Introduction to R

Nilotpal Sanyal

(nilotpal.sanyal@gmail.com)

Bayesian and Interdisciplinary Research Unit

(currently Interdisciplinary Statistical Research Unit)

Indian Statistical Institute

[R commands are in red and outputs are in blue]

- ✓ R is a great programming language – easy to learn, user-friendly, funny, and absolutely free! Play with it!

R Website; Download R (Latest version R-3.1.2): http://www.r-project.org/

Topics:

R Windows	Variables	Plots and images	Random samples
Run from editor	Vectors	R loops: if, for	Density, distribution function, quantum and random samples from distributions
Arithmetic	Useful commands	Read and write files	Defining a function
Useful commands	Matrices	Save and load console	R package

R Windows

- ✓ R opens as a large window named RGui (Graphical user interface), inside which you will see a smaller window named 'R Console'. In this console window the R codes run.

- ✓ You can write the codes directly into console and press enter to run. But, in console, editing option is very limited. So, better to open 'R Editor' window by choosing File → New script. This editor window is like notepad with flexible editing options. You can save the new script file with usual Ctrl+s from keyboard. The default saved file extension is .R (can be opened later by R or Notepad both).

How to run code from editor

- ✓ Write the code in editor.
- ✓ Select whole code in editor with Ctrl+a from keyboard, or using mouse just select a portion of the code that you want to run.

- ✓ Then, to run the code either press Ctrl+r from keyboard or click the button that looks like  . The code will run in the console and output will be in console.

Do Simple Arithmetic operations

Type in editor the following and run to see the result in console.

```
34 + 56*45 / 45
[1] 90
(2 + 4)/(5 - 7)
[1] -3
```

Tips: Practice using R (instead of calculator) for everyday arithmetic calculations.

Useful R commands

```
Ctrl + l          # clears the console screen
version           # shows R software version, platform etc.
# bla bla bla     # Anything after # is treated as comment and not run
builtins()        # lists all built-in functions (come installed with R)
```

Define variables and perform common mathematical operations

```
x <- 2.5857   # a variable x receives the value 2.5857

y <- -5.95          # a variable y receives the value -5.95
```

Note: In above codes, you could also use = in place of <- and get the same result. However, in general there is a difference between using = and <- (for later discussion. Remind me!).

```
x + y               # adds x and y
[1] -3.3643

x * y               # multiplies x and y
[1] -15.38492

(x - y)^2           # squares the difference between x and y
[1] 72.85817

(x + 2*y)^10        #
```

```
[1] 4914758904

sqrt(x)          # returns square root of x
[1] 1.608011

sign(x)          # returns sign of x (1 for positive, -1 for negative)
[1] 1

floor(x)         # returns the highest integer < or = x
[1] 2

ceiling(x)       # returns the smallest integer > or = x
[1] 3

log(x)           # returns logarithm of x with base e
[1] 0.9499963

log2(x)          # returns logarithm of x with base 2
[1] 1.370555
```

```
log10(x)          # returns logarithm of x with base 10
[1] 0.4125781

exp(x)            # returns exponential of x
[1] 13.27258

sin(x)            #
[1] 0.5277018

cos(x)            #
[1] -0.8494297

tan(x)            #
[1] -0.6212424

round(x,2)        # rounds x to 2 digits after decimal
[1] 2.59

round(x,3)        # rounds x to 3 digits after decimal
[1] 2.586
```

```
abs(y)              # returns the absolute value of y
[1] 5.95
```

Defining vectors and various operations with vectors

```
x <- c(1,2,2,3)     # a variable x receives a vector of 4 elements

y <- c(4,6,9,10)    # a variable y receives a vector of 4 elements

x + y               # adds vectors x and y element-wise
[1]  5  8 11 13

x * y               # multiplies x and y element-wise
[1]  4 12 18 30

x^2                 # squares x element-wise
[1] 1 4 4 9

z <- c(x,y)         # combines x and y in a new vector z
```

```
 z
[1]  1  2  2  3  4  6  9 10
```

```
w <- 1:10            # returns all integers from 1 to 10
w
[1]  1  2  3  4  5  6  7  8  9 10
```

```
m <- seq(from=1, to=10, by=1)
```

returns a sequence of numbers from 1 to 10 with increment 1

```
m
[1]  1  2  3  4  5  6  7  8  9 10
```

```
n <- seq(from=1, to=10, length=4)
```

