R Language Essentials

Statistical Packages

STAT 1301 / 2300, Fall 2014

Sungkyu Jung
Department of Statistics
University of Pittsburgh

E-mail: sungkyu@pitt.edu
http://www.stat.pitt.edu/sungkyu/stat1301/

Section 1

R basics

R basics

R commands are entered at the prompt in the R console window.

```
> a <- 2;
> a
[1] 2
> x <- c(1,2,3)
> x
[1] 1 2 3
> a+2
[1] 4
> xsq = x^2
> rnorm(15)
 [1]  0.9175425  0.4328707  0.4734853 -1.4772252  0.4950732
 [6] -1.8456475  0.1006685 -0.9263071 -1.3785625 -0.4507060
[11] -0.9963989 -0.1078937  0.5003022 -0.5151607  0.3139086
```

- Create an object with the assignment operator <- or =.
- Names of variables (objects) can be built from letters, digits, and the period (dot) symbol, and are case-sensitive.
- `c` and `rnorm` are called functions.

Elementary Operators

Arithmetic

`+`	Addition
`-`	Subtraction, sign
`*`	Multiplication
`/`	Division
`^`	Raise to power
`%/%`	Integer division
`%%`	Remainder from integer division

Logical and relational

`==`	Equal to
`!=`	Not equal to
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`is.na(x)`	Missing?
`&`	Logical AND
`|`	Logical OR
`!`	Logical NOT

R Basic Operations

General R command syntax

```
> object <- object[arguments]
> object <- function_name(arguments)
> object
```

Examples of elementary functions

```
sum(x)
length(x)
mean(x)
sd(x)
plot(x)
```

R Basic Operations: An Exercise

- Compute the mean and standard deviation of data `weight`,
- Compute the BMI $= \mathit{weight}/(\mathit{height})^2$.
- Whose BMI is greater than 22?
- Plot weight vs height.
- Test whether the mean BMI equals 22.5.

```
> weight <- c(60, 72, 57, 90, 95, 72)
> height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

Data Types

Numeric data: 1, 2, 3

```
> x <- c(1, 2, 3); x
[1] 1 2 3
> is.numeric(x)
[1] TRUE
> as.character(x)
[1] "1" "2" "3"
```

Character data: "a", "b", "c"

```
> x <- c("1", "2", "3"); x
[1] "1" "2" "3"
> is.character(x)
[1] TRUE
> as.numeric(x)
[1] 1 2 3
```

Data Types

Complex data

```
> c(1, "b", 3)
[1] "1" "b" "3"
```

Logical data

```
> x <- 1:10 < 5
> x
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
> !x
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> which(x) # Returns index for the 'TRUE' values in logical vector
[1] 1 2 3 4
```

Vectors (1D)

```
> x <- c(red="Huey", blue="Dewey", green="Louie")
> x
    red    blue   green
 "Huey" "Dewey" "Louie"
> names(x)
[1] "red"   "blue"  "green"
> x[1] <- "Hey"; x
    red    blue   green
  "Hey" "Dewey" "Louie"
>
> y <- 1:10; names(y) <- letters[1:10]
> y[c("b", "d", "f")]
b d f
2 4 6
```

Vectors (1D)

```
> 1:7
[1] 1 2 3 4 5 6 7
> 6:2
[1] 6 5 4 3 2
> seq(from = 10, to = 20, by = 2)
[1] 10 12 14 16 18 20
> seq(-2, -1, 0.1)
 [1] -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1.0
> seq(0, 5, length=7)
[1] 0.0000000 0.8333333 1.6666667 2.5000000 3.3333333 4.1666667
[7] 5.0000000
>
> rep(x = 1, times = 5)
[1] 1 1 1 1 1
> rep(1,5)
[1] 1 1 1 1 1
> rep(c(1,2),3)
[1] 1 2 1 2 1 2
> rep(1:2,c(3,5))
[1] 1 1 1 2 2 2 2 2
```

Creating Matrices

Matrices (2D): two dimensional structures with data of same type

