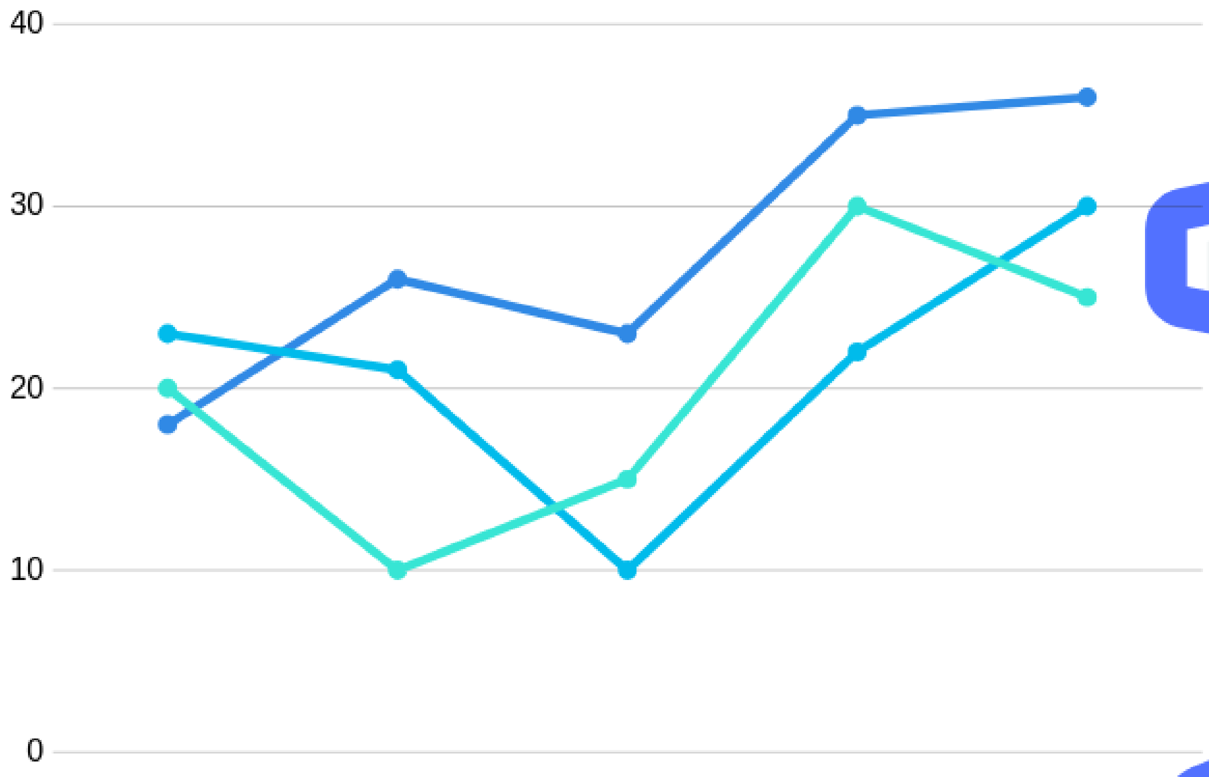
19200129

R PROJECT

THAMU MNYULWA

PART A



Contents

INTRODUCTION TO R: 2019

FINAL ASSIGNMENT: PART A

Exercise 1.7.a

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
> library(MASS)
> search()
 [1] ".GlobalEnv"        "package:MASS"      "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Exercise 1.7.b

```
> objects(pos=2)
  [1] "abbey"                "accdeaths"          "addterm"
  [4] "Aids2"                "Animals"            "anorexia"
        :                           :                       :
        :                           :                       :
 [163] "width.SJ"            "write.matrix"       "wtloss"
```

"objects(pos=2)", shows us all the objects on position 2 and can be used interchangeably with " objects("package:MASS") ", which is a better method.

To show the counting of objects use length function as in "length(objects("package:MASS")) " .

However, the default of the function objects, is that if an object starts with a period it will not be removed unless the argument ALL.NAMES in the function is set to true, as in "ALL.NAMES = TRUE".

```
> length(objects("package:MASS"))
[1] 165

> length(objects("package:MASS", all.names = TRUE))
[1] 166
```

Above we see how the argument all.names set to true shows ".Depends" and does not ignore it.

Exercise 1.7.c

```
objects(pos=2, pat=".")
```

Will return a list of all objects in the package MASS (because of pos 2).

"." Is the modifier and represents any pattern, any object,any character call is the same as objects("package:MASS")

```
objects(pos=2, pat="\\.")
```

will return all objects in the MASS package(because of pos 2) that have a period in the name e.g. "as.fractions"

It is worth noting that the all.names argument is set to FALSE on default and does not show ".Depends" because of it beginning with a period.

Exercise 1.7.d

```
#Notice this question says nothing about what comes after the digit
> objects("package:MASS", pat="^[A-Za-z]{3}[0-9]")
[1] "con2tr" "kde2d"  "npr1"
```

Exercise 1.7.e

```
> objects("package:MASS", pat = "^[A-Za-z]{3}[0-9]$")
[1] "npr1"
```

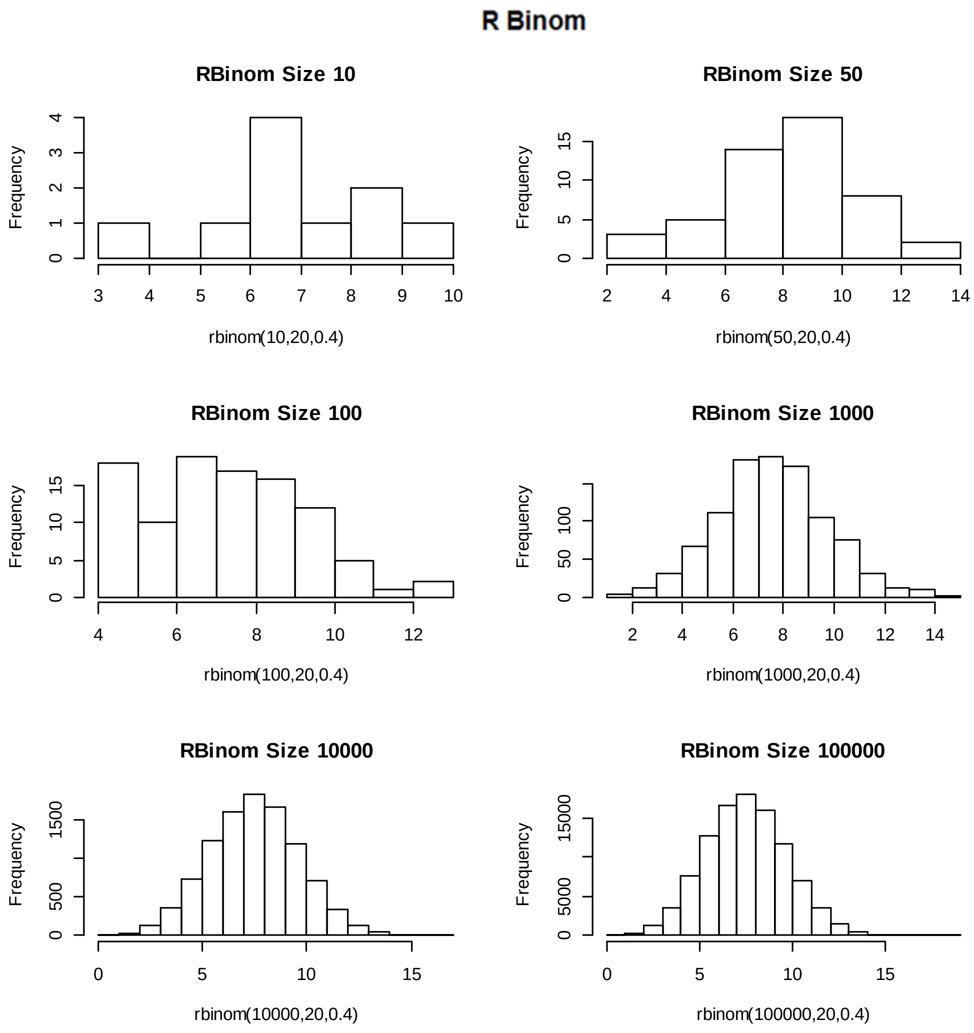
Note: A bit of ambiguity as one can interpret the question as pat = "[A-Za-z]{3}[0-9]$".

Exercise 1.7.f

```
> objects("package:MASS",pat =  "\\.[A-Za-z]{3,4}$")
 [1] "bandwidth.nrd" "contr.sdif"    "cov.mcd"       "cov.mve"       "cov.rob"
 [6] "cov.trob"      "lm.gls"        "nclass.freq"   "neg.bin"       "rms.curv"
```

Exercise 2.10.1

```
> # n=10, p=0.4;success/heads, size=20
># plot histograms , below we enter the commands into a function that does the
># same thing
>  par(mar=c(4,4,5,3) )
>  #rbinom(n, size, prob)
>  b1 <- rbinom(10,20,0.4)
>  b2 <- rbinom(50,20,0.4)
>  b3 <- rbinom(100,20,0.4)
>  b4 <- rbinom(1000,20,0.4)
>  b5 <- rbinom(10000,20,0.4)
>  b6 <- rbinom(100000,20,0.4)
>  par(mfrow=c(3,2))
>  hist(b1,main = "RBinom Size 10", xlab ="rbinom(10,20,0.4)")
>  hist(b2,main = "RBinom Size 50", xlab ="rbinom(50,20,0.4)")
>  hist(b3,main = "RBinom Size 100", xlab ="rbinom(100,20,0.4)")
>  hist(b4,main = "RBinom Size 1000", xlab ="rbinom(1000,20,0.4)")
>  hist(b5,main = "RBinom Size 10000", xlab ="rbinom(10000,20,0.4)")
>  hist(b6,main = "RBinom Size 100000", xlab ="rbinom(100000,20,0.4)")
>  par(mfrow=c(1,1))
>        par(new=T)
>        par(mar=c(2,2,1,7))
>        plot(1:10,1:10,type = "n",xlab="",ylab = "",axes = F)
>        title(main=paste("R Binom"))
```



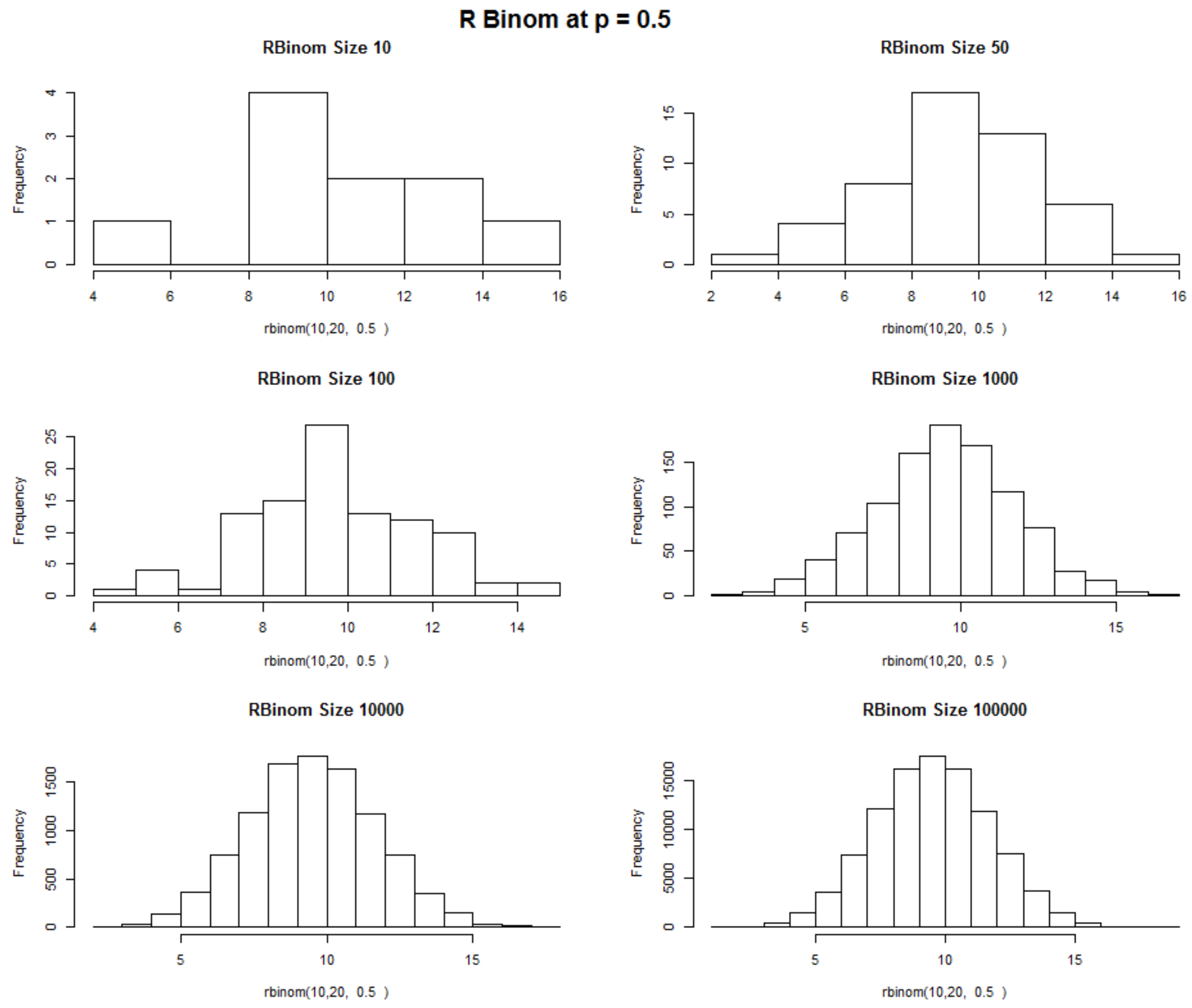
The above histograms show that as the number of observations increases the data will become approximately bell shaped.

This is consistent with the central limit theorem.
Below we build a function and further elaborate in detail with an interpretation.

```
> fix(bi.fun)
```

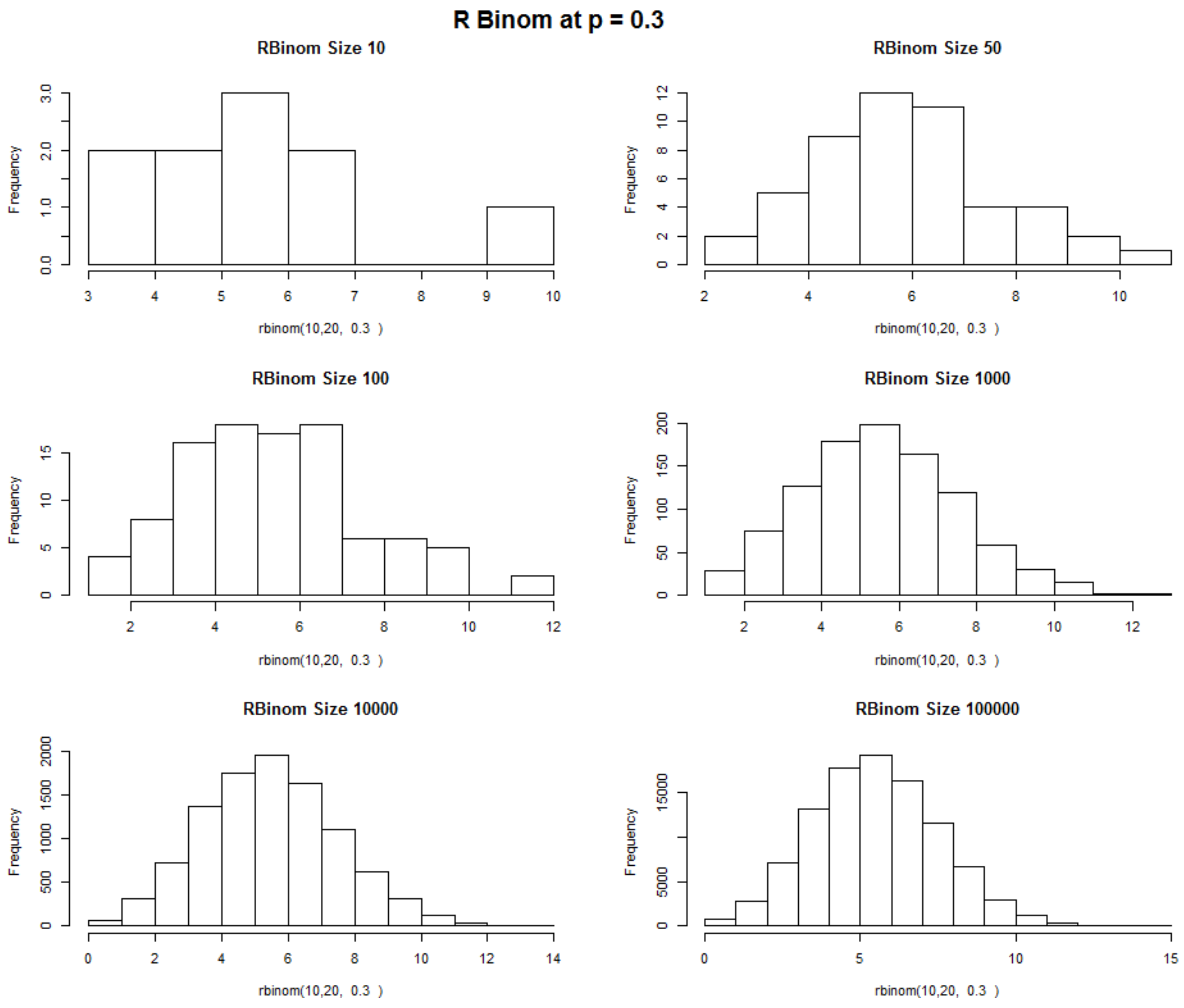
```
function (p)
{   # Function takes in the varable p: the  probability of success
    # Function sends p into each 'rbinom(n, size, prob)'
    # Function assigns each to a object that has a different sample     size(n)
    # Function then prints each on histograms with par(mfrow=c(3,2))
    # We use this function to repeats the exercise for each  p
    par(mfrow=c(3,4),mar=c(4,4,5,3) )
    b1 <- rbinom(10,20,p)
    b2 <- rbinom(50,20,p)
    b3 <- rbinom(100,20,p)
    b4 <- rbinom(1000,20,p)
    b5 <- rbinom(10000,20,p)
    b6 <- rbinom(100000,20,p)
    par(mfrow=c(3,2))
    hist(b1,main = "RBinom Size 10", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    hist(b2,main = "RBinom Size 50", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    hist(b3,main = "RBinom Size 100", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    hist(b4,main = "RBinom Size 1000", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    hist(b5,main = "RBinom Size 10000", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    hist(b6,main = "RBinom Size 100000", xlab = paste("rbinom(10,20, ", p,"
)"),ylab="Frequency")
    par(mfrow=c(1,1))
    par(new=T)
    par(mar=c(2,2,1,7))
    plot(1:10,1:10,type = "n",xlab="",ylab = "",axes = F)
    title(main=paste("R Binom at p =",p))                          }
```

```
> bi.fun(0.5)
```

