# R tutorial

Dr. K. Asokan

College of Engineering, Trivandrum

# What is R ?

- R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

# Entering R environment

- In linux, start a new terminal and at the command prompt enter
  $: R
  This will put you in the R environment with a '$>$' prompt
  $>$
- cntrl +l to clear screen
- To exit the environment type
  $>$ q()

# Variables

- A variable provides us with named storage that our programs can manipulate
- A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.
- Which of the following are valid variable names ?
  1. `var_name2.`
  2. `var_name%`
  3. `2var_name`
  4. `.2var_name`
  5. `_var_name`

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a data type of the same variable again and again when using it in a program.

# Basic arithmatics and functions I

Basic arithmatics

| Operator | Description | Example |
|---|---|---|
| x + y | y added to x | 2 + 3 = 5 |
| x -y | y subtracted from x | 8 - 2 = 6 |
| x * y | x multiplied by y | 3 * 2 = 6 |
| x / y | x divided by y | 10 / 5 = 2 |
| x ^y (or x ** y) | x raised to the power y | 2 ^ 5 = 32 |
| x %% y | remainder of x divided by y (x mod y) | 7 %% 3 = 1 |
| x % /% y | x divided by y but rounded down (integer divide) | 7 %/% 3 = 2 |

Basic numeric functions

# Basic arithmatics and functions II

| Functions | Descriptions |
|---|---|
| abs(x) | absolute value |
| sqrt(x) | square root |
| ceiling(x) | ceiling(3.475) is 4 (integer on the right) |
| floor(x) | floor(3.475) is 3 (integer on the left) |
| trunc(x) | trunc(5.99) is 5 (remove decimal) |
| round(x, digits=n) | round(3.475, digits=2) is 3.48 |
| cos(x), sin(x), tan(x) | trigonometric functions (x in radians) |
| acos(x), cosh(x), acosh(x) | inverse trig functions |
| log(x) | natural logarithm |
| log10(x) | common logarithm |
| exp(x) | $e^x$ |

Experiment with negative numbers in floor(x),trunc(x),ceiling(x) Execute the following

```
> pi
> cos(3.14)
> cos(pi)
> acos(-1)
> tan(pi/2)
> sqrt(8)
> 8^(1/2)
> 8^(1/3)
> (-8)^(1/3)
> (-8+0i)^(1/3)
```

# Data types

Most important data types include

| Data Type | Example |
|---|---|
| Logical | TRUE, FALSE |
| Numeric | 12.3, 5, 999 |
| Integer | 2L, 34L, 0L |
| Complex | 3 + 2i |
| Character | 'a' , '"good", "TRUE", '23.4' |

# Assignment

The variables can be assigned values using leftward, rightward and equal to operator. For example execute the following commands

```
> x=1.0
>y <- 2+3i
>"linux"->z
```

The variables x ,y and z are the variables. In the last two cases the arrow must point to the variable direction

To see the value of the variable use the `print()` command

```
> print(x)
[1] 1
```

You can get a list of the defined variables using `ls()` and remove any variable by `rm()`;eg. `rm(x)` removes x

# Data Objects in R

The most important data objects in R are

1. Vectors
2. Matrices
3. Data Frames
4. Lists
5. Factors

# Vectors

Vectors are ordered items of the same type. Most common are vectors of numbers or vectors of strings.
Vectors are created by the command c().

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

# Other ways of entering vectors

```
x<-2:10  # vector of numbers from 2 to 10
y<- seq(from = 2, by = -0.1, length.out = 4)
z<-rep(1:4, 2)
x<-rep(1:4, each=2)
y<-rep(1:4, each = 2, times = 3)
z<-rep(1:4, each = 2, len = 9)
w<-rep("x",4)
a<-c(rep("x",4),rep("y",3))
b<-c(rep(1:4,2),10,11)
c<-x=rep(c(1,4,2),2) # repeating a whole vector
```

## the scan() function

The command `scan()` reads the input from the command line one by one. A blank line stops the scan

```
>x=scan()    #reads keyboard input into vector x
```

By default the input is numeric. If we want to read in a vector of strings add the option `what="character"` or initialise by an empty character: `what=""` (empty quotes). No need to quote the individual inputs.

```
>x=scan(what="")    #reads a vector x of strings
```