```
> x <- 1:12
> dim(x) <- c(3,4);x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> x <- matrix(1:12,nrow = 3, byrow = T)
> rownames(x) <- month.abb[1:3] ; x
    [,1] [,2] [,3] [,4]
Jan    1    2    3    4
Feb    5    6    7    8
Mar    9   10   11   12
```

Useful functions that operate on matrices : `rownames`, `colnames`, and the transposition function `t`, `as.vector`, `cbind`, `rbind`.

Creating Matrices

Matrices (2D): two dimensional structures with data of same type

```
> cbind(A=1:4,B=5:8,C=9:12)
     A B  C
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> x <- rbind(A=1:4,B=5:8,C=9:12); x
  [,1] [,2] [,3] [,4]
A    1    2    3    4
B    5    6    7    8
C    9   10   11   12
> t(x)
     A B  C
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
```

Factors (1D): vectors with grouping information

Categorical variables should be specified as factors in R.

```
> pain <- c(0,3,2,2,1)
> as.factor(pain)
[1] 0 3 2 2 1
Levels: 0 1 2 3
> fpain <- factor(pain,levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
> fpain
[1] none severe medium medium mild
Levels: none mild medium severe
> as.numeric(fpain)
[1] 1 4 3 3 2
> levels(fpain)
[1] "none" "mild" "medium" "severe"
```

If the variable is ordinal, use `ordered`.

Lists

List: to combine a collection of objects into a larger composite object

```
> intake.pre <- c(5260,5470,5640,6180,6390,
+ 6515,6805,7515,7515,8230,8770)
> intake.post <- c(3910,4220,3885,5160,5645,
+ 4680,5265,5975,6790,6900,7335)
> mylist <- list(before=intake.pre,after=intake.post)
> mylist$new <- as.matrix(c("n"))
> mylist
$before
 [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770

$after
 [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335

$new
     [,1]
[1,] "n"
```

Many of R's built-in functions compute more than a single vector of values and return their results in the form of a list.

Data Frames

Data Frames (2D) corresponds to what other statistical packages call a "data matrix" or a "data set".

```
> d <- data.frame(intake.pre,intake.post)
> head(d)
  intake.pre intake.post
1       5260        3910
2       5470        4220
3       5640        3885
4       6180        5160
5       6390        5645
6       6515        4680
> d$intake.pre
 [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

Indexing

Subsetting by indices

```
> intake.pre[5]
[1] 6390
> intake.pre[c(3,5,7)]
[1] 5640 6390 6805
> intake.pre[-c(3,5,7)]
[1] 5260 5470 6180 6515 7515 7515 8230 8770
> x <- matrix(1:12,nrow = 3, byrow = T)
> x[1:2,]
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Subsetting by logical selection

```
> intake.post[intake.pre > 7000]
[1] 5975 6790 6900 7335
> x[,(1:4) == (1:4)^2]
[1] 1 5 9
```

Missing Values

In SAS, Missing Values are denoted by '.' (period).

In R, missing values are denoted by a special NA value.

```
> x <- 1:10
> x[2] <- NA
> is.na(x)
 [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> x>0
 [1] TRUE    NA TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> x <- c(red="Huey", blue="Dewey", green="Louie")
> x[4] <- NA
> x
    red     blue   green
 "Huey" "Dewey" "Louie"       NA
```

Sorting

The function `sort` returns a vector in ascending or descending order

```
> sort(10:1)
 [1]  1  2  3  4  5  6  7  8  9 10
```

The function `order` returns a sorting index for sorting an object

```
> sortindex <- order(iris[,1], decreasing = FALSE)
> sortindex[1:12]
 [1] 14  9 39 43 42  4  7 23 48  3 30 12
> iris[sortindex,][1:3,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14          4.3         3.0          1.1         0.1  setosa
9           4.4         2.9          1.4         0.2  setosa
39          4.4         3.0          1.3         0.2  setosa
```

- Here, we are using `iris`, a data set preloaded for any R system.

Section 2

The R environment

Getting Around

List objects in current R session

```
> ls()
```

Delete some of the objects

```
> rm(height, weight)
> rm(list=ls())
```

Return content of current working directory