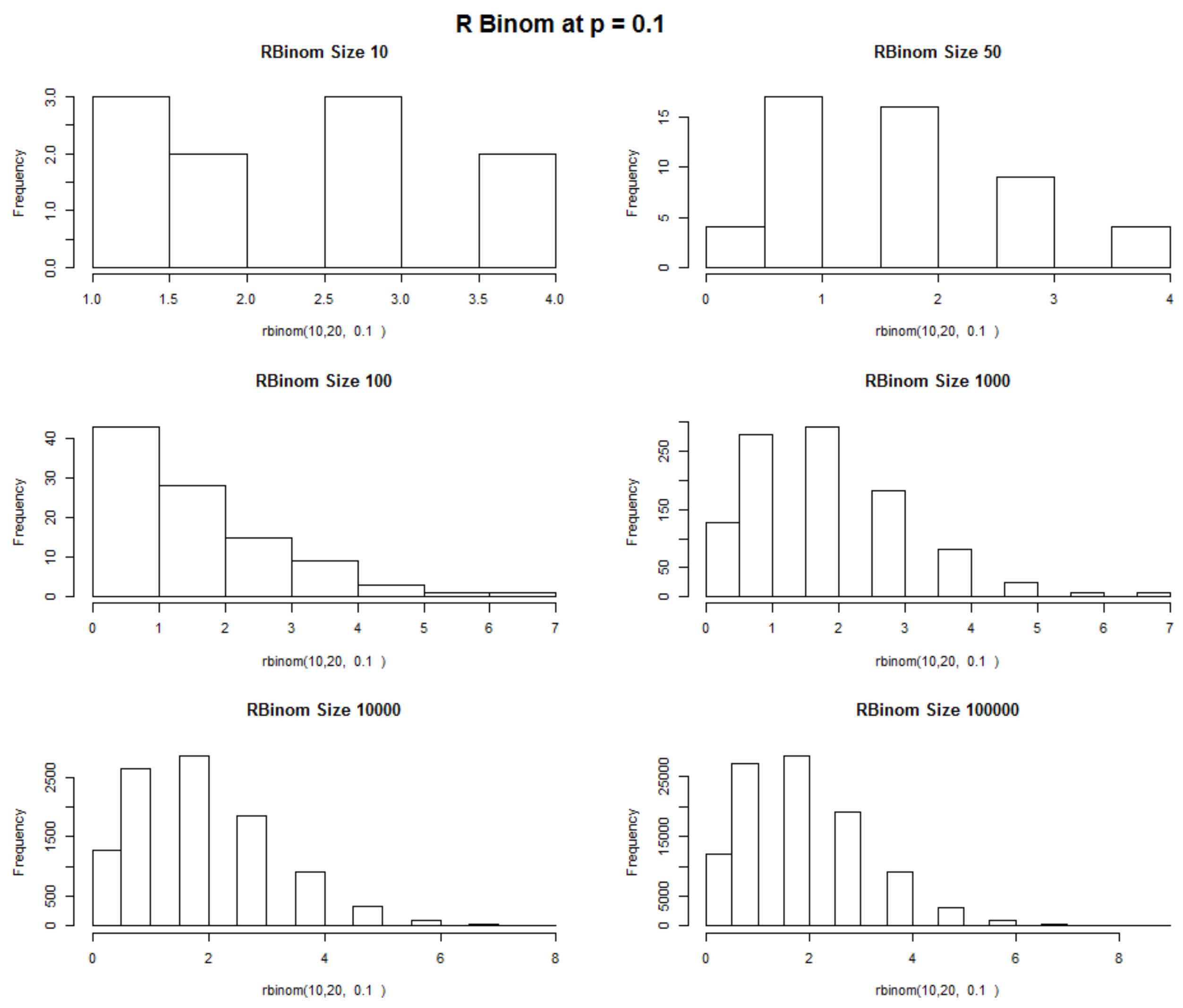


R Binom at p = 0.5
RBinom Size 10
RBinom Size 50
RBinom Size 100
RBinom Size 1000
RBinom Size 10000
RBinom Size 100000
Frequency
rbinom(10,20, 0.5)

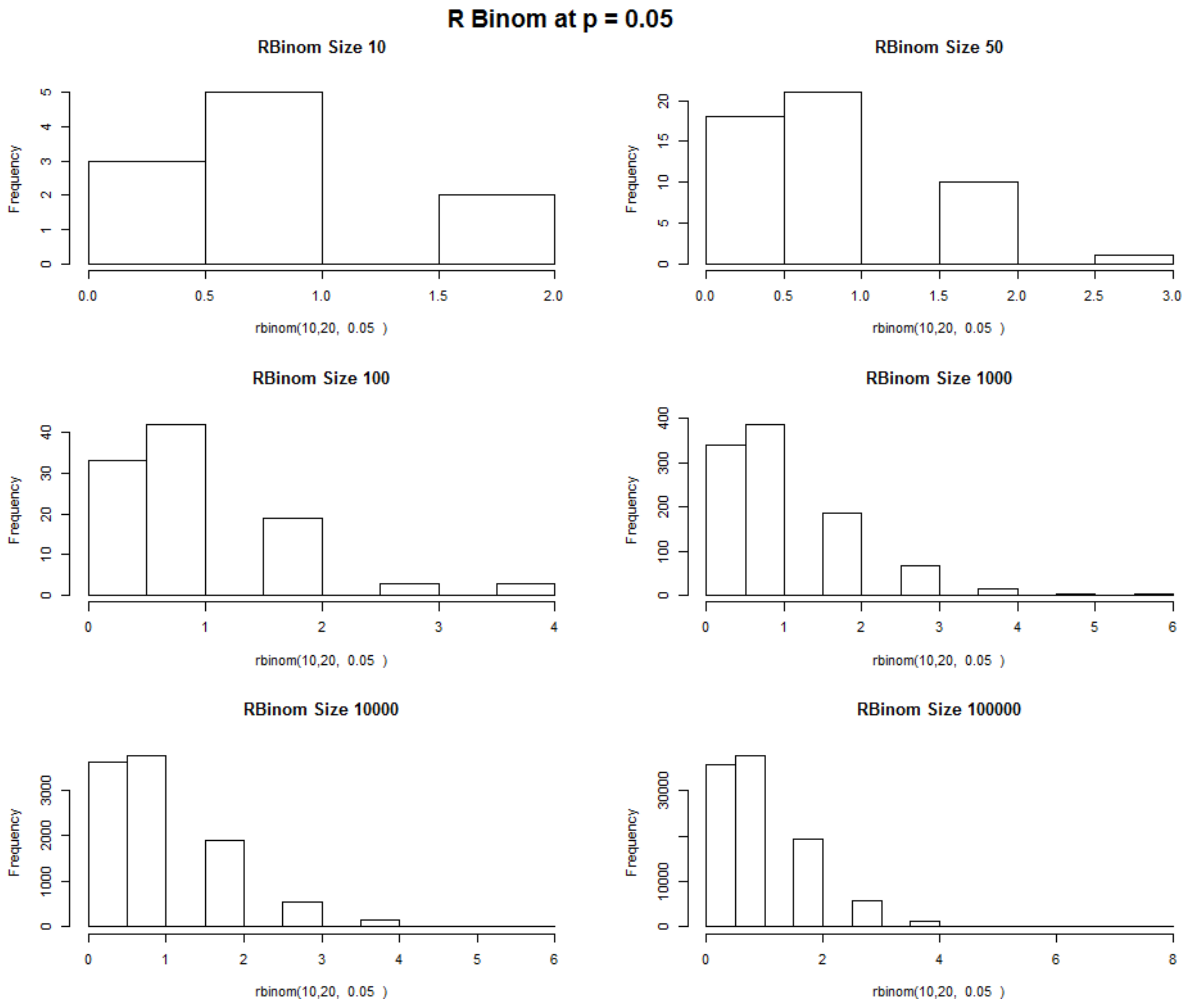
```
> bi.fun(0.3)
```


R Binom at p = 0.3
RBinom Size 10
RBinom Size 50
RBinom Size 100
RBinom Size 1000
RBinom Size 10000
RBinom Size 100000
Frequency
rbinom(10,20, 0.3)

```
> bi.fun(0.1)
```


R Binom at p = 0.1
RBinom Size 10
RBinom Size 50
RBinom Size 100
RBinom Size 1000
RBinom Size 10000
RBinom Size 100000
Frequency
rbinom(10,20, 0.1)

```
> bi.fun(0.05)
```


R Binom at p = 0.05
RBinom Size 10
RBinom Size 50
RBinom Size 100
RBinom Size 1000
RBinom Size 10000
RBinom Size 100000
Frequency
rbinom(10,20, 0.05)

Theory
As we see for each probability the sample becomes more normally distributed as the sample size "n" increases, We also see as the probability of success decreases the sample becomes less normally distributed.

Let Y be the number of successes out of n independent trials, each with success probability p.
Then $Y\sim binom(n,p)$ can be written as the the sum of n i.i.d Bernoulli random variables. i.e $= U_n = \sum_{i=1}^{n} X_i$. Therefore $X_i \sim bernoulli\ (p),\ i = 1,2,\dots n$

The central limit theorem (CLT) tells us that the actual distribution of $Y = U_n = \sum_{i=1}^{n} X_i$ (which is binomial) can be approximated by the normal distribution if n is large.

So $Y = U_n = \sum_{i=1}^{n} X_i \approx N(np, np(1-p))$

The data agrees that the binomial distribution (sum of Bernoulli) can be approximated by the Normal distribution if n is large (larger than 30 observations). As we see the histogram become more bell shaped as we move from 10 observations to a sample of 100 000 observations for each p. This is consistent with the CLT.

From Observation
We also observe the effect of ***n***, the number of independent trials
p, the probability of success on shape.

From observation of p=0.1 and n=10 one can see that for a **small *p*** and **small *n***, the binomial distribution is "**skewed right**". When we have a **large *p*** and **small *n***, the binomial distribution is "**skewed left**" according to the theory (as we see p=0.5 and n=100000 is relatively more left skewed than when p=0.1 and n=10). When **p =0.5** the distribution is somewhat more **symmetrical**.

The histograms suggest **assuming a high probability** is present the sample size n has a larger effect on the normality of a histogram (e.g. 0.5). Furthermore **at lower probabilities**, the increase in sample size produces a more left skewed histogram and the histograms are fairly less dense.

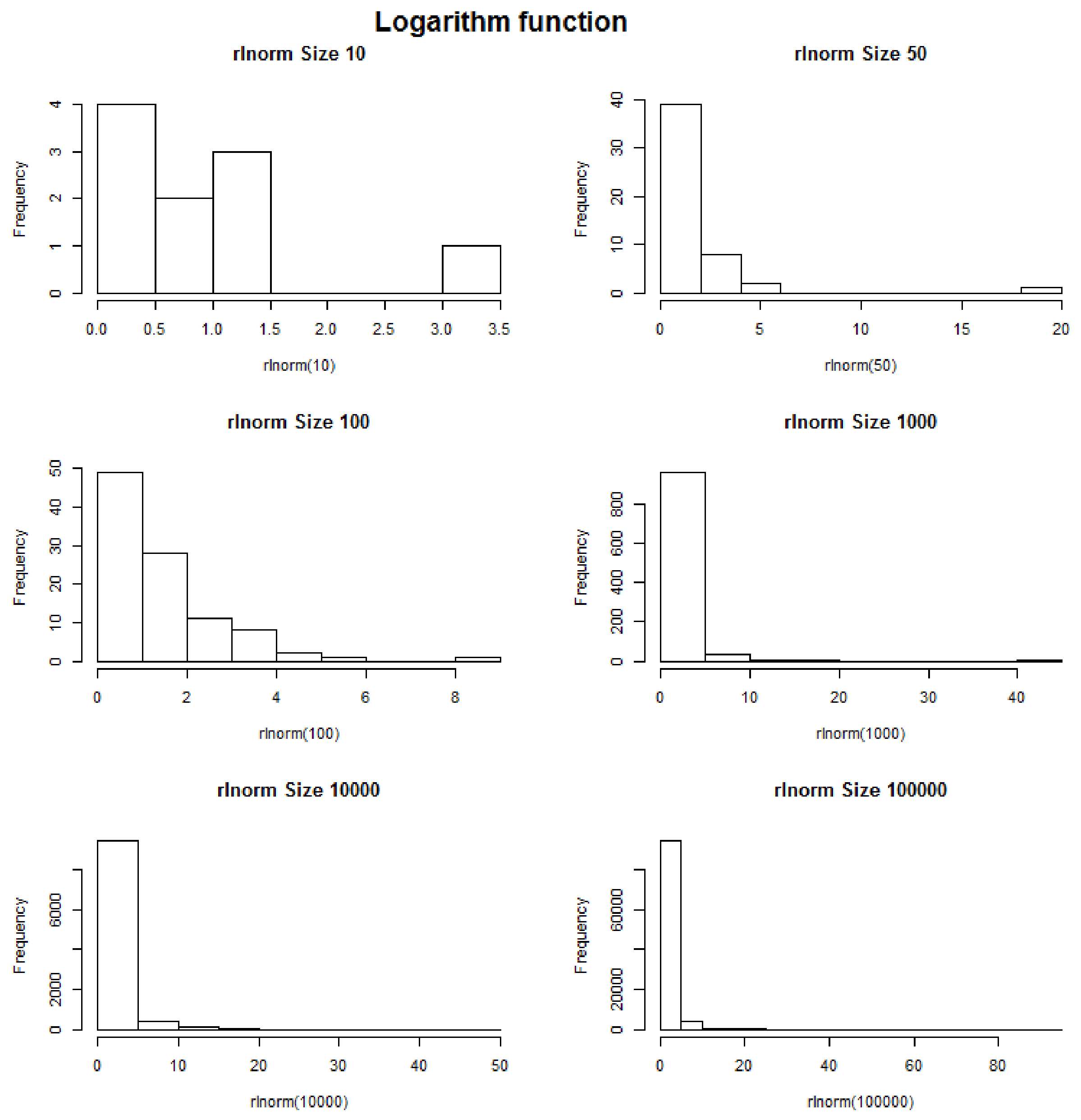
From inspection a decrease in the sample size n, leads to more randomness in the data illustrated by the histograms at each probability. This is expected and once again agrees with the CLT.

Logarithm function

```
> > fix(Ex.2.10.1.log)
```

```
 function (m=0,sdlog=1,...)
{
    # Function excecutes histograms for different rlnorm , at different n
    # Function prints, rlnorm(n, meanlog = 0, sdlog = 1)
    # n : number of observations (10,50,100,1000,10000,100000 here)
    # Function then prints each on histograms with par(mfrow=c(3,2))
      # Function allows one to change the mean and standard deviation
    par(mfrow=c(3,4),mar=c(4,4,5,3) )

    ln1 <- rlnorm(10,m=m,sdlog=sdlog)
    ln2 <- rlnorm(50,m=m,sdlog=sdlog)
    ln3 <- rlnorm(100,m=m,sdlog=sdlog)
    ln4 <- rlnorm(1000,m=m,sdlog=sdlog)
    ln5 <- rlnorm(10000,m=m,sdlog=sdlog)
    ln6 <- rlnorm(100000,m=m,sdlog=sdlog)
    par(mfrow=c(3,2))
    hist(ln1,main = "rlnorm Size 10", xlab = 'rlnorm(10)',...)
    hist(ln2,main = "rlnorm Size 50", xlab = 'rlnorm(50)',...)
    hist(ln3,main = "rlnorm Size 100", xlab = 'rlnorm(100)',...)
    hist(ln4,main = "rlnorm Size 1000", xlab = 'rlnorm(1000)',...)
    hist(ln5,main = "rlnorm Size 10000", xlab = 'rlnorm(10000)',...)
    hist(ln6,main = "rlnorm Size 100000", xlab = 'rlnorm(100000)',...)

    par(mfrow=c(1,1))
    par(new=T)
    par(mar=c(2,2,1,7))
    plot(1:10,1:10,type = "n",xlab="",ylab = "",axes = F)
    title(main=paste("Logarithm function"))

}
```

```
> Ex.2.10.1.log()
```



We see the log normal distribution above. Our function allows us to change our standard deviation and the mean of our graphs at different sizes of n.

