**Function for Yong**

**Yong Lecture 1**

vector("logical", n) # Create a logical vector

vector("numeric", n) # Create a numeric vector

vector("character", n) # Create a character vector

logical(n) # Create a logical vector shortcut

numeric(n) # Create a numeric vector shortcut

character(n) # Create a character vector shortcut

vector("list", n) # Create a list with vectors

Ia.na() # Test for the presence of missing values.

is.null() # Test for the presence of missing vectors.

c(A = 1) # Associate a vector of names and elements of vector

list(A = 1:4) # Associate a vector of names and elements of list

names() # Manipulating Names of vector or list

x [ i ] # subsetting by specifying indices

x [ - i ] # subsetting by excluding indices

x [ c(TRUE, FALSE) ] # subsetting by logical selection

x [ c (var1,var2) ] # subsetting by column/variable names

x [ ! is.na (x) ] # Remove the NA values from a vector

x$a # Access a column/variable from list/dataframe (same as x[["a"]] )

A$a = b # Reassign values to subsets

A$a <- b # Reassign values to subsets

x[i] = list(b) # To replace elements of a list by b

is.na(x) # Test for NA (missing value )

is.nan(x) # Test for NaN (for not a number)

is.infinite(x) # Test for infinities

is.finite(x) # Test for finite numbers

x + y # Sum of x and y

x - y # Difference of x and y

x \* y # Product of x and y

x / y # Quotient of x and y

x ^ y # Exponentiation (raising to a power)

x %/% y # Integral division

x %% y # Remainder after integral division

dnorm, pnorm, qnorm # Normal distibution

dchisq, pchisq, qchisq # Chi-squared distribution

dt, pt, qt # t distribution

dF, pF, qF # F distribution

dbinom, pbinom, qbinom # Binomial distribution

dpois, ppois, qpois # Poisson disibution

min() # Return the minimum

max() # Return the maximum

range() # Returns a vector of minimum and maximum

sum() # Returns the sum of all the elements

prod() # Returns the product of all the elements

cumsum() # Returns the cumulative sum of all the elements

cumprod() # Returns the cumulative product of all the elements

cummax() # Returns the cumulative maximum of all the elements

cummin() # Returns the cumulative minimum of all the elements

pmin() # Return the minimum of multiple vectors

pmax() # Return the maximum of multiple vectors

x:y() # Create a sequence from x to y by 1

seq() # Create a custom sequence

rep() # Create a custom repeating sequence

< # Less than

<= # Less or equal

== # Equal

!= # Not equal

> # Great than

>= # Greater or equal

& # And (Good for logical vectors)

&& # first value that satisfy X & Y condition (use && in an If-condition)

| # Or

! # Not

any() # see if any the elements of a logical vector are true.

all() # see if all the elements of a logical vector are true.

c() # Combine

nchar() # Returns the lengths the character

substring() # Substrings can be extracted

paste() # Strings can be “glued together” by using paste

as.logical() # interpret as logic

as.numeric() # interpret as number

as.complex() # interpret as complex number

as.character() # interpret as character

**Yong Lecture 2a**

function (arglist) { # Simple function

body}

(function (arglist) { # Anonymous function (use only once without creation)

body})(arg)

for(variable in vector){ # for loop, used to calculate a range of different numbers

expression}

for(variable in vector) { # for loop start again from the expression 1 if condition is meet

expression1

if (condition)

next

expression2}

for(variable in vector) { # for loop stop at expression 1 if a condition is meet

expression1

if (condition)

break

expression2}

while (condition){ # repeat a calculation until a particular condition is false

expression }

ask tutor how to write?

repeat { # It simply repeats an expression indefinitely.

if( ! condition ) break # To avoid an infinite loop, expression must contain a break expression statement to exit the loop.

}

Ask tutuor how to write?

lapply () # applies a given function to a list and returns the computed values in a list.

sapply () # applies a given function to a list and returns the computed values in a vector

function (arglist) { # vectorised function

body}

ask tutor for example

function (arglist) { # un-vectorised function

body} ask tutor for example

**Yong Lecture 3**

matrix(x , nrow =3 , ncol = 2) # create a 3 \*2 matrix with vector x

matrix(byrow=TRUE) # vector goes long row instead of going down column

nrow() # gives number of row of matrix

ncol() # gives number of column of matrix

dim() # gives dimension of matrix

rbind() # bind vectors as rows

cbind() # bind vectors as column

dimnames() # give names for matrix rows and then columns

dimnames() # Extract row and column names

rownames() # Extract row names

colnames() # Extract column names

apply(matrix, 1, fun) # apply function to matrix row

apply(matrix, 2, fun) # apply function to matrix column

sweep() # subtract out those summaries to obtain residual values

ask tutuor how to write

outer(1:5,1:5,f) # call a function on the outer values, default is multiplication

A %\*% B # matrix multiplication

x[row(x) == col(x)] = 2 # assign new value to matrix diagonals, we can customise

t() # Transpose of matrix

solve() # solves equation a %\*% x = b for x, where b is a vector or matrix.