```
> dir()
```

Return path of current working directory

```
> getwd()
```

Change current working directory

```
> setwd("/home/user")
```

ALWAYS USE FORWARD SLASHES (/).

R script: similar to a SAS program

Beyond a certain level of complexity, you will not want to work with R on a line-by-line basis. It is better to work with R scripts, collections of lines of R code stored either in a file or in computer memory somehow.

Listing 1: `my_script.R`

```
intake.pre <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,
intake.post <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900
mylist <- list(xbefore=intake.pre,after=intake.post)
d <- data.frame(intake.pre,intake.post)
head(d)
```

Execute an R script from within R

```
> source("my_script.R")
```

Execute an R script interactively

You can work with a script editor window, which allows you to submit one or more lines of the script to a running R, which will then behave as if the same lines had been entered at the prompt.

- Mac: cmd + enter, Windows: F5

Packages

An R installation contains one or more libraries of packages. Some of these packages are part of the basic installation. Others can be downloaded from CRAN.

Installation of CRAN Packages

```
> install.packages(c("ISwR", "pkg2"))
> install.packages("pkg.zip", repos=NULL)
```

Load a library

```
> library(ISwR)
```

Lists all functions and datasets defied by a library

```
> library(help="ISwR")
```

Getting Help

- R also comes with extensive online help in text form as well as in the form of a series of HTML files that can be read using a Web browser
- The help pages can be accessed via “help” in the menu bar on Windows and by entering `help.start()` on any platform.
- From the command line, you can always enter `help(aggregate)` to get help on the aggregate function or use the prefix form `?aggregate`
- The function `help.search` is similar but uses fuzzy matching and searches deeper into the help pages, so that it will be able to locate, for example, Kendall’s correlation coefficient in `cor.test` if you use `help.search("kendal")`.

Data Entry: Built-in Data

- Many packages, both inside and outside the standard R distribution, come with built-in data sets.
- lazy loading: the built-in data sets are loaded when they are referenced for the first time.
- With this mechanism, data are *just there*. For example, if you type `thuesen` or `data(thuesen)`, the *data frame* of that name is displayed.
- Example: Load and view first few lines of data frames `melanom`, `iris`, `USArrests`, and `cabbages` from MASS package.

Data Inspection

```
> head(thuesen)
  blood.glucose short.velocity
1          15.3           1.76
2          10.8           1.34
3           8.1           1.27
4          19.5           1.47
5           7.2           1.27
6           5.3           1.49
> nrow(thuesen)
[1] 24
> ncol(thuesen)
[1] 2
> dim(thuesen)
[1] 24  2
> str(thuesen)
'data.frame': 24 obs. of  2 variables:
 $ blood.glucose : num  15.3 10.8 8.1 19.5 7.2 5.3 9.3 11.1 7.5 12
 $ short.velocity: num  1.76 1.34 1.27 1.47 1.27 1.49 1.31 1.09 1.
> summary(thuesen)
 blood.glucose    short.velocity
 Min.   : 4.200   Min.   :1.030
 1st Qu.: 7.075   1st Qu.:1.185
 Median : 9.400   Median :1.270
 Mean   :10.300   Mean   :1.326
 3rd Qu.:12.700   3rd Qu.:1.420
 Max.   :19.500   Max.   :1.950
                  NA's   :1
```

Data Manipulations

attach and detach

You can make R look for objects among the variables in a given data frame

```
> plot(thuesen$blood.glucose,thuesen$short.velocity)
> attach(thuesen); search();
> plot(blood.glucose,short.velocity)
> detach(thuesen); search();
```

subset, transform, and within

```
> thue2 <- subset(thuesen,blood.glucose<7)
> thue3 <- transform(thuesen,log.gluc=log(blood.glucose))
> thue4 <- within(thuesen,{
+ log.gluc <- log(blood.glucose)
+ m <- mean(log.gluc)
+ centered.log.gluc <- log.gluc - m
+ rm(m)
+ })
```

Data Entry: Manual Entry

- A table of gas mileage on four new models of Japanese luxury cars.
- To test whether the four models give the same gas mileage.

A	B	C	D
22	28	29	23
26	24	32	24
	29	28	

Create data frame `mileages`