Through visual inspection of the above results we can see that the increased sample size 'n' results in the distribution being **increasingly skewed to the right.**

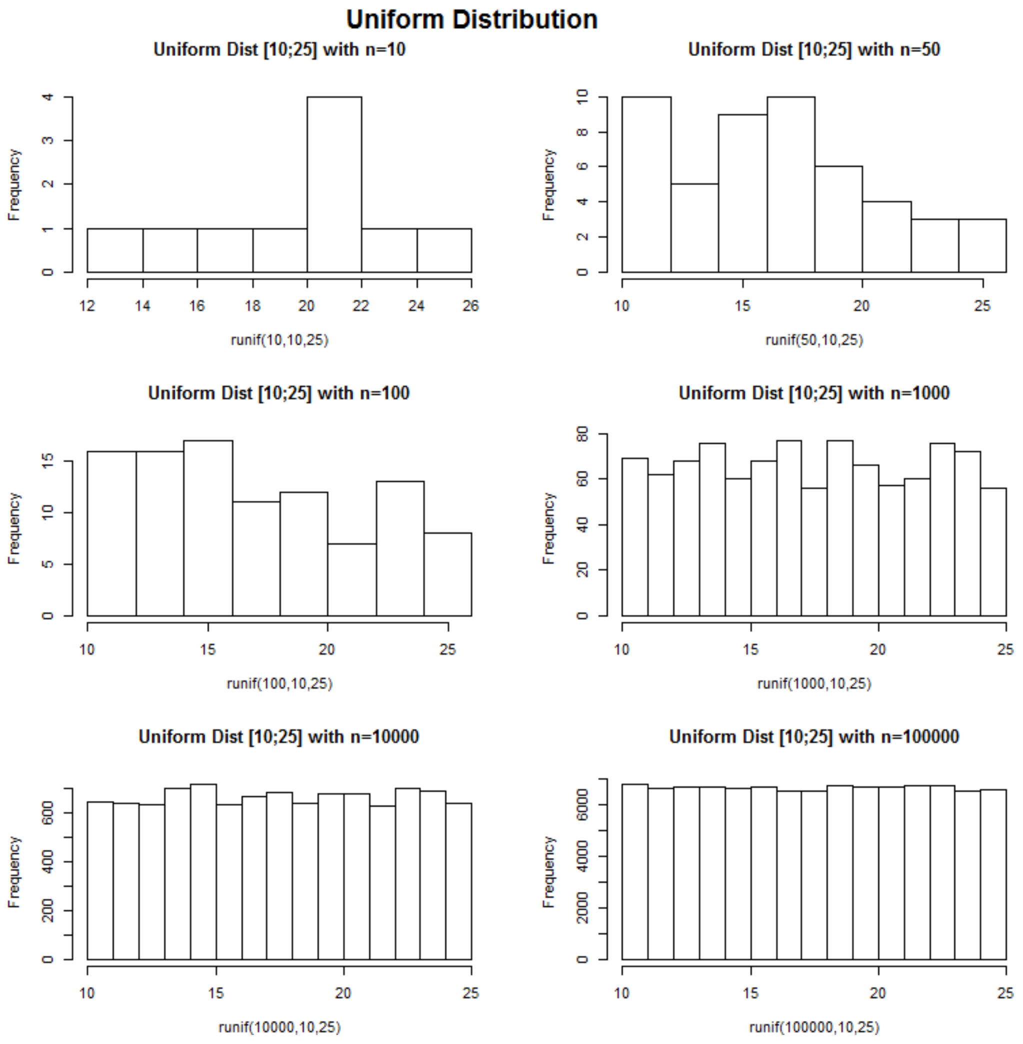
This is consistent with the expectations as log distribution does not have the assumption of independent and identically distributed variables, hence the central limit theorem cannot be applied.
The standard log normal distribution **does not** converge to a **normal distribution as the sample size increase**

Uniform Distribution over the interval [10;25]

```
> fix(Ex.2.10.1.uni)
```

```
function()
{
     # Function excecutes histograms for
     # Uniform Distribution over the interval [10;25]
     # runif(n, min = 0, max = 1), here: runif(n,10,25)
     # n : number of observations (10,50,100,1000,10000,100000
here )
     # Function then prints each on histograms with
par(mfrow=c(3,2))
 u1 <- runif(10,10,25)
 u2 <- runif(50,10,25)
 u3 <- runif(100,10,25)
 u4 <- runif(1000,10,25)
 u5 <- runif(10000,10,25)
 u6 <- runif(100000,10,25)
par(mfrow=c(3,2))
par(mar=c(4,4,5,3))
hist(u1,main = "Uniform Dist [10;25] with n=10", xlab =
'runif(10,10,25)')
hist(u2,main = "Uniform Dist [10;25] with n=50", xlab =
'runif(50,10,25)')
hist(u3,main = "Uniform Dist [10;25] with n=100", xlab =
'runif(100,10,25)')
hist(u4,main = "Uniform Dist [10;25] with n=1000", xlab =
'runif(1000,10,25)')
hist(u5,main = "Uniform Dist [10;25] with n=10000", xlab =
'runif(10000,10,25)')
hist(u6,main = "Uniform Dist [10;25] with n=100000", xlab =
'runif(100000,10,25)')

par(mfrow=c(1,1))
        par(new=T)
        par(mar=c(2,2,1,7))
        plot(1:10,1:10,type = "n",xlab="",ylab = "",axes = F)
        title(main=paste("Uniform Distribution"))

}
```

```
> Ex.2.10.1.uni()
```



Through visual inspection of the above results we can see that the increased sample size 'n' results in the histogram being more evenly distributed amongst all observations.

This is consistent with the expectations of the uniform distribution as the theory suggests that in a uniform distribution outcome are equally likely to happen.

We see fair randomness when n is small (10) and as n increases our histogram becomes fairly more flat.

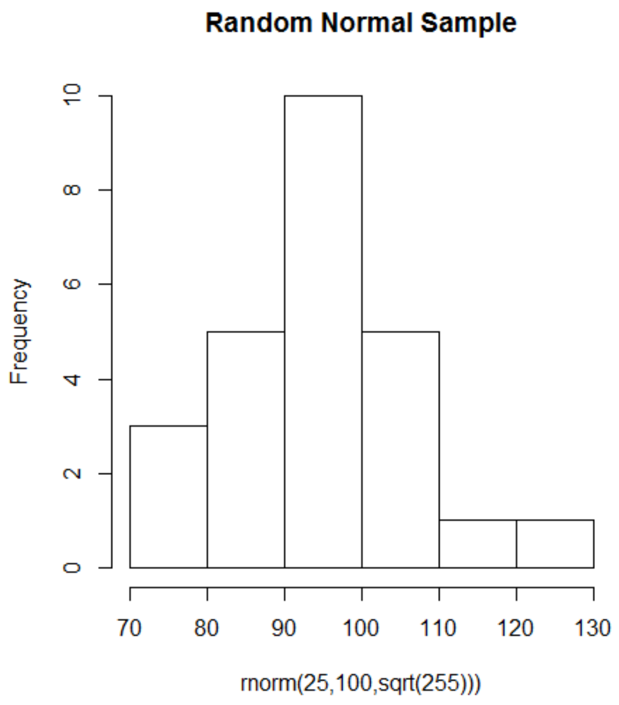
The Uniform distribution does not does not converge to a normal distribution as sample size increases, i.e it cannot be written as a sum of a distribution thus it does not follow the assumptions that must be available for the central limit to apply. Hence it does not become more normally distributed as we increase our number of variables n.

Exercise 2.10.2.i

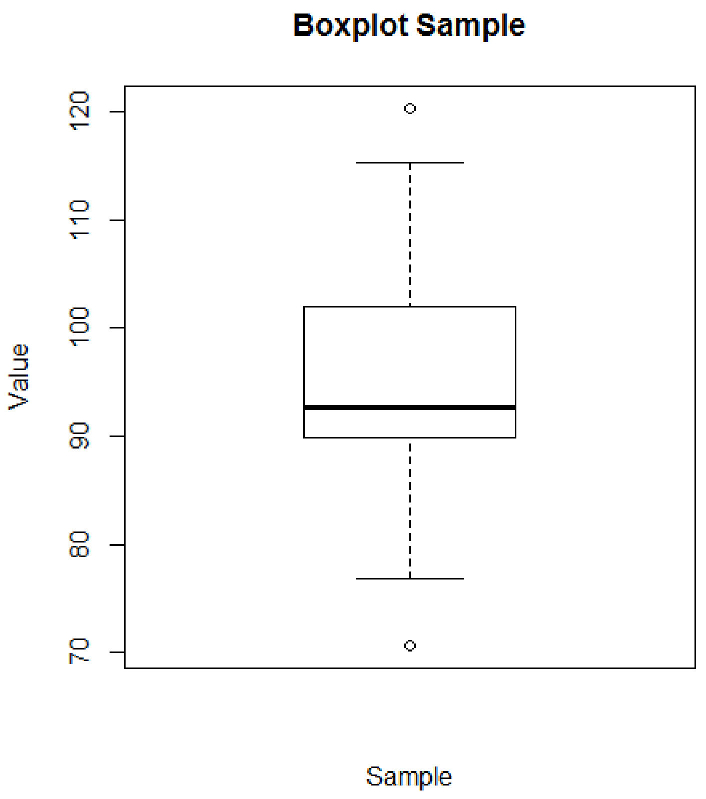
```
> my.sample<-rnorm(n=25,mean=100,sd=sqrt(255))
```

Exercise 2.10.2.ii

```
> hist(my.sample,main="Random Normal Sample",
xlab='rnorm(25,100,sqrt(255))')
```



```
> boxplot(my.sample,ylab = "Value", main ="Boxplot Sample",xlab="Sample")
```



```
> summary(my.sample)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  63.74   86.24   99.33   97.24  110.88  125.35
```

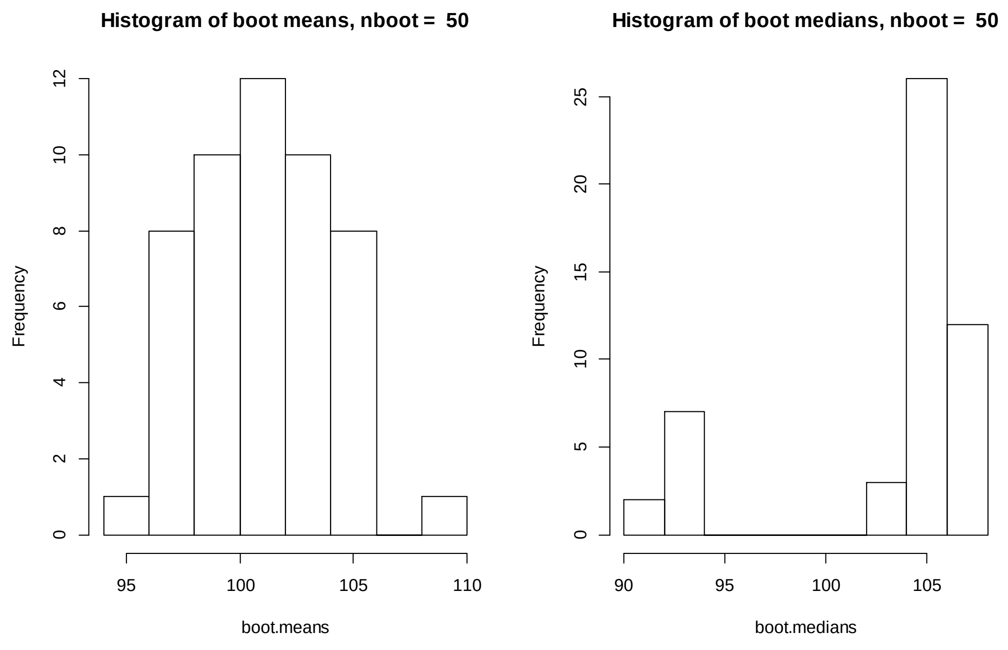
The histogram suggest that our sample is normally distributed and it is bell shaped around a mean 97.24. Our box plot and summary statistics tell us that our first quartile is at 86.24, our third is at 110.88, the minimum amount is 63.74 and the max at 125.35.

Exercise 2.10.2.iii & iv

```
> fix(bootstrap)
```

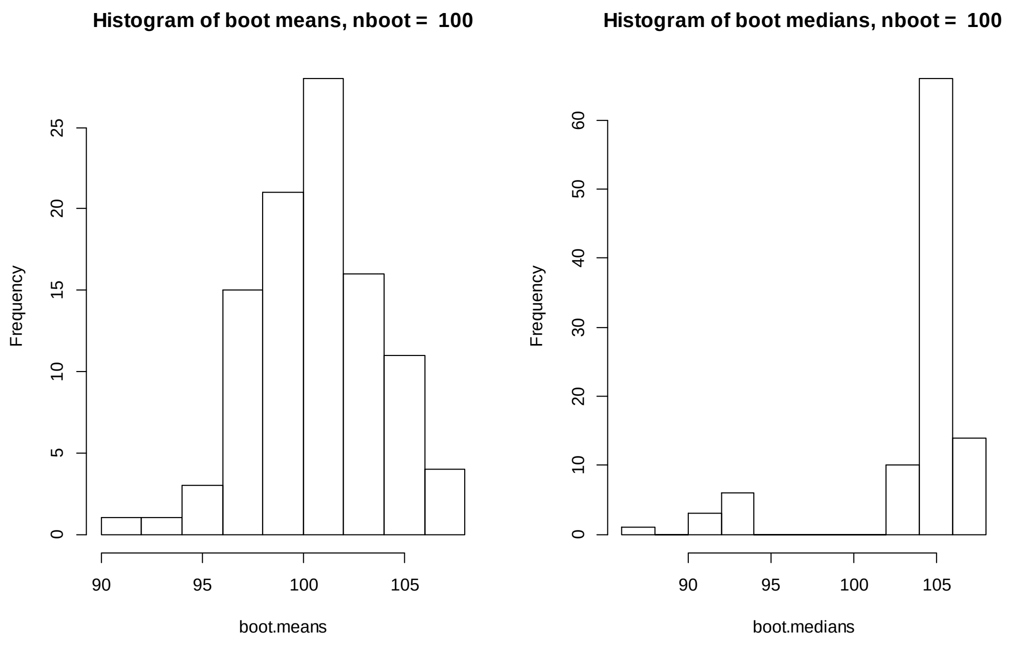
```
function(data = my.sample , nboot=100)
{
    boot.data <- matrix(0,nrow=nboot ,ncol=length(data))

    #We are attempting to take a sample of a sample (bootstrap sampling)
    # First, create an empty mat
    # create column with n rows and number of columns = length(data)
    #data is preset to 'my.sample' , nbootis  preset to '100'
    # the function allows one to increase the number of bootstrap samples
    # important to note that bootstrap sampling samples with replacement
    # This means we can expect overlapping values within each bootstrap sample

    for(i in 1:nboot){
        boot.data[i,] <- sample(data,replace=TRUE)
    }

    boot.means=apply(boot.data,1,mean)
    boot.medians<-apply(boot.data,1,median)

    par(mfrow=c(1,2))
    hist(boot.means,main=paste("Histogram of boot means, nboot = ",nboot))
    #dev.new() #Replaced with par(mfrow=c(1,2))

    hist(boot.medians,main=paste("Histogram of boot medians, nboot = ",nboot))

    list( var.mean=var(boot.means),
var.median=var(boot.medians),sd.mean=sd(boot.means),sd.medians=sd(boot.medians) )

}
```

```
> bootstrap(nboot = 50)
$`var.mean`
[1] 8.620104

$var.median
[1] 24.6132

$sd.mean
[1] 2.936001

$sd.medians
[1] 4.961169
```

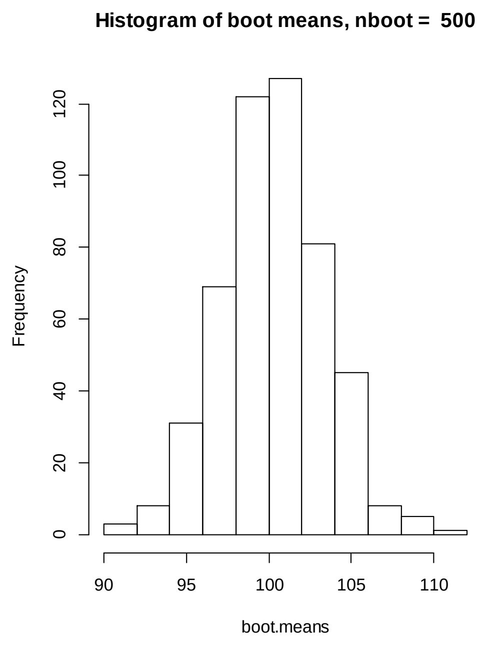


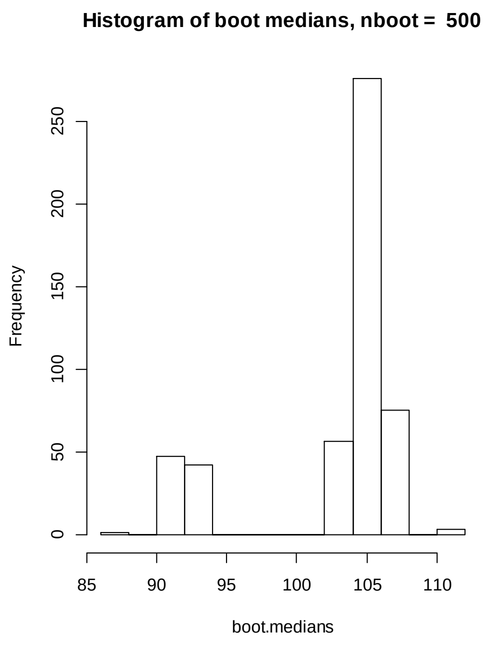
```
> bootstrap(nboot = 100)
```

```
$`var.mean`
[1] 8.677689

$var.median
[1] 16.59521

$sd.mean
[1] 2.945792

$sd.medians
[1] 4.073722
```



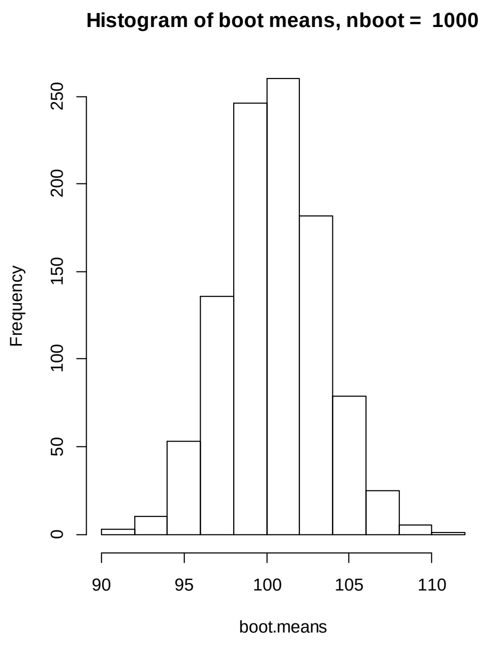
```
> bootstrap(nboot = 500)

$`var.mean`
[1] 9.590019

$var.median
[1] 25.63754

$sd.mean
[1] 3.096776

$sd.medians
[1] 5.063353
```

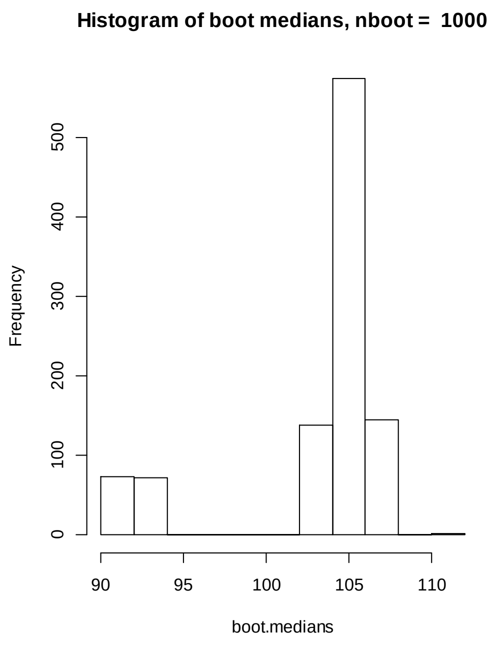




```
> bootstrap(nboot = 1000)
```

```
$`var.mean`
[1] 8.81539

$var.median
[1] 20.68206

$sd.mean
[1] 2.969072

$sd.medians
[1] 4.547754
```





Interpretation: Boot mean and boot median

Theoretical Standard error of the mean is $\frac{\sigma}{\sqrt{n}}$.

```
> sqrt(255)/sqrt(25)
[1] 3.193744
```

The standard deviation of the sampling distribution of the sample mean is often called the standard deviation of the mean or the standard error of the mean.

