Final Exam

Arun Ram Sankaranarayanan

December 14, 2015

1. The five main Rs of EDA are

1. Resistance
2. Residuals
3. Re-expression
4. Revelations
5. Reiteration

2.

There are two functions that are used to calculate the five number summary in R 1.fivenum() 2.Summary()

```
dat = seq(1, 9, by = 2)

fivenum(dat)
```

```
## [1] 1 3 5 7 9
```

```
summary(dat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       3       5       5       7       9
```

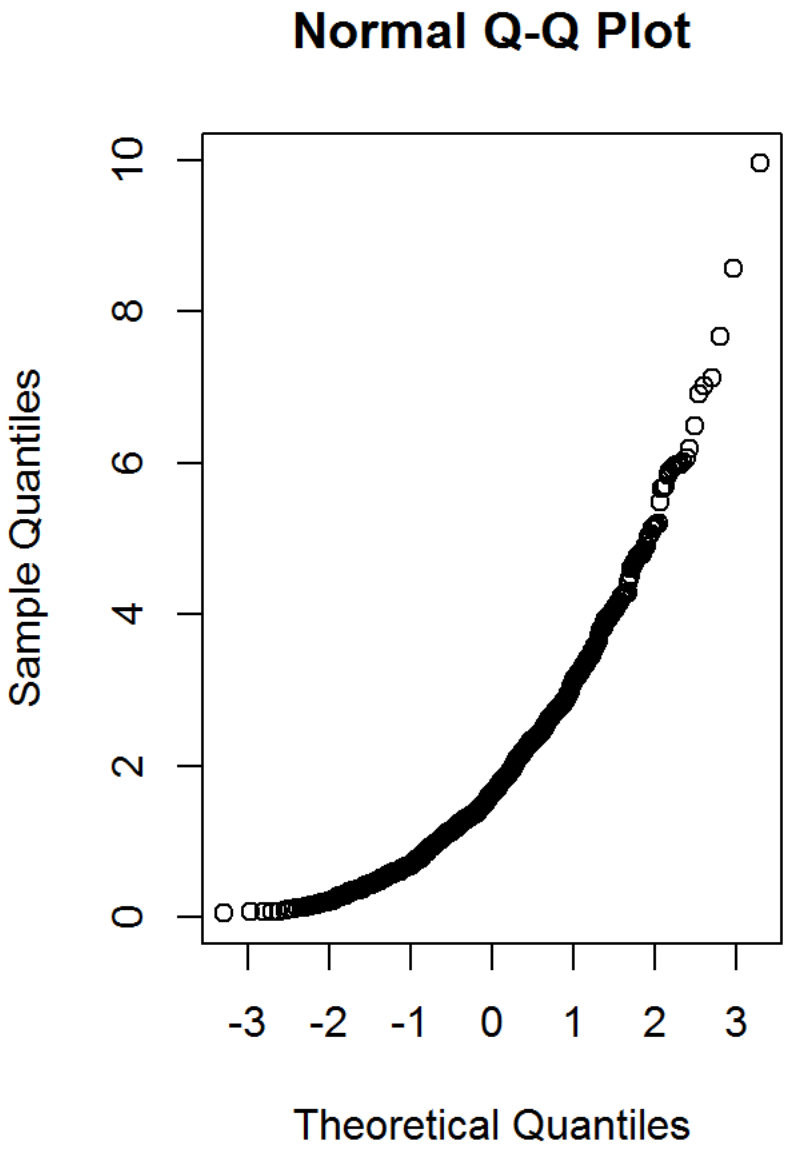
3.

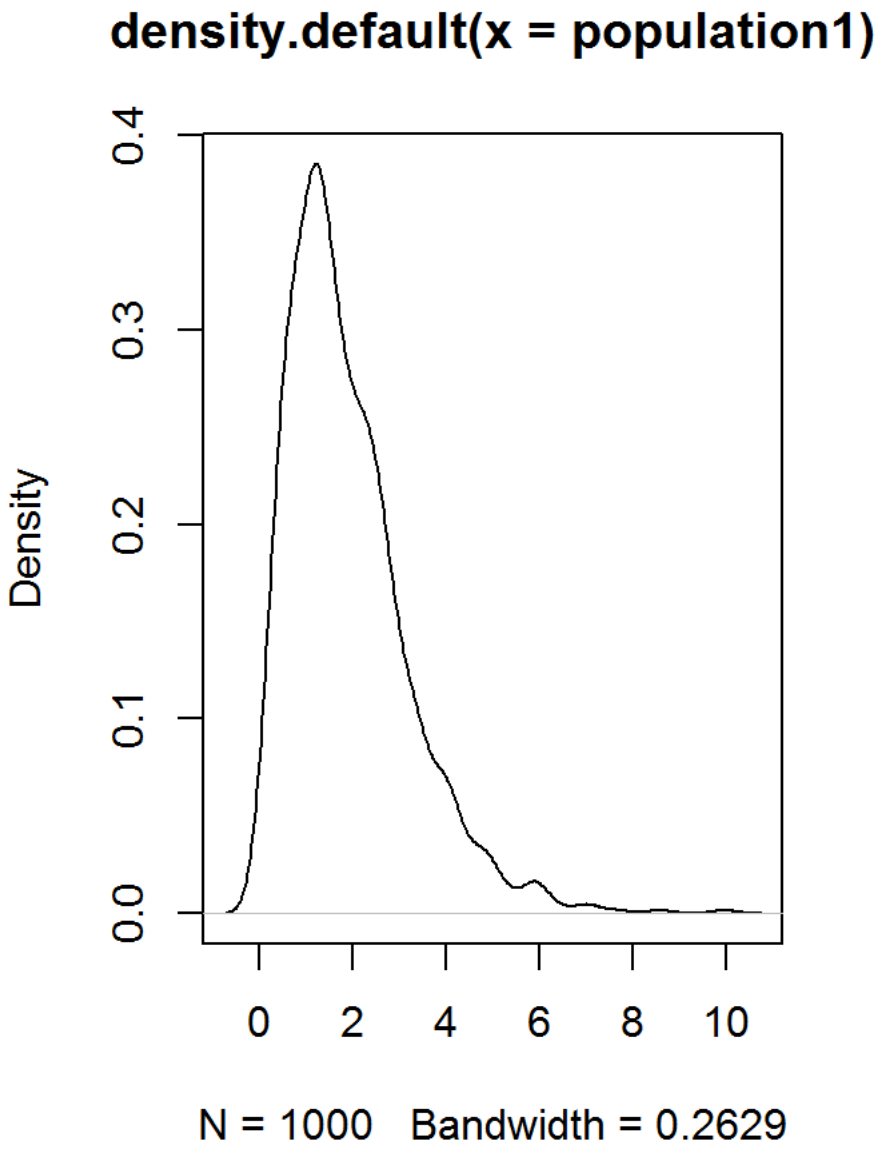
1. TO make the distribution more symmetric - We can easily summarice the center of the distribution that way
2. To make spread of several groups more alike - That way we can compare groups that share common spread 3.To transform the data to normality
3. Detect outliers

4.