returns a sequence of numbers of length 4 from 1 to 10 with equal difference between the numbers

```
n
[1]  1  4  7 10
```

```
x1 <- rep(4, 10)          # repeats the number 4 ten times
x1
```

```
 [1] 4 4 4 4 4 4 4 4 4 4

x2 <- c(rep(2,4),rep(9,5))   #
x2
[1] 2 2 2 2 9 9 9 9 9

x[3]        # returns the third element of vector x
[1] 2

length(x)       # returns the length of vector x
[1] 4

max(x)          # returns the maximum element of x
[1] 3

min(x)          # returns the minimum element of x
[1] 1

range(x)        # returns the maximum and minimum of x
[1] 1 3
```

```
unique(x)      # returns only the distinct elements of x
[1] 1 2 3

rev(x)         # returns vector x in the reverse order
[1] 3 2 2 1

sort(x)        # sorts the elements of x in increasing manner
[1] 1 2 2 3

sort(x, decreasing=T)       # sorts the elements of x in decresing manner
[1] 3 2 2 1

sum(x)         # returns sum of the elements of x
[1] 8

mean(x)        # returns mean/average of the elements of x
[1] 2

median(x)      # returns median of the elements of x
```

```
[1] 2

sd(x)           # returns standard deviation of the elements of x
[1] 0.8164966

var(x)          # returns variance of the elements of x
[1] 0.6666667

summary(x)      # returns minimum, maximum and the three quartiles of the
                  elements of x
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.75    2.00    2.00    2.25    3.00

quantile(x, .56)        # returns 56% quantile of x
56%
  2

cor(x,y)        # returns the correlation coefficient of x and y
[1] 0.8894992
```

```
which(y > 5) # elements at which positions of y are greater than 5
[1] 2 3 4

which(y == max(y))        # element at which position of y is the maximum
                          element of y
[1] 4
```

Useful R commands

```
ls()          # shows all objects currently in the R workspace
rm(x)         # remove x from R workspace
help(mean)    # opens the R help page for the function 'mean'
date()        # shows current date and time
```

Defining matrices and various operations with matrices

```
A <- matrix(c(1,2,4,2,4,5), nrow=2, ncol=3)    # Variable A receives a matrix
                                               of 6 elements with 2 rows and
                                               3 columns
```

```
A
     [,1] [,2] [,3]
[1,]    1    4    4
[2,]    2    2    5

dim(A)          # shows the dimension of matrix A
[1] 2 3

B <- matrix(5:10, 2, 3)      # Variable B receives a matrix of 6 elements with 2
                             rows and 3 columns
B
     [,1] [,2] [,3]
[1,]    5    7    9
[2,]    6    8   10

C <- matrix(c(20:27,29), byrow=F, nrow=3, ncol=3)      # …elements enter by
                                                       column
C
     [,1] [,2] [,3]
[1,]   20   23   26
```

```
[2,]   21   24   27
[3,]   22   25   29

A[2,3]          # (2,3)th element of matrix A
[1] 5

B[2,2] * C[1,4]
Error in C[1, 4] : subscript out of bounds

A + B           # adds matrices A and B element-wise
     [,1] [,2] [,3]
[1,]    6   11   13
[2,]    8   10   15

A + C
Error in A + C : non-conformable arrays

A * B           # multiplies matrices A and B element-wise
     [,1] [,2] [,3]
[1,]    5   28   36
```

```
[2,]   12   16   50
```

```
A * C
Error in A * C : non-conformable arrays
```

```
A %*% C      # multiplies matrices A and B (matrix multiplication)
     [,1] [,2] [,3]
[1,]  192  219  250
[2,]  192  219  251
```

```
t(A)       # returns the transpose of matrix A
     [,1] [,2]
[1,]    1    2
[2,]    4    2
[3,]    4    5
```

```
det(C)          # returns the determinant of matrix C
[1] -3
```

```
solve(C)       # returns inverse of matrix C
```

```
     [,1]      [,2] [,3]
[1,]   -7  5.666667    1
[2,]    5 -2.666667   -2
[3,]    1 -2.000000    1
```

```
diag(4)          # returns a diagonal matrix of order 4 with diagonal elements 1
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