Theoretically this standard error is 3.193744 which is approximately the bootstrap samples standard error for the means.

From inspection and consistent with the central limit theorem, as the number of observations increases the bootstrap sample means move closer to the density of the normal distribution, regardless of the underlying distribution.

This is not the case for the medians, as we observe from above histograms the medians remain relatively more skewed left as we increase our observations n. The standard errors of the medians are higher. This is also observed in the histograms as they seem to have a wider rage than the mean.

Exercise 3.6.5(p)

$f(n) = \Gamma\left(\frac{n-1}{2}\right) / \left\{\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{n-2}{2}\right)\right\}$ for n=-10,10,100,500

```
>fix(Ex3.6.5.p.alt)
```

```
function (n) {
# Function inputs the function above
# Restriction 1: n must be a non nrgative number
# Restriction 2: n must be a integer

if( n<0 ) return( "'n' must be a non-negative number.")
if( !n%%floor(n) == 0 ) return( "Argument 'n' must be an integer." )

# floor : first integer smaller than n,
# %% : 'modulas' returns 'the rest'

#Class Question: Could(!is.integer(n)) be used for testing this?
#Ans: No we must use "!n%%floor(n) == 0"

if(!(n>1)) stop("n must be at least equal to 2 \n")

#Property of the gamma distribution  if alpa = 1,
#you get a exponential distribution

temp <- lgamma((n-1)/2) - lgamma(1/2) - lgamma((n-2)/2)

exp(temp) # compute the exponential value of a number or number vector


}
```

```
> Ex3.6.5.p.alt(-10)
[1] "'n' must be a non-negative number."

> Ex3.6.5.p.alt(10)
[1] 1.09375

> Ex3.6.5.p.alt(100)
[1] 3.939265

> Ex3.6.5.p.alt(500)
[1] 8.898293
```

Exercise 3.6.7.e
Explain the usage of match(), by considering difference.

Case A

```
> match (c(10,2,7,20,5,8,9,20,9,1,15), c(10,20,15))
 [1]  1 NA NA  2 NA NA NA  2 NA NA  3
```

Case B

```
> match (c(10,20,15), c(10,2,7,20,5,8,9,20,9,1,15))
[1]  1  4 11
```

```
> args(match)
function (x, table, nomatch = NA_integer_, incomparables = NULL)
```

The match argument requires two arguments; the first ‘x’ is a vector of values to be matched. ‘table’ which is second is a vector of values that should be searched for possible matches.

If elements don’t match the functions default behavior is to return NA (a missing value).

The return value will always be the same length vector as the first argument.

Here we refer to the first vector as vec 1 and the second as vec 2 as in ‘match(vec1,vec2).’

In case A the values returned 1 indicates the 10 that is in the first position of the second vector. The values returned 2 indicate the value 20 which is in the second position of the second vector is in that position of the first. The value 15 is in the 3rd position of the second vector and the last of the first. The NA’s in the output indicate that the second vector does not have that particular number in the first.

In case B, the match function matches everything in the first vector (c(10,20,15)) to everything in the second argument (vector 2). It then returns the indices of their position in the second vector.

This is also different to **pmatch()** which is the partial match. We put an example below.

```
> day.w <-c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday")
```

```
> pmatch(c("Fri"),day.w)
[1] 5
```

```
> day.w[5]
[1] "Friday"
```

Section 3.6.8 (a) – (i).

Section 3.6.8 (a)

```
> # checked and saw there are no missing values , hence no previsions necessary
> mean(state.x77[ ,4],na.rm=TRUE)
[1] 70.8786
```

Section 3.6.8 (b)

```
> mean(state.x77[,"Illiteracy"], trim=0.05)
[1] 1.136957
```

Section 3.6.8 (c)

```
> cor(state.x77[,"Illiteracy"], state.x77[,"Income"])
[1] -0.4370752
```

Section 3.6.8 (d)

```
> cov(state.x77)
```

```
             Population        Income   Illiteracy      Life Exp       Murder       HS Grad        Frost          Area
Population 19931683.7588   571229.7796  292.8679592 -4.078425e+02  5663.523714  -3551.509551 -77081.97265  8.587917e+06
Income       571229.7796   377573.3061 -163.7020408  2.806632e+02  -521.894286   3076.768980   7227.60408  1.904901e+07
Illiteracy      292.8680     -163.7020    0.3715306 -4.815122e-01     1.581776     -3.235469    -21.29000  4.018337e+03
Life Exp       -407.8425      280.6632   -0.4815122  1.802020e+00    -3.869480      6.312685     18.28678 -1.229410e+04
Murder         5663.5237     -521.8943    1.5817755 -3.869480e+00    13.627465    -14.549616   -103.40600  7.194043e+04
HS Grad       -3551.5096     3076.7690   -3.2354694  6.312685e+00   -14.549616     65.237894    153.99216  2.298732e+05
Frost        -77081.9727     7227.6041  -21.2900000  1.828678e+01  -103.406000    153.992163   2702.00857  2.627039e+05
Area        8587916.9494 19049013.7510 4018.3371429 -1.229410e+04 71940.429959 229873.192816 262703.89306  7.280748e+09
>
```

```
> round(cov(state.x77),2)
             Population      Income Illiteracy  Life Exp   Murder   HS Grad     Frost          Area
Population 19931683.76   571229.78     292.87   -407.84  5663.52  -3551.51 -77081.97    8587916.95
Income       571229.78   377573.31    -163.70    280.66  -521.89   3076.77   7227.60   19049013.75
Illiteracy      292.87     -163.70       0.37     -0.48     1.58     -3.24    -21.29       4018.34
Life Exp       -407.84      280.66      -0.48      1.80    -3.87      6.31     18.29     -12294.10
Murder         5663.52     -521.89       1.58     -3.87    13.63    -14.55   -103.41      71940.43
HS Grad       -3551.51     3076.77      -3.24      6.31   -14.55     65.24    153.99     229873.19
Frost        -77081.97     7227.60     -21.29     18.29  -103.41    153.99   2702.01     262703.89
Area        8587916.95 19049013.75    4018.34 -12294.10 71940.43 229873.19 262703.89 7280748060.84
```

Section 3.6.8 (e)

```
> range(state.x77[,"Murder"])
[1]  1.4 15.1

> max(state.x77[,"Murder"])-min(state.x77[,"Murder"])
[1] 13.7
```

Section 3.6.8 (f)

```
> state.x77[sample(1:nrow(state.x77),10,replace=FALSE),]
              Population Income Illiteracy Life Exp Murder HS Grad Frost  Area
South Dakota         681   4167        0.5    72.08    1.7    53.3   172 75955
Maine               1058   3694        0.7    70.39    2.7    54.7   161 30920
Rhode Island         931   4558        1.3    71.90    2.4    46.4   127  1049
Virginia            4981   4701        1.4    70.08    9.5    47.8    85 39780
Illinois           11197   5107        0.9    70.14   10.3    52.6   127 55748
Maryland            4122   5299        0.9    70.22    8.5    52.3   101  9891
New Hampshire        812   4281        0.7    71.23    3.3    57.6   174  9027
West Virginia       1799   3617        1.4    69.48    6.7    41.6   100 24070
Minnesota           3921   4675        0.6    72.96    2.3    57.6   160 79289
Oklahoma            2715   3983        1.1    71.42    6.4    51.6    82 68782
>
# Random sample without replacement: replace=FALSE
```

Section 3.6.8 (g)

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

Section 3.6.8 (h)

Kurtosis

```
> fix(Ex.3.6.8.h)
```

```
function() {
#function to calculate the coefficient of kurtosis
#We then test this function on Frost variable in state.x77
#We know kurtosis is the 4th non-central moment


data <- state.x77[,"Frost"] #take the data as Frost

n <- length(data)
mean <- mean(data)
var <- var(data)

kurturtosis <- sum((data-mean)^4)/n #kurtosis

coeff.kurt <- kurtosis/var^2   #coff of kurtosis
return(coeff.kurt)
}
```

```
> Ex.3.6.8.h()
[1] 2.056163
```

Section 3.6.8 (i)

Skewness

```
> fix(Ex.3.6.8.i)
```

```
function() {
#function to find the coefficient of skewness for data
#Tested on Murder variable in state.x77
#skewness is the 2nd non-central moment

data <- state.x77[,"Murder"]

n <- length(data)
mean <- mean(data)
var <- var(data)

skewness <- sum((data-mean)^3)/n

coeff.skewness <- skewness/( var^(3/2)) #coefficient of skewness
return(coeff.skewness)
}
```

```
> Ex.3.6.8.i()
[1] 0.1293391
```

Section 3.6.10 (f)
cut()

We worked through this to illustrate the usage of cut function. We tried to illustrate how changing it affects boundaries of groups.

```
> cut(state.x77[ ,"Illiteracy"],breaks = c(0,0.5,1,1.5,2,5),include.lowest =
TRUE, right=FALSE)
 [1] [2,5]   [1.5,2) [1.5,2) [1.5,2) [1,1.5) [0.5,1) [1,1.5) [0.5,1) [1,1.5)
[2,5]   [1.5,2)
[12] [0.5,1) [0.5,1) [0.5,1) [0.5,1) [0.5,1) [1.5,2) [2,5]   [0.5,1) [0.5,1)
[1,1.5) [0.5,1)
[23] [0.5,1) [2,5]   [0.5,1) [0.5,1) [0.5,1) [0.5,1) [0.5,1) [1,1.5) [2,5]
[1,1.5) [1.5,2)
[34] [0.5,1) [0.5,1) [1,1.5) [0.5,1) [1,1.5) [1,1.5) [2,5]   [0.5,1) [1.5,2)
[2,5]   [0.5,1)
[45] [0.5,1) [1,1.5) [0.5,1) [1,1.5) [0.5,1) [0.5,1)
Levels: [0,0.5) [0.5,1) [1,1.5) [1.5,2) [2,5]
```

```
> illitgrp <- cut(state.x77[ ,"Illiteracy"],breaks =
c(0,0.5,1,1.5,2,5),include.lowest = TRUE, right=FALSE)
```

```
# Note, args of cut()
#include.lowest : indicating if an ‘x[i]’ equal to the lowest (or highest, for
#right = FALSE) ‘breaks’ value should be included.
#
#right : indicating if the intervals should be closed on the right (and open on
#the left) or vice versa. Default is “right=TRUE” ->  Levels: (0,0.5]
#
#Instead of table(cut(x, br)), hist(x, br, plot = FALSE) is more efficient and
#less memory hungry.
#Instead of cut(*, labels = FALSE), findInterval() is more efficient.
```

Section 3.6.10 (g)
Two - Way Table

```
# table: table uses the cross-classifying factors to build a contingency table
of # the counts at each combination of factor levels.

# get areagp # Note Intervals  Levels: (0,1e+04] (1e+04,1e+05] (1e+05,Inf]
```

```
> areagp<-
cut(state.x77[,'Area'],
breaks = c(0,10000,100000,Inf),
labels=c("Small","Medium","Large"))   # state.x77[ ,Area]
```

```
> head(areagp)
[1] Medium Large  Large  Medium Large  Large
Levels: Small Medium Large
```

```
> table(areagp,illitgrp)   # Two-way table / Contingency table
        illitgrp
areagp   [0,0.5) [0.5,1) [1,1.5) [1.5,2) [2,5]
  Small        0       4       4       1     0
  Medium       0      18       6       4     5
```

Large	0	3	1	2	2

Exercise 3.8.1.

Standard Deviation of Unity: Less than 1

From CLT 68% of data from normal distribution with mean zero and sd of unity will have absolute value less than unity.

Find proportion of n random normal (0; 1) variables whose absolute values is less than 1.0.

```
function (n)
# n :  number of random normal (m=0, sd=1) variables to generate
# n must be positive and an integer
{
if(n < 0) return( cat("n, must be positive.") )
if(!n%%floor(n) == 0) return( cat("n, must be an integer.") )

num <- sum( abs(rnorm(n,m=0,sd=1)) < 1 )        #Important part of function
den <- n
percentage <- num/den

return( result = cat("In a rnorm(m=0,sd=1) sample of",n,"proportion with abs < 1
is :",percentage ) )

}
```

```
> Ex.3.8.1(2)
In a rnorm(m=0,sd=1) sample of 2 proportion with abs < 1 is : 1
> Ex.3.8.1(100)
In a rnorm(m=0,sd=1) sample of 100 proportion with abs < 1 is : 0.65
> Ex.3.8.1(10000)
In a rnorm(m=0,sd=1) sample of 10000 proportion with abs < 1 is : 0.684
> Ex.3.8.1(10000000)
In a rnorm(m=0,sd=1) sample of 1e+07 proportion with abs < 1 is : 0.6828499
> Ex.3.8.1(100000000)
In a rnorm(m=0,sd=1) sample of 1e+08 proportion with abs < 1 is : 0.6826973
```

From the above results we infer that there is a fairly large proportion (≈ 0.68) with an absolute value below unity in a rnorm distribution with mean 0 and standard deviation 1.

As numbers generated n increases in size the proportion of n(0,1) variables that meet the absolute value less than unity condition move closely towards 68%. This is consistent with what would be expected as the central limit theorem suggest that as the sample size increases the sum of i.i.d variables veers towards normal distribution and 68% of values lie less than 1 standard deviation from the mean.

Exercise 5.7.4. Subscripting
Use subscripting to find the largest proportion of over 75 in those countries with a dpi of less than 1000 in the LifeCycleSavings data set. Determine also countries having this pop75value.

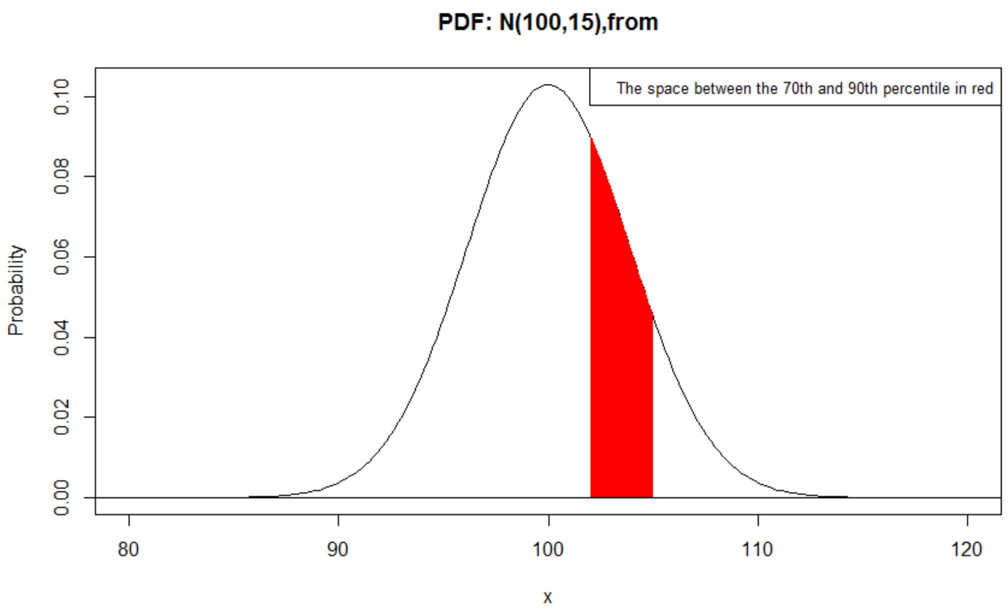
```
>Ex.5.7.4
```

```
function ()
{
# Function finds the largest proportion  of over 75 in those countries
#with a dpi of less than 1000 in the LifeCycleSavings

subset <- LifeCycleSavings[ LifeCycleSavings[ ,"dpi"] < 1000,  ]
a <- subset[,"pop75"]

b <- seq(along = a)[a == max(a)]
names <- rownames( subset )[b]

 list(Max.Pop = max(a), CountryNames = names)
}
```

```
> Ex.5.7.4()
$`Max.Pop`
[1] 3.1

$CountryNames
[1] "Greece"
```

Exercise 6.2.2.
Display the pdf of a Normal (100, 15) distribution with the area under the density bounded by 70th and 90th percentiles in red.

```
function(lower=0.7 , upper=0.9){
# This function plots a normal distribution and superimposes red lines that
#colour the 70th to 90th percentiles in red. This are user defined so that one can
# change them to less or more from the global environment or wherever
# the calling environment is.
# argument lower is the lower limit to highlight and upper the upper limit.
# We plot using lower level plotting lines function
x <- seq(from=80,to=120, length=200)
dens <- dnorm(x, m=100, sd=sqrt(15))

plot(x=x,y=dens, main= paste("PDF: N(100,15),from"),typ="l", ylab="Probability" )

qlower = qnorm(lower,m=100,sd=sqrt(15)) # x-value lower
qupper = qnorm(upper,m=100,sd=sqrt(15)) # x-value upper

# where y is the lower density to upper density that needs shading i
pts <- seq( from =qlower, to = qupper , length=200) # vector of x values

#for loop drawing the line inbetween all the available values
for(i in pts){
dens <- dnorm(i, m=100, sd=sqrt(15))
lines(x = c(i,i), y=c(0,dens), col="red")
}
abline(h=0) # horizontal line at y = 0
# textbox explaining graph,
legend("topright", legend=c("The space between the 70th and 90th percentile in
red"),cex=0.8)
}
```



The graph is normally distributed with the space between the 70th and 90th percentile in red.

