimated model parameters. You can predict the process at the following two locations: (5, 5) and (7, 7).

> pred.df=data.frame(xcoord=c(5,7), ycoord=c(5,7))

> pred.df$response=predict(zxy.lm,pred.df)

> pred.df

1. **Plot predictions**

To graphically view the predictions, you can plot them together with the original data:

> plot(data.df$xcoord,data.df$ycoord,type="n", main="spatial process")

> symbols(data.df$xcoord,data.df$ycoord,

circles=(data.df$response-min(data.df$response))/50,

inches=FALSE,add=TRUE)

> symbols(pred.df$xcoord,pred.df$ycoord,

circles=(pred.df$response-min(data.df$response))/50,

inches=FALSE,bg=1,add=TRUE)

Ask yourself how the predicted values (now shown as filled in circles) appear to fit in with the observed data.

**Assignment I**

* Read the Columbus crime dataset named "columbus.shp". The data description is available in file "columbus.html".
* Create a histogram, boxplot, and QQ plot for each of the following variables: CRIME, HOVAL, and INC. Discuss the normality of the three variables.
* Create a scatterplot matrix for CRIME, HOVAL, and INC variables. Discuss the relationship of the variables.
* Fit a regression model with CRIME as dependent variable, and HOVAL and INC variables as independent variables, i.e., one model like CRIME = *a*\*HOVAL+ *b*\*INC, where *a* and *b* are the coefficient.
* From the result of the regression model, generate two residual plots (i.e., residuals vs. each independent variable), histogram of the residuals, and qqplot of the residuals. Discuss if the assumptions of linear regression are fitted well or not.