```
y1 = c(22, 26)
y2 = c(28, 24, 29)
y3 = c(29, 32, 28)
y4 = c(23, 24)
y = c(y1, y2, y3, y4)
Model = c(rep("A", 2), rep("B", 3), rep("C", 3), rep("D", 2))
mileages = data.frame(y, Model)
str(mileages)
mileages
```

Exercise: Change the variable name “y” to “mileage”

Data Entry: Reading from and Writing to text files

Read from files: read.table, read.csv and read.delim

To read example data files webhits.txt, flicker.txt, twins.txt

```
> setwd("C:/R")
> webhits <- read.table("webhits.txt", header = TRUE)
> flicker <- read.table("flicker.txt", header = TRUE)
```

Arguments of read.table(file,...)

- file: path + file name.
  If it does not contain an absolute path, the file name is relative to the current working directory. ALWAYS USE FORWARD SLASHES (/).
- header = FALSE: a logical value indicating whether the file contains the names of the variables as its first line.
- sep = "": the field separator character. "" (white space), "," (comma separated data) or "\t" (tab-separated data).
- na.strings = "NA": a character vector of strings which are to be interpreted as NA values.

Data Entry: Reading from and Writing to text files

read.table

Type conversions: by default, the read.table guesses and converts the data types of the different columns (e.g. number, factor, character). There are options as.is and colClasses to control this. read the online help or type ?read.table.

Exercise: Read data from twins.txt

Write a data fame to a file: write.table and write.csv

```
> write.table(twins, "twins2.txt")
```

Section 3

Probability and Distributions

An Example: Poisson density (mass) function

Prussian horse kicks

von Bortkiewicz's (1898) data on death of soldiers in the Prussian army from kicks by horses and mules. The data pertain to 10 army corps, each observed over 20 years. In 109 corps-years, no deaths occurred; 65 corps-years had one death, etc.

```
# Prussian horsekick data
k = c(0, 1, 2, 3, 4)
x = c(109, 65, 22, 3, 1)
p = x / sum(x)          #relative frequencies
print(p)
par(mfrow = c(1,2))
barplot(x, names= k); pie(x)
r = sum(k * p)   #mean
v = sum(x * (k - r)^2) / 199  #variance
f = dpois(k, r)
print(cbind(k, p, f))
```

Random sampling

`sample()` function for random sampling from finite population

```
> sample(1:40,5)
[1] 4 30 28 40 13
> sample(c("H","T"), 10, replace=T)
 [1] "T" "T" "T" "T" "T" "H" "H" "T" "H" "T"
> sample(c("s", "f"), 10, replace=T, prob=c(0.9, 0.1))
 [1] "s" "s" "s" "s" "s" "f" "s" "s" "s" "s"
```

Exercise:

- Suppose there are five red, six blue, and seven green balls in a jar. Randomly sample five balls from the jar.
- Randomly sample 50 birthdays, and inspect whether these birthdates are all distinct.

The built-in distributions in R

Four fundamental items can be calculated for a statistical distribution

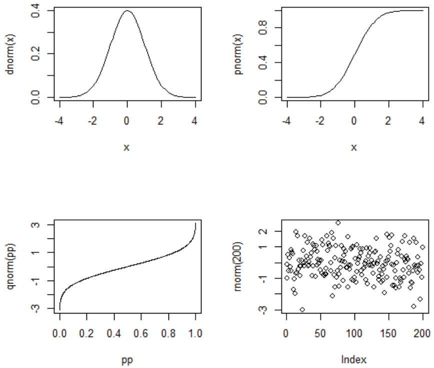
- Density or point probability
- Cumulated probability, distribution function
- Quantiles
- Pseudo-random numbers

For example, for the normal distribution, these are named `dnorm`, `pnorm`, `qnorm`, and `rnorm` (*d*ensity, *p*robability, *q*uantile, and *r*andom, respectively).

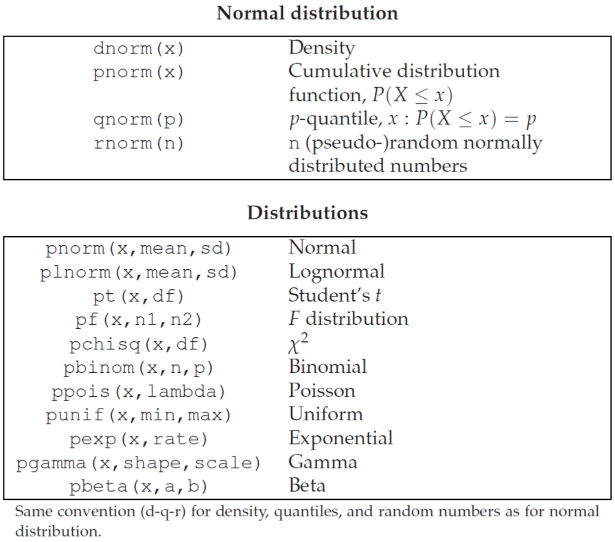
The built-in distributions in R

dnorm, pnorm, qnorm, and rnorm