We used high level plotting to plot the graph and superimposed red line using low level plotting "lines" function.

Exercise 6.2.6.
Write your own function for calculating the **harmonic mean** and use it to calculate the harmonic mean of variable 'dpi' in the LifeCycleSavings data set.

I give **two Answers** for the **Harmonic mean,** one an adaptation of the harmonic mean question done in class (Ex.3.6.8.j) Ex.6.2.6.i and another Ex.6.2.6.i.alt, a function written solely by myself. This is to illustrate both ways the first one has more functionality as it sets provisions for both matrices and vectors and has been commented on to explain how it works.

```
> Ex.6.2.6.i()
[1] 431.4708
```

```
>Ex.6.2.6.i.alt()
```

```
>fix(Ex.6.2.6.i)
function ()
{
#Function creates a function  harmonic mean that
#calculates the harmonic mean
```

```
#"harmonic.mean()" takes in arguments: x, na.rm, eps

#x : is either a vecor or matrix that is then used to calculate the
h.mean
#na.rm : argument is to specify the handling of missing values
# arg removes all 'NA' arguments in the vector or matrix at FALSE
#eps : is set to almost 0, it takes in the machines specifications ands
sets it to almost 0
harmonic.mean <- function(x, na.rm = TRUE,eps=.Machine$double.eps^(1/4))
{
#x: numeric matrix or vector
#below object x is checked as either a vector or matrix
#if ncol(x) is not null and if more than 1 column it's assigned as a
matrix
#otherwise its assigned as a vector
#If x is not numeric argument "x must be numeric" is returned
#If neither a matrix or a vector "x must be a numeric vector or matrix"
is returned
#####################################################################

if(!is.null(ncol(x))) {if(ncol(x)>1) x <- as.matrix(x)}
else x <- as.vector(x)


if(!is.numeric(x))stop("x must be numeric \n")

out.temp <- is.matrix(x)
out.temp <- c(out.temp, is.vector(x))

if(all(out.temp==FALSE)) stop("x must be a numeric vector or matrix \n")
#####################################################################
# 2 options , if x is a vector  or a matrix
# In our example x is a vector column of ‘LifeCycleSavings’ hence, vector
#
if(is.vector(x))
{
if(any(na.omit(x) <= eps)) stop("Har.mean requires positive numbers only
\n")
else out <- 1/ mean(1/x,na.rm=na.rm)
}
if(is.matrix(x))
{
temp <- ifelse(is.na(x),5,x)

if(any(temp <= eps)) stop("Har.mean requires positive numbers only \n")

else out <- 1/(apply(1/x,2,mean,na.rm = na.rm))
}
out
}
```

```
## send dpi column to harmonic mean function above
harmonic.mean(x = LifeCycleSavings[,"dpi"]  )
}
```

```
>fix(Ex.6.2.6.i.alt)
```

```
function (x)
{
# Function returning the harmonic mean
#takes in argument x which is a vector
#x can have NA values but are removed otherwise
# x must be a numeric vector
#returned is the harmonic mean based on calculation provided

if(!is.vector(x)) return("Revise input, this function takes in a vector")
#ensure as a vector
if(!is.numeric(x)) return("Revise input, this function takes in a numeric
vector")
if(any(is.na(x))){ cat("Warning: Function removes NA values"); x <-
x[!is.na(x)]}

mean <- mean(x,na.rm=TRUE)
1/mean(1/x) # harmonic mean

}
```

```
> Ex.6.2.6.i.alt(LifeCycleSavings[ ,"dpi"] )
[1] 431.4708

> Ex.6.2.6.i()
[1] 431.4708
```

Here we see that both functions give us the harmonic mean of 431.4708. We comment further below.

Exercise 6.2.6 (ii)

Calculate the ordinary mean of variable dpi in the LifeCycleSavings data set. Compare the answer with the answer in (a). Which answer would you use in practice? Motivate.

```
> fix(Ex.6.2.6.ii)
```

```
function ()
{
mean(x = LifeCycleSavings[,"dpi"]  )
}
```

```
> Ex.6.2.6.ii() # normal mean
[1] 1106.758
> Ex.6.2.6.i() # Harmonic mean
[1] 431.4708
```

```
> Ex.6.2.6.i.alt(LifeCycleSavings[,"dpi"])
[1] 431.4708
```

Normal Mean (Arithmetic Mean)

$$\bar{X}_{arithmetic} = \frac{\sum x_i}{n}$$

The arithmetic mean, uses addition and is calculated as the addition of objects, divided by the sum of objects. This is the most often referred to “average”, the “mean” function is used to calculate it.

Harmonic Mean

$$\bar{X}_{harmonic} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=i}^{n} 1/x_i}$$

When the arithmetic mean requires addition, the harmonic mean uses reciprocals. The reciprocal of n is simply $1/_n$.

Another average, not discussed here is the geometric mean, this is favored in investments and time value of money calculations as it allows better representations of results, it is the compounding average.

Since the harmonic mean makes use of reciprocals allows another layer of multiplication / division over the geometric mean. This allows it to deal well with fractions such as rates or ratios over varying period lengths of times. We calculate the harmonic mean using our function above.

```
> Ex.6.2.6.ii() # normal mean
[1] 1106.758
> Ex.6.2.6.i() # Harmonic mean
[1] 431.4708
```

We find our normal mean at 1106.758, which is much higher than our harmonic mean at 431.4708 because of the different ways that they are computed.

LifeCycleSavings is data on the savings ratio 1960–1970 and ‘dpi’ is a variable showing the real per-capita disposable income. Since the data is of a ratio the harmonic mean would be a better average to use in practice.

Exercise 6.2.7.i

```
Fisher’s linear discriminate function in the case of two groups is
defined as follows.
```

Exercise 6.2.7 (i)

```
## takes “company.10var” in as a list, then assign it as data.frame
> company.10var <- source("clipboard") # source brings in as a list
> company.10var <- as.data.frame(company.10var[[1]]) # behave as a dataframe

> head(company.10var,2)
  Successful   X1   X2   X3   X4   X5   X6   X7   X8   X9  X10
1          1 0.09 0.13 0.03 0.25 0.25 0.04 0.03 1.61 0.74 1.38
2          1 0.04 0.07 0.02 0.06 0.04 0.02 0.01 1.39 0.85 1.98

> tail(company.10var,2)
   Successful   X1   X2   X3   X4   X5   X6   X7   X8   X9    X10
78          2 0.15 0.27 0.16 0.22 0.37 0.58 0.27 2.84 1.75 112.38
79          2 0.13 0.27 0.14 0.30 0.55 0.28 0.26 1.67 0.73   4.46
```

Exercise 6.2.7.ii

```
#Notice that column Successful has 1 for successful, 2 for unsuccessful
#subscript “Successful” column into “company.10var.faliure”
#&“company.10var.success”

> company.10var.success <- company.10var[company.10var[,'Successful']==1, ]

> head(company.10var.success,2)
  Successful   X1   X2   X3   X4   X5   X6   X7   X8   X9  X10
1          1 0.09 0.13 0.03 0.25 0.25 0.04 0.03 1.61 0.74 1.38
2          1 0.04 0.07 0.02 0.06 0.04 0.02 0.01 1.39 0.85 1.98

> # assuming 2 is for success and 1 is for faliure
> company.10var.faliure <- company.10var[company.10var[,'Successful']==2, ]
> head(company.10var.faliure,2)
   Successful   X1   X2   X3   X4   X5   X6   X7   X8   X9  X10
25          2 0.17 0.18 0.11 0.24 0.34 0.19 0.15 2.36 1.26 5.20
26          2 0.13 0.18 0.09 0.19 0.29 0.18 0.14 1.85 1.13 8.73

# remove “Successful” column and make covariance matrices for subscripts
# changed data.frame into matrix for future use
# rounded for publication purposes

> cov.success<- cov(company.10var.success)
> cov.faliure <- cov(company.10var.faliure)

round(cov(company.10var.success),3)# Successful companies
       X1    X2    X3     X4     X5    X6    X7     X8     X9    X10
X1  0.018 0.009 0.009  0.115  0.116 0.012 0.010  0.022  0.035  0.495
X2  0.009 0.011 0.009  0.081  0.082 0.012 0.010  0.018  0.014  0.522
X3  0.009 0.009 0.009  0.105  0.106 0.011 0.010  0.022  0.020  0.340
X4  0.115 0.081 0.105 13.509 13.451 0.105 0.108 -0.208 -0.018  1.413
X5  0.116 0.082 0.106 13.451 13.396 0.106 0.110 -0.205 -0.016  2.008
X6  0.012 0.012 0.011  0.105  0.106 0.015 0.012  0.023  0.023  0.713
```

```
X7  0.010 0.010 0.010  0.108  0.110 0.012 0.010  0.023  0.021   0.488
X8  0.022 0.018 0.022 -0.208 -0.205 0.023 0.023  0.232  0.199   4.939
X9  0.035 0.014 0.020 -0.018 -0.016 0.023 0.021  0.199  0.216   5.750
X10 0.495 0.522 0.340  1.413  2.008 0.713 0.488  4.939  5.750 694.850
```

```
X10 0.495 0.522 0.340  1.413  2.008 0.713 0.488  4.939  5.750 694.850
```

```
> round(cov(company.10var.faliure),3)# Unsuccessful companies
        X1     X2     X3     X4     X5    X6     X7     X8     X9      X10
X1   0.004  0.003  0.002  0.003  0.005 0.003  0.003 -0.008  0.002    0.298
X2   0.003  0.005  0.003  0.005  0.009 0.005  0.005 -0.014 -0.005    0.841
X3   0.002  0.003  0.002  0.003  0.004 0.004  0.003 -0.005 -0.002    0.824
X4   0.003  0.005  0.003  0.008  0.013 0.002  0.005 -0.037 -0.021    0.053
X5   0.005  0.009  0.004  0.013  0.025 0.001  0.009 -0.057 -0.029    0.176
X6   0.003  0.005  0.004  0.002  0.001 0.026  0.007  0.117  0.055    6.275
X7   0.003  0.005  0.003  0.005  0.009 0.007  0.006 -0.008 -0.002    1.439
X8  -0.008 -0.014 -0.005 -0.037 -0.057 0.117 -0.008  1.462  0.661   30.686
X9   0.002 -0.005 -0.002 -0.021 -0.029 0.055 -0.002  0.661  0.380   16.472
X10  0.298  0.841  0.824  0.053  0.176 6.275  1.439 30.686 16.472 3919.422
```

Exercise 6.2.7.iii

```
> mean.vec.success <-apply(company.10var.success,2,mean)
> mean.vec.faliure <-apply(company.10var.faliure,2,mean)
```

```
> round(mean.vec.success,3) # (a) Successful Companies
    X1     X2     X3     X4     X5     X6     X7     X8     X9    X10
 0.026  0.004 -0.052 -1.014 -1.004 -0.060 -0.049  1.375  0.789 12.607
```

```
> round(mean.vec.faliure,3) # (b) Unsuccessful Companies
    X1     X2     X3     X4     X5     X6     X7     X8     X9    X10
 0.120  0.191  0.103  0.231  0.375  0.230  0.166  1.941  1.016 40.367
```

```
> mean.vec.success  # full numbers
          X1           X2           X3           X4           X5           X6           X7
 0.025833333  0.004166667 -0.052083333 -1.014166667 -1.003750000 -0.060000000 -0.049166667
          X8           X9          X10
 1.375416667  0.789166667 12.607083333
> mean.vec.faliure
        X1         X2         X3         X4         X5         X6         X7         X8
 0.1203636  0.1914545  0.1032727  0.2312727  0.3750909  0.2298182  0.1656364  1.9410909
        X9        X10
 1.0161818 40.3665455
```

Exercise 6.2.7.iv LDA

```
>fix(Ex.6.2.7.a)
```

```
> Ex.6.2.7.a()
```

```
function (company.10var=company.10var)
{
# function takes in company.10 var and splits it
#into success = company.10var.success
#into faliure = company.10var.faliure
x <- c(1:10)

#Subscripting to split company.10var, remove col_1
company.10var.success<-company.10var[company.10var[,1]==2 ,c(-1) ]
company.10var.faliure<-company.10var[company.10var[,1]==1 ,c(-1) ]

#Get mean vectors of two groups and difference
mean.s1 <- apply(company.10var.success,2,mean,na.rm=TRUE)
mean.f2 <- ap
ply(company.10var.faliure,2,mean,na.rm=TRUE)

#Get number of rows of two groups
n.s1 <-nrow(company.10var.success) # Can we use nrow / length
n.f2 <-nrow(company.10var.faliure)

#Get covariance matrixes two groups
S.1.s <-cov(company.10var.success)
S.2.f <-cov(company.10var.faliure)

##Group Variance
num <- ((n.s1-1)*S.1.s)+ ((n.f2-1)*S.2.f)
denom  <- (n.s1-n.f2-2)
S <- num/denom

#Group Mean
x.bar<- apply(company.10var[,1:10], 2, mean)

 # Classiffication Func & LDF   hard code
LDF<-(t(mean.s1-mean.f2))%*% solve(S)*x.bar #Double CHECK
Function <- ((t(mean.s1-mean.f2))%*%solve(S)*x.bar) -
(0.5*(t(mean.s1-mean.f2))%*%solve(S)*(mean.s1-mean.f2))

#Coefficients of linear discriminant function (LDF)
 LDF.coeff <- t(mean.s1-mean.f2)%*% solve(S)

 #LDF 2nd
 LDF <- LDF.coeff%*%x
 #CDF 2nd
 CDF <- LDF-0.5*LDF.coeff%*%(mean.s1+mean.f2)
list("Linear discriminant
coefficients"=LDF.coeff,LDF=LDF,CDF=CDF,2LDF.Ex.6.2.6.iv.a=LDF,2LDF
.coeff=LDF.coeff)
}
```

```
> Ex.6.2.7.a()
```

```
$`Linear discriminant coefficients`
            X1        X2       X3        X4       X5       X6        X7         X
8         X9        X10
[1,] -6.179782 -11.46436 37.33781 -18.75429 18.84393 6.805857 -16.07244 -0.121394
5 0.02672031 -0.00662445

$LDF
         [,1]
[1,] 29.63856

$CDF
         [,1]
[1,] 29.54862

$LDF.Ex.6.2.6.iv.a
         [,1]
[1,] 29.63856

$LDF.coeff
            X1        X2       X3        X4       X5       X6        X7         X
8         X9        X10
[1,] -6.179782 -11.46436 37.33781 -18.75429 18.84393 6.805857 -16.07244 -0.121394
5 0.02672031 -0.00662445
```

Exercise 6.2.11 (iv)

Theory Behind Ellipsoid (Bivariate case: Ellipse)

i. **Give mathematical expression in matrix notation that describes ellipsoid in p dimensions**

$$for\ all\ \underline{x}:\ \sqrt{\left(\underline{x}-\underline{\mu}\right)'\underline{\Sigma}^{-1}\left(\underline{x}-\underline{\mu}\right)} = c$$

ii.

iii. **Describe the axes in terms of eigen values and eigenvectors.**

The eigenvectors give the directions of the ellipse and the eigenvalues are for drawing the outer bounds of an ellipse.

iv. **Let p = 2. Simplify the expression for the ellipse concerned in terms of scalar quantities.**

When the ellipsoid has p=2 we have a bivariate case , i.e a ellipse.