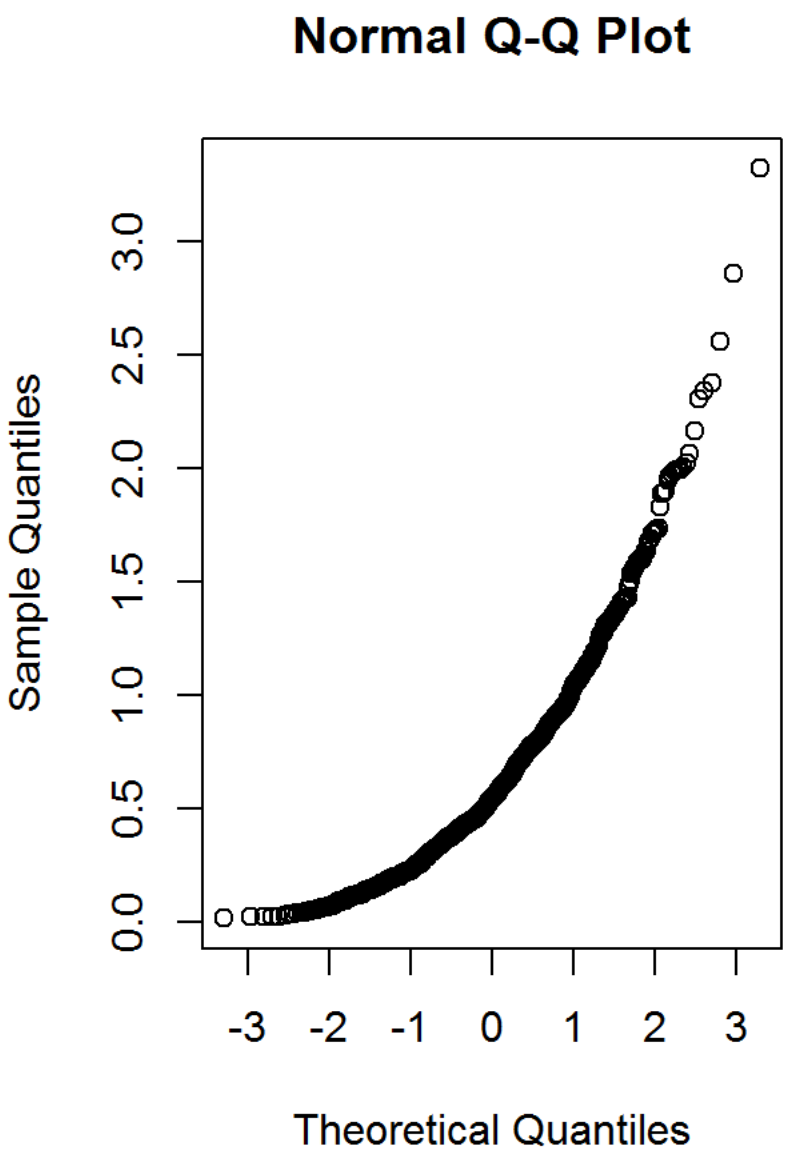
Generate a tailed distribution

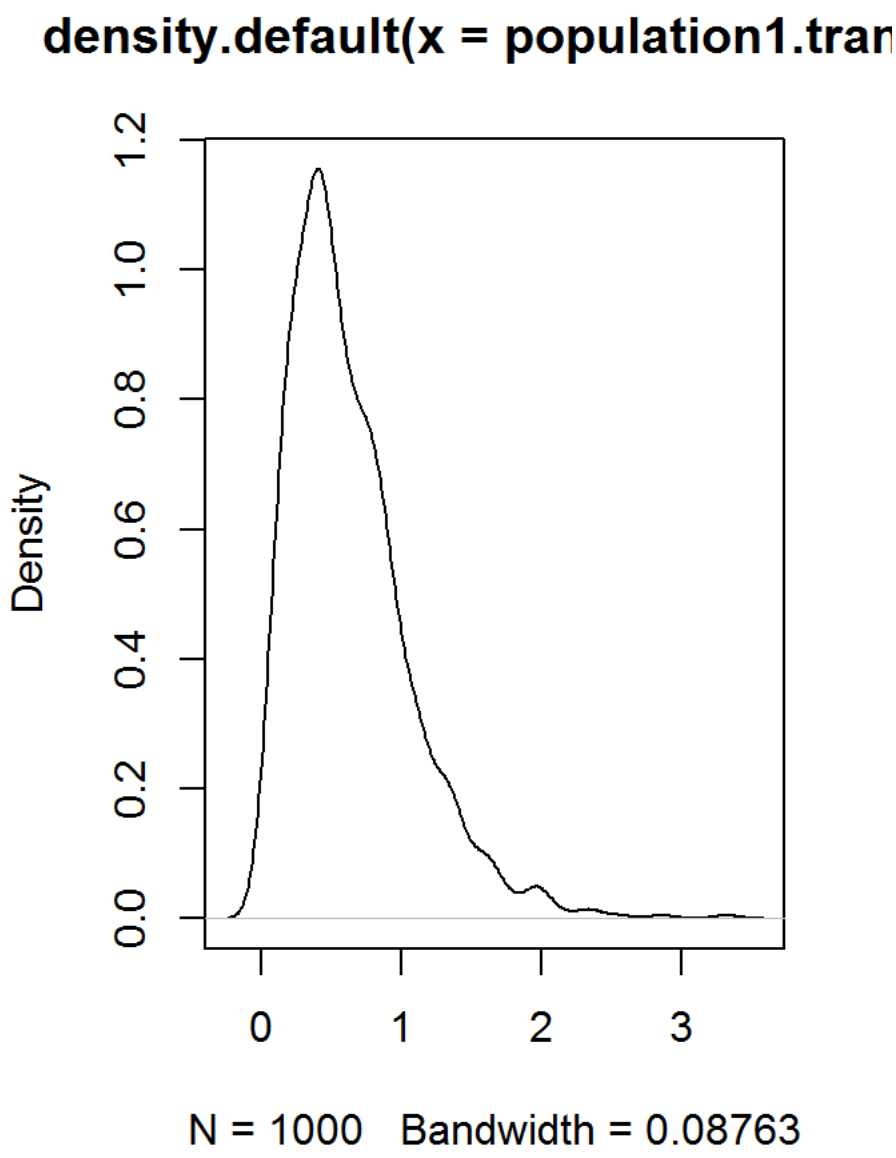
```
population1<-rgamma(1000,shape=2)
par(mfrow=c(1,1),mfcol=c(1,2))

qqnorm(population1)
plot(density(population1))
```






```
population1.trans<-(population1)^1/3

qqnorm(population1.trans)
plot(density(population1.trans))
```





From the qq plot

we can see the long tailness of data also with the help of gh estimators the heavy tailness can be detected

5.

For g-H distributions

when g=0, h=0 Gaussian Data #No skewness #No Long Tails when g<0.25,h>0 slight skewness with Long Tail when gâ¼1,h>0 very skewed with Long tail

The given gh estimates are (-0.5,0.3) , (0.5,0.3), (1,0.6)

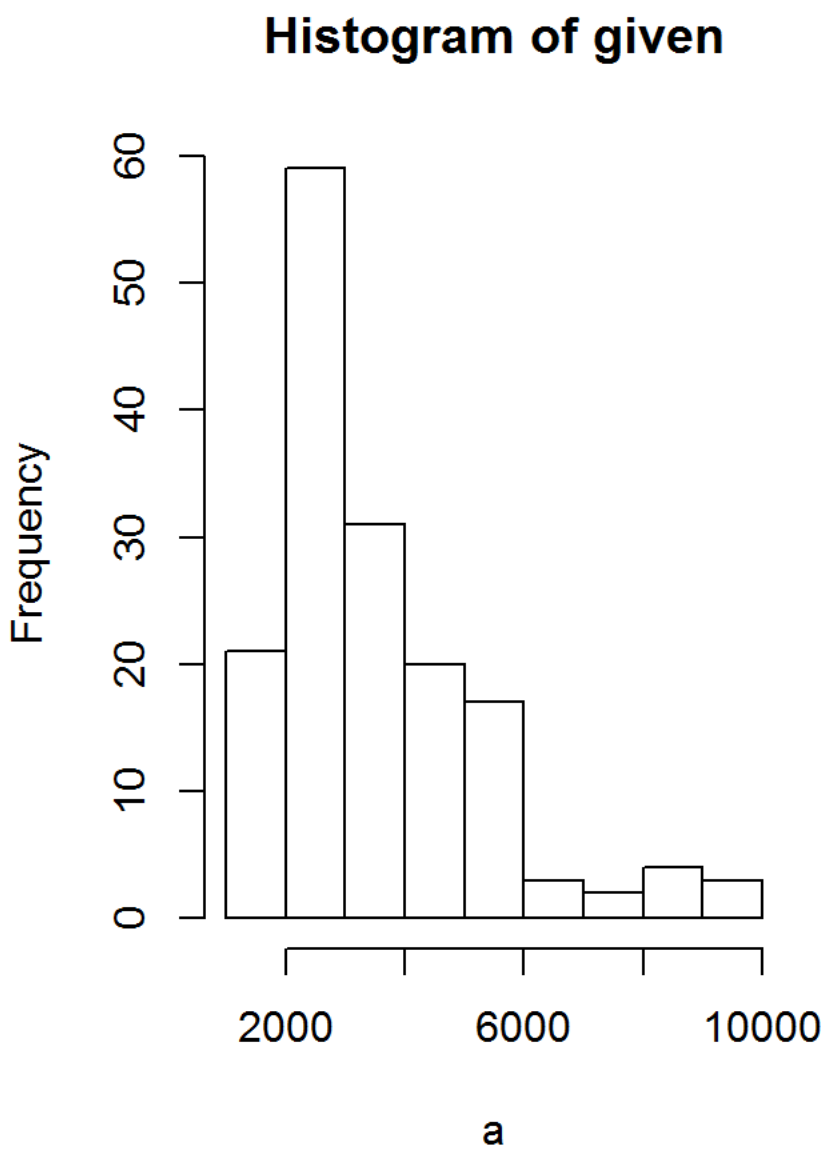
(g,h) -> (-0.5,0.3) is left skewed and has moderate tails (g,h) -> (0.5,0.3) is right skewed and has moderate tails (g,h) -> (1,0.6) is highly right skewed and has heavy tails

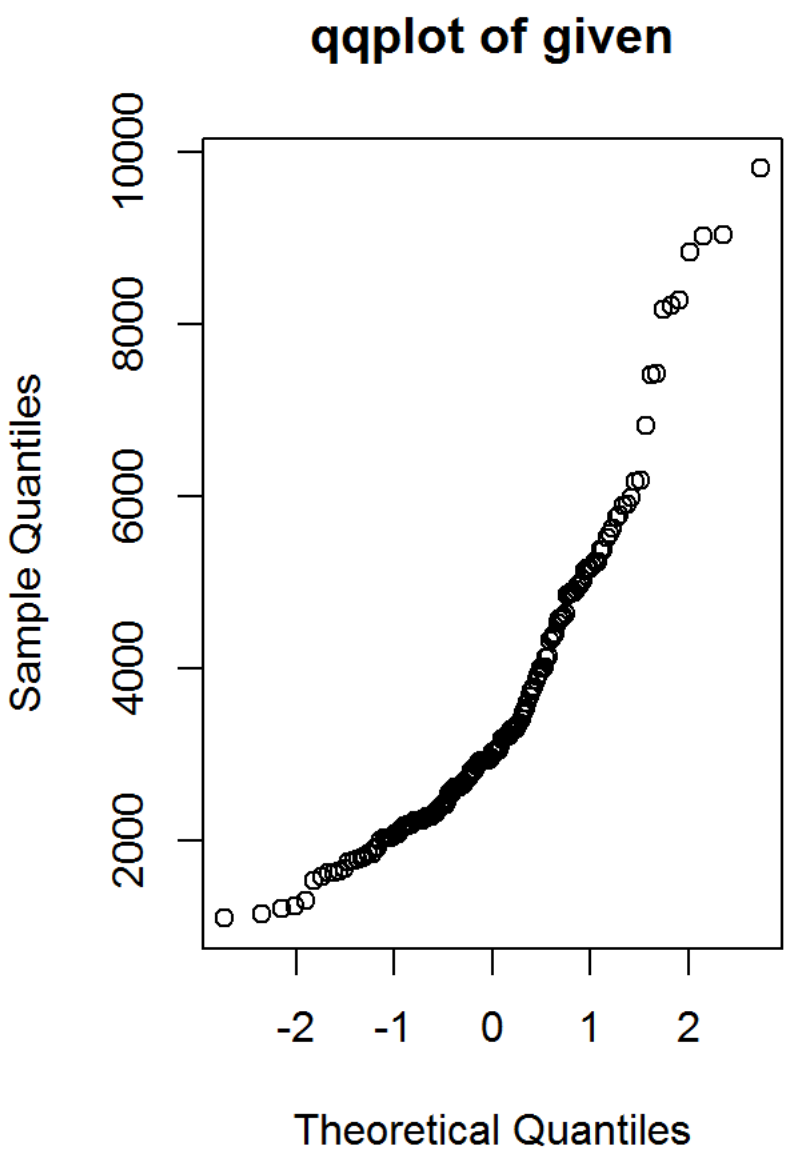
6.

Given

```
a=c(1092,1137,1197,1237,1301,1523,1577,1619,1626,1644,
    1672,1748,1768,1780,1796,1816,1843,1844,1902,1919,
    1993,2025,2028,2032,2036,2072,2078,2090,2137,2162,
    2163,2180,2185,2194,2225,2230,2233,2234,2235,2265,
    2270,2274,2281,2289,2319,2322,2357,2381,2398,2421,
    2421,2443,2522,2549,2552,2581,2618,2618,2620,2624,
    2642,2647,2666,2705, 2721,2740,2804,2819,2823,2860,
    2873,2906,2913,2926,2929,2931,2931,2934,2939,2961,3020,3023,3044,3047,
    3048,3096,3174,3190,3199,3204,3222,3225,3278,3287,
    3292,3300,3339,3361,3412,3462,3503,3530,3589,3672,
    3734,3749,3783,3854,3901,3932,3995,4001,4006,4118,
    4134,4320,4346,4385,4401,4522,4565,4581,4593,4629,
    4855,4868,4878,4885,4907,4962,4975,5021,5127,5155,
    5160,5183,5229,5242,5379,5383,5513,5555,5619,5755,
    5774,5890,5899,5988,6161,6185,6818,7406,7419,8175,
    8220,8282,8827,9027,9042,9805)


#a. Construct a QQ plot
```

```
plot.new()
par(mfrow=c(1,2))


#Constructing Histogram
hist(a,main="Histogram of given")

#constructing qq plot


qqnorm(a, main="qqplot of given")
```






```
#b use g distribution and estimate A,B,g for these data
```

```
getwd()
```

```
## [1] "C:/Users/Arun Ram/Documents/R/mid term/Exam/Final"
```

```
source("lvalprogs.r")



#lettevalueplot
lvp<-lval(a)

n<-length(a)

print(lvp)
```

```
##   Depth  Lower  Upper     Mid Spread pseudo-s
## M  80.5 2990.5 2990.5 2990.50    0.0    0.000
## F  40.5 2267.5 4543.5 3405.50 2276.0 1687.201
## E  20.5 1956.0 5448.0 3702.00 3492.0 1517.800
## D  10.5 1658.0 6501.5 4079.75 4843.5 1578.592
## C   5.5 1412.0 8251.0 4831.50 6839.0 1835.745
## B   3.0 1197.0 9027.0 5112.00 7830.0 1817.655
## A   2.0 1137.0 9042.0 5089.50 7905.0 1634.914
## Z   1.5 1114.5 9423.5 5269.00 8309.0 1561.803
## Y   1.0 1092.0 9805.0 5448.50 8713.0 1509.720
```

```
a1 <- 1/2^(1:9)
g1 <- abs(qnorm(a1))
```

```
a2 <- (lvp[,1]-1/3)/(n + 1/3)
g2 <- abs(qnorm(a2))

e2.g <- log((lvp[,3] - lvp[1,2])/(lvp[1,2]-lvp[,2]))/g2


plot(1:(dim(lvp)[1]-1), e2.g[-1],
     xlab="Letter value(1=F, 2=E, 3=D, ... , 8=Y)",
     ylab="g estimate")
abline(h=median(e2.g[-1]))



#g estimate

e.g <- median(e2.g[-1])


cat("Estimate of g:",e.g)
```