```
diag(c(1, 5, 3, 7.3))
```

returns a diagonal matrix of order 4 with given diagonal elements

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0  0.0
[2,]    0    5    0  0.0
[3,]    0    0    3  0.0
[4,]    0    0    0  7.3
```

```
rbind( c(1,2,3), c(4,5,6) )        # Binds the two vectors as two rows
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

cbind( c(1,2,3), c(4,5,6) )        # Binds the two vectors as two columns
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

rbind(A, B)         # Binds the rows of the two matrices A and B
     [,1] [,2] [,3]
[1,]    1    4    4
[2,]    2    2    5
[3,]    5    7    9
[4,]    6    8   10

cbind(A, B)         # Binds the columns of the two matrices A and B
     [,1] [,2] [,3] [,4] [,5] [,6]
```

```
[1,]   1   4   4   5   7   9
[2,]   2   2   5   6   8  10
```

```
rowSums(C)        # Returns the sums of the rows of matrix C
[1] 69 72 76
```

```
rowMeans(C)       # Returns the means of the rows of matrix C
[1] 23.00000 24.00000 25.33333
```

```
colSums(C)        # Returns the sums of the columns of matrix C
[1] 63 72 82
```

```
colMeans(C)       # Returns the means of the columns of matrix C
[1] 21.00000 24.00000 27.33333
```

Characters

```
x <- "a"                    # x receives a character element “a”
```

```
y <- letters[1:6]          # y receives a character vector with first six alphabets as
                           elements
class(x)                   # shows the class of x
[1] "character"
```

Data frames

```
x <- data.frame(1,2)          # x receives a data frame of two elements
class(x)
[1] "data.frame"
y <- data.frame(m=1,n=2)          # including names of data columns
y$n                           # gives the data column named n of data frame y
[1] 2
data.frame(a=1,b=2:5)
  a b
1 1 2
```

```
2 1 3
3 1 4
4 1 5
```

Lists

```
x <- list(2,3)          # x receives a list of two numeric elements
class(x)
[1] "list"
length(x)
[1] 2
x[1]
[[1]]
[1] 2

x[[1]]
[1] 2
y <- list(2,"f")        # y receives a list of two elements, one numeric and one
                          character
y[[2]]
```

```
[1] "f"
z <- list(a=2:7, b="f")        # z receives a list of two numeric elements, one
                               numeric and one character
z
$a
[1] 2 3 4 5 6 7

$b
[1] "f"

names(z)                       # shows the names of the elements of z
[1] "a" "b"
z$a                            # shows the element with name a of list z
[1] 2 3 4 5 6 7
x <- c(x, 45)                  # adding one element to existing list x

is.vector(x)
[1] FALSE
is.character(x)
[1] FALSE
```

```
is.matrix(x)
[1] FALSE
is.data.frame(x)
[1] FALSE
is.list(x)
[1] TRUE
```

```
as.vector(c(1,2))
[1] 1 2
as.character(c(1,2))
[1] "1" "2"
as.matrix(c(1,2))
     [,1]
[1,]    1
[2,]    2
as.data.frame(c(1,2))
  c(1, 2)
1       1
```

```
2       2
as.list(c(1,2))
[[1]]
[1] 1

[[2]]
[1] 2
```

Set-theoretic mathematical functions

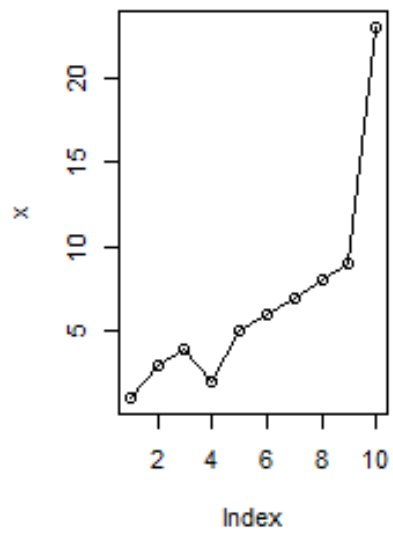
```
choose(5,2)        # the number of ways to choose 2 elements out of 5 = 5C2
[1] 10
factorial(4)
[1] 24
x1 <- c(1,2,3,4)
x2 <- c(3,4,5,8)
union(x1,x2)       # union of two sets
[1] 1 2 3 4 5 8
intersect(x1,x2)   # intersection of two sets
```

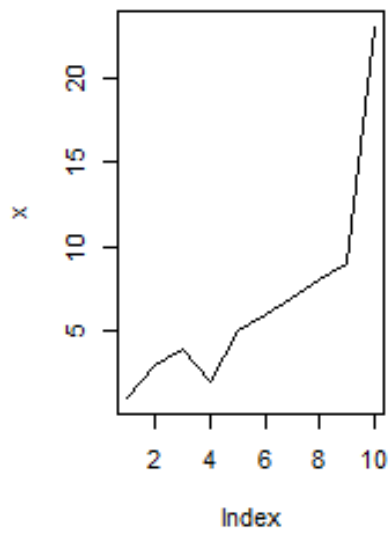
```
[1] 3 4
setdiff(x1,x2)      # Set x1 difference Set x2
[1] 1 2
setdiff(x2,x1)      # Set x2 difference Set x1
[1] 5 8
setequal(x1,x2)     # checks if sets x1 and x2 are equal
[1] FALSE
setequal( union(x1,x2), c( setdiff(x1,x2), intersect(x1,x2), setdiff(x2,x1) ) )
[1] TRUE
is.element(4,x1)    # checks if 4 is element of set x1
[1] TRUE
is.element(12,x2)
[1] FALSE
```