**Yong Lecture 4**

plot.new() # start a new bottom right hand window for custom plots

plot(x,y) # Create a scatter plot using 2 equal length vectors

plot(main=””) # Add a title

plot(sub = "a”) # Add a sub-title

plot(xlab =”a”, ylab=”b”) # Add x axis and y axis labels

plot(xlab =””, ylab=””) # Delete x axis and y axis labels

plot(pch= x) # Controls the plotting symbol(s) used by plot

plot(pch=0) # No symbol

plot(col=) # Custom colours through recycling or specific vectors of colour

colors() # obtained A complete list of colour names within expression

plot(type=) # produce other types of plot by describing their content

plot(lty=, lwd=) # Custom line type and line width

plot(xlim=,ylim=) # custom x and y limits with vector of 2

points() # draw points

lines() # draw connected line segments

abline() # add a straight line to a plot

segments() # draw disconnected line segments

arrows() # add arrows to a plot

rect() # add rectangles to a plot

polygon() # add polygons to a plot ask tutor to show how polygon works

text() # add text to a plot

**Yong Lecture 4**

boxplot() # draw a boxplot

pie() # draw a pie chart

barplot() # draw a bar plot

barplot(horiz = TRUE) # draw a horizontal bar plot

dotchart() # Create a dot chart

plot.new() # To obtain a clear area (new piece of paper) to plot

plot.window(xlim=, ylim=) # To set up the coordinates in the plotting area

plot.window(xaxs="i") # internal x axis

plot.window(yaxs="i") # internal y axis

plot.window(asp=1) # equal scale axes

lines() # Draw lines (x y)

points() # Draw points (x y)

axis() # Draw axis(1,2,3,4)

box() # Draw the graph boarder

title(main,xlab ylab) # Write title and axis labels

par(mai=c()) # Set the margin sizes in inches.

par(mar=c()) # Set the margin sizes in lines of text.

par(pin=c()) # Set the plot width and height in inches

par(mfrow=c()) # number of plot within a plot (layout)

layout() #Create a custom layout format

layout(matrix) #Create a matrix for the layout

layout(height=) # Height of each row

layout(width=) #width of each column

layout(x=lcm()) #Custom scale with cm or just number for a proportion

mtext() # Draws text in the margins of a plot.

strwidth() # compute the x dimensions of the bounding box of the text.

strheight() # compute the y dimensions of the bounding box of the text.

**Yong Slide 7**

factor() # Create a categorical variable

factor(levels=) # Set levels for a categorical variable

ordered() # Create an ordered categorical variable

is.factor() # To tell whether a value is a factor

table() # Create a table or cross table for 1 or more variables

tapply() # applies a given function to subsets

cut() # cut continuous numeric variable into intervals of ordered category

data.frame() # Create a data frame

transform() # Adding Derived Variables to Data Frames

subset() # Subset data

sort() # sort dataframe in order of a variable

order() # reorder data frames in order of a variable

**Functions for Mehdi**

**Mehdi Lecture 2**

readLines() # Read a text file and generate a character vector.

read.table() # Read a delimited text file and generate a data frame.

read.csv() # Read a CSV text file and generate a data frame.

read.fwf() # Read a fixed-width file and generate a data frame.

head() # Show just the first few values in a data structure.

**Mehdi Lecture 3**

grep() # Find the lines that contain data.

data [ line ] # Extract the first line of data.

strsplit() # Break up that line into pieces.

strspliting() # Break up that line into pieces via vector.

as.numeric() # Convert the second piece into a number.

**Mehdi Lecture 4**

sub() # Search text for a pattern (first location).

gsub() # Search text for a pattern.

substr() # Extract a substring from text.

substring() # Extract a substring from text via vector.

nchar() # Count the number of characters in text.

**Mehdi Lecture 5**

regexpr() # Search text for pattern location (first).

gregexpr() # Search all pattern location.

regmatches() # Extract or replace matched substring.

**Mehdi Lecture 7**

print() # Print text output.

debug() # Mark a function for debugging.

undebug() # Remove debugging for a function.

ls() # Shows us which symbols are visible within the function.

vector() # Create an empty vector data structure with a known length.

lapply() # Call a function on each component of a list and return the results.

as.list() # Coerce an object to a list.

**Mehdi lecture 8**

unlist() # Remove the list structure [[ ]] and give a vector structure [ ].

rbind() # Combine two or more vectors as rows of a matrix.

do.call() # Call a named function, specifying the arguments as a list.

dim() # Report the number of rows and columns in a matrix

.