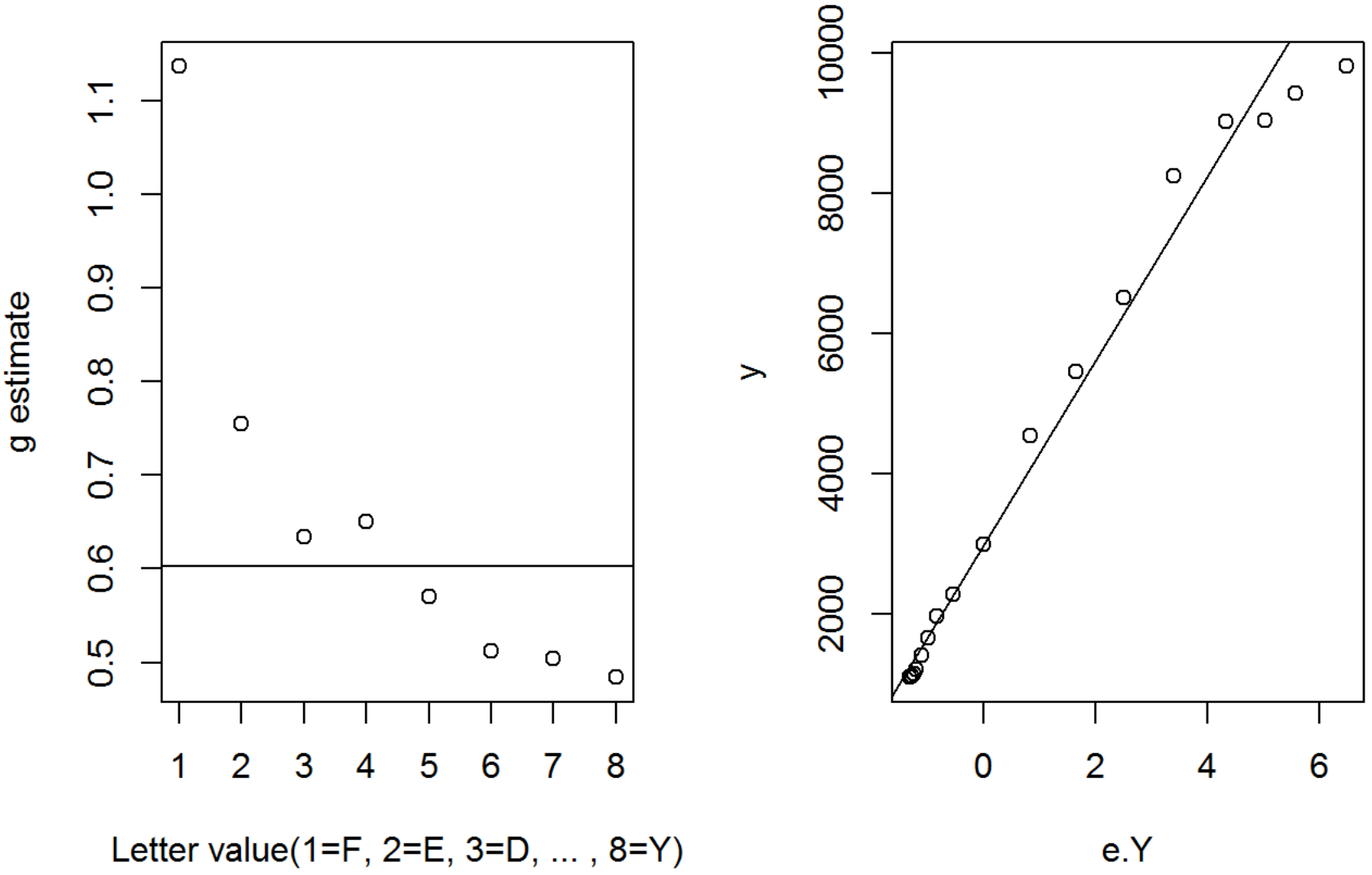
```
## Estimate of g: 0.6023315
```

```
#estimate a and b


source ("rrline.r")


pol <- c(rev(qnorm(a2)),abs(qnorm(a2)))
```

```
e.Y <- (exp(e.g*pol)-1)/e.g

y<-c(rev(lvp[,2]),lvp[,3])
plot(e.Y,y)


rline<-run.rrline(e.Y,y)

abline(rline$a, rline$b)


cat("G estimate =",e.g, ",A estimate=" , rline$a, " and B estimate =" ,rline$b)
```

```
## G estimate = 0.6023315 ,A estimate= 2981.679  and B estimate = 1317.049
```

```
#c Bootstrap estimates


library(ggplot2)
```

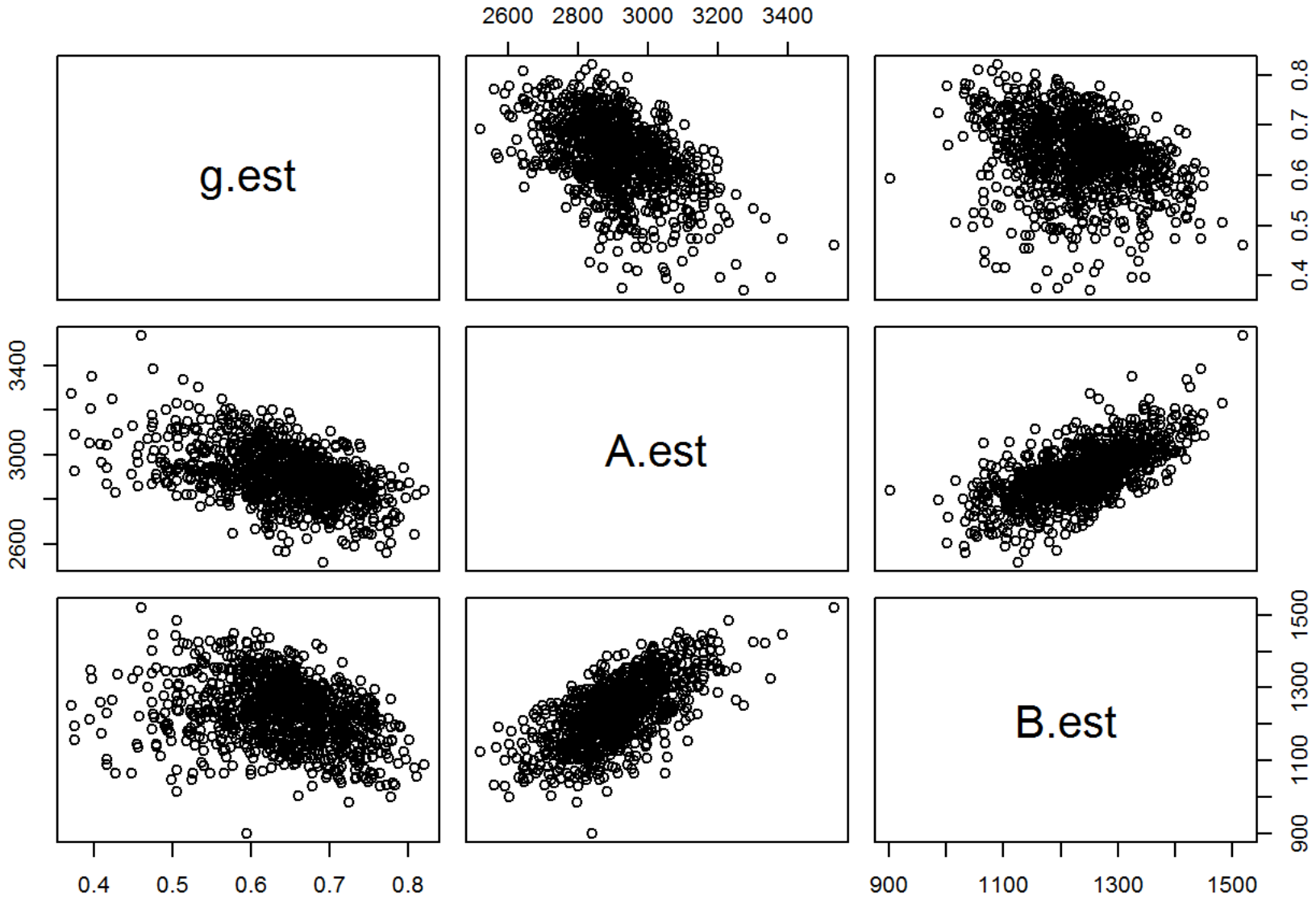



```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 3.2.3
```

```
g.dist.estimates<-function(sample.pop){
  source("lvalprogs.r")
  source("rrline.r")
  ll<-lval(sample.pop)
  pp1 <- 1/2^(1:nrow(ll)-1)
  gau1 <- abs(qnorm(pp1))
  pp2 <- abs((pp1-1/3)/(nrow(ll)-1 + 1/3))
  gau2 <- abs(qnorm(abs(pp2)))
  est2.g <- log((ll[,3] - ll[1,2])/(ll[1,2]-ll[,2]))/gau2

  # Estimation of g
  est.g <- median(est2.g[-1])
  p <- c(0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.975, 0.99, 0.995)
  zp <- qnorm(p)
  est.Y <- (exp(est.g*zp)-1)/est.g
  rr <- run.rrline(est.Y,quantile(sample.pop,p))
  #Run Resistant Regression for A and B Estimates
  return (list(g=est.g,A=rr$a,B=rr$b))
}


bootstrap.g<-function(pops,sims){
  g.est <- c()
  A.est <- c()
  B.est <- c()
  for (i in 1:sims){
    boot.sample<-sample(pops,length(pops),replace = TRUE)
    r.val <- g.dist.estimates(boot.sample)
    g.est[i]<-r.val$g
    A.est[i]<-r.val$A
```

```
    B.est[i]<-r.val$B
  }
  best.g <- mean(g.est)
  g.lower <- best.g - qt(0.975,df=length(g.est)-1)*sd(g.est)
  g.upper <- best.g + qt(0.975,df=length(g.est)-1)*sd(g.est)
  best.A <- mean(A.est)
  A.lower <- best.A - qt(0.975,df=length(A.est)-1)*sd(A.est)
  A.upper <- best.A + qt(0.975,df=length(A.est)-1)*sd(A.est)
  best.B <- mean(B.est)
  B.lower <- best.B - qt(0.975,df=length(B.est)-1)*sd(B.est)
  B.upper <- best.B + qt(0.975,df=length(B.est)-1)*sd(B.est)
  cor.est <- cor (cbind(g.est,A.est,B.est))
  es.plt <- pairs(as.data.frame(cbind(g.est,A.est,B.est)))
  return(list(g=best.g,a=best.A,b=best.B,g.lower,g.upper,A.lower,A.upper,B.lower,B.upper,cor.es
t,es.plt))
}
a<-c(1092,1137,1197,1237,1301,1523,1577,1619,1626,1644,1672,1748,1768,1780,1796,1816,1843,1844,
1902,1919,1983,1993,2025,2028,2032,2036,2072,2078,2090,2137,2162,2163,2180,2185,2194,2225,2230,
2233,2234,2235,2265,2270,2274,2281,2289,2319,2322,2357,2381,2398,2421,2421,2443,2522,2549,2552,
2581,2618,2618,2620,2624,2642,2647,2666,2705,2721,2740,2804,2819,2823,2860,2873,2906,2913,2926,
2929,2931,2931,2934,2939,2961,3020,3023,3044,3047,3048,3096,3174,3190,3199,3204,3222,3225,3278,
3287,3292,3300,3339,3361,3412,3462,3503,3530,3589,3672,3734,3749,3783,3854,3901,3932,3995,4001,
4006,4118,4134,4320,4346,4385,4401,4522,4565,4581,4593,4629,4855,4868,4878,4885,4907,4962,4975,
5021,5127,5155,5160,5183,5229,5242,5379,5383,5513,5555,5619,5755,5774,5890,5899,5988,6161,6185,
6818,7406,7419,8175,8220,8282,8827,9027,9042,9805)
bs.val <- bootstrap.g(a,1000)
```




```
print(paste("The g Estimate is ",bs.val[1]))
```

```
## [1] "The g Estimate is  0.640785440184216"
```

```
print(paste(" and Confidence interval is between",bs.val[4]," and ",bs.val[5]))
```

```
## [1] " and Confidence interval is between 0.494667472879012  and  0.786903407489421"
```

```
print(paste("The A Estimate is ",bs.val[2]))
```

```
## [1] "The A Estimate is  2911.14232974624"
```

```
print(paste(" and Confidence interval is between",bs.val[6]," and ",bs.val[7]))
```

```
## [1] " and Confidence interval is between 2673.35085809729  and  3148.93380139518"
```

```
print(paste("The B Estimate is ",bs.val[3]))
```

```
## [1] "The B Estimate is  1233.91371501814"
```

```
print(paste(" and Confidence interval is between",bs.val[8]," and ",bs.val[9]))
```

```
## [1] " and Confidence interval is between 1063.50010183516  and  1404.32732820111"
```