Here $' = [x_1, x_2]$, $\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12} & \Sigma_{22} \end{bmatrix}$, $\mu' = [\mu_1, \mu_2]$

Hence

$$x_1^2\Sigma_{11} + x_1\ x_2\Sigma_{12} + x_2^2\Sigma_{22} = c$$

```
   function ( mu.vec=c(0,0), covmat=diag(2), c.sq = 1, ...)
{
    par(mar=c(5,3,3,3))

    # Ex.6.2.11.iv Elliptic shape using class notes and Johnson and Wichern
    #Ellipse of form (x-mu.vec)'(covmat^-1)(x-mu.vec) = c.sq
    # This function takes in a vector, the covariance matrix and the c
    # It then draws an ellipse. It only works in bivariate case i.e p=2

    # Restriction 1: covmat must be matrix
    # Restriction 2: covmat must of size 2 x 2
    # Restriction 3: covmat must be symmetrical
    # Restriction 4: covmat must be positive definite
    # Restriction 5: mu.vec is a numeric vector
    # Restriction 6: c.sq is a numeric integer
    mu.vec <- matrix(mu.vec, ncol=1, nrow=2)
    # Restrictions
    if(!is.matrix(covmat)) stop("Cov must be a matrix.")
    if(nrow(covmat) !=2 | ncol(covmat) != 2) stop("covmat must of size 2 x 2")
    if(covmat[1,2] != covmat[2,1] )  stop("covmat must be symmetrical")

    if(covmat[1,1] < 0 | covmat[2,2] < 0 | det(covmat) <= 0) stop("covmat must be
positive definite")

    for (i in 1:length(mu.vec)){
        if(!is.numeric(mu.vec[i])) stop("mu.vec must be a numeric vector.")  }

    if( !c.sq%%floor(c.sq) == 0 | !is.numeric(c.sq)) stop("c.sq must be a numeric
integer value")

    #Starting of formula and plot

    a <- (0:6283)/1000 # 2*pi = 6.238

    Y <- cbind(cos(a), sin(a))*sqrt(c.sq)
    covmat.svd <- svd(covmat)        # Singular Value Decomposition covmat

    Y <- Y%*%diag(sqrt(covmat.svd$d)) #stretch axes differentialy
    Y <- Y%*%t(covmat.svd$v) # rotate in direction of eigenvector
    # associated with largest eigen value

    Y <- Y + matrix(rep(1,6284),ncol=1)%*%t(mu.vec)
    plot(Y, xlim=range(Y[,1]), ylim=range(Y[,2]), type="l", xlab="x1", ylab="x2",
main = paste("Ellipse with ",covmat[1,2]," correlation"),xpd=TRUE)

    # low level plotting to put line on [0,0] horizontal and verticle
    # shows the centre of the matrix as you move around mu
    abline( h=0, col="grey",lty=3)
    abline( v=0, col="grey",lty=3)

}
```

Here , $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mu' = [0,0]$

```
> Ex.6.2.11()
```

Figure 1: Circle, positive Ellipse centred, with c = 1 and no correlation

We take note of how the eigen values and eigen vectors create an ellipse with equal size boundaries here, which is in turn a circle.

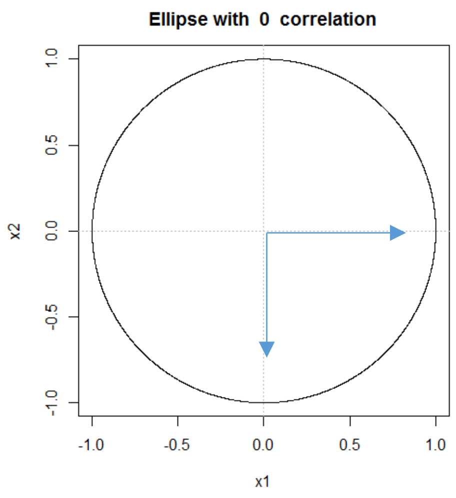
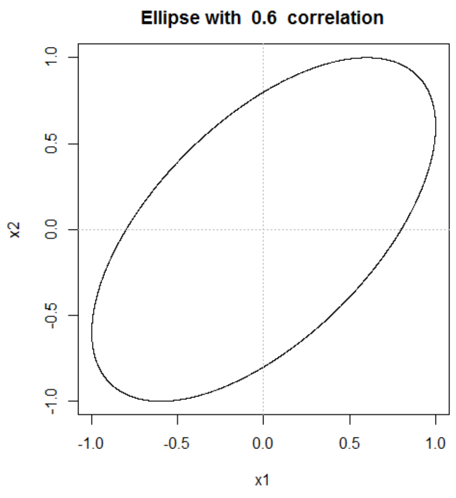


Figure 2: Ellipse centred , with positive correlation 0.6

Here , $\Sigma = \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}$, $\mu' = [0,0]$ c=1

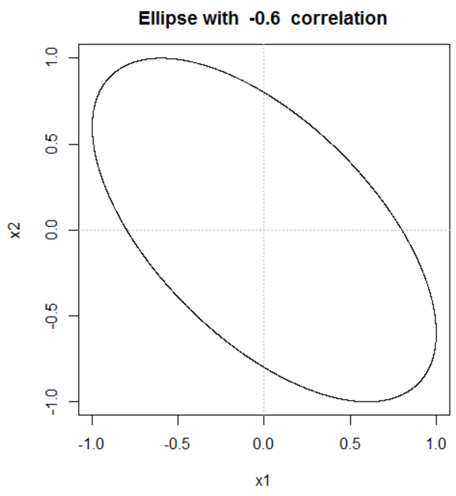
```
> Ex.6.2.11(mu.vec = c(0,0),covmat =
matrix(c(1,0.6,0.6,1),nrow=2),asp=1)
```



<u>Figure 3: Ellipse centred , with negative correlation -0.6</u>

Here , $\Sigma = \begin{bmatrix} 1 & -0.6 \\ -0.6 & 1 \end{bmatrix}$, $\mu' = [0,0]$ c=1

```
> Ex.6.2.11(mu.vec = c(0,0),covmat = matrix(c(1,-0.6,-
0.6,1),nrow=2),asp=1)
```

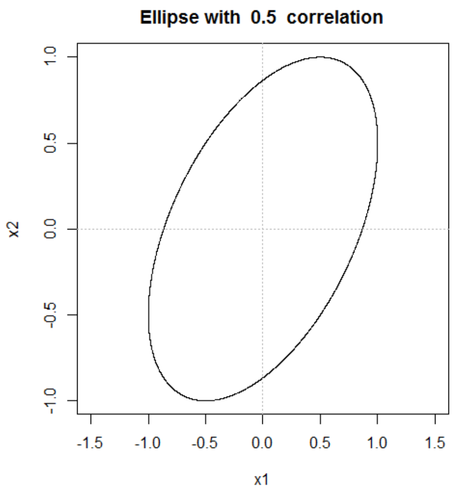


Exercise 6.2.11 (v)

plot(asp=1) vs plot()

<u>Figure 4: Ellipse asp ≠ 1 , i.e default</u>

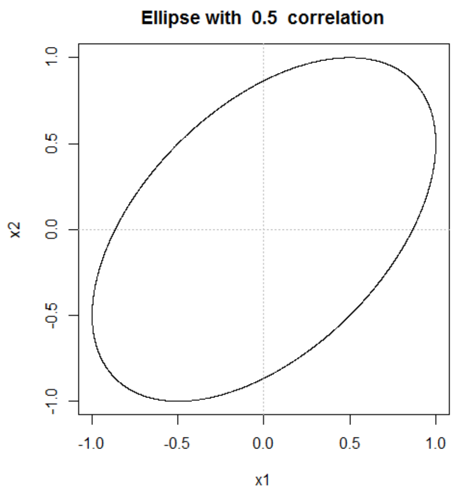
```
> Ex.6.2.11(mu.vec = c(0,0),covmat = matrix(c(1,0.5,0.5,1),nrow=2))
```



We change the equation by adding a asp command to plot and set it equal to 1 .

Figure 5: Ellipse asp = 1

```
Ex.6.2.11(mu.vec = c(0,0),covmat =
matrix(c(1,0.5,0.5,1),nrow=2),asp=1)
```



asp - the y/x aspect ratio
If asp = 1 then the aspect ratio is 1.
We see that the plot has different ratios when the aspect ratio is at a default and not set to one.

Exercise 6.2.12.
This challenge is optional. If you have attempted it but could not get it to work properly, hand in your attempt.

Game

Consider the following game. You are given a computer screen containing a rectangle filled at random with evenly spaced letters. Repetitions of the same letter are allowed. The challenge is to select sequentially the first n letters of the alphabet as quickly as possible. You must read each line from left to right and from top to bottom. Going backwards is not allowed. The time to complete the task is taken as well as if the rules have been obeyed. Program an R version of this game.

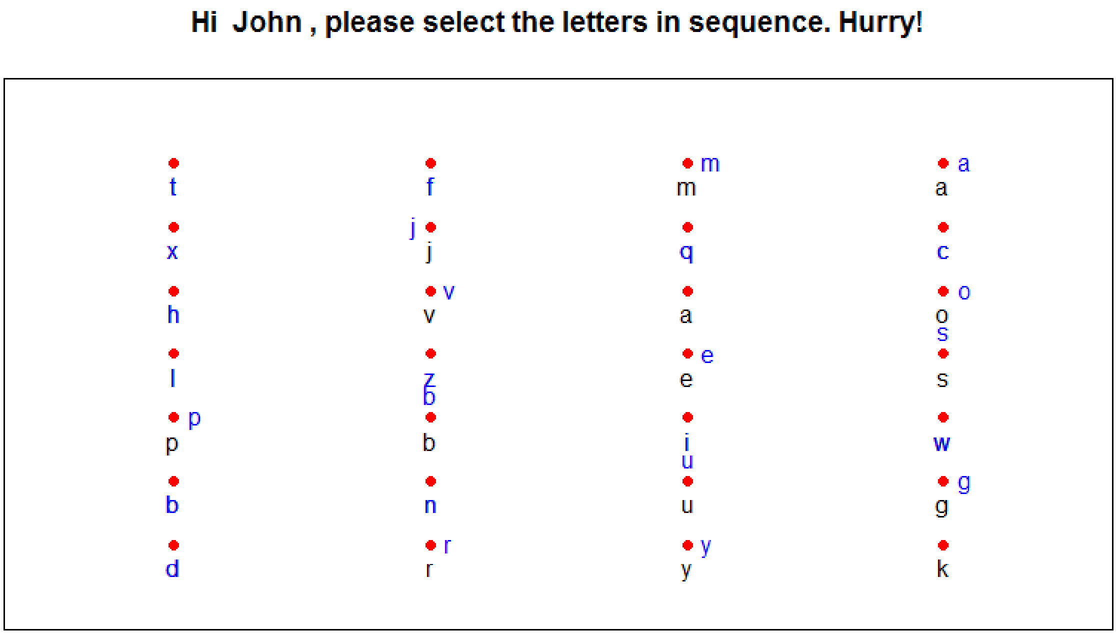
Try this one as best as you can. If you have attempted it but could not get it to work properly, hand in your attempt.

```
fix(Ex.6.2.12)
```

```
function ()
{
    ## Game instructions given to user in console
    print("The goal of this game is to pick the letters on the console in
sequence. You will be timed and the aim is to complete is as soon as
possible. Your time begins after inputting your name.Let the games
begin!!!")

    # Prompt user to enter name and the game begins
    name <- readline(prompt="Enter your name: ")
    print("Your time starts now")
    start.time <- proc.time()

    # Create the empty canvas with the letters and points below, note
that the letters are
    #In a different random order everytime one restarts the game
    plot(c(0, 200), c(0, 400), type= "n", xlab = "", ylab =
"",yaxt="n",xaxt="n", main= paste("Hi ",name,", please select the letters
in sequence. Hurry!"))

    # Create the points to plot and sample with replacement to make the
points random in how they are presented to the user
    # reorder sampling of x values
    # The game uses sampling with replacement to create a new order to
plot the letters which gives the user
    #The illusion of the game creating a new random canvas
    pts <- seq(25,175,50)
    pts.replace <- sample(pts,4,replace = FALSE)
    pts.fin <- rep(pts.replace,7)
    #reorder sampling of y values
    y.val <- seq(50,350,50)
    y.sample <- sample(y.val,7,replace = FALSE)
    y.fin <- rep(y.sample,4)
    # Plot the letters for the user
    points(pts.fin, y.fin,col="red",pch=16)

    text(x=pts.fin,y=y.fin,labels=letters,pos=1)

    temp1 <- identify(x=pts.fin,y=y.fin,
                      labels=letters,n=3,col="lightblue")



#Possible sequences that would result in a Success message once game is
complete
    vec1 <- c(1:26)
    vec2 <- c(27,2:26)
    vec3 <- c(1,28,3:26)
    vec4 <- c(27,28,3:26)


    #Check whether the vector that we get as an answer agrees with the
sucess vectors
    #Print a message to the user congradulating him on sucess, then give
the user his elapsed time
    #Give the user the sequence of his results
    if(temp1==vec1 || temp1==vec2 || temp1==vec3 || temp1==vec4)
        result <- c(paste("Congradulations,",name," completed the game
successfully"))
    else
        result <- c("Unfortunately your selected sequence was incorrect,
please try again")

    end.time <- proc.time()

    time.taken <- end.time[3]-start.time[3]

    list(result = result,time_taken = time.taken,Your_Sequence_Was =
temp1)
}
```

We use the name John and choose the wrong sequence

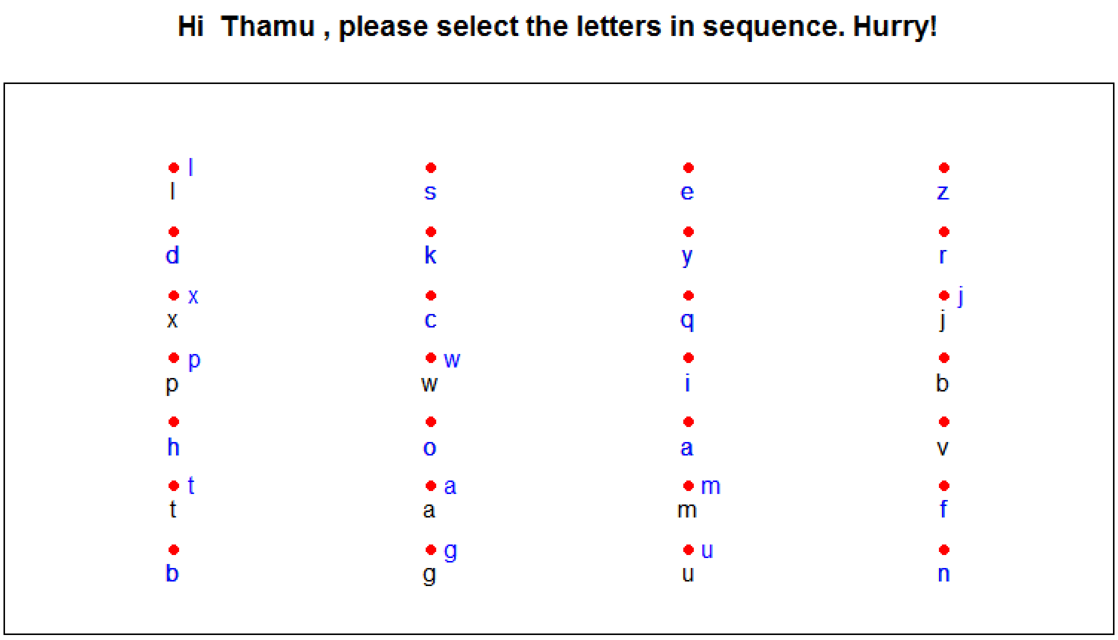
```
> Ex.6.2.12 ()
[1] "The goal of this game is to pick the letters on the console in sequence. Yo
u will be timed and the aim is to complete is as soon as possible. Your time beg
ins after inputting your name. Let the games begin!!!"
Enter your name: John
[1] "Your time starts now"
```



```
$`result`
[1] "Unfortunately your selected sequence was incorrect, please try again"

$time_taken
elapsed
 163.24

$Your_Sequence_Was
 [1]  2  3  4  5  6  7  8  9 10 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28
```

We use the name Thamu and choose the correct sequence

```
> Ex.6.2.12 ()
[1] "The goal of this game is to pick the letters on the console in sequence. Yo
u will be timed and the aim is to complete is as soon as possible. Your time beg
ins after inputting your name. Let the games begin!!!"

Enter your name: Thamu
[1] "Your time starts now"

$`result`
[1] "Congradulations, Thamu  completed the game successfully"

$time_taken
elapsed
 106.76

$Your_Sequence_Was
 [1]  1  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 23 24 25 26 27
28
```



Further Description
The identify function allows us to find the closest observation to add to the sequence, we use this to allow the user to specify which letter he would like to add to the sequence

Exercise 7.4.

```
>fix(x)
```

```
function (x)
#Exercise takes in a numeric vector
#returns mean,median,variance, min, max, coeff. of var for sample data
#as a list
#x: arg is a numeric vector that is checked
{
if(!is.vector(x)) return("Please enter a vector.")
if(!is.numeric(x)) return("Please enter a numeric vector argument.")

#################################
#for loop looks through x and checks if any args are missing
for (i in 1:length(x)){
if(is.na(x[i])) print(cat("Please note, you have missing args.\n This
function manipulates them"))
}
# create a vector containing only the non-missing values in x
x<-x[!is.na(x)]
#################################
mean.x <-mean(x)            #arithmetic mean
median.x<-median(x)         #median
var.x<-var(x)               #variance
min.x<-min(x)               #minimum amount
max.x<-max(x)               #maximum amount
# Note CV is return per unit risk in investments.
cv<-mean.x/(sqrt(var.x))     # measure of dispersion

list(mean=mean.x,median=median.x,var=var.x,max=max.x,min=min.x)
}
```

```
> Ex.7.4(rnorm(1000))
$`mean`
[1] 0.02864813

$median
[1] -0.01334336

$var
[1] 1.017243

$max
[1] 3.644958

$min
[1] -3.253617

$coeff.of.var
[1] 0.02840428

> Ex.7.4(rnorm(1000))[[1]]
[1] -0.01172802
> Ex.7.4(rnorm(1000))$mean #mean
[1] 0.02453634
```

```
> Ex.7.4(rnorm(1000))$median
[1] -0.02220202
> Ex.7.4(rnorm(1000))$median #median
[1] -0.00530339
```

Output in first running of output

Mean	Median	var	max	min	coeff.of.var	SD
0.02864813	-0.01334336	1.017243	3.644958	-3.253617	0.02840428	1.008585

Difference between coeff of variation and variance?

$Variance: Var(X) = E[(X-\mu)^2]$

$$CV: \frac{Mean}{\sqrt{Var(X)}} = \frac{Mean}{Standard\ Deviation}$$

Coefficient of variation CV is the relative standard deviation. I is a standardised measure of dispersion. It is the ratio of the standard deviation to the mean (more correctly the absolute value of the mean because the mean could be negative). It is often used in investments in determining the volatility in the price of an asset as both negative and positive deviations are viewed as risk.

The variance is the standard deviation squared, i.e the squared deviation of a random variable from its mean.