```
x = seq(-4,4,0.1); pp = seq(0,1,0.001);
par(mfrow = c(2,2))
plot(x,dnorm(x),type = "l")
plot(x,pnorm(x),type = "l")
plot(pp,qnorm(pp),type = "l")
plot(rnorm(200))
```



The built-in distributions in R



Normal distribution

`dnorm(x)`	Density
`pnorm(x)`	Cumulative distribution function, $P(X \leq x)$
`qnorm(p)`	p-quantile, $x : P(X \leq x) = p$
`rnorm(n)`	`n` (pseudo-)random normally distributed numbers

Distributions

`pnorm(x,mean,sd)`	Normal
`plnorm(x,mean,sd)`	Lognormal
`pt(x,df)`	Student's t
`pf(x,n1,n2)`	F distribution
`pchisq(x,df)`	χ^2
`pbinom(x,n,p)`	Binomial
`ppois(x,lambda)`	Poisson
`punif(x,min,max)`	Uniform
`pexp(x,rate)`	Exponential
`pgamma(x,shape,scale)`	Gamma
`pbeta(x,a,b)`	Beta

Same convention (d-q-r) for density, quantiles, and random numbers as for normal distribution.

Exercises

- Inspect the densities of Beta distributions for various values of parameters (a, b) and compute $P(X < 0.5)$ for X Beta$(1, 2)$.

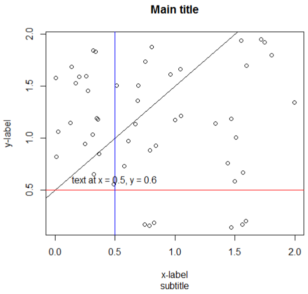
- Plot the masses (densities) of Binomial distributions for various values of parameters (n, p). Compare with normal densities $N(np, np(1-p))$. Compute the probability that Binomial (or normal) random variable is less than mean $-$ 2s.d.

Section 4

R Graphics

Example 1: plot, text and abline

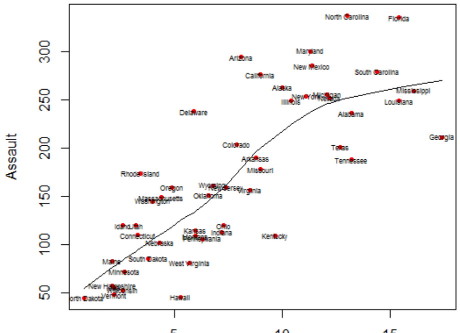
```
par(mfrow = c(1,1))
x <- runif(50,0,2);y <- runif(50,0,2)
plot(x, y, main="Main␣title", sub="subtitle",
    xlab="x-label", ylab="y-label")
abline(a = 0.5, b = 1)
abline(h = 0.5, col = 'red')
abline(v = 0.5, col = "blue")
text(0.5,0.6,"text␣at␣x␣=␣0.5,␣y␣=␣0.6")
```



Generic x - y plot

Example 2: plot, text and lines

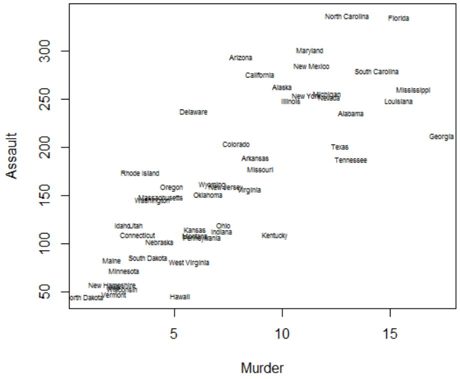
```
head(USArrests)
attach(USArrests)
plot(Murder,Assault, pch = 20, col = "red")
text(Murder,Assault,rownames(USArrests),cex = 0.5)
localregfit <- lowess(Assault~Murder)
lines(localregfit$x, localregfit$y)
```



Generic x - y plot

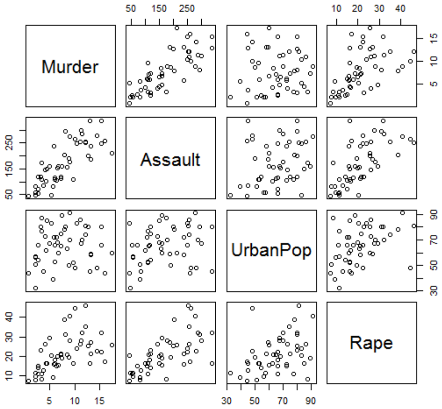
Example 3: Print the rownames instead of symbols