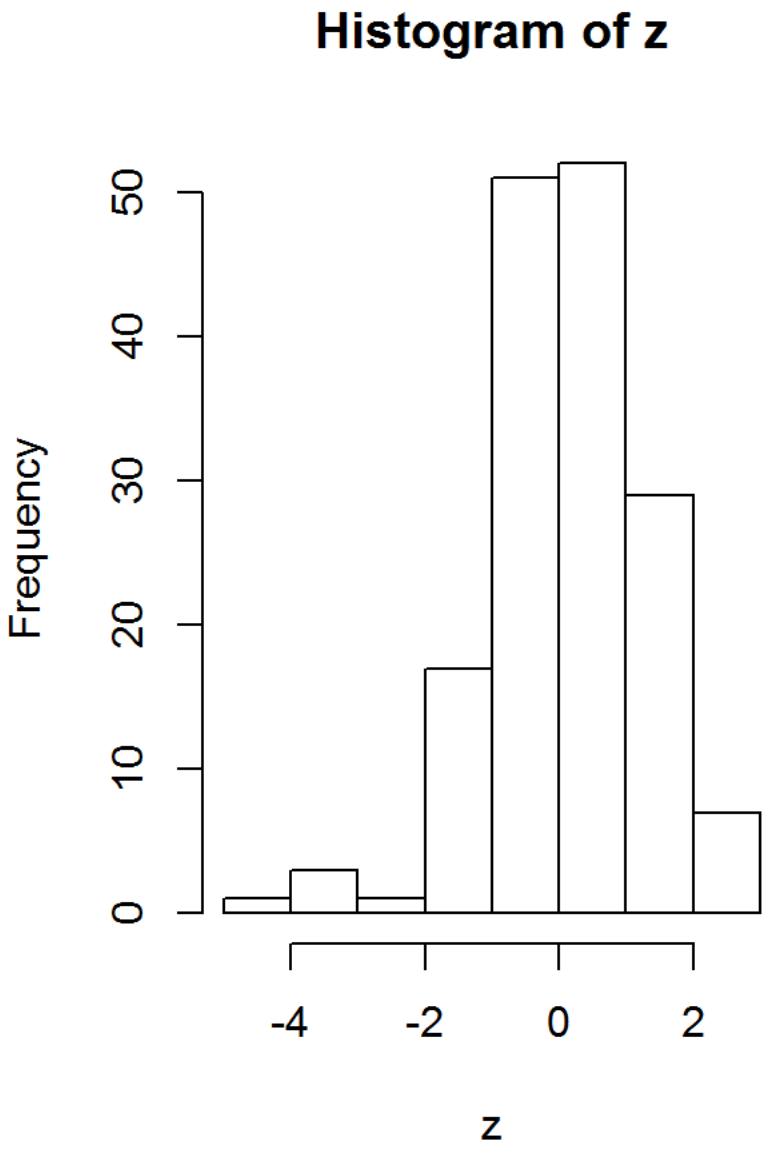
```
print(bs.val[10])
```

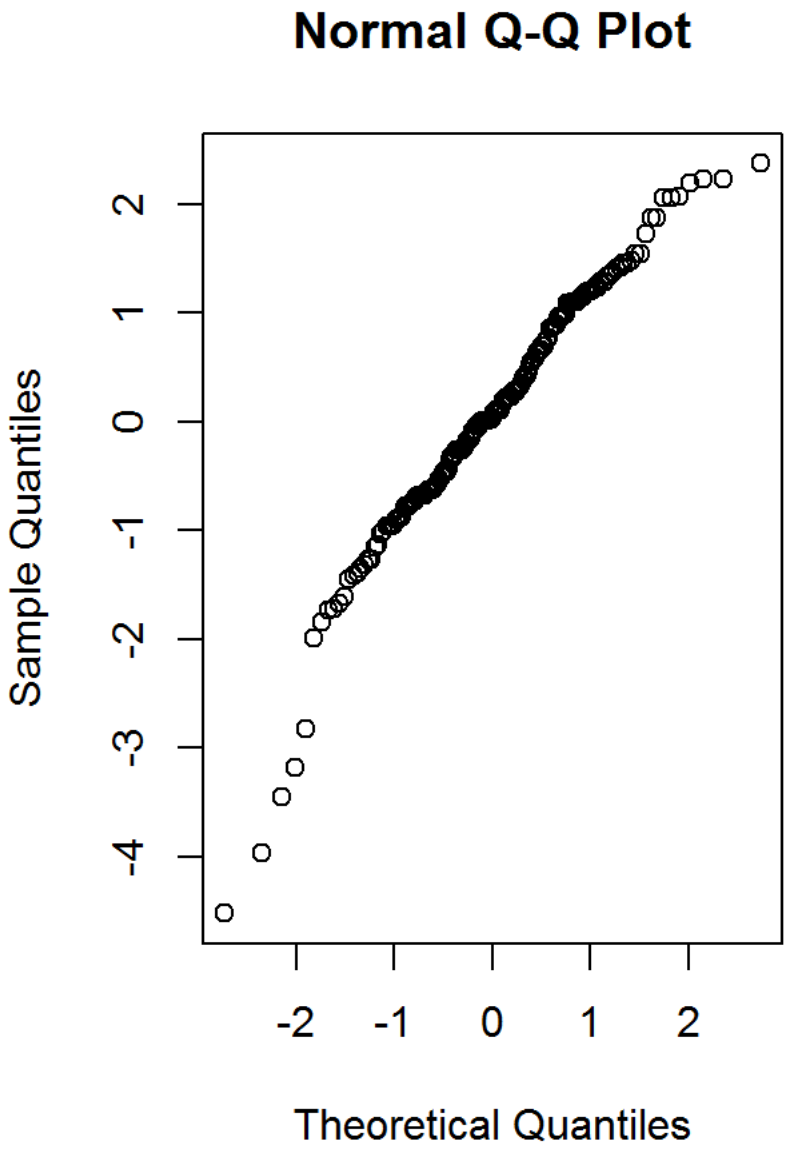
```
## [[1]]
##            g.est      A.est      B.est
## g.est  1.0000000 -0.4721578 -0.2793019
## A.est -0.4721578  1.0000000  0.6520740
## B.est -0.2793019  0.6520740  1.0000000
```

```
print(bs.val[11])
```

```
## [[1]]
## NULL
```

```
g<-bs.val$g
A<-bs.val$a
B<-bs.val$b



#d


z<- 1/g*log(((a-A)*g)/B +1)
par(mfrow=c(1,2))
hist(z)
qqnorm(z)
```






```
# The transformed data is still skewness in the left tail


#e
```

```
gof.pearson=function (x,nbins) {
  n = length(x)
  m = floor(n/nbins)
  k = n - m*nbins # This is the remainder
  xx=sort(x)
  index = rep(1:nbins,m)
  if(k >0){ d=sample(1:nbins,k,replace=FALSE);
  index=c(index,d) }
  bincount=as.numeric(table(index))
  binindicies = cumsum(bincount)
  binbreaks = rev(rev(xx[binindicies])[-1])
  binbreaks = c(-Inf,binbreaks,Inf)
  bins=cut(x,breaks=binbreaks)
  internalbreaks = rev(rev(xx[binindicies])[-1])
  p = pnorm(internalbreaks,mean(x),sd(x))
  p = c(p[1],diff(p),1-pnorm(max(internalbreaks),mean(x),sd(x)))
  exp = n*p
  df = data.frame(bin=levels(bins),bincount=bincount,prob=p,expectedcount=exp)
  chisqstat = sum((bincount - exp)^2/exp)
  pval = 1- pchisq(chisqstat,nbins-1)
  output = list(df=df,chisq=chisqstat,pval=pval)
  output = list(df=df,chisq=chisqstat,pval=pval)
}
out.p<-gof.pearson(z,2*sqrt(length(z))) #usinf Velleman rule
out.p
```

```
## $df
##                  bin bincount        prob expectedcount
## 1       (-Inf,-1.99]        6 0.040439487      6.510757
```

```
## 2      (-1.99,-1.45]        6 0.059924102      9.647780
## 3      (-1.45,-1.16]        7 0.050143709      8.073137
## 4     (-1.16,-0.952]        6 0.045092946      7.259964
## 5    (-0.952,-0.769]        6 0.045940287      7.396386
## 6    (-0.769,-0.681]        6 0.023968190      3.858879
## 7    (-0.681,-0.627]        6 0.015402015      2.479724
## 8    (-0.627,-0.503]        6 0.036911175      5.942699
## 9    (-0.503,-0.325]        6 0.055802513      8.984205
## 10   (-0.325,-0.252]        6 0.023880069      3.844691
## 11   (-0.252,-0.145]        6 0.035539294      5.721826
## 12 (-0.145,-0.00417]        6 0.047765007      7.690166
## 13 (-0.00417,0.0184]        6 0.007691519      1.238335
## 14    (0.0184,0.107]        7 0.030228786      4.866834
## 15     (0.107,0.236]        7 0.043536490      7.009375
## 16     (0.236,0.328]        6 0.030747912      4.950414
## 17      (0.328,0.52]        6 0.062152388     10.006534
## 18      (0.52,0.697]        7 0.053896570      8.677348
## 19     (0.697,0.887]        7 0.053039569      8.539371
## 20     (0.887,0.995]        6 0.027626489      4.447865
## 21      (0.995,1.14]        7 0.033155355      5.338012
## 22       (1.14,1.24]        7 0.021529942      3.466321
## 23       (1.24,1.42]        6 0.033466003      5.388026
## 24       (1.42,1.73]        7 0.046142554      7.428951
## 25       (1.73, Inf]        6 0.075977629     12.232398
##
## $chisq
## [1] 40.41894
##
## $pval
## [1] 0.0217523
```

```
library("goftest")
```

```
## Warning: package 'goftest' was built under R version 3.2.3
```

```
#Correlation of the QQ Data test
qqnorm(z)
qqline(z)

#Shapiro Wilk's Test
shapiro.test(z)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z
## W = 0.96109, p-value = 0.0001755
```

```
#Anderson-Darling Test
ad.test(z,"pnorm")
```

```
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: Normal distribution
##
## data:  z
## An = 1.7235, p-value = 0.1311
```