We can also read data from file in the current directory. Data may be a single column or row

```
>x=scan(file = "data.txt")
>x=scan(file = "data.txt",what="")
```

If the file is not in the current directory we can give the full path to the file as argument to `file=` option

```
>x=scan(file = "/home/asok/RLab/data.txt",what="character")
```

If the data is a single line of comma separated items use `sep=` option (default is space)

```
>x=scan(file = "data.txt",what="", sep=",")
```

If the first few lines of the file are to be skipped use `skip=` option;

```
>x=scan(file = "data.txt",skip=2)   # skips 2 lines (default 0)
```

type `help(scan)` for more options

# Extracting data from a vector

If x is a vector x[i] is the i-th element from the vector. eg. x[2] is the second element.
Execute the following

> x=c(10,20,30,40,50)
> x[1]*x[3]
> x[2]+x[4]

etc Other options

| | |
|---|---|
| how many elements? | length(x) |
| ith element | x[2] (i = 2) |
| all but ith element | x[-2] (i = 2) |
| first k elements | x[1:5] (k = 5) |
| last k elements | x[(length(x)-5):length(x)] (k = 5) |
| specific elements. | x[c(1,3,5)] (First, 3rd and 5th) |
| all greater than some value x | [x>3] (the value is 3) |
| maximum of x | max(x) |
| which indices have x=2 | which(x==2) |

The output of any of this can be saved as a new vector.

Try all these on the vector x created by
> x=rep(1:5, 3)

# Naming elements of a vector

The elements of a vector can be given names using the names() function. The argument of the function is a vector. The names list is usually a vector of strings.

```
> x=c(10,20,30)
> print(x)
> names(x)=c("id1","id2","id3")
> print(x)
```

Now we can refer to the second element as either x[2] or x["id2"]

```
> print(x[2])
> print(x["id2"])
```

# Operations on numeric vectors I

Suppose x and y are numeric vectors of same length then

x*y, x+y,x-y,x/y does the respective operations element wise

If x and y are of incompatible length the elements of the shorter vector are repeated before applying the operation

If x is a vector and a is a number
x*a multiplies each element of x with the number 'a' successively
Similarly for x+a,x-a,x/a, x^a etc.
Applying a numeric function, such as sin(), on a vector executes it on all the elements of the vector. Execute the following commands >x=rep(1:5,5)

```
>y=rep(11:15,5)
>length(x)
>length(y)
>x+y
>x-y
>x*y
>x/y
>x+2
>(x+2)/3
>x^2
>sin(x)
>x=c(1,2,3)
```

# Operations on numeric vectors II

```
> y=c(1,2,3,4,5)
> x+y
```

If x is a numeric vector The sum(x) gives the sum of all the elements of x and mean(x) gives the mean of the elements of x

Compare the outputs of the following

```
> x=rep(1:5,5)
> y=rep(11:15,5)
> sum(x)
> sum(y)
> sum(x+y)
> mean(x)
> mean(y)
> (x+y)/2
```

Can you see why sum(x+y)= sum(x)+sum(y)

# Problems

- A family consists of two members A and B. The monthly cell phone bills of A and B in (in Rs.) for one year is as follows

  A: 153 162 173 184 178 140 182 190 144 193 174 180
  B: 220 221 240 271 211 236 240 271 211 275 282 231 Enter this data as two vectors in R.
  Using appropriate R functions or operations calculate the following.
  (1) The total bill and average bill of the family for each month
  (2) The yearly total and average bill of A and B

- Write a one-line command in R to find the sum of the squares of the first 100 natural numbers.

- Let the vector X has the following elements
  1, 8, 2, 6, 3, 8, 5, 5, 5, 5
  Let $X_i$ denote the $i$-th element of x. Use R to compute the following.
  - $(X_3 + X_4 + \cdots X_8)/10$
  - $(X_3^2 + X_4^2 + \cdots X_8^2)/10$
  - $\log_{10} X_i$ for each $i$
  - Find $(X_i - 4.4)/3.4$ for each $i$
  - Find $\sin(X_1^2) + \sin(X_2^2) + \cdots + \sin(X_{10}^2)$

# Entering a matrix I

The basic syntax for creating a matrix in R is

matrix(data, nrow, ncol, byrow)

where

- data is the input vector which becomes the data elements of the matrix
- nrow is the number of rows to be created.
- ncol is the number of columns to be created.
- byrow is a logical clue. If TRUE then the input vector elements are arranged by row.