**Mehdi lecture 9 & 10**

is.na() # Determine which values in a data are NA (result logical vector).

which() # Determine which values in a logical vector are T (result numeric vector).

which(!) # Determine which values in a logical vector are F (result numeric vector).

data.frame() # Create a data frame.

grepl() # Search text for a pattern; similar to grep() but the result is a logical vector.

ifelse() # Select values from two vectors based on values in a logical vector; when

the logical vector is TRUE, take a value from the first vector, otherwise

take a value from the second vector.

**Mehdi lecture 11 &12**

list.files() # Generate a character vector of the names of files in a directory.

file.path() # Generate a file path from path components.

rep() # Generate a sequence that has a pattern.

as.character() # Convert values to character values.

sprintf() # Control how a value is converted to a character value. (with C language).

paste() # Combine two or more character values.

as.Date() # Create a date object.

?strptime # Explanation of the special format codes used in date formats.

Sys.Date() # Returns today's date as a \date".

seq() # Generate a sequence from a start value to an end value.

format() # Control how a value is converted to a character value.

**Mehdi Lecture 13**

split() # Break a data frame into pieces.

tapply() # Split a vector into a list of sub vectors and call a function on each.

aggregate() # Call a function on each sub-vector, gives results as data frame.

table() # Generate a table of counts.

**Mehdi Lecture 14**

all() # are all logical values in the arguments TRUE ?

any() # are any logical values in the arguments TRUE ?

warnings() # Show most recent warning messages.

options("warn") # Control the behaviour of R warnings.

traceback() # Show the call stack after an error.

recover() # Navigate within the call stack.

trace() # Insert debugging code within a function.

untrace() # Remove code inserted by trace().

debug() # Mark a function for debugging.

undebug() # Remove debugging for a function.

**Mehdi Lecture 15**

warnings() # Shows the warning messages

options() # Custom R option from default to custom

options(warning=1) #

options(warning=2) # stop as soon as warning occur, so we can trace it back

traceback() # Shows all the functions before and until the error

recover() # Within the debugger, move within the call stack

ls() #

q # quit from debug mode

trace() # Insert debugging code within a function.

untrace() # Remove code inserted by trace().

debug() # Mark a function for debugging.

undebug() # Remove debugging for a function.

quote() # returns an unevaluated expression.

**Mahdi Lecture 16 17**

merge() # Combine two data frames side-by-side by matching rows.

Reduce() # Call a function recursively, just like a cum sum

mapply() # Call a function for each component of x and each component of y

lapply() # only calls a function for each component of x).

library(reshape2)

melt() # Convert a data frame to long form.

dcast() # Convert a long form data frame to wide form.

**Mahdi Lecture 18**

library(lattice)

xyplot() #Function for plotting using x and y coordinates

**Meta characters**

\* zero or more [0-9] digits

+ one or more | or

? zero or one ^ at the start

. any character $ at the end

[a-zA-Z] letters

// literal meaning

Plus brackets, ( and ), to make sub-patterns.

Examples:

sub(" **(^[a-zA-Z ]+?)** ->.+"," **\\1** ",splitData)

"[0-9]" one digit

"[0-9]+" one or more digits

"^[0-9]+" one or more digits at the start

"^[0-9]|[0-9]$" a digit at the start or a digit at the end

(i) A function is **vectorised** if the result of applying it to a vector is identical to the result of applying the function to the separate elements and combining the resultS into a vector.

(ii) **Recycling rule** is when we applying the function to vectors of different lengths, elements of the shorter length are recycled to the length of the longest vector, so we get 2 equal length vector for the analysis.

Plot(las=0) # axis labels always parallel to the axis [default],

Plot(las=1) # axis labels always horizontal,

Plot(las=2) # axis labels always perpendicular to the axis,

Plot(las=3) # axis labels always vertical.

substr(x, start, stop)

**par()**

par(mar = c(5.1, 4.1, 0.1, 2.1)) # margins

par(oma = c(0, 0, 4, 0)) # outer margins

par(mai=c()) # Set the margin sizes in inches.

par(mar=c()) # Set the margin sizes in lines of text.

par(pin=c()) # Set the plot width and height in inches

par(mfrow=c()) # number of plot within a plot (layout)

mtext("Male", adj = 1) # draw at right margin

mtext("Male", adj = 0) # draw at left margin

> layout(rbind(c(0, 4, 4, 0),

c(0, 2, 0, 0),

c(0, 1, 3, 0),

c(0, 0, 0, 0)),

height = c(lcm(2), lcm(2), 1, lcm(2)),

width = c(lcm(2), 1, lcm(2), lcm(1)))

> layout.show(4)

> box("outer", lty = "11")

> plot.new()

> plot.window(xlim = range(x),

ylim = range(y))

> points(x, y)

> axis(1)

> axis(2)

> box()

> title(main = "A Simple Plot", xlab = "x", ylab = "y")