Intro to R

Jonathan M. Lees
University of North Carolina, Chapel Hill
Department of Geological Sciences
CB #3315, Mitchell Hall
Chapel Hill, NC 27599-3315
email: jonathan.lees@unc.edu
ph: (919) 962-1562

July 23, 2012

1 Getting Started

This set of routines is intended to show show how to make something **R** .

R can be downloaded and installed on linux, windows or macOS machines by extracting files from the website and following the instructions.

Once the code is installed you may add on a variety of packages that enhance the base installation. Which packages you install among the thousands that are available depends on your needs and what you are doing. I do not recommend downloading all the packages. In some cases a function on one package may interfere with the same named function in another, unrelated package.

Install **R** from: `http://www.r-project.org/`

The **R** foundation updates the base **R** software about every 6 months. I recommend keeping the most current version available on your system. In some cases if you upgrade to a more recent version you may have to also re-install packages.

2 Installing Packages

If you have administrator access on your computer you may install packages and update them. This is easily done by starting **R** as administrator and telling **R** to exxtract a package and install it. In many cases one package may depend on others so the install process will also extract those packages that the original package depends on. You may extract several packages at once.

Once the packages are installed on a computer they do not need to be installed again. However, many packages are constantly being upgraded and I recommend also installing the updates.

To find out which version of **R** you are using you may query by,

```
vers = R.Version()
print(paste("I am using", vers$version.string))
```

```
[1] "I am using R version 2.15.1 (2012-06-22)"
```

```
Repository="http://lib.stat.cmu.edu/R/CRAN"
```

```
install.packages("akima")
packageStatus()
update.packages()
```

To see all the packages available go to the CRAN web site and see the link the PACAKGES.

In **R** you can download the list of packages by executing:

```
tennREPOS = "http://mirrors.nics.utk.edu/cran"
options(repos=tennREPOS)
AV =  available.packages(contriburl = contrib.url(getOption("repos")),
                         method, fields = NULL)
LAV = length(AV[,1])
```

There are 0 packages in CRAN at the time of the creation of this document.

After a package is installed it can be invoked in an **R** session by calling the library function.

```
library(GEOmap)
```

If you are constantly using the same libraries you can put these commands in a file that is executed (sourced) every time **R** starts on your system.

3 Packages used in Seismology

There are several packages installed at CRAN that may be very helpful for investigating seismic data. These include software for reading and storing seismic data, exploratory analysis, interpretation and creation of publication quality figures. The strategy is to create useful and reproducible software that can serve as the foundation of the study of earthquakes and seismic waves.

- RPMG
- RSEIS
- Rquake
- GEOmap

- RFOC
- RTOMO
- geophys
- zoeppritz

These packages include numerous functions that depend on each other and on other **R** packages that can be installed from CRAN. In most cases when one installs a package that has dependencies they are resolved and the the associated packages are downloaded during the initial installation. Occasionally packages need to be updated to make sure they are compatable with the current **R** version.

When a package is loaded using the *library* or *require* command all the sublibraries required are also loaded and made ready for use. In some cases different libraries may use the same name for a function, and a warning will be issued. Usually this is not a problem and conflicts can be resolved by explicity tell **R** which version to use.

As an example, when we start library Rquake the session will load the following libraries:

```
library(Rquake)
```