If data has the right number of elements one of nrow or ncol is enough Execute the following and check the resulting matrices

```
> x=c(3:14)
> M= matrix(x, nrow = 4)
> M = matrix(c(3:14), nrow = 4, byrow = TRUE)
> M= matrix(c(3:14), ncol = 4, byrow = TRUE)
> N=matrix(1:12,3,4,TRUE)
```

The last function is in an abbreviated form which should be entered in the order matrix(data, nrow, ncol, byrow). 1:12 is an abbreviation for the vector rep(1:12) or c(1:12)

M[i,j] represents the element in the i-th row and j-th column of M
Thus M[2,3] is the element in the 2-nd row and 3-rd column of M

# Entering a matrix II

M[i, ] represents the ith row & M[, j] the j-th column, both vectors.
Similar to vectors, you can add names for the rows and the columns of a matrix using the commands rownames() and colnames().

Each name is a vector of strings.

```
> rownames(M) = c("row1", "row2", "row3", "row4")
> colnames(M) = c("col1", "col2", "col3")
> M
```

The commands rowSums() gives the sum of the rows as a vector.
Similarly colSums() gives the sum of the columns

# Augmenting a matrix

- If M is a matrix and x and y are vectors of suitable length, the command rbind(M,x,y) adds to the matrix M, the rows x and y in that order.
- We can also combine matrices M1, M2, etc of suitable sizes by rbind(M1,M2,M3) etc.
- cbind(M,x,y) does the same thing to columns.

Experiment with the following commands

```
> M= matrix(c(3:14), ncol = 4, byrow = TRUE)
> x=rowSums(M)
> y=colSums(M)
> M1=cbind(M,x)
> M2=rbind(y,M)
> N= matrix(c(3:14), ncol = 4, byrow = TRUE)
> rbind(M,N)
```

# Matrix operations

If M and N are matrices M+N, M-N, M*N, M/N does the respective operations element wise, provided the matrices are of the same size.
If M is a matrix and a is a number M+a, M-a M*a, M/a, M^a does the respective operation with a on each element of M

```
> M= matrix(c(3:14), ncol = 4, byrow = TRUE)
> N=M/2
> M+N
> M-N
> M*N
> M/N
> (M/N)-2
> M^2
```

Use M %*% N for ordinary matrix multiplication of conformable matrices or vectors and t(M) for transpose of M. Other matrix operations

| | |
|---|---|
| t(A) | Transpose |
| diag(x)(x vector) | Creates diagonal matrix with x along diagonal |
| diag(A)(A vector) | returns a vector of diagonal entries of A |
| diag(k) (k integer) | creates a k x k identity matrix. |
| solve(A) | Inverse of A where A is a square matrix. |
| solve(A, b) | Returns vector x in the equation b = Ax |
| eigen(A) | Returns a list of eigen values and igen vectors |
| svd(A) | Singular value decomposition of A (as a list). |
| qr(A) | QR decomposition of A (as a list). |

# Lists I

Lists are the R objects which contain elements of different types like âĹŠ numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using list() function.

Create a list containing strings, numbers, vectors and a logical values.

```
> listdata = list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
> print(listdata)
```

Each item within the list can be accessed by its index

```
> listdata[2]
> listdata[3]
```

The list elements can be given names and they can be accessed using these names.

```
> listdata = list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
list("green",12.3))
> listdata
> names(listdata) <- c("Months", "Matrix", "InnerList")
```

Each item within the list can be accessed by its name or index.

```
> listdata$Months
> listdata[1]
> listdata$Matrix
> listdata[2]
```

The scan() function can be used to read a simple list of vectors from a file.