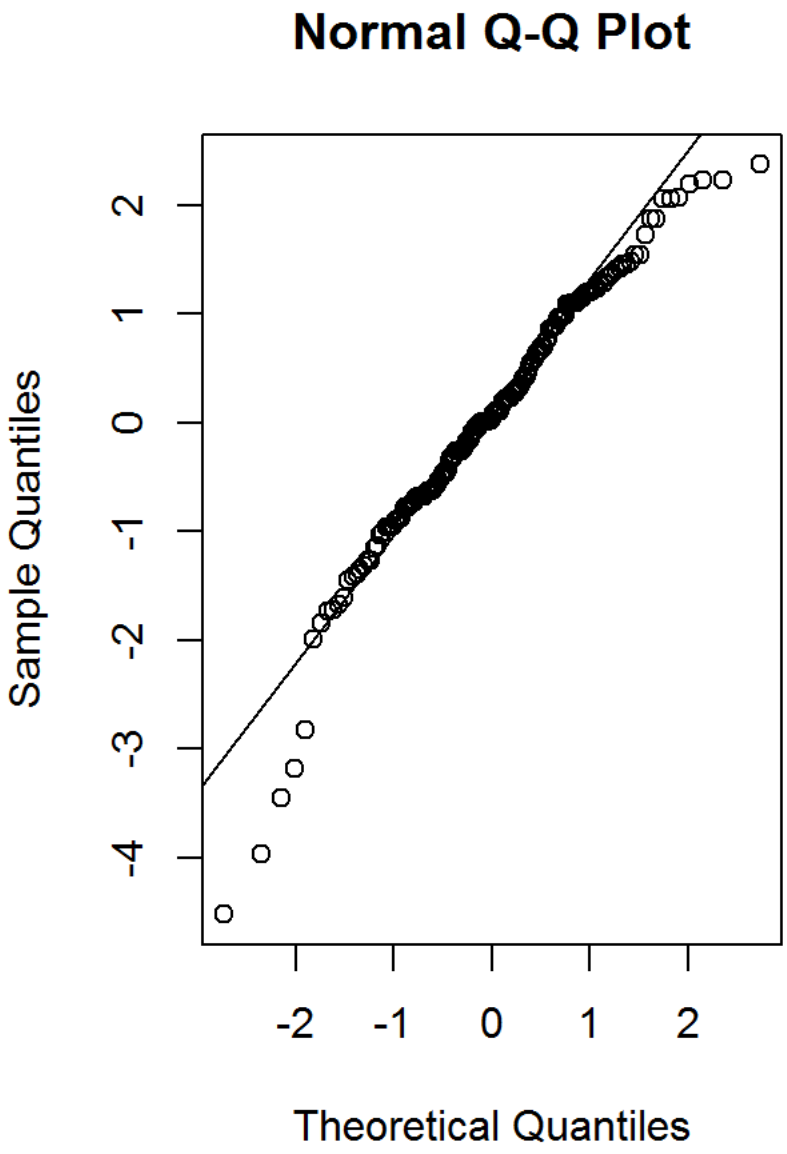
```r
#Kolmogorov Test
ks.test(z,"pnorm")
```

```
## Warning in ks.test(z, "pnorm"): ties should not be present for the
## Kolmogorov-Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  z
## D = 0.085544, p-value = 0.1894
## alternative hypothesis: two-sided
```

```r
#Cramer-von-Mises Test
cvm.test(z,"pnorm")
```

```
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: Normal distribution
##
## data:  z
## omega2 = 0.20535, p-value = 0.2575
```



Shapiro Wilkbs Test -The null-hypothesis of this test is that An issue with the Shapiro-Wilkbs test is that when feed with more data, the chances of the null hypothesis being rejected becomes larger. So what happens is that for large amounts of data even very small deviations from normality can be detected, leading to rejection of the null hypothesis event though for practical purposes the data is more than normal enough

ECDF Based Test Statistics -Empirical testing has found[5] that the Anderson-Darling test is not quite as good as Shapiro Wilk test

Pearson goodness of Fit Test - A sample with a sufficiently large size is assumed - If a Pearson Goodness of fit test is conducted on a sample with a smaller size, then the Pearson Goodness of fit test will yield an inaccurate inference -The observations are always assumed to be independent of each other

QQPlot -Easiest to Interpret for skewness and Heavy/Light tails and normality -The Q-Q plot doesnbt give a strong indication of non-normality

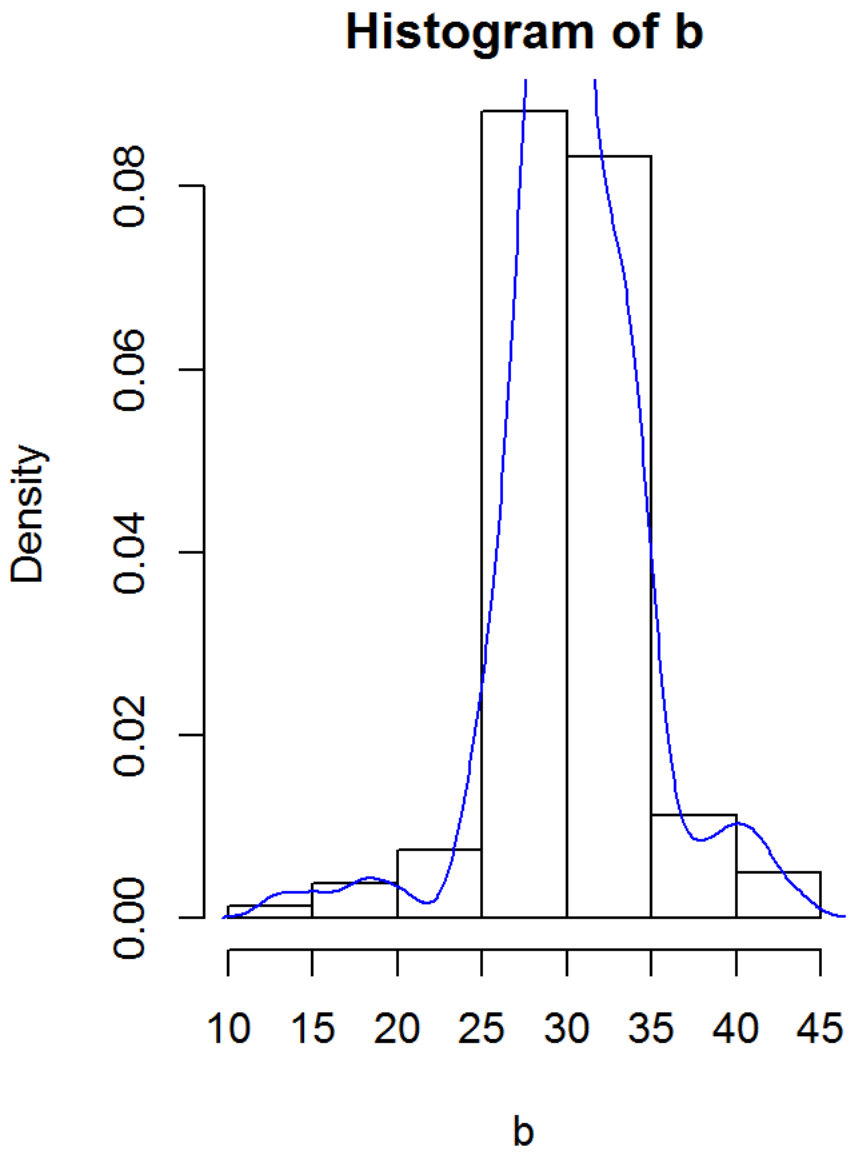
7.