Observing the first value we see the standard deviation at 1.008585 and the coefficient of variation at 0.02840428.

Section 8.1 (f) (v) and (vi).

Section 8.1 (f) .v

Exercise 8.1 (f) (v)

```
#function obtains a table of the minimum and maximum values
#of each column in LifeCycleSavings
#min.vec = column vector of minimums; max.vec = column vector of max
```

```
function ()
{
min.vec<-apply(LifeCycleSavings,2,min)
max.vec<-apply(LifeCycleSavings,2,max)
table<-cbind(min.vec,max.vec)
#rename columns
colnames(table)=c("Minimum","Maximum") # no need to use format, equal size
table
}
```

```
> Ex.8.1.f.v()
      Minimum Maximum
sr       0.60   21.10
pop15   21.44   47.64
pop75    0.56    4.70
dpi     88.94 4001.89
ddpi     0.22   16.71
```

Section 8.1 (f) .vi

```
> fix(Ex.8.1.f.vi)
```

```
function ()
{
#function obtains a table of the minimum and maximum values
#of each column in airquality
# function then calculates the min and max vectors with na omitted.

    min.vec<-apply(airquality,2,min)
    max.vec<-apply(airquality,2,max)
    min.vec.narm<-apply(airquality,2,min,na.rm=T)
    max.vec.narm<-apply(airquality,2,max,na.rm=T)

    table<-cbind(min.vec,max.vec,min.vec.narm,max.vec.narm)

    #rename columns
    colnames(table)=c("Minimum rm.na=F","Maximumrm.na=F","Minimum
rm.na=T","Maximum rm.na=T")

    return(table)
}
```

```
> Ex.8.1.f.vi()
        Minimum rm.na=F Maximum rm.na=F Minimum rm.na=T Maximum rm.na=T
Ozone                NA              NA             1.0           168.0
Solar.R              NA              NA             7.0           334.0
Wind                1.7            20.7             1.7            20.7
Temp               56.0            97.0            56.0            97.0
Month               5.0             9.0             5.0             9.0
Day                 1.0            31.0             1.0            31.0
```

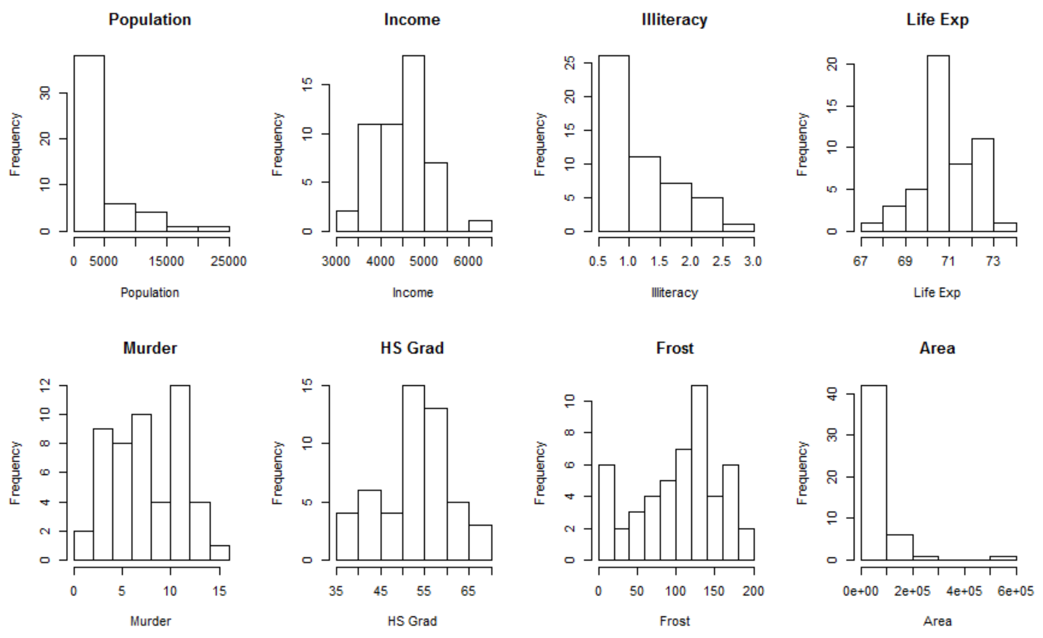
Note:
Most statistical summary functions (mean, var, sum, min, max, etc.) accept the argument na.rm=, which can be set to TRUE if you want missing values to be removed before the summary is calculated.
Above we see that the min function and the max cannot obtain the min and max in the data as the data does not have the na.rm set to false but on default true.

Section 8.2 (c)

```
fix(Ex.8.2.c)
```

```
function ()
{
#function prints histograms of every column

col.names<- dimnames(state.x77)[[2]]       #variable with column names
      #return(col.names), if dimnames was set to 1 ,where its 2, would be
rownames

par(mfrow=c(2,4))                               # 2 rows, 4 columns
lapply(col.names, function(x){hist(state.x77[,x],main=x,xlab=x)}) # apply
function on every column
invisible()             # don't populate screen with unnecessary output
}
```

```
> Ex.8.2.c()
```



Exercise 8.15

```
function (x)
{
# function checks if restrictions for a vector are met and then
# returns a vector of the prime numbers and a frequency table
# of the prime numbers, since one can repeat a prime number
#multiple times.
#The only input is x ; this is the vector to check whether it
#has prime numbers or not.

if(!is.vector(x)) return("Please enter a vector!")

for(i in 1:length(x)){ if( x[i] <= 0 ) return("Please enter a
positive!") }

if(any(x>10^10))stop("All elements must be in range specified")

x <- x[!is.na(x)]  # remove any missing values

vec<-numeric()
for(i in 1:length(x))
{
if(!any((x[i]/(2:(x[i]-1)))%%1==0))
if( x[i] - x[i] %/%1 == 0)
vec[i] <- x[i]
}
if(length(vec)>0)
{
list("Prime numbers in the vector
are:"=sort(vec[!is.na(vec)]),Frequency_Table=
format(table(vec[!is.na(vec)],dnn=c("Prime Numbers")))
}
else  {  return("No prime numbers are in the vector!")  }
}
```

```
> Ex.8.15(c(1:20))
$`Prime numbers in the vector are:`
[1]  3  5  7 11 13 17 19


$Frequency_Table
Prime Numbers
 3  5  7 11 13 17 19
 1  1  1  1  1  1  1
```

```
> Ex.8.15(c(-2,1,2))
[1] "Please enter a positive!"
```

Exercise 9.11.2.

Print a table from the state.x77 data set such that for each variable, an **asterix is placed after the maximum value for that variable.**

The numbers must line up correctly. Pg 158

```
>fix(Ex.9.11.2)
```

```
function(data=state.x77)
 # Goal of function is to identify the largest element in each column and
 #then to add an astrix at the end of each of the max elements in
 #original data

    # requirements include formatted cells
{
    max.vec <- apply(state.x77,2,max)

    #empty matrix
    mat.max <- matrix( rep(" ",length(data)) , ncol=ncol(data))

    max.vec <- apply(state.x77,2,max)

# Find where in state is the max, then replace with astrix
    state.x77[sweep(state.x77,2,max.vec,'==')] <- '*'

    for(i in 1:ncol(data)) #ncol(data)= 8 i.t.c
    {
        data[data[,i]==max.vec[i],i] <-
(paste(c(max.vec[i],"*"),sep="",collapse=""))

    }

# Use format to improve aesthetics through spacing
    data<-data.frame(format(data))
    print(data)
}
```

```
> Ex.9.11.2()
```

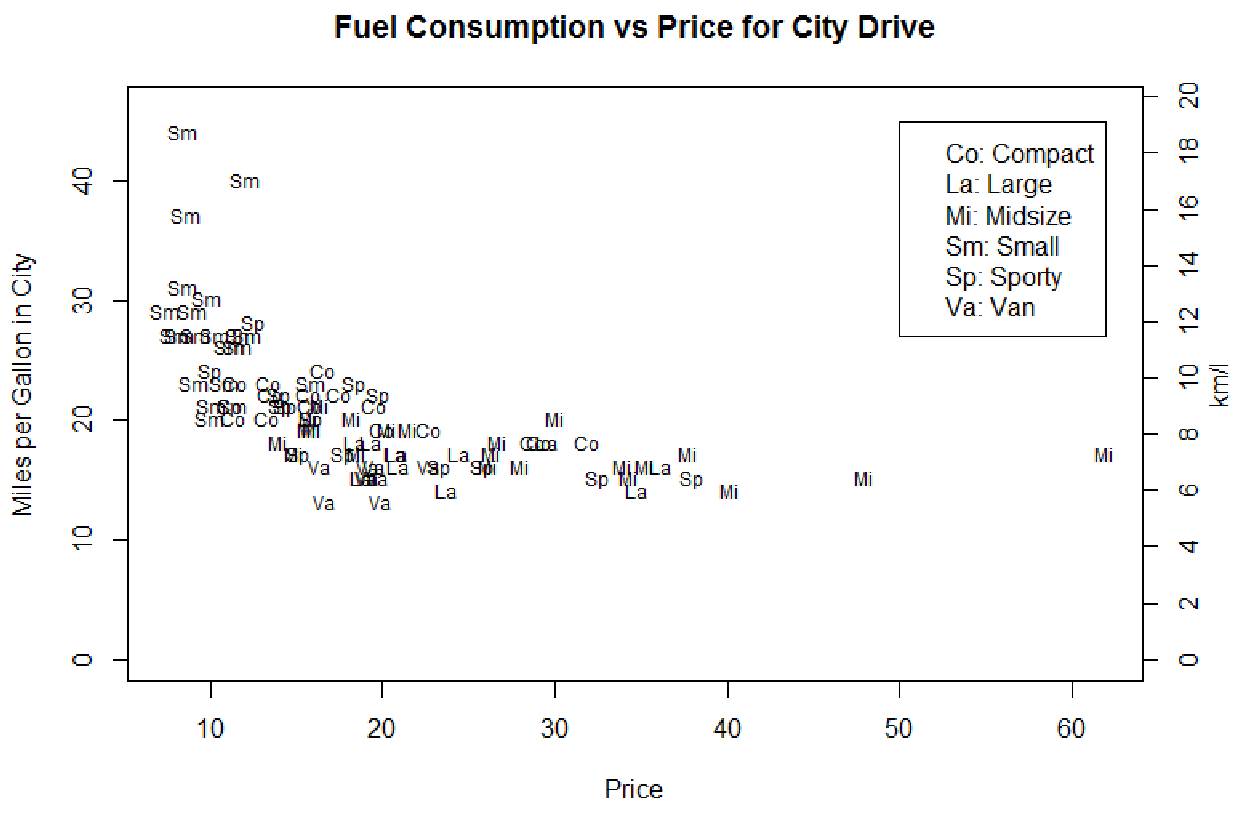
```
> Ex.9.11.2()
```

```
              Population  Income Illiteracy Life.Exp  Murder HS.Grad   Frost    Area
Alabama          3615     3624        2.1      69.05   15.1*   41.3    20      50708
Alaska           365      6315*       1.5      69.31   11.3    66.7    152     566432*
Arizona          2212     4530        1.8      70.55   7.8     58.1    15      113417
Arkansas         2110     3378        1.9      70.66   10.1    39.9    65      51945
California       21198*   5114        1.1      71.71   10.3    62.6    20      156361
Colorado         2541     4884        0.7      72.06   6.8     63.9    166     103766
Connecticut      3100     5348        1.1      72.48   3.1     56      139     4862
Delaware         579      4809        0.9      70.06   6.2     54.6    103     1982
Florida          8277     4815        1.3      70.66   10.7    52.6    11      54090
Georgia          4931     4091        2        68.54   13.9    40.6    60      58073
Hawaii           868      4963        1.9      73.6*   6.2     61.9    0       6425
Idaho            813      4119        0.6      71.87   5.3     59.5    126     82677
Illinois         11197    5107        0.9      70.14   10.3    52.6    127     55748
Indiana          5313     4458        0.7      70.88   7.1     52.9    122     36097
Iowa             2861     4628        0.5      72.56   2.3     59      140     55941
Kansas           2280     4669        0.6      72.58   4.5     59.9    114     81787
Kentucky         3387     3712        1.6      70.1    10.6    38.5    95      39650
```

Louisiana	3806	3545	2.8*	68.76	13.2	42.2	12	44930
Maine	1058	3694	0.7	70.39	2.7	54.7	161	30920
Maryland	4122	5299	0.9	70.22	8.5	52.3	101	9891
Massachusetts	5814	4755	1.1	71.83	3.3	58.5	103	7826
Michigan	9111	4751	0.9	70.63	11.1	52.8	125	56817
Minnesota	3921	4675	0.6	72.96	2.3	57.6	160	79289
Mississippi	2341	3098	2.4	68.09	12.5	41	50	47296
Missouri	4767	4254	0.8	70.69	9.3	48.8	108	68995
Montana	746	4347	0.6	70.56	5	59.2	155	145587
Nebraska	1544	4508	0.6	72.6	2.9	59.3	139	76483
Nevada	590	5149	0.5	69.03	11.5	65.2	188*	109889
New Hampshire	812	4281	0.7	71.23	3.3	57.6	174	9027
New Jersey	7333	5237	1.1	70.93	5.2	52.5	115	7521
New Mexico	1144	3601	2.2	70.32	9.7	55.2	120	121412
New York	18076	4903	1.4	70.55	10.9	52.7	82	47831
North Carolina	5441	3875	1.8	69.21	11.1	38.5	80	48798
North Dakota	637	5087	0.8	72.78	1.4	50.3	186	69273
Ohio	10735	4561	0.8	70.82	7.4	53.2	124	40975
Oklahoma	2715	3983	1.1	71.42	6.4	51.6	82	68782
Oregon	2284	4660	0.6	72.13	4.2	60	44	96184
Pennsylvania	11860	4449	1	70.43	6.1	50.2	126	44966
Rhode Island	931	4558	1.3	71.9	2.4	46.4	127	1049
South Carolina	2816	3635	2.3	67.96	11.6	37.8	65	30225
South Dakota	681	4167	0.5	72.08	1.7	53.3	172	75955
Tennessee	4173	3821	1.7	70.11	11	41.8	70	41328
Texas	12237	4188	2.2	70.9	12.2	47.4	35	262134
Utah	1203	4022	0.6	72.9	4.5	67.3*	137	82096
Vermont	472	3907	0.6	71.64	5.5	57.1	168	9267
Virginia	4981	4701	1.4	70.08	9.5	47.8	85	39780
Washington	3559	4864	0.6	71.72	4.3	63.5	32	66570
West Virginia	1799	3617	1.4	69.48	6.7	41.6	100	24070
Wisconsin	4589	4468	0.7	72.48	3	54.5	149	54464
Wyoming	376	4566	0.6	70.29	6.9	62.9	173	97203

Section 10.4.3 (b).

```
function()
{
# Note :This function converts miles per gallon into kilometres per litre via the
# axis , we move tick marks to allow the first and second y axis to show the
# marks
require(MASS)
oldstate <- par(no.readonly = TRUE)
on.exit(par(oldstate))
graphics.off()

cargrp <- Cars93[ , "Type"]
price <- Cars93[ , "Price"]
mpg.city <- Cars93[ , "MPG.city"]

par(mar=c(5,4,4,4))    # ensure everything is in the plot

plot(price, mpg.city, type = "n", ylim = c(0, max(mpg.city)), main = "Fuel
Consumption vs Price for City Drive", xlab = "Price", ylab = "Miles per Gallon in
City")

char <- substring(as.character(cargrp),1,2)
text(x=price,y=mpg.city, labels=char, pos=1,cex=0.75)
labs <- paste(substring(levels(cargrp),1,2),levels(cargrp), sep=": ")

legend(x=50,y=45, legend=labs) # move legend to position of choice
## Mile per gallon
#1 mile = 1.6093 km  (Note)
#1US gallon =0.83267 UK gallons
#1 US gallon = 3,78541 litres
# 1 mile per gallon = (1.6093/3.78541) km per l (0.42514285)

km.l <- seq(from=0,to=20, by=2)
axis(side=4, at= km.l*(3.78541/1.60934) , labels = km.l)
mtext(side=4, line=2, "km/l") # create axis label and move out

}
```

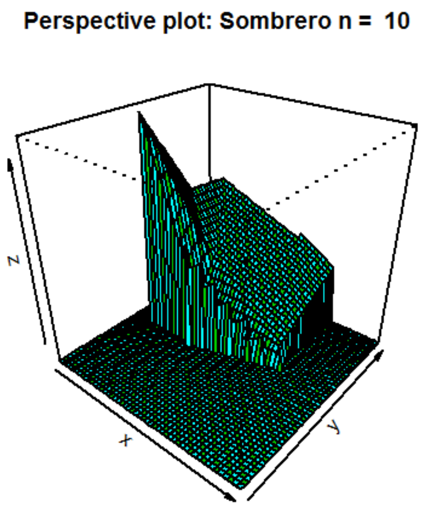


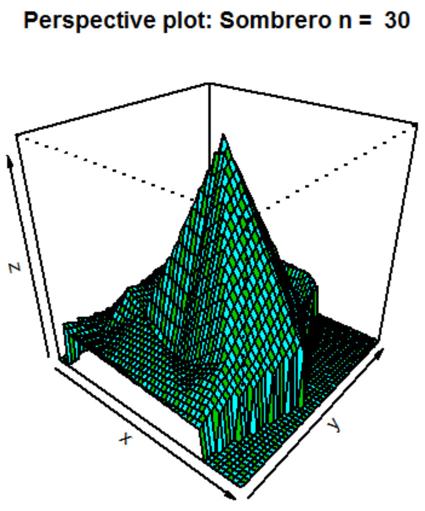
Exercise 10.10.1.

```
fix( Ex.10.10.1)
```