Suppose we have a text file 'data.txt' containing the following

```
textid, name and age of candidates
10    Kumar      12/11/1971
11    Kraksh     01/06/1970
12    Pradeep    26/03/1972
```

# Lists II

```
>newlist <- scan("data.txt", what=list(id=0,name="",date=""),skip=1)
>newlist
```

The whole data is read as a list. First column is read as a numeric vector and assigned the name 'id' within the list and can be accessed as newlist$id. Similarly for other columns.

```
>newlist <- scan("data.txt", what=list(id=0,name="",date=""),skip=1)
>newlist$date
```

In the above, date is stored just as a string, which is not good. For manipulating dates it should be stored in the correct format.

In R, dates are represented as the number of days since 1970-01-01, with negative values for earlier dates

You can use the as.Date( ) function to convert character data to dates. The format is as.Date(x, "format")

```
> dob <- as.Date(newlist$date, "%d/%m/%Y")
> dob # a vector of dates
```

If the dates are in correct format we can do various date operations

```
> dob[3]-dob[2]
> as.numeric(dob[3]-dob[2]) # no of days
```

# Data frames I

- A data frame is like a matrix in that it represents a rectangular array of data, but each column in a data frame can be of a different mode, allowing numbers, character strings and logical values to coincide in a single object in their original forms.
- Data frames are similar to database tables. Each row of a data frame represents an observation; the elements in a given row represent information about that observation. Each column, taken as a whole, has all the information about a particular variable for the data set.
- Important;
  1. The column names should be non-empty.
  2. Each column should contain same number of data items.

Data frames are created by data.frame() function. The arguments are vectors, each with an assigned name.

```
> emp.data <- data.frame( id = c (1:5),
  name = c("asok","kumar","syam","prakash","pradeep"),
  salary = c(623.3,515.2,611.0,729.0,843.25),
  entrydate = as.Date(c("2012-01-01", "2013-09-23",
    "2014-11-15", "2014-05-11", "2015-03-27")))
> emp.data
```

The structure of the data frame can be seen by using str() function.

```
> str(emp.data)
```

# Data frames II

The statistical summary and nature of the data can be obtained by applying summary() function.

```
> summary(emp.data)
```

emp.data$salary or emp.data[,3] gives 3rd column. emp.data[3] gives the same in column format

```
> emp.data[3]
> emp.data$salary
>emp.data[,3]
```

emp.data[1,] extracts the first row.

```
> emp.data[1,] #  extract the first row
> emp.data[1:2,] # extract the first 2 rows, returns a data frame
```

To add a new column just add the column vector using a new column name.

```
>emp.data$dept <- c("IT","Operations","IT","HR","Finance")
>emp.data
```

To add new rows create new data frame using the same name for columns and use rbin() function

```
# Create the second data frame
>emp.newdata <- data.frame( id = c (6:8),
   name = c("Rasmi","Pranab","Tusar"),
   salary = c(578.0,722.5,632.8),
   entrydate = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
```

# Data frames III

```
  dept = c("IT","Operations","Fianance"))
# Bind the two data frames.
>emp.data <- rbind(emp.data,emp.newdata)
```

# Creating data frame from external files I

If we have a data in a text file in free format with columns separated by space we can use read.table() function for reading the data.
Suppose we have a text file 'data.txt' containing the following information

```
# textid, name and age of candidates
10   Kumar     12/11/1971
11   Kraksh    01/06/1970
12   Pradeep   26/03/1972
```

This can be read as a dataframe using

```
>mydf=read.table("data.txt")
>mydf
```

We see that the first line which begins with # is treated as a comment and hence ignored. The columns are automatically given names V1, V2, etc. If we want to skip a few lines at the beginning (which are not commented), use the skip option

```
>mydf=read.table("data.txt",skip=2)   # skips the first 2 lines
```

If the data already contains names for columns, these can be read as names for the columns in the database using header=TRUE.

```
>mydf=read.table("data.txt",header=TRUE)
    # the first row are names for columns
```

# Creating data frame from external files II

If the data file does not have a header, but we want to supply names for the columns use the option col.names as a list.

```
>mydf=read.table("data.txt",col.names=c("ID", "Name","DoB"))
    # the first row are names for columns
```

- Use read.csv() to read file with data where columns are separated by comma.
- Files containing fixed width records can be read using read.fwf(). The width of the columns have to be supplied as a vector through the width= option. eg.

  ```
  >mydf=read.fwf("data.txt",width=c(4,10,10))
  ```

# Input/Output redirection I

- All the commands in an R session can be stored in a file, say 'rsession.R' (the extension .R is preferable, not necessary), and executed all at once from R shell using the source() function.