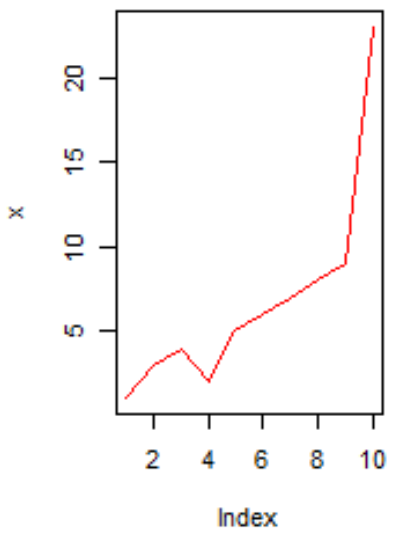
Plots and images

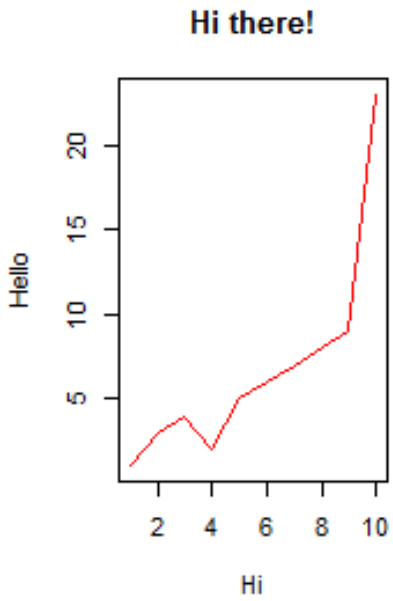
```
x <- c(1,3,4,2,5,6,7,8,9,23)
par(mfrow=c(2,3))          # divides the plot region as a 2 by 3 matrix for 6 plots
plot(x)          # point plot of x
```

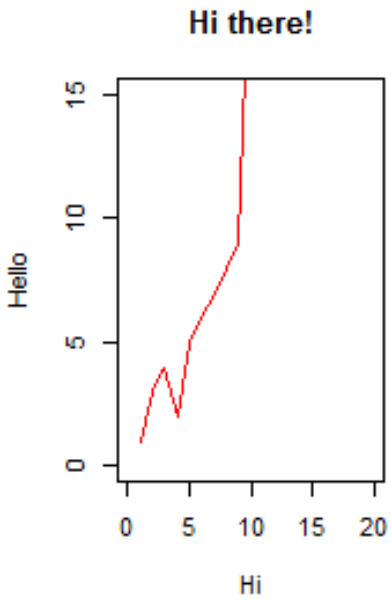
```
lines(x)          # adds a line joining the points to an existing point plot of x
plot(x, type="l")  # line plot of x
plot(x, type="l", col="red")      # plots the line with color red
plot(x, type="l", col="red", main="Hi there!", xlab="Hi", ylab="Hello")
                  # …adds a title ‘Hi there’ to the plot, adds x axis label ‘Hi’ and y
                  axis label ‘Hello’
plot(x, type="l", col="red", main="Hi there!", xlab="Hi", ylab="Hello",
+      xlim=c(0,20),ylim=c(0,15))     # specifies the limits of x and y axes
?par              # for all possible parameters
```