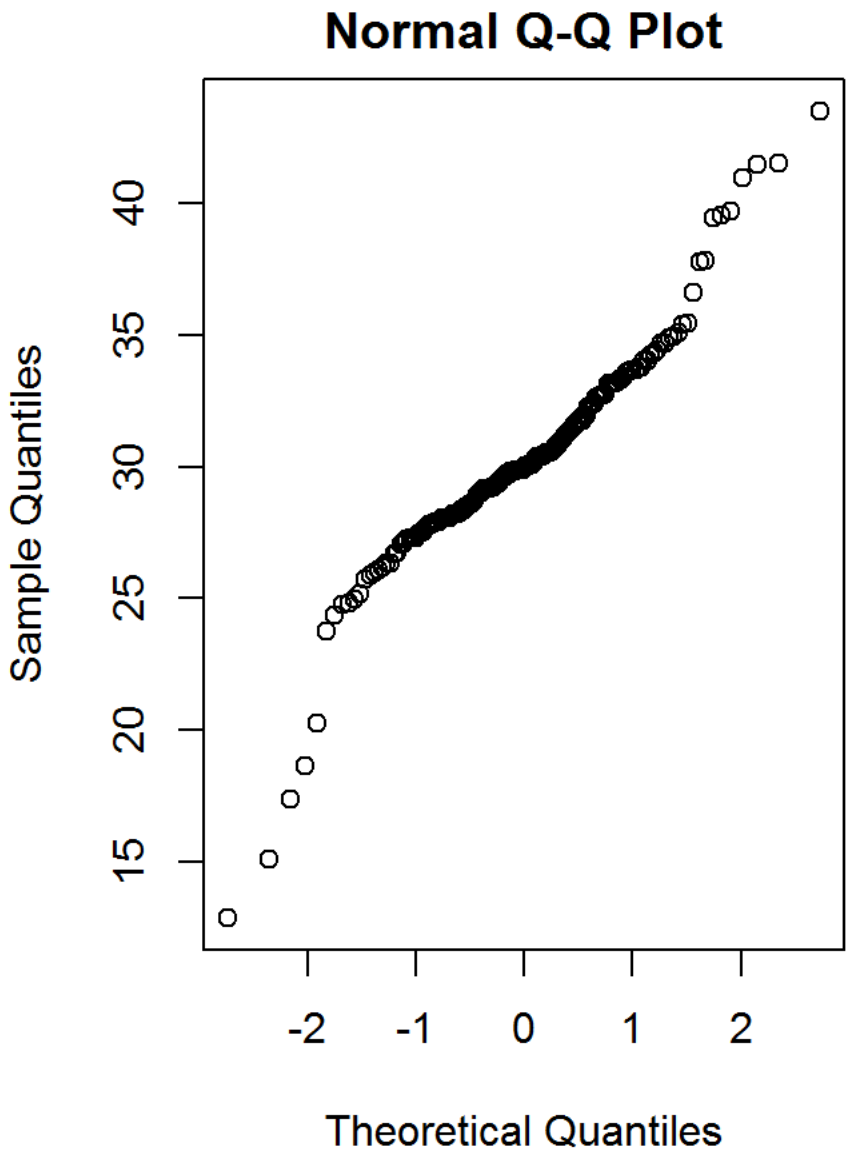
```
library(boot)
```

```
## Warning: package 'boot' was built under R version 3.2.3
```

```
b =c(12.87,15.09,17.39,18.62,20.24,23.76,24.35, 24.74,24.81,24.96,25.19,25.75,25.89,25.97, 26.0
7,26.19,26.35,26.36,26.67,26.76,27.07, 27.12,27.26,27.28,27.30,27.31,27.46,27.49, 27.54,27.72,2
7.81,27.82,27.88,27.90,27.93, 28.03,28.05,28.06,28.07,28.07,28.17,28.19, 28.20,28.22,28.25,28.3
4,28.35,28.46,28.53,28.58,28.64,28.65,28.70,28.92,28.99,29.00, 29.07,29.16,29.16,29.17,29.18,29
.22,29.23, 29.28,29.37,29.40,29.45,29.59,29.62,29.63, 29.71,29.74,29.81,29.82,29.85,29.86,29.86
, 29.86,29.87,29.88,29.92,30.04,30.05,30.09, 30.09,30.10,30.19,30.34,30.37,30.38,30.39, 30.43,3
0.43,30.53,30.55,30.55,30.57,30.64, 30.68,30.77,30.86,30.93,30.98,31.08,31.22, 31.32,31.35,31.4
1,31.52,31.60,31.65,31.76, 31.76,31.77,31.96,31.98,32.28,32.33,32.39, 32.42,32.61,32.68,32.71,3
2.73,32.79,33.15, 33.18,33.19,33.20,33.24,33.33,33.35,33.43, 33.60,33.65,33.66,33.70,33.77,33.8
0,34.03, 34.03,34.26,34.33,34.44,34.68,34.71,34.91, 34.93,35.09,35.40,35.44,36.63,37.81,37.84,
39.47,39.58,39.72,41.00,41.49,41.52,43.50)

par(mfrow=c(1,2),mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))
hist(b,prob=TRUE)
```

```
lines(density(b),col="blue")
qqnorm(b)
```



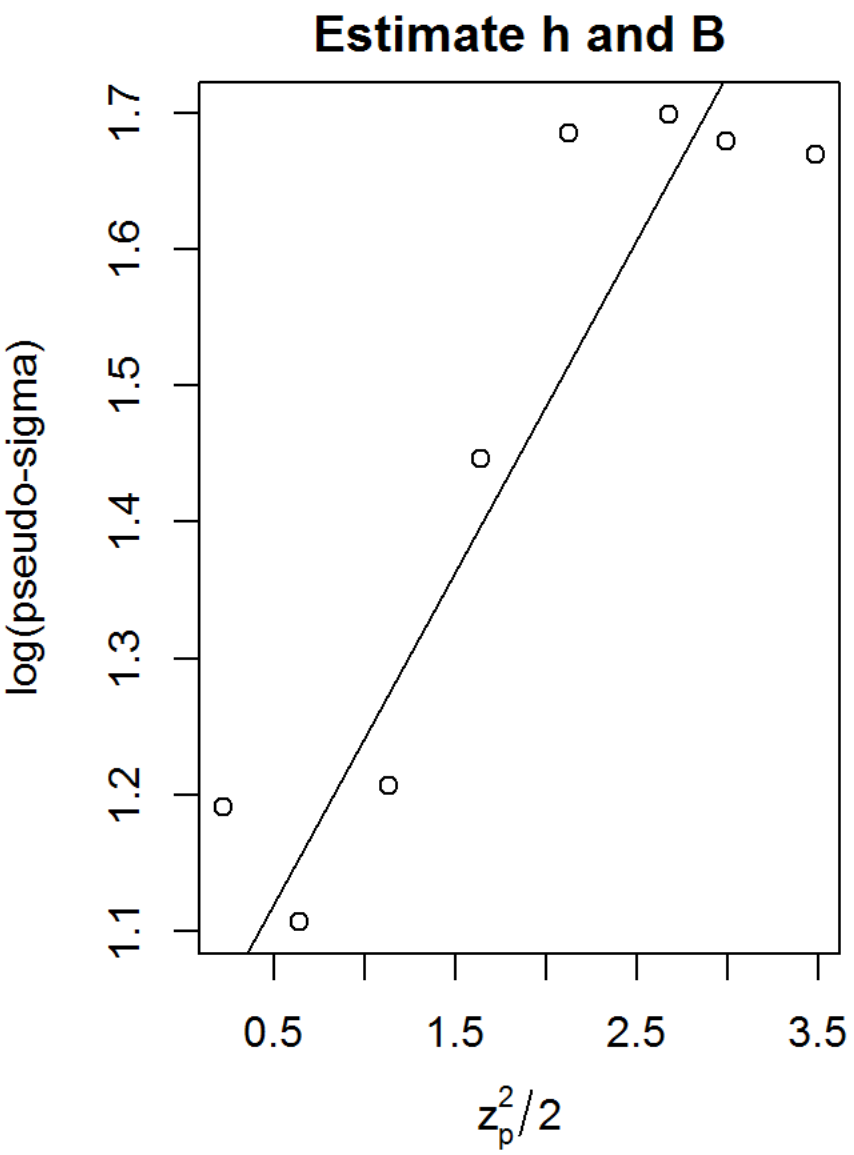


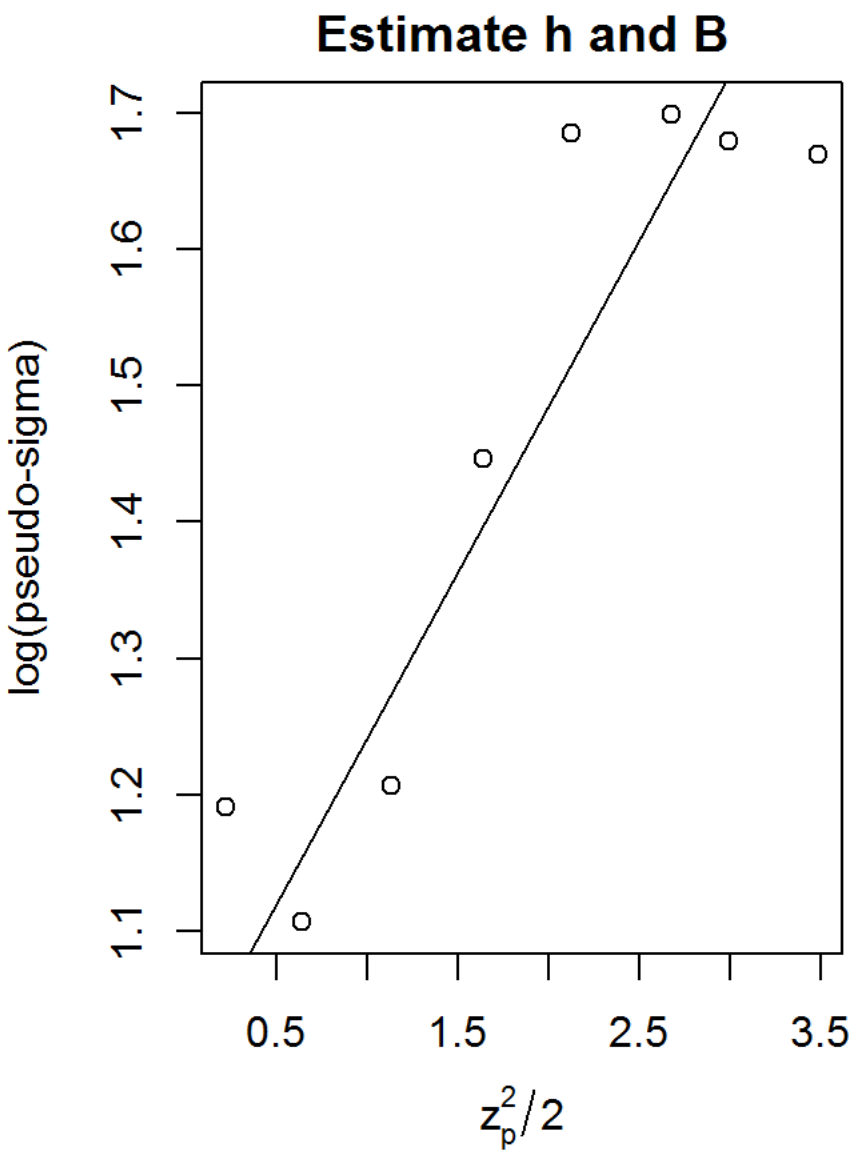
```
ll <- lval(b);
#ll
n<-length(b)
gh2.data <- b
ll.gh2 <- lval(gh2.data)
yy.gh2 <- log(ll.gh2[-1,6])
xx.gh2 <- (qnorm((ll.gh2[-1,1] - 1/3)/(161 + 1/3)))^2/2
plot(xx.gh2,yy.gh2,main="Estimate h and B",
     ylab="log(pseudo-sigma)", xlab=expression(z[p]^2/2),
     sub="rrline: 2.71 + 0.24x => B = 2.71, h = 0.24")
rr <- run.rrline(xx.gh2,yy.gh2);

abline(rr$a, rr$b)

cat( "A=", median(b), ",B=" ,exp(rr$a) , ",h=" ,rr$b)
```

```
## A= 29.92 ,B= 2.711262 ,h= 0.2440427
```

```
est.h<-function(b){
  source("lvalprogs.r")
  source("rrline.r")
  ll <- lval(b)
  n<-length(b)
  gh2.data <- b
  ll.gh2 <- lval(gh2.data)
  yy.gh2 <- log(ll.gh2[-1,6])
  xx.gh2 <- (qnorm((ll.gh2[-1,1] - 1/3)/(161 + 1/3)))^2/2
  plot(xx.gh2,yy.gh2,main="Estimate h and B",
       ylab="log(pseudo-sigma)", xlab=expression(z[p]^2/2))
  rr <- run.rrline(xx.gh2,yy.gh2)
```

```
  abline(rr$a, rr$b)
  return(list(h=rr$b,A=median(b),B=exp(rr$a)))
}
normal.h<-est.h(b)
```






```
print (paste("H estimate is",normal.h$h))
```

```
## [1] "H estimate is 0.24404266941269"
```

```
print (paste("A estimate is",normal.h$A))
```

```
## [1] "A estimate is 29.92"
```

```
print (paste("B estimate is",normal.h$B))
```

```
## [1] "B estimate is 2.71126231385263"
```

```
est.h<-function(b){
  source("lvalprogs.r")
  source("rrline.r")
  ll <- lval(b)
  n<-length(b)
  gh2.data <- b
  ll.gh2 <- lval(gh2.data)
  yy.gh2 <- log(ll.gh2[-1,6])
  xx.gh2 <- (qnorm((ll.gh2[-1,1] - 1/3)/(161 + 1/3)))^2/2
  #plot(xx.gh2,yy.gh2,main="Estimate h and B",
  #  ylab="log(pseudo-sigma)", xlab=expression(z[p]^2/2))
  rr <- run.rrline(xx.gh2,yy.gh2)

  return(list(h=rr$b,A=median(b),B=exp(rr$a)))
}

bootstrap.h<-function(pop,sims){
  library(GGally)
  est.h <-c()
```

```
  est.A <-c()
  est.B <-c()
  for (i in 1:sims){
    b<-sample(pop,length(pop),replace = TRUE)
    b.sample <- est.h(b)
    est.h[i]<- b.sample$h
    est.A[i]<- b.sample$A
    est.B[i]<- b.sample$B
  }
  best.h <- mean(est.h)
  g.lower <- best.h - qt(0.9,df=length(est.h)-1)*sd(est.h)
  g.upper <- best.h + qt(0.9,df=length(est.h)-1)*sd(est.h)
  best.A <- mean(est.A)
  A.lower <- best.A - qt(0.9,df=length(est.A)-1)*sd(est.A)
  A.upper <- best.A + qt(0.9,df=length(est.A)-1)*sd(est.A)
  best.B <- mean(est.B)
  B.lower <- best.B - qt(0.9,df=length(est.B)-1)*sd(est.B)
  B.upper <- best.B + qt(0.9,df=length(est.B)-1)*sd(est.B)
  cor.est <- cor (cbind(est.h,est.A,est.B))
  es.plt <- ggpairs(as.data.frame(cbind(est.h,est.A,est.B)))
  return(list(h=best.h,A=best.A,B=best.B,gcil=g.lower,gciu=g.upper,Acil=A.lower,Aciu=A.upper,bc
il=B.lower,bciu=B.upper,cor.est=cor.est,es.plt=es.plt))
}
s <- bootstrap.h(b,1000)
cat("The A estimate is ",s$A,",C.I is between",s$Acil," and ",s$Aciu )
```

```
## The A estimate is  30.00446 ,C.I is between 29.70892  and  30.3
```

```
cat("The B estimate is ",s$B, ",C.I is between",s$bcil," and ",s$bciu)
```

```
## The B estimate is  2.938939 ,C.I is between 2.45211  and  3.425769
```