  `>source("rsession.R")`

  This is similar to a normal C program session.
  Useful for repeating the same analysis over and again.

- We can redirect all outputs of an R session to a file instead of the terminal using the sink() command.

  ```
  >sink("output.txt",split=TRUE)
  >sink()
  ```

  split=TRUE shows the output on the terminal also.
  append=TRUE appends to the file instead of overwriting. The last sink() gets the output back on terminal only.

- print()
  A generic function for printing a single object to the terminal.

  `>print(x) # prints x`

  print() prints with indexes and quotes for strings.
  quote=FALSE turns off quotes.
  digits= specifies the number of significant digits to be printed.

# Input/Output redirection II

- cat()-A more useful wrapper around print()
  It combines several objects and then prints.
  Prints strings without quotes and vectors without index.
  Most useful for printing one line of text along with a vector

  ```
  >x=c(1:5)
  >cat("the vector x is", x,"\n")
  >cat("the vector y is", format(y,digits=3),"\n")
  ```

  The "\n" at the end starts a new line. Note the use of format()

cat() can also write to a file with the file= option. Use the append=TRUE option to append to the end of the file and sep="," option to separate the objects with coma.

```
>x=c(1:5)
>cat("the vector x is", x,"\n",file="out.dat",append=TRUE)
```

To write to the same file several times it is better to open a connection to the file and write the output to the connection:

```
con <- file("analysisReport.out", "a")
cat(data, file=con)    # 'data' is a vector
cat(results, file=con)    # 'results' is a vector
close(con)
```

The "a" signifies that connection is opened for appending in text mode (as against binary mode). In fact the above file command is a short form of
`file(description="analysisReport.out", open="a")`

# Input/Output redirection III

cat() is not useful for printing matrices, since it will combine its columns into a single vector before printing.

To print a vector like a table or matrix use write(). However, it must have a file= argument. file="" is terminal

```
>write(x,file="file.out",ncolumns=3,sep=",")
  # writes in 3 columns,comma separated.
```

For printing matrices and data frames use write.table()

```
> write.table(x,file="output.txt")
  # 'x' is a matrix or data frame
```

OPTIONS
append=TRUE (append to the file, not overwrite)
sep="\t" (separate columns by a tab)
row.names=FALSE (do not write row names)
row.names=c("name1","name2",...) (custom name for rows)
  (similar options for col.names)
quote=FALSE (do not quote character strings)

# Pie Charts I

The data consists of a single vector. Most common plots are

1. Pie Chart
2. Bar Charts
3. Histogram

Pie charts are created with the function pie(x, labels=) where x is a non-negative numeric vector indicating the area of each slice and labels= notes a character vector of names for the slices.

```
# Simple Pie Chart
> slices <- c(10, 12,4, 16, 8)
> lbls <- c("US", "UK", "Australia", "Germany", "France")
> pie(slices, labels = lbls, main="Pie Chart of Countries")
```

The main= option sets the label for the chart
An enhanced plot with custom colors and more information

```
# More options
> slices <- c(10, 12,4)
> lbls <- c("India", "China", "US")
> pct <- round(slices/sum(slices)*100)
> lbls <- paste(lbls, pct) # add percents to labels
> lbls <- paste(lbls,"%",sep="") # add % to labels
> pie(slices,labels = lbls, col=c("blue","red","black"), main="Pie Chart of
Countries")
```

# Pie Charts II

The col= option sets the colors of the divisions
check help(pie) for more options

# Bar charts I

Create barplots with the barplot(height) function, where height is a vector or matrix.

- If height is a vector, the values determine the heights of the bars in the plot.
- If height is a matrix and the option beside=FALSE then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked sub bars
- If height is a matrix and beside=TRUE, then the values in each column are juxtaposed rather than stacked.