x
Index
2
4
6
8
10
5
10
15
20

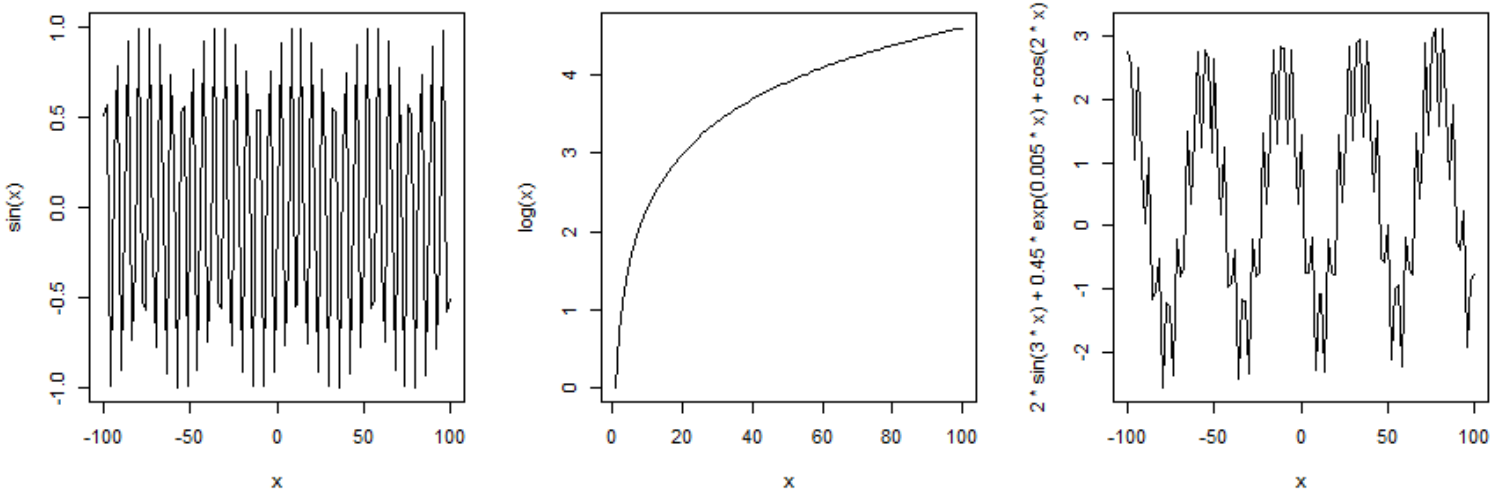

x
Index
2
4
6
8
10
5
10
15
20


x
Index
2
4
6
8
10
5
10
15
20


Hi there!
Hello
Hi
2
4
6
8
10
5
10
15
20

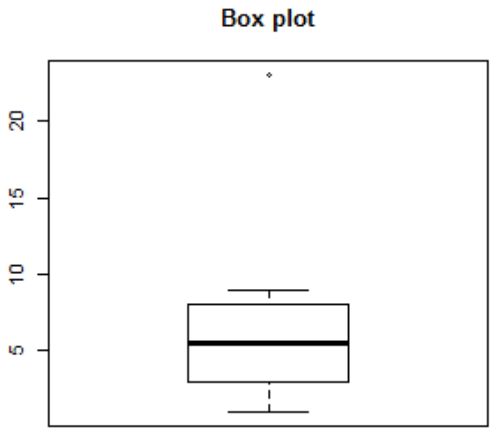

Hi there!
Hello
Hi
0
5
10
15
20
0
5
10
15

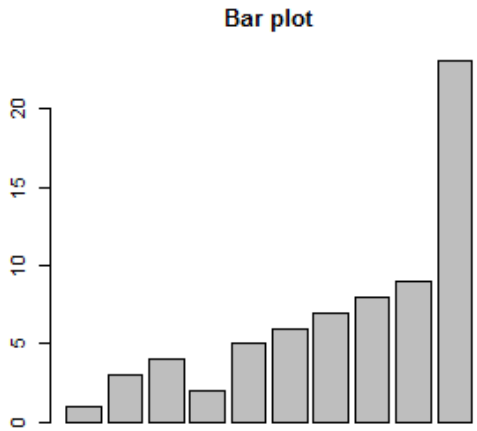
```
par(mfrow=c(1,3))
curve( sin(x), -100, 100 )          # draws curve of sin(x) with x values between -100 and 100
curve( log(x), 1, 100 )             # draws curve of log(x) with x values between 1 and 100
curve( 2*sin(3*x) + .45*exp(.005*x) + cos(2*x), -100, 100 )          #
```

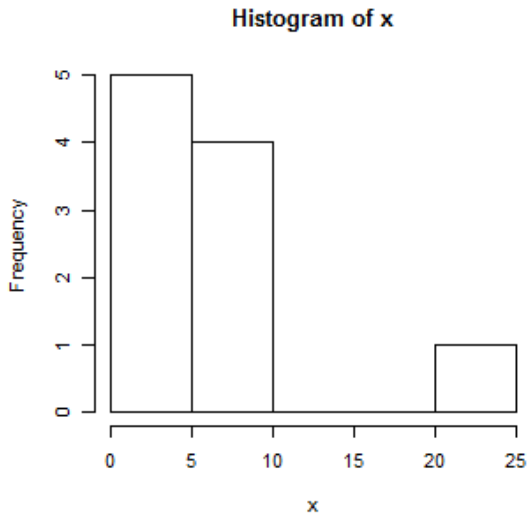



```
x <- c(1,3,4,2,5,6,7,8,9,23)
par(mfrow=c(1,3))
boxplot(x, main="Box plot")          # Draws box plot of x
```

```
barplot(x, main="Bar plot")      # Draws bar plot of x
hist(x)                          # Draws histogram of x
```