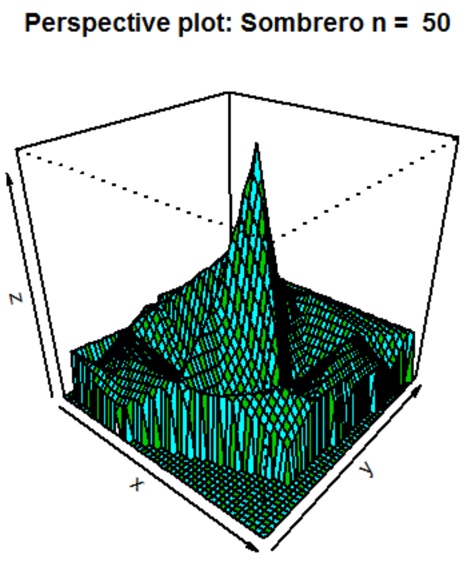
```
 function (n=10,col=c(5,3,5),theta=40,phi=30...)
{
    #Graphic parameters: Function takes in a n sample size and uses it to
#construct a plot of a sombrero
    #Function uses persp to construct a plot
    #We also make use of expand.grid and interp notably.

# cannot take a sample larger than the population when 'replace = FALSE'
    # Restriction
    require(akima)
    pt1 <- seq(from=-10,to=10,length=50)   # x
    pt2 <- pt1   # y
    # The function we plot for our sombrero, function within function
    # takes a x and a y value and creates a zvalue
    zvalue <- function(x,y){
        z <- sqrt(x^2+y^2)
        sin(z)/z
    }

# creates a data frame from all combinations of the supplied vectors or factors.
    # these values then used as coordnates to plot.
    coordinates <- expand.grid(pt1,pt2) # create values of pt1 and pt2 to plot


    sample.coordinates<-coordinates[sample(1:nrow(coordinates),size=n), ]

    zsp <- zvalue(sample.coordinates[,1],sample.coordinates[,2])

    temp <-
interp(sample.coordinates[,1],sample.coordinates[,2],zsp,xo=pt1,yo=pt2)

    zval<-temp$z
    zval[is.na(zval)] <- min(zval[!is.na(zval)])

    #  function draws perspective plots of a surface over the x--y plane
# theta:
# phi
#
persp(pt1,pt2,zval,lty=c(1,2,3),col=col,theta=theta,lwd=c(2,2,2),phi=phi,r=sq
rt(6),xlab="x", ylab="y", zlab="z"
          , main=paste("Perspective plot: Sombrero n = ",n),...)

    invisible() # invisible removes unnecessary output in console
}
```

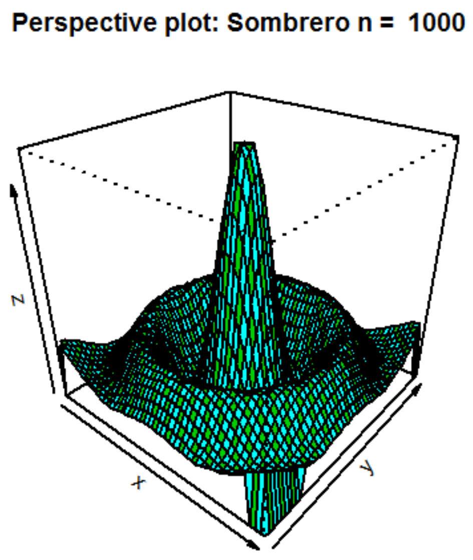
```
> Ex.10.10.1 (n=10)

> Ex.10.10.1 (n=30)
```





```
> Ex.10.10.1 (n=50)

> Ex.10.10.1 (n=1000)
```





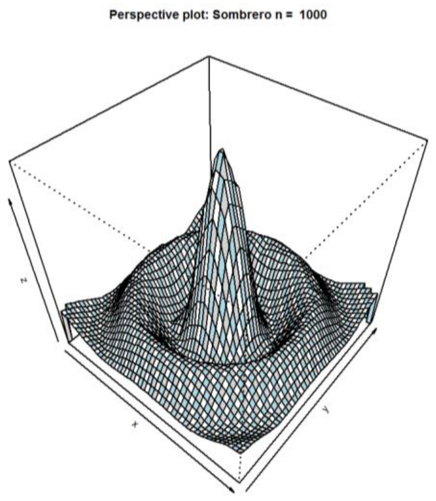
As the sample size increases we see that the perspective plot becomes starts to show the sombrero with more detail. Even when the sample size is 30 we begin to see the sombrero shape.

When the sample size increases it is smoother as it has more observations.

Theta and phi controls the viewing direction and angles.

Below we alter the

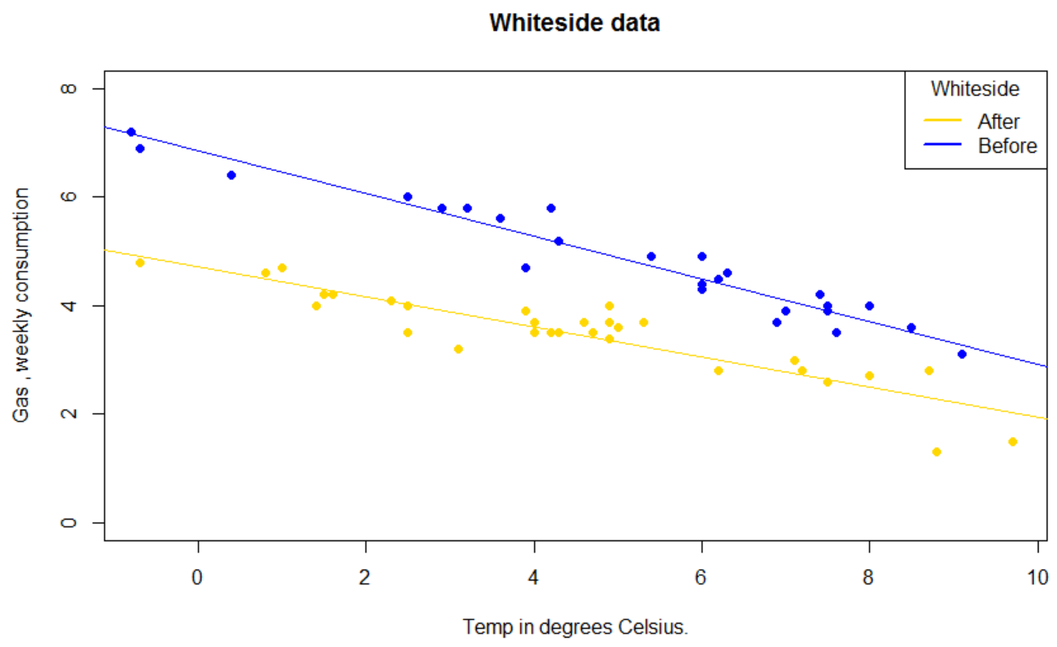
```
> Ex.10.10.1(n=1000,theta=,phi=45,col=c("light grey","white","light blue"))
```



Section 11.7 (d).

Origin on Left

```
function ()
{
# Need MASS package and whiteside data
# detach MASS on exit
require(MASS)
attach(whiteside)
on.exit(detach(whiteside))

## separate data
Insul.after <- whiteside[ whiteside[,1]=="After", ]
Insul.before <- whiteside[ whiteside[,1]=="Before", ]
## plot After points in gold and add axis labels
#  High Level plotting
plot(y=Insul.after[,3],  x=Insul.after[,2], col="gold", pch=16,
ylim=c(0,8),main="Whiteside data", xlab="Temp in degrees
Celsius.",ylab="Gas , weekly consumption")

# Before points in a scatter plot,
# low level plotting to add Before in blue points
par(new=T)

points(y=Insul.before[,3], x=Insul.before[,2], col="blue", pch=16)

# fit linear models for After and Before
lm1 <- lm(Insul.after[,3] ~ Insul.after[,2] )
lm2 <- lm(Insul.before[,3] ~ Insul.before[,2])

#low level plotting to add simple regression lines After and Before
abline(lm1,col="gold")
abline(lm2,col="blue")

legend("topright", legend=c("After","Before"),
col=c("gold","blue"),
lty=c(1,1),lwd=c(2,2),title = "Whiteside")

}
```



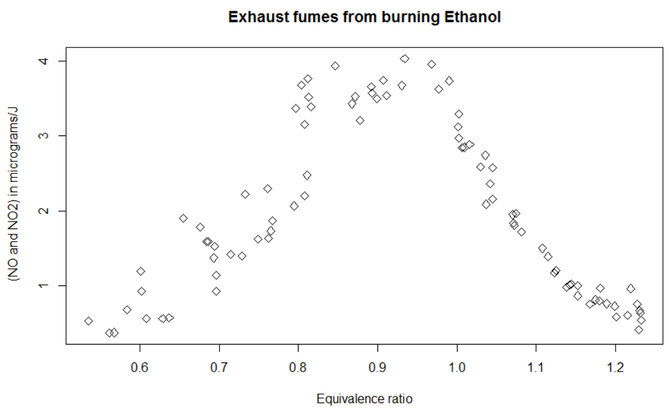
Interpretation:

We can see that the Before (before cavity-wall insulation was installed) has a higher intercept and a steeper decreasing slope in its regression line than the After (after cavity-wall insulation was installed) line. This shows us that the weekly consumption of gas decreased thanks to the installation of insulation. However gas consumption decreases at a faster rate as temperature increases before insulation rather than after insulation.

The object of the exercise was to assess the effect of the insulation on gas consumption.

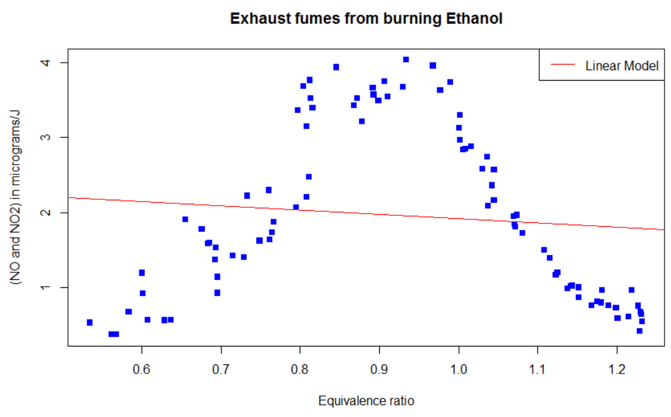
Section 11.9 (d). i

```
function ()
{
require(lattice)
attach(ethanol)

plot(x=E ,y=NOx, main="Exhaust fumes from burning Ethanol", xlab =
"Equivalence ratio", ylab = "(NO and NO2) in micrograms/J")
    on.exit(detach(ethanol))

}
```



Section 11.9 (d). ii

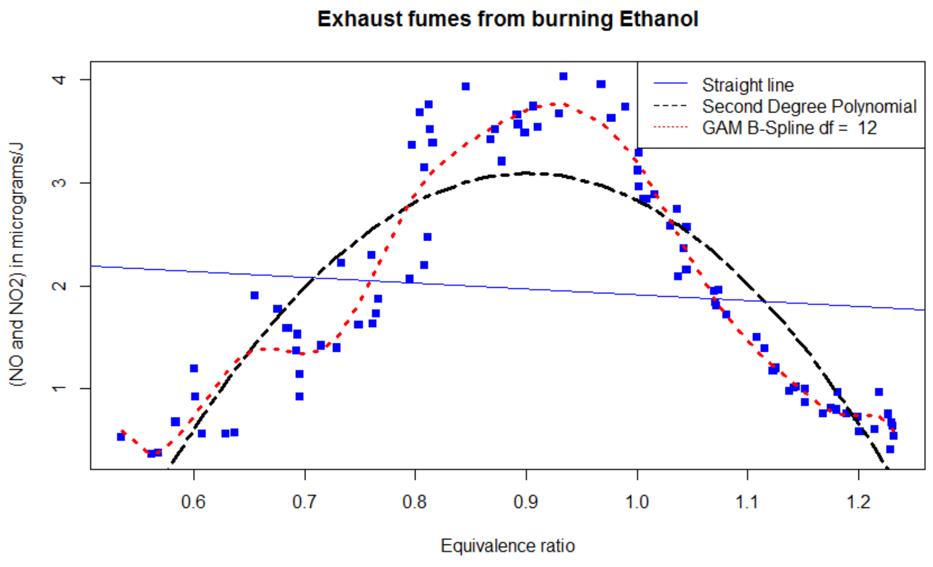
```
function ()
{
   require(lattice)

   attach(ethanol)
   on.exit(detach("ethanol"))

   plot(x=E ,y=NOx, main="Exhaust fumes from burning Ethanol", xlab
= "Equivalence ratio", ylab = "(NO and NO2) in
micrograms/J",col="blue",pch=15)
    fit.1 <- lm(NOx ~ E ,data=ethanol)
   abline(fit.1, col="red")
   legend("topright",legend=c("Linear Model"), lty = 1, col="red")
}
```



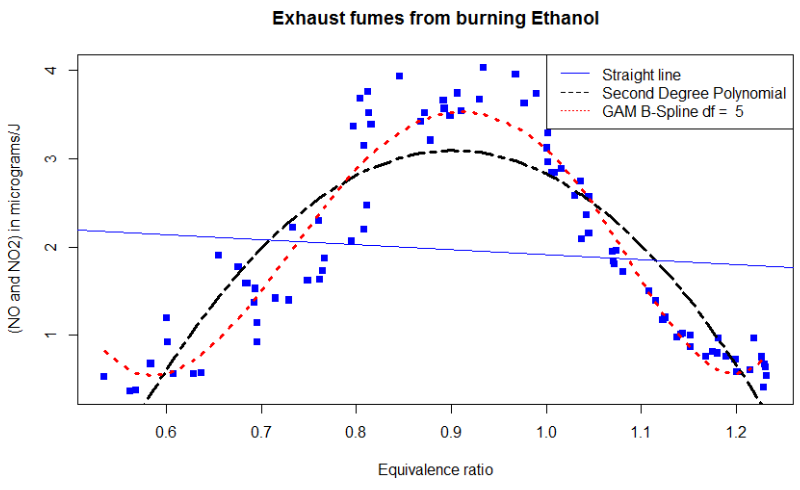
Section 11.9 (d). iii

Note: Need to order the x variable first before fitting it for the plot to work.

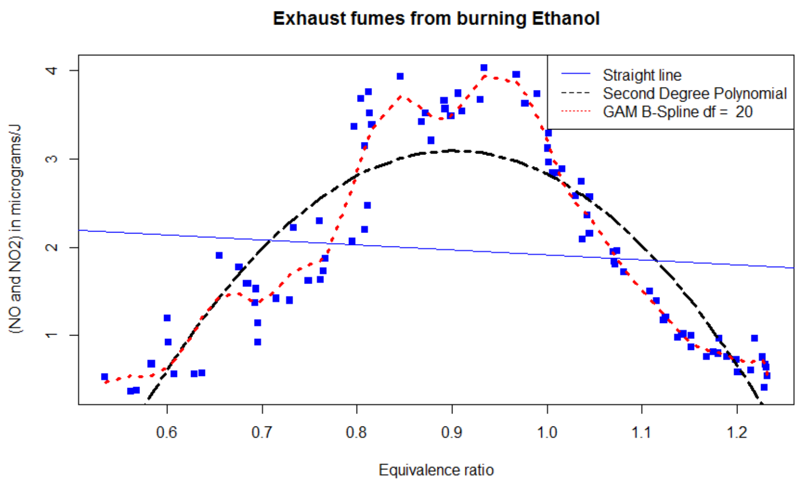
```
function (df1=12)
{
# Add required packages and attach ethanol data set,
# also remove it on exit
#argument df1 is degrees of freedom of GAM B-spline, set to 12 at default

require(lattice)
require (mgcv)
require(splines)

attach(ethanol)
on.exit(detach("ethanol"))

# (High Level plotting) Plot points and canvas of exhaust fumes from
# burning Ethanol
plot(x=E ,y=NOx, main="Exhaust fumes from burning Ethanol", xlab =
"Equivalence ratio", ylab = "(NO and NO2) in
micrograms/J",col="blue",pch=15)

# models for straight line and polynomial
lm.1 <- lm(NOx ~ E ,data=ethanol)
ethanol.lm2 <- lm(NOx ~ E + I(E^2) ,data=ethanol)

# the data from the models
#data.lm.1 <- cbind(E,fitted(lm.1))
data.lm2 <- cbind(E,fitted(ethanol.lm2))

# ordering the data
#data.lm.1 <- data.lm.1[order(data.lm.1[,1]) ,]
data.lm2.o <- data.lm2[ order(data.lm2[,1]) ,]

#Plot the lines of the straight line and second degree polynomial
#lines(x=data.lm.1[,1] , y=data.lm.1[,2],lty=3)
lines(x=data.lm2.o[,1] , y=data.lm2.o[,2],lty=6,col="black",lwd=3 )

abline(lm.1, col="blue")              #Straight line

# creating the plot after with the the gam
require (mgcv)
require(splines)

#df is degrees of freedom , df1 = 12
ethanol.gam <- mgcv::gam(NOx ~ bs(E, df=df1), data=ethanol)
eth <- cbind( E , fitted(ethanol.gam) )
eth.o <- eth[ order(eth[ ,1]) , ]
lines( x=eth.o[ ,1], y=eth.o[ ,2], lty=3,col="red", lwd=3)

#
legend("topright",legend=c("Straight line","Second Degree
Polynomial","GAM B-Spline")
     ,col=c("blue","black","red"), lty=c(1,2,3))
}
```



```
> Ex.11.9(df=5)
```



```
> Ex.11.9(df=20)
```



Above we see the straight line as fitting poorly as the data does not follow a linear pattern. We move to another parametric measure, the second degree polynomial which fits better than the straight line as the data does follow a similar pattern. However we do not conclude this is the best fit.

From observation we conclude that the best model is the nonparametric GAM B-spline as it is far more flexible and follows the data well. The degrees of freedom is a quantity that summarizes the flexibility of a curve. A more restricted and hence smoother curve has fewer degrees of freedom. We consider the GAM B-spline at different degrees of freedom and conclude that the GAM B-spline at a degree of freedom of 20 is overfitting as it starts to model the error term in the data.

From observation we conclude that the best model is the GAM B-spline at 12 degrees of freedom.