To see all the libraries installed a=on the computer at any given time try: *library()*. On my computer here, the following results are returned:

```
LL = library()
print(LL[[2]][,1])
```

```
 [1] "ade4"          "adegenet"      "adimpro"
 [4] "akima"         "base"          "batch"
 [7] "bitops"        "boot"          "bvls"
[10] "Cairo"         "cda"           "CircStats"
[13] "class"         "clue"          "cluster"
[16] "codetools"     "compiler"      "compositions"
[19] "datasets"      "diagram"       "digest"
[22] "doBy"          "e1071"         "editrules"
[25] "EMD"           "energy"        "ETOPO"
[28] "evaluate"      "fields"        "filehash"
[31] "foreign"       "formatR"       "GEOmap"
```

```
 [34] "geomapdata"    "geophys"       "graphics"
 [37] "grDevices"     "grid"          "gstat"
 [40] "highlight"     "KernSmooth"    "knitr"
 [43] "lattice"       "lme4"          "lpSolve"
 [46] "mapdata"       "mapproj"       "maps"
 [49] "maptools"      "markdown"      "MASS"
 [52] "Matrix"        "mcclust"       "methods"
 [55] "mgcv"          "minpack.lm"    "multcomp"
 [58] "mvtnorm"       "ncdf"          "nlme"
 [61] "NMF"           "nnet"          "nnls"
 [64] "parallel"      "parser"        "PEIP"
 [67] "PET"           "pixmap"        "plotrix"
 [70] "plyr"          "png"           "pracma"
 [73] "R.matlab"      "R.methodsS3"   "R.oo"
 [76] "R2HTML"        "randtoolbox"   "raster"
 [79] "Rcpp"          "RcppArmadillo" "RCurl"
 [82] "reshape2"      "RFOC"          "rgeos"
 [85] "rgl"           "rjson"         "rngWELL"
 [88] "robustbase"    "rpanel"        "rpart"
 [91] "RPMG"          "Rquake"        "RSAGA"
 [94] "RSEIS"         "RTOMO"         "Rwave"
 [97] "shape"         "shapefiles"    "signal"
[100] "snow"          "sp"            "spacetime"
[103] "spam"          "spatgraphs"    "spatial"
[106] "splancs"       "splines"       "statmod"
[109] "stats"         "stats4"        "stringr"
[112] "survival"      "TauP.R"        "tcltk"
[115] "tcltk2"        "TELES"         "tensorA"
[118] "testthat"      "tikzDevice"    "tkrgl"
[121] "tkrplot"       "tools"         "utils"
[124] "WORLDMAP2"     "XML"           "xtable"
[127] "xts"           "zoeppritz"     "zoo"
```

4 Data Storage in R

All data are stored locally in the directory where R-was started, unless it is not specified. Data in **R** are usually organized as objects in one of the following kinds:

scalars numbers,

vectors sequences of numbers

matrices arrays of numbers

lists combinations of several objects

dataframe matrix of mixed mode objects (must have same length)

array higher dimensional matrix or array

Data are accessed by typing the name of the variable or by issuing a print command.

If you read in a vector of character strings that should be numeric you must convert the vector to mode numeric.

Here are some examples of setting these types of variables:

4.1 Scalar Examples

```
y = 3
x = 10
z = pi
w = x*y*z
h = x-y^z
```

4.2 Vector Examples

```
y = 1:10
h = rep(6.6, times=10)
x = y/30
z = sqrt(y)
w = z* y
w
```

```
[1]  1.000000  2.828427  5.196152  8.000000 11.180340
[6] 14.696938 18.520259 22.627417 27.000000 31.622777
```

Note: the bracketed numbers on the left are the index values of vector (or matrix).

Note: the colon operator creates a sequence of numbers.

Note: if you multiply 2 vectors they elements are multiplied element-by-element and a vector is returned. If you want the "dot" product you must sum the results

```
dotyz = sum(w)
print(dotyz)
```

```
[1] 142.6723
```

Other ways to create vectors:

```
x= runif(10)
y = seq(from=12, to=50, by=3)
```

4.3 Matrix Examples

```
y = matrix(1:20, ncol=4, nrow=5)
y
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

```
x =  matrix(runif(20) , ncol=4, nrow=5)
x
```

```
          [,1]       [,2]      [,3]       [,4]
[1,] 0.2644211 0.04856993 0.6225951 0.91758671
[2,] 0.5941977 0.65803509 0.9121923 0.64520937
[3,] 0.3460684 0.14155841 0.8941195 0.10508868
```

```
[4,] 0.7104925 0.11026551 0.1785283 0.00329403
[5,] 0.6454219 0.54138437 0.2002159 0.22526565
```

Note: the bracketed numbers on the left and top are the row and column index values of the matrix.

4.4 List Examples

Lists are heavily used in **R** . The are mixed vectors of several objects, perhaps scalars, vectors, matrices and other lists. They are very useful for conglomerating a lot of information into one object. Each member of a list may have a different mode (character, vector, numeric, etc.)

```
tel = 9995552456
ssn = "666-77-9898"
street = "555 Pensylvania Ave."
data1 = rnorm(5)
data2 = trunc(runif(10, 10, 100))
L1 = list(tel=tel, ssn=ssn, d1 =data1, d2=data2)
print(L1)
```

```
$tel
[1] 9995552456

$ssn
[1] "666-77-9898"

$d1
[1]  3.1387802 -0.8173817 -2.1770847  2.1484234  0.8920154

$d2
 [1] 69 32 70 83 76 52 54 96 11 71
```

4.5 Data Frame

A dataframe is a matrix (or a list) where each row has the same number of elements. A dataframe is basically a table, like a spreadsheet, with numbers, characters mixed together.

for example, we start with a list,