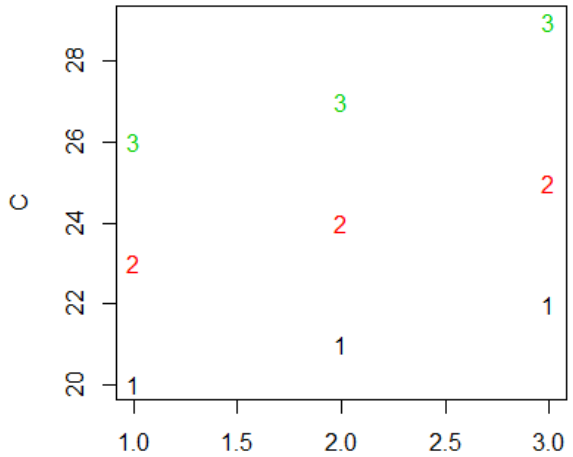


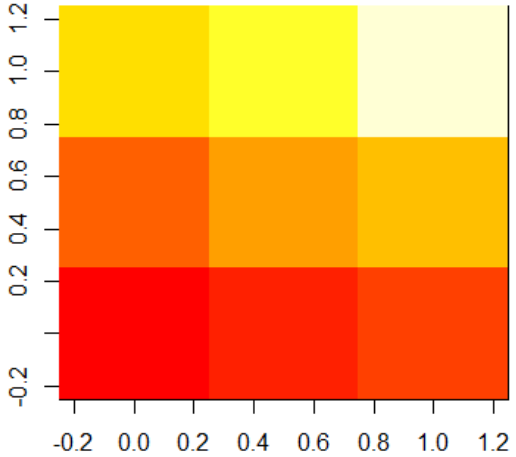

```
stem(x)                          # Produces stem-and-leaf plot of x
```

The decimal point is 1 digit(s) to the right of the |

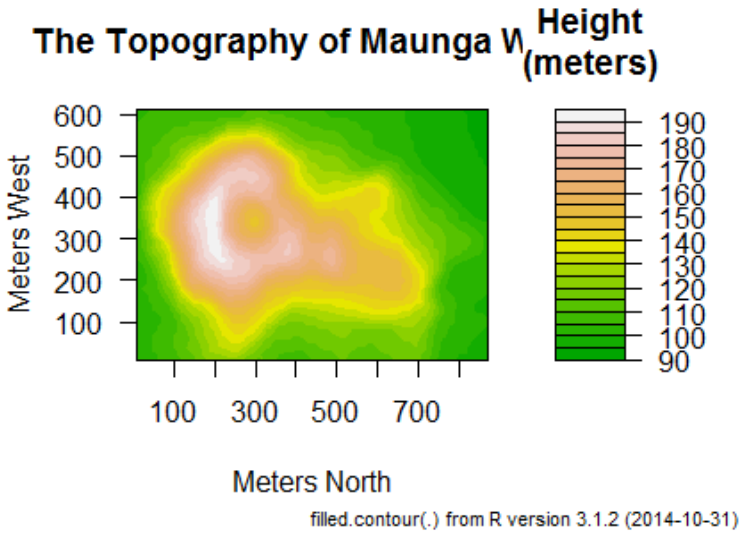
```
0 | 1234
0 | 56789
1 |
1 |
2 | 3
```

```
C <- matrix(c(20:27,29), byrow=F, nrow=3, ncol=3)
C
     [,1] [,2] [,3]
[1,]   20   23   26
[2,]   21   24   27
[3,]   22   25   29
par(mfrow=c(1,2))
matplot(C)              # produces a matrix plot of C
image(C)                # produces an image of C
```





```
x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
filled.contour(x, y, volcano, color = terrain.colors,
plot.title = title(main = "The Topography of Maunga Whau",
xlab = "Meters North", ylab = "Meters West"),
plot.axes = { axis(1, seq(100, 800, by = 100))
axis(2, seq(100, 600, by = 100)) },
key.title = title(main = "Height\n(meters)"),
key.axes = axis(4, seq(90, 190, by = 10))) # maybe also asp = 1
mtext(paste("filled.contour(.) from", R.version.string),
side = 1, line = 4, adj = 1, cex = .66)
```