```
plot(Murder,Assault, type = "n")
text(Murder,Assault,rownames(USArrests),cex = 0.5)
```



Example 4: Scatterplot matrix for multi-variate data frame

```
plot(USArrests)
pairs(USArrests)
```



Remarks

Add more points to the current plot by `points(x,y)`

Commonly used graphical parameters

- `type = "p"`: what type of plot should be drawn. To get more info, type `?plot`
- `pch = 1`: plotting character, i.e., symbol to use. To get more info, type `?points`
- `cex = 1`: A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default.
- `col = "black"` Colors can be specified in several different ways. The simplest way is with a character string giving the color name (e.g., "red"). A list of the possible colors can be obtained with the function `colors`. The functions `rgb`, `hsv`, `hcl`,`gray` and `rainbow` provide additional ways of generating colors.

Example:

```
> pie(rep(1, 12), col = rainbow(12))
```

Remarks

The par function allows incredibly fine control over the details of a plot
The par settings allow you to control line width and type, character size and font, colour, style of axis calculation, size of the plot and figure regions, clipping, etc. (These can also be set by individual plotting functions.)
It is possible to divide a figure into several subfigures by using the mfrow and mfcol parameters.

Commonly used arguments

- `mfrow = c(1,1)`: muli-frame, row-wise order = `c(rownum,colnum)`
- `mfcol = c(1,1)`: muli-frame, column-wise order = `c(rownum,colnum)`
- `mar = c(5,4,4,2)`: margin sizes = `c(bottom, left, top, right)`

Example:

```
> par(mfrow = c(1,2))
> plot(Murder, Assault)
> plot(Rape, UrbanPop)
```

Important high-level plotting functions

Commonly used functions

- plot: generic x-y plotting
- barplot: bar plots
- boxplot: box-and-whisker plot
- hist: histograms
- pie: pie charts
- dotchart: cleveland dot plots
- qqnorm, qqline, qqplot: distribution comparison plots
- pairs: display of multivariant data

An example from textbook Section 2.2.4 (p.43)

```
> h <- hist(x, plot=F)
> ylim <- range(0, h$density, dnorm(0))
> hist(x, freq=F, ylim=ylim)
> curve(dnorm(x), add=T)
```

Section 5

R Programming Basics

R programming Basics: Flow Control

Grouped expressions

- R is an expression language in the sense that its only command type is a function or expression which returns a result.
- Commands may be grouped together in braces, `{expr 1, . . ., expr m}`, in which case the value of the group is the result of the last expression in the group evaluated.

Conditional Executions: `if` statement

```
> if( condition ) {   statements_1   }
+ else {  statements_2   }
```

Example:

```
> if(1==0) {
+ print(1)
+ } else {
+ print(2)
+ }
[1] 2
```

R programming Basics: Flow Control

Conditional Executions: `ifelse` statement operates on vectors

```
> ifelse( condition(s) , true_value , false_value)
Example
> x <- 1:10
> ifelse(x<5, x, 0)
[1]  1
```

Example:

```
> x <- 1:10
> ifelse(x<5, x, 0)
 [1] 1 2 3 4 0 0 0 0 0 0
```

R programming Basics: Flow Control

for Loops

Loops over a fixed set of values

```
> for(variable in sequence) {
+ statements
+ }
```

Example:

```
> x <- seq(0, 1,.05)
> plot(x, x, ylab="y", type="l")
> for ( j in 2:8 ) lines(x, x^j)
```

R programming Basics: Flow Control

`while` Loops

Iterates as long as a condition is true

```
> while(condition) {
+ statements
+ }
```

Example:

```
> z <- 0
> while(z<5) {
+ z <- z + 2
+ print(z)
+ }
[1] 2
[1] 4
[1] 6
```

R programming Basics: Flow Control

while Loop with break statement

```
> while(TRUE) {
+ statements
+  if (condition) break
+ }
> repeat {
+ statements
+  if (condition) break
+ }
```

Example:

```
> z <- 0
> while(T) {
+  z <- z+ 1
+  print(z)
+  if (z > 2) break
+  }
[1] 1
[1] 2
[1] 3
```

Implicit loops

The `apply` function family

A common application of loops is to apply a function to each element of a set of values or vectors and collect the results in a single structure. In R, `apply` allows you to apply a function to the rows or columns of a matrix.

`apply`

```
> apply(X, MARGIN, FUN, ARGs)
```

Arguments

- X: array, matrix or data.frame
- MARGIN: 1 for rows, 2 for columns
- FUN: one or more functions to be applied
- ARGs: possible arguments for functions

Example (mean computed over each column of USArrests data)

```
> apply(USArrests, 2, mean)
  Murder  Assault UrbanPop     Rape
   7.788  170.760   65.540   21.232
```

Implicit loops

To operate on vector or list objects, use `lapply` and `sapply`. The former always returns a *l*ist, whereas the latter tries to *s*implify the result to a vector or a matrix if possible.

`lapply`, `sapply`

```
> lapply(X, FUN, ARGs)
> sapply(X, FUN, ARGs)
```

Arguments

- X: vector, list or data.frame
- FUN: one or more functions to be applied
- ARGs: possible arguments for functions

Example (mean computed over variable of thuesen data)

```
> lapply(thuesen, mean, na.rm=T)
$blood.glucose
[1] 10.3
$short.velocity
[1] 1.325652
> sapply(thuesen, mean, na.rm=T)
blood.glucose short.velocity
10.300000 1.325652
```

Implicit loops

Also, the function `tapply` allows you to create *t*ables of the value of a function on subgroups defined by its second argument, which can be a factor or a list of factors.

`tapply`

```
> tapply(vector, factor, FUN)
```

Example (median of energy expenditure for each level (lean, obese))

```
> data(energy)
> tapply(energy$expend, energy$stature, median)
lean obese
7.90 9.69
```

Your Own R functions

A very useful feature of the R environment is the possibility to expand existing functions and to easily write custom functions. In fact, most of the R software can be viewed as a series of R functions.

To define functions

```
> myfn <- function(arg1, arg2, ...) {
+ function_body
+ }
```

To call functions

```
> myfn(arg1=..., arg2=...)
```

R function example 1

Define a sample function

```
> myfn <- function(x1, x2=5) {
+     z1 <- x1 / x1
+     z2 <- x2 * x2
+     myvec <- c(z1, z2)
+     return(myvec)
+     }
```

Function usage

```
>
> myfn(x1 = 2, x2 = 3)
[1] 1 9
> myfn(2,3)
[1] 1 9
> x <- myfn(5); x
[1]  1 25
```

R function example 2

```
> hist.with.normal <- function(x, xlab=deparse(substitute(x)),...)
+    {
+      h <- hist(x, plot=F, ...)
+      s <- sd(x)
+      m <- mean(x)
+      ylim <- range(0,h$density,dnorm(0,sd=s))
+      hist(x, freq=F, ylim=ylim, xlab=xlab, ...)
+      curve(dnorm(x,m,s), add=T)
+      }
```

R function exercise

- Write a function that standardizes all variables in a data frame.
- Apply the function to the data loaded from `twins.txt`.