```
N = 5
names = vector(length=N, mode="character")
for(i in 1:N) { k=trunc(runif(1, 5, 10)); names[i]= paste(sample(letters, k), collapse="'
Area = rep(919, 5)
Pref = rep(555, 5)
tel = trunc( runif(5, 1000, 9000) )
data.frame(list(Name=names, Area=Area, Pref=Pref, tel=tel))
```

```
       Name Area Pref  tel
1   tyizpxa  919  555 6096
2 cmqstzwou  919  555 1075
3    ezauqy  919  555 1616
4    sgxthj  919  555 7289
5 mzwcfajdo  919  555 5325
```

5 Functions in R

All functions are called by using parentheses.

5.1 Files, IO

Getting data into an R session and saving data from a current R session are operations that often cause frustration with beginners. There are several ways to store data and this can be confusing. Here is a small bit of advice about data storage.

Data is usually stored as either ASCII (text) or a binary files. If the data is in a binary format not rcreated by **R** you will need to use a specialized program to read it in and you will need detailed information on how the data was written on the original computer where it was created.

If the data is in ASCII (text) format, there are a variety of ways to read it into **R** , depending on how it is organized in the file. There are specialized functions in **R** for reading files that are comma separated tables (CSV files). These are common for the output of spread sheet programs like MS-Excel.

read.table read a table of text

read.csv read a comma separated table

read.delim read a delimited table

readLines read a file line by line

scan low level read a file (see below)

5.2 Working Directory

After installation you should choose a working directory for **R** .

Windows Note

On a windows installation of **R** you will have a shortcut to Rgui.exe on your desktop and/or somewhere on the Start menu file tree, and perhaps also in the Quick Launch part of the taskbar (Vista and earlier). Right-click each shortcut, select Properties... and change the 'Start in' field to your working directory.

Two **R** commands can be used to get and set the directory where the user is working:

```
getwd()
setwd()
```

You may wish to always work in the default directory that you get when you start **R** . I do not recommend this approach.

Rather, for each project create a directory and work exclusively in that folder for the duration of that project. Keep your data and output files in that folder or other sub-folders there. When you start an **R** session, depending on which project you plan on working on, switch to that folder and load what you need there. Or, you may create a ".first" file and execute it so that it loads the appropriate libraries, data files, etc.

6 Important Commands

There are a few very important commands in R. These are used over and over in packages and at the command line. Some of these commands should be memorized since they are so basic.

```
ls()
help(ls)
save(file="R_mystuff")
history()
help.start()
set.seed(2)
rep()
seq(from=1, to=20, by=2)
plot(x,y)
par()
```

You may write all your **R** code into a file, save the file and execute it with the source command

```
source("filename.R")
```

7 Plotting and Graphics

Here is a plot:

```
JPOST(file="/Users/lees/Mss/SEIS_BOOK/Intro/FIGS/STUBBER.eps", width = 8,   height = 10)
## par(mai=c(.2, .2, .2,.2))
plot(rnorm(10), rnorm(10))
dev.off()
```

7.1 Plots: par()

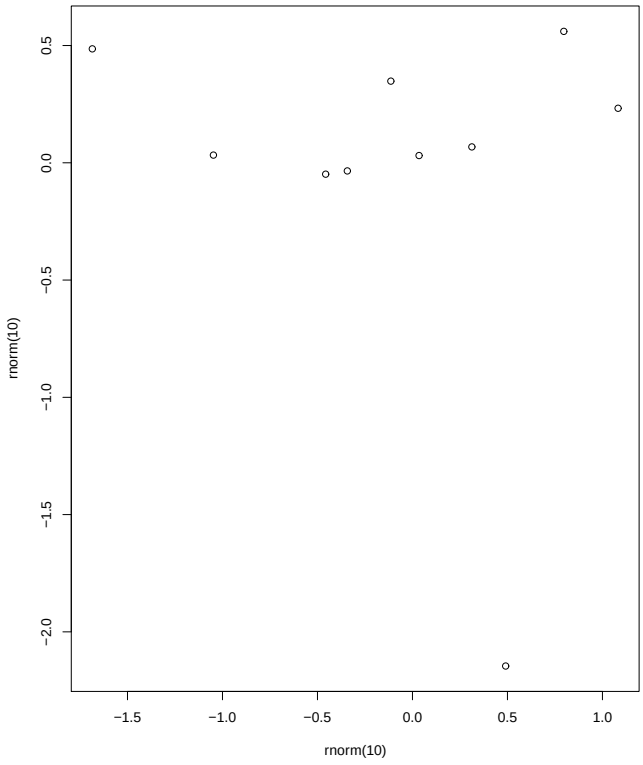

Figure 1: GEOmap Example **R**