Try at home using R:

1) Compute the sum of squares of all integers from 1 to 100.
2) Generate a sequence of 100 numbers between 1 and 10. Call the sequence x. Produce a line plot of x in reverse order.
3) Produce matrix plot of a matrix which has 10 rows and has as elements all the numbers divisible by 5 in between 1 and 200.
4) Compute the mean, median, standard deviation and 82^{nd} quantile of all the numbers in between 1 and 50 which are divisible by 2.25.

R loops: if, if-else, for

```
i <- 9
if(i > 2) j <- 4          # if i is less than 2, then j receives value 4
j
[1] 4

if(i >= 10) k <- 2 else k <- 4          # if i is greater than or equal to 8, then k receives 2, else k receives 4
k
[1] 4

if(k==4) l <- 10          # if k is equal to 4, then l receives 10
l
[1] 10

if( (i > 2) & (l < 11) ) m <- 15          # if i is greater than 2 and less than 11 (2<i<11), then m receives 15
m
[1] 15
```

```
if( (i > 10) | (j < 3) ) n <- 20 else n <- 0
```

if i is greater than 10 or j is less than 3, then n receives 20, else n receives 0

```
n
[1] 0

if(!n==1) p <- 25 else p <- 4
```

if n is not equal to 1, then p receives 25, else p receives 4

```
p
[1] 25

x1 <- c()
for(i in 1:10)
 x1[i] <- 2 + i
```

for each i in 1 to 10 (integers), i'th element of vector x1 is 2+i

```
x1
[1]  3  4  5  6  7  8  9 10 11 12
```

```
x2 <- x3 <- c()
for(i in 1:20)
{
 x2[i] <- i^2 + 2*log(i+1)
 x3[i] <- i^3 + exp(i+1)
}
x2
 [1]   2.386294   6.197225  11.772589  19.218876  28.583519  39.891820
 [7]  53.158883  68.394449  85.605170 104.795791 125.969813 149.129899
[13] 174.278115 201.416100 230.545177 261.666427 294.780744 329.888878
[19] 366.991465 406.089045
x3
 [1] 8.389056e+00 2.808554e+01 8.159815e+01 2.124132e+02 5.284288e+02
 [6] 1.312633e+03 3.323958e+03 8.615084e+03 2.275547e+04 6.087414e+04
[11] 1.640858e+05 4.441414e+05 1.204801e+06 3.271761e+06 8.889486e+06
[16] 2.415905e+07 6.566488e+07 1.784881e+08 4.851721e+08 1.318824e+09
```

Read and write files

```
getwd()                 # shows the current working directory
[1] "C:/Users/Sunny/Documents"
setwd("C:/Users/Sunny/Desktop")        # sets the current working directory to
                                       the user-chosen directory
list.files()            # list all files in the current working directory
x <- 1:100
write(x, "test.txt", ncolumns=1)       # write x in a file test.txt in one column
write(x, "test.txt", ncolumns=4)       # write x in a file test.txt in four columns
y <- matrix(1:100, nrow=20)
write(t(y), "test.txt")                # write matrix y in a file test.txt
write(t(y), "test.txt", sep=",")       # write matrix y in a file test.txt with comma
                                       separation
write(t(y), "test.txt", sep="\t")      # write matrix y in a file test.txt with tab
                                       separation
read.table("test.txt")                 # read test.txt as a data frame
  V1 V2 V3 V4  V5
1  1 21 41 61  81
2  2 22 42 62  82
```

```
3   3 23 43 63  83
4   4 24 44 64  84
5   5 25 45 65  85
6   6 26 46 66  86
7   7 27 47 67  87
8   8 28 48 68  88
9   9 29 49 69  89
10 10 30 50 70  90
11 11 31 51 71  91
12 12 32 52 72  92
13 13 33 53 73  93
14 14 34 54 74  94
15 15 35 55 75  95
16 16 36 56 76  96
17 17 37 57 77  97
18 18 38 58 78  98
19 19 39 59 79  99
20 20 40 60 80 100
rownames(y) <- letters[1:20]        # assign names for the rows of matrix y
colnames(y) <- LETTERS[1:5]         # assign names for the columns of matrix y
```

```
write(t(y), "test.txt")
write.table(y, "test.txt")
```