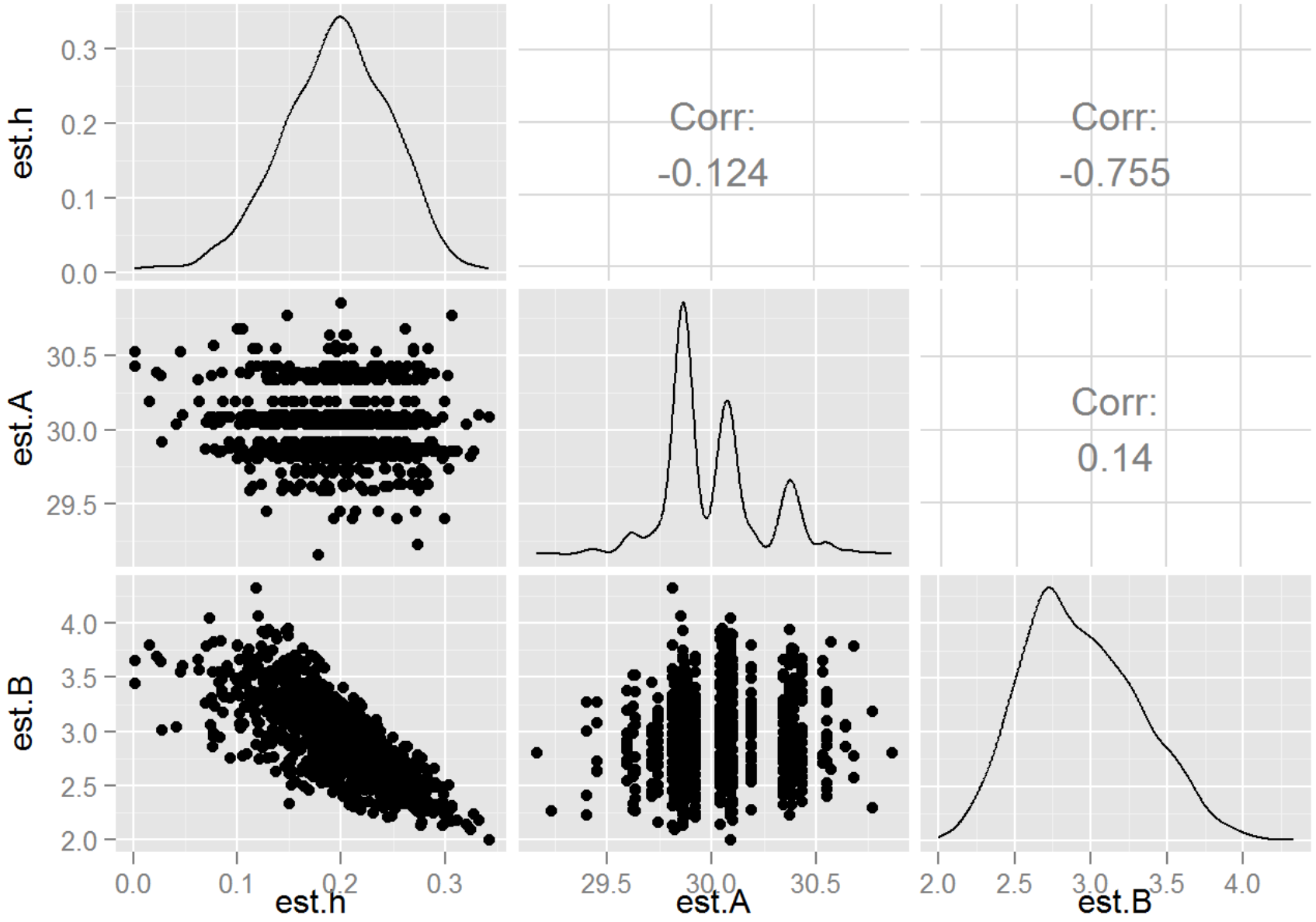
```
cat("The H estimate is ",s$h,",C.I is between",s$gcil," and ",s$gciu)
```

```
## The H estimate is  0.1950324 ,C.I is between 0.1276594  and  0.2624054
```

```
#Co-Relation Pairs

s$cor.est
```

```
##            est.h      est.A      est.B
## est.h  1.0000000 -0.1235600 -0.7550774
## est.A -0.1235600  1.0000000  0.1404717
## est.B -0.7550774  0.1404717  1.0000000
```