A simple bar plot

```
# Simple Bar Plot
>counts <- c(100,80,50)
>barplot(counts, main="Population",
  xlab="Countries",names.arg = c("China","India","US"))
```

xlab= sets label for x-axis
names.arg= sets labels for bars
Adding the option horiz=TRUE would plot the bars horizontally.
Counts is a matrix $\begin{pmatrix} 100 & 80 & 50 \\ 60 & 42 & 28 \end{pmatrix}$.
Each column is plotted as two stacked sub bars

```
# Bar plot of a matrix
>counts <- matrix(c(100,60,80,42,50,28),nrow = 2)
>barplot(counts, main="Total Population and female population",
```

# Bar charts II

xlab="Countries",names.arg = c("China","India","US"))

Adding the option beside=TRUE would position the sub bars side by side.

# Histograms I

Create histograms with the function hist(x) where x is a numeric vector of values to be plotted.

```
>x=rnorm(100)    # 100 N(0,1) random numbers
>hist(x)    # draws the histograms
```

OPTIONS

probability=TRUE plots probability densities instead of frequencies

breaks=10 Number of bins 10; (suggestion only)

# Plotting two-dimensional data I

If x and y are numeric vectors of same length plot(x,y) plots a scatter graph of x versus y

```
> x=seq(from=-3,to=3,by=0.1)   # set up x vector
>y=dnorm(x)   # y is a vector of N(0,1) values of points in x
>plot(x,y)
```

OPTIONS

Option type= can take any of the following values

| | |
|---|---|
| p | points |
| l | lines |
| o | overplotted points and lines |
| b, c | points (empty if "c") joined by lines |
| s, S | stair steps |
| h | histogram-like vertical lines |
| n | does not produce any points or lines |

If type="l" we can set the line type by lty= option, line width by lwd= option and line color by col= option. All of these accepts integer values. col also accepts values like "blue", "green" etc.

```
> x=seq(from=-3,to=3,by=0.1)
>y=dnorm(x)
>plot(x,y,type="l",lty=3,lwd=2,col="blue")
```

If type="p" we can set the point character type by pch= option which accepts an integer from 1 to 25 or a custom character like pch="t".

# Plotting two-dimensional data II

The size of character can be set as a character expansion factor by cex= option and line color by col= option.

```
> x=seq(from=-3,to=3,by=0.1)
>y=dnorm(x)
>plot(x,y,type="p",pch=3,cex=1.5,col="red")
```

# Adding more graphs to a plot I

Some of the additional general options include

xlim=c(-3,3) X-axis extends from -3 to 3
ylim=c(0,3) Y-axis extends from 0 to 3
xlabel="Time" X-axis label is 'Time'(similarly ylabel=)
font= Integer specifying font to use for text. 1=plain, 2=bold, 3=italic, 4=bold italic
text(2,3,"My text") custom text at position (2,3)

Many of these options can be set globally using par() functions lines(x,y) or points(x,y) adds a line graph or a points graph to an existing graph without changing the axes or labels

```
> x=seq(from=-3,to=3,by=0.1)
>y=dnorm(x,0,0.5)
>plot(x,y,type = "l", col="red")
>lines(x,dnorm(x,0,1), col="blue")
>points(x,dnorm(x,0,3), pch="+", col="green")
>legend(1.5,0.7,c("SD=0.5","SD=1","SD=2"),lty=c(1,1,0),
  pch=c("","","+"), col=c("red","blue","green"))
```

The legend() function adds a legend to the graph at position (1.5,0.7)

# Subplots I

A plot can be designed to have several subplots arranged like a matrix.
The function mfrow=c(m,n) sets up positions for $m \times n$ subplots within a single plot like an $m \times n$ array. This function must be set using par() function as
par(mfrow=c(m,n)

```
>x=seq(from=-3,to=3,by=0.1)
>par(mfrow=c(2,2))
>y=dnorm(x,0,0.5)
>hist(y)
>plot(x,y,type = "l")
>plot(x,dnorm(x,1),type="l")
>plot(x,dnorm(x,2),type="l")
>plot(x,dnorm(x,3),type="l")
```

In par(mfrow=c(m,n) the plots are arranged by row. If we want them arranged by column use mfcol=c(m,n) instead.

# Problems for practice I

- The file 'smoker.csv' contains results of a survey conducted on 356 persons in a city. Each record contains the ID of the person, smoking habit (current-current smoker, former-former smoker, never-never a smoker) and his economic status(Low,middle,high). Write an R program to read the data and construct a two-way table of smoking status versus economic status. Also calculate the proportion of people belonging to each category. Write the output to a text file.