write matrix y in a file with rows and columns names

```
read.table("test.txt")
   A  B  C  D   E
a  1 21 41 61  81
b  2 22 42 62  82
c  3 23 43 63  83
d  4 24 44 64  84
e  5 25 45 65  85
f  6 26 46 66  86
g  7 27 47 67  87
h  8 28 48 68  88
i  9 29 49 69  89
j 10 30 50 70  90
k 11 31 51 71  91
l 12 32 52 72  92
m 13 33 53 73  93
n 14 34 54 74  94
o 15 35 55 75  95
```

```
p 16 36 56 76  96
q 17 37 57 77  97
r 18 38 58 78  98
s 19 39 59 79  99
t 20 40 60 80 100
write.table(y, "test.txt", quote=F)     # write matrix y in a file with unquoted rows
                                        and columns names

z <- matrix(c('Person','Familysize',1,2,3,4), byrow=T, nrow=3)
write(t(z), "test.txt", ncolumns=2)
read.table("test.txt")
      V1         V2
1 Person Familysize
2      1          2
3      3          4
read.table("test.txt",  header=T)           # read table identifying header names
  Person Familysize
1      1          2
2      3          4
```

```
y <- matrix(1:100, nrow=20)          # write in csv file
write.csv(y, "test.csv")
read.csv("test.csv")                 # read csv file
```

Save and load console image

```
save.image("R_image")
load("R_image")
```

Drawing random samples

```
x <- 1:12
sample(x)          # draws a random sample of size 12 from x without
                   replacement, or in other words, produces a random
                   permutation of the elements of x
[1]  8 10  6  9  1  5 11  7  4  3 12  2
sample(x, replace = TRUE)     # draws a random sample of size 12 from x with
                   replacement
[1]  5 10 12 10  3  7  3  9  8  2  8  9
```

```
sample(x, 5)        # draws a random sample of size 5 from x without replacement
[1]  3  2  9 12  7
sample(x, replace = TRUE)     # draws a random sample of size 5 from x with replacement
[1] 12  4 12  5  8 12  1  8 12  1  1  6
set.seed(5)         # sets the seed (for random number generation) to user-given value
```

Density, distribution function, quantum and random samples from distributions

Normal distribution:

```
dnorm(3)                    # evaluates at 3 the density function of standard normal distribution (mean=0, sd=1)
dnorm(3, mean=2, sd=3)      # evaluates at 3 the density function of normal distribution with mean=2 and sd=3
pnorm(3, mean=2, sd=3)      # evaluates at 3 the distribution  function of normal distribution with mean=2 and sd=3
```

```
qnorm(.56, mean=2, sd=3)# evaluates 56th percentile of normal distribution with
                                mean=2 and sd=3
rnorm(100, mean=2, sd=3)        # Generates 100 random samples from normal
                                distribution with mean=2 and sd=3
```

t distribution:

```
dt(3, 5)
pt(3, 5)
qt(.56, 5)
rt(100, 5)
```

Chi-squared distribution:

```
dchisq(3, 5)
pchisq(3, 5)
qchisq(.56, 5)
rchisq(100, 5)
```

F distribution:

```
df(3, 5, 4)
pf(3, 5, 4)
qf(.56, 5, 4)
rf(100, 5, 4)
```

Binomial distribution:

```
dbinom(4, 10, .3)
pbinom(4, 10, .3)
qbinom(.4, 10, .3)
rbinom(100, 10, .3)
```

Poisson distribution:

```
dpois(4, 3)
ppois(4, 3)
qpois(.4, 3)
rpois(100, 3)
```

Defining a function

Define an R function to compute $f(x) = 2\sin(x) - \log(x) + (1-x^3)^4$.

```
myfunc <- function(x)
{
   return( 2*sin(x) - log(x) + (1-x^3)^4 )
}

myfunc(45)
[1] 6.89495e+19
myfunc(.056)
[1] 3.993643
myfunc(1234.456)
[1] 1.252297e+37
```

Downloading, installing and loading an R package

```
.Library        # shows the location of the current R library in your system
[1] "C:/PROGRA~1/R/R-31~1.2/library"

# For installing a package within C drive, you may need administrator privilege.
#For that, right click on R icon and 'run as administrator'. Without administrtair
#privilege, you may choose to install in some other folder.

install.packages("tree")    # both downloads and installs the package in the
                            current R library (if without administrator privilege,
                            you will be asked to choose a folder where the
                            package will be installed). Also, choose any mirror
                            from the list of mirrors that will be shown

library(tree)       # loads the package tree (must be already installed)
```