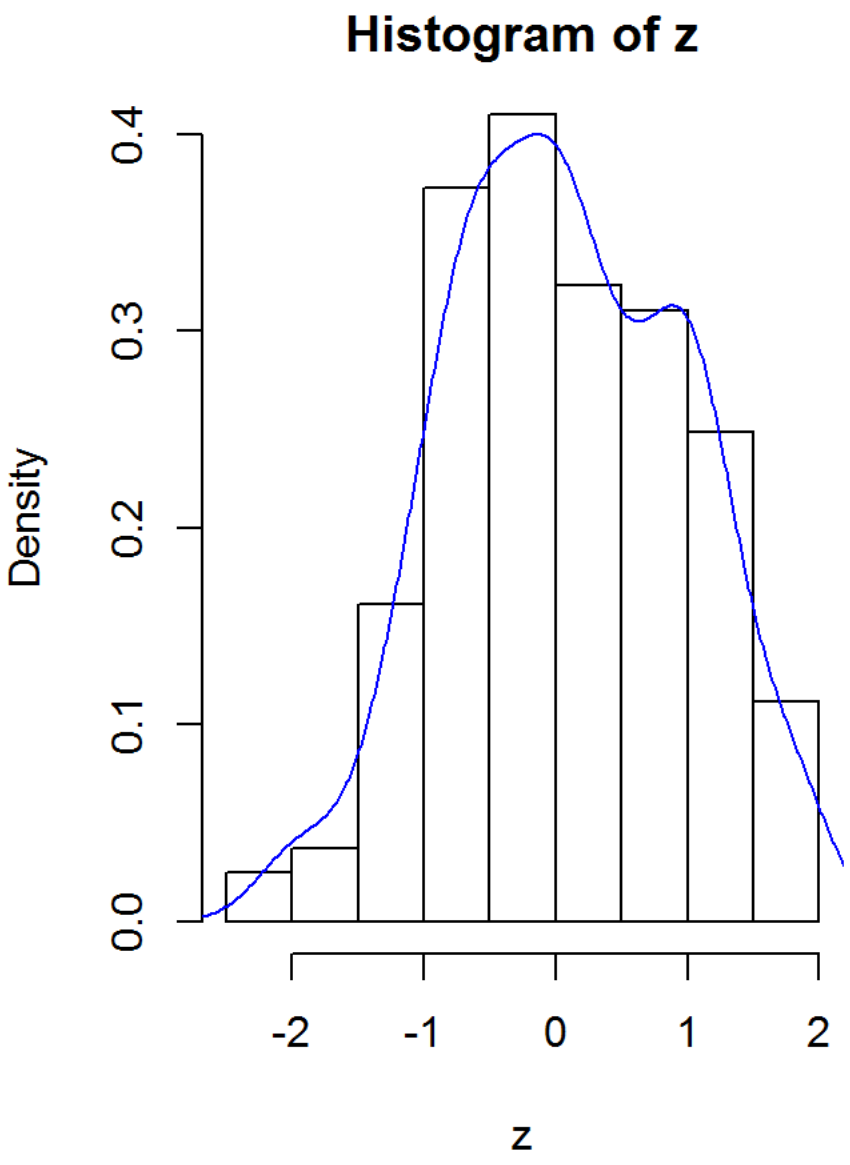
```
#Pairs Plot
print(s$es.plt)
```

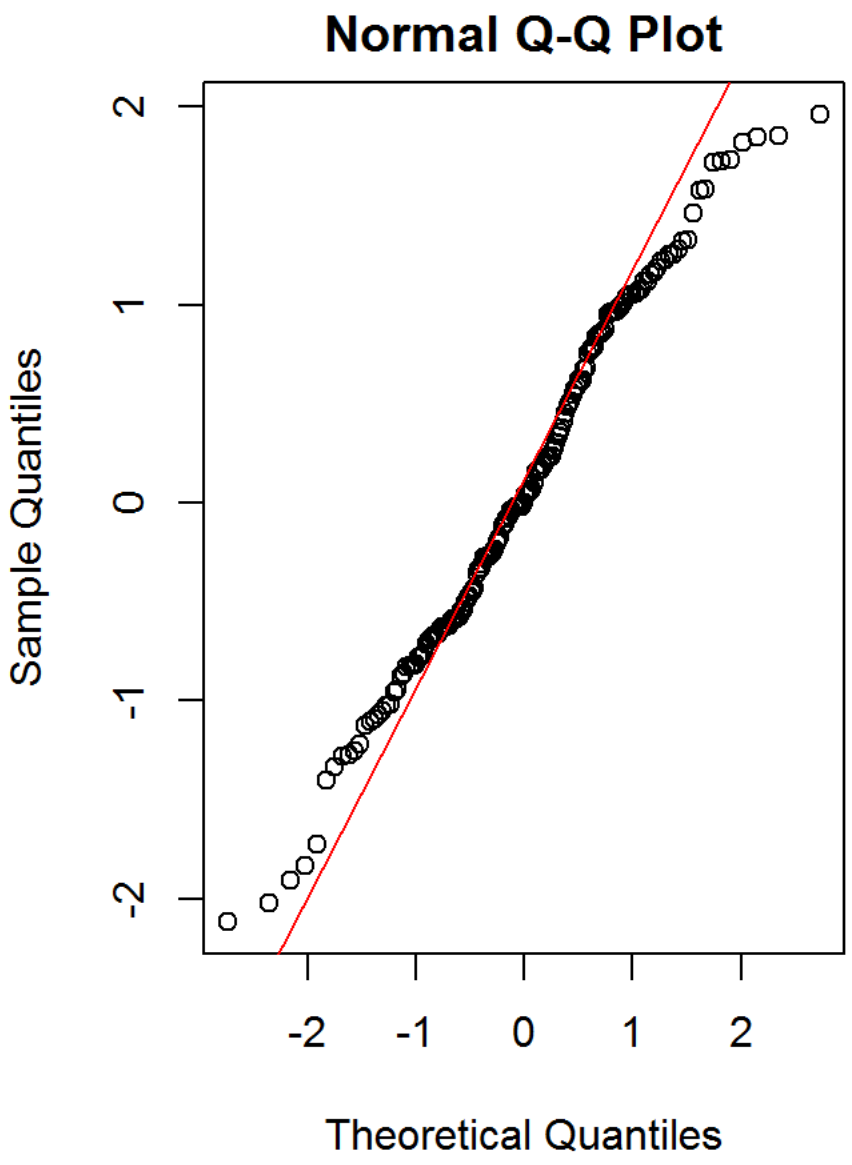


8.

```
HDistBackXform=function(h,A,B,data){
  ####################################
```

```
  #  This function will allow you to back solve for Z
  #  under any H-distribution transform
  #  the Values h, A, and B are the estimated values of
  #  the H-distribution parameters.  In this program
  #  data is a vector of data.
  ###################################

  n=length(data)
  #using Veleman's rule
  output=numeric(n)
  g=function(z){z*exp(h*z^2)-((x-A)/B)}
  # Begin loop on i where data[i] is the ith data value
  for(i in 1:n){
    x=data[i]
    obj=uniroot(g,interval=c(-6,6))
    output[i]=obj$root
  }
  return(output)
}

h<-normal.h$h
A<-normal.h$A
B<-normal.h$B

z<-HDistBackXform(h,A,B,b)
par(mfrow=c(1,2),mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))
hist(z,prob=TRUE)
lines(density(z),col="blue")
qqnorm(z)
qqline(z,col="red")
```






```
#Veleman's Rule for Pearson's GoF

noofbins=2*sqrt(length(z))
out<-gof.pearson(z,noofbins)
```

```
out
```

```
## $df
##                  bin bincount       prob expectedcount
## 1       (-Inf,-1.4]        6 0.04775222      7.688107
## 2      (-1.4,-1.13]        6 0.03979728      6.407362
## 3     (-1.13,-1.02]        6 0.02108262      3.394302
## 4    (-1.02,-0.825]        6 0.04636156      7.464211
## 5   (-0.825,-0.716]        6 0.03096361      4.985141
## 6   (-0.716,-0.632]        6 0.02626791      4.229134
## 7   (-0.632,-0.587]        6 0.01518388      2.444605
## 8   (-0.587,-0.484]        7 0.03626236      5.838240
## 9    (-0.484,-0.33]        7 0.05940056      9.563490
## 10   (-0.33,-0.251]        7 0.03285142      5.289078
## 11  (-0.251,-0.107]        7 0.06192364      9.969705
## 12 (-0.107,-0.0221]        7 0.03743175      6.026511
## 13 (-0.0221,0.0626]        7 0.03794367      6.108930
## 14   (0.0626,0.168]        6 0.04743777      7.637482
## 15    (0.168,0.229]        6 0.02707514      4.359097
## 16    (0.229,0.378]        7 0.06432602     10.356489
## 17    (0.378,0.572]        7 0.07886522     12.697300
## 18    (0.572,0.679]        6 0.03965494      6.384446
## 19    (0.679,0.853]        6 0.05769841      9.289444
## 20    (0.853,0.962]        6 0.03183651      5.125678
## 21     (0.962,1.04]        6 0.02075762      3.341977
## 22      (1.04,1.12]        6 0.01808076      2.911002
## 23      (1.12,1.22]        6 0.02261619      3.641206
## 24      (1.22,1.47]        6 0.03916901      6.306210
## 25      (1.47, Inf]        6 0.05925995      9.540851
##
```

```
## $chisq
## [1] 25.64573
##
## $pval
## [1] 0.3920897
```

```
#9 Estimation of Mode


data = rnorm(100, 3, 2)

getGaussianMax = function(data){
  d = density(data, kernel="gaussian")
  index = which(d$y == max(d$y), arr.ind =TRUE)
  ans = d$x[index]
  return(ans)
}

calculatePseudoValues = function(data) {
  n = length(data)
  y.all = getGaussianMax(data)
  PV = numeric(n)
  for( i in 1:n) {
    yminusi = getGaussianMax(data[-i])
    PV[i] = n*y.all - (n-1)*yminusi
  }
  return(PV)
}

PVAll = calculatePseudoValues(data)
n = length(PVAll)
```

```
jackKnifeEstimate = mean(PVAll)
varJK = sum((PVAll - jackKnifeEstimate)^2)/(n*(n-1))
seJK = sqrt(varJK)

getbootstrapestimate = function(data, sims) {
  theta = numeric(sims)
  varTheta = numeric(sims)

  n = length(data)
  index = 1:n
  for (i in 1:sims){
    sampleindex= sample(index,n,replace=TRUE)
    theta[i] = mean(getGaussianMax(data[sampleindex]))
  }

  return(list(thetaBS = mean(theta), varBS = var(theta), seBS = sqrt(var(theta))))
}

BS = getbootstrapestimate(data, 100)$seBS

print(paste("The Mode is ",mean(PVAll) ))
```

```
## [1] "The Mode is  3.43317633321731"
```

```
print(paste("The Standard Error of Jackknife Estimator",seJK))
```

```
## [1] "The Standard Error of Jackknife Estimator 0.225121552512169"
```

```
print(paste("The Standard Error of Bootstrap Estimates",BS))
```

```
## [1] "The Standard Error of Bootstrap Estimates 0.680111858303124"
```

9.

```
q9data = rnorm(100, 3, 2)

getGausEstimate = function(data){
  d = density(data, kernel="gaussian")
  index = which(d$y == max(d$y), arr.ind =TRUE)
  ans = d$x[index]
  return(ans)
}

calculatePseudoValues = function(data) {
  n = length(data)
  yall = getGausEstimate(data)
  PV = numeric(n)
  for( i in 1:n) {
    yminusi = getGausEstimate(data[-i])
    PV[i] = n*yall - (n-1)*yminusi
  }
  return(PV)
}

# We first use jackknife
PVAll = calculatePseudoValues(q9data)
n = length(PVAll)
```

```
mean(PVAll)
```

```
## [1] 1.960042
```

```
jackKnifeEstimate = mean(PVAll)
varJK = sum((PVAll - jackKnifeEstimate)^2)/(n*(n-1))
seJK = sqrt(varJK)
seJK
```

```
## [1] 0.2636036
```

```
getbootstrapestimate = function(data, nsim) {
  theta = numeric(nsim)
  varTheta = numeric(nsim)

  n = length(data)
  index = 1:n
  for (i in 1:nsim){
    sampleindex= sample(index,n,replace=TRUE)
    theta[i] = mean(getGausEstimate(data[sampleindex]))
  }

  output = list(thetaBS = mean(theta), varBS = var(theta),
                seBS = sqrt(var(theta)))
  output
}
seBS = getbootstrapestimate(q9data, 100)$seBS
seBS
```

```
## [1] 0.5338059
```

10.

1 The first step is to sort the values of x and divide the n points into 3 nearly equal groups 2 The summary points (median x, median y) will be estimated in outer groups R andL 3 We know slope = (yR-yL)/(xR-xL) , Intercept= median(yi-bxi)

Advantages of fitting RR line 1.Robustness 2.Easiest way to obtain linearity of data 3. It is Asymptotically efficient than OLS per step

Disadvantages of fitting RR line

1. Larger datasets are complex to operate using this
2. We dont arrive at a unique definite solution
3. It operates based on many iterations

11.

1. Bootstrapping is resamplinng tchnique which helps when there is a doubt in the accutacy of usual distributional assumptions and asymptotic results 2 . Jackniffing is a special case of bootstrapping. It helps in variance and bias estimation 3 . Bootstrpping follows subsample data with replacement while the jackniffing resampling follows leave one out and calculate rest approach
2. In understanding variability and bias bootstrapping plays a major role as it is more accurate, the jackniffing helps in understanding bias of a point esstimator 5 Bootstrapping is more reliable when compared to jackniffing.

12.

Given Number of scientific inventions for each year between 1860 and 1959

a. What distribution is appropriate for these count data

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.2.3
```

```
library(vcd)
```

```
## Warning: package 'vcd' was built under R version 3.2.3
```

```
## Loading required package: grid
```

```
invent <- c(5,3,0,2,0,3,2,3,6,1,2,1,2,1,3,3,3,5,2,4, 4,0,2,3,7,12,3,10,9,2,3,7,7,2,3,3,6,2,4,3,
5,2,2,4,0,4,2,5,2,3,3,6,5,8,3,6,6,0,5,2, 2,2,6,3,4,4,2,2,4,7,5,3,3,0,2,2,2,1,3,4, 2,2,1,1,1,2,1
,4,4,3,2,1,4,1,1,1,0,0,2,0)
library(kequate)
```

```
## Warning: package 'kequate' was built under R version 3.2.3
```

```
## Loading required package: ltm
```

```
## Warning: package 'ltm' was built under R version 3.2.3
```

```
## Loading required package: MASS
## Loading required package: msm
```

```
## Warning: package 'msm' was built under R version 3.2.3
```

```
##
```

```
## Attaching package: 'msm'
##
## The following object is masked from 'package:boot':
##
##     cav
##
## Loading required package: polycor
```

```
## Warning: package 'polycor' was built under R version 3.2.3
```

```
## Loading required package: mvtnorm
```

```
## Warning: package 'mvtnorm' was built under R version 3.2.3
```

```
## Loading required package: sfsmisc
```