  [Hint: You may use the table()function to construct the table. If $x$ and $y$ are vectors table (x) gives a frequency table of the items in $x$ and table(x,y) constructs a two-way table of frequencies. The command addmargins(tb) calculates the marginal sums of the table object tb. The command prop.table(tb) calculates the proportion of each category.]

- write an R program to find the sum of the first $n$ terms of the Fourier series

  $$\frac{\pi}{2} + 2\left[\frac{sinx}{1} + \frac{sin3x}{3} + \frac{sin5x}{5} + \cdots\right]$$

  The program should include a function to evaluate the sum with $n$ as an argument. Also plot the graphs of the sum for $n = 3, 10, 20$ and 50.

- File 'sales.txt' contains data of price, discount factors and sales volume (number of units sold) of 3 items in three different Indian States. Read the data and compute the total sales bill of each item in each state. write the output with details in an output file

# User defined functions I

The structure of a function is given below.

```
myfunction <- function(arg1, arg2, ... ){
statements
return(object)
}
```

- myfunction is the name of the function
- arg1, arg2 are arguments to the function
- the body of the function is enclosed in braces
- thee return command returns the desired result

here is a function to find the sum of squares of its arguments

```
sumsquares<-function(x,y) {
z=x^2 +y^2
return(z) }
```

Once the above code is executed the function sumsquares is available. We may call it with scalar arguments or vector arguments (of compatible length) and accodingly it will return a scalar or vector.

```
u=sumsquares(2,3)
print(u)
a=1:3
b=11:13
v=sumsquares(a,b)
print(v)
```

# if,else I

Commonly used control structures include

- if, else
- for loop
- while loop

the syntax of an if statement is as follows

```
if (condition) {
statements 1
} else {
statements 2
}
```

- If the condition is true, statements 1 are executed
- If the condition is false, statements 2 are executed
- if want to execute only statements 1 if condition is true, the else part can be removed removed

# if,else II

Example

```
x = 1:15
if (sample(x, 1) <= 10) {
print("x is less than 10")
} else {
print("x is greater than 10")
}
```

If we execute the above code multiple times we might get different results depending on which value from the vector x is sampled.

# For loop I

A for loop is used to iterate over a vector, in R programming. Usually used when the number of iterations is known in advance
Syntax of for loop

```
for (i in sequence) {
statement
}
```

- sequence is a vector, usually of integers
- The index i takes values from the sequence successively and for each value of i the statement is executed once

Example 1: The following code counts the number of even numbers in the vector x.

```
x = c(2,5,3,9,8,11,6)
count = 0
for (i in x) {
if(i %% 2 == 0) count = count+1
}
print(count)
```

Example 2: A program to find the sum of the first hundred odd integers: $\sum_{i=1}^{100}(2i-1)^2$

```
sum=0
for (i in 1:100)
```

# For loop II

```
{
sum=sum+(2*i-1)^2
}
print(sum)
```

However, the same could be achieved in R without using for loop as follows

```
x=seq(from=1, by=2,length.out = 100)
sum(x^2)
```

The usual practice in R is to avoid loops as far as possible.

# While loop I

In R programming, while loops are used to loop until a specific condition is met.
Usually used when the number of iterations is not known in advance
Syntax of while loop

```
while (test-expression) {
statement
}
```

- test-expression is evaluated and the body of the loop is entered if the result is TRUE.
- The statements inside the loop are executed and the flow returns to evaluate the test-expression again.
- This is repeated each time until test-expression evaluates to FALSE, in which case, the loop exits.

Example 1: The following code prints the integers 1 to 5

```
i = 1
while (i < 6) {
print(i)
i = i+1
}
```

# break and next I

A break statement is used inside a loop (for, while) to stop the iterations and flow the control outside of the loop.

Example 1: The following code will only print 1 and 2.

```
x = 1:5
for (val in x) {
if (val == 3){
break
} print(val)
}
```

The loop terminates when it encounters the break statement. A next statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R skips further evaluation and starts next iteration of the loop.

Example: Observe the change when break is replaced by next in the previous code. It prints the integers 1 to 5 skipping 3.

```
x = 1:5
for (val in x) {
if (val == 3){
next
} print(val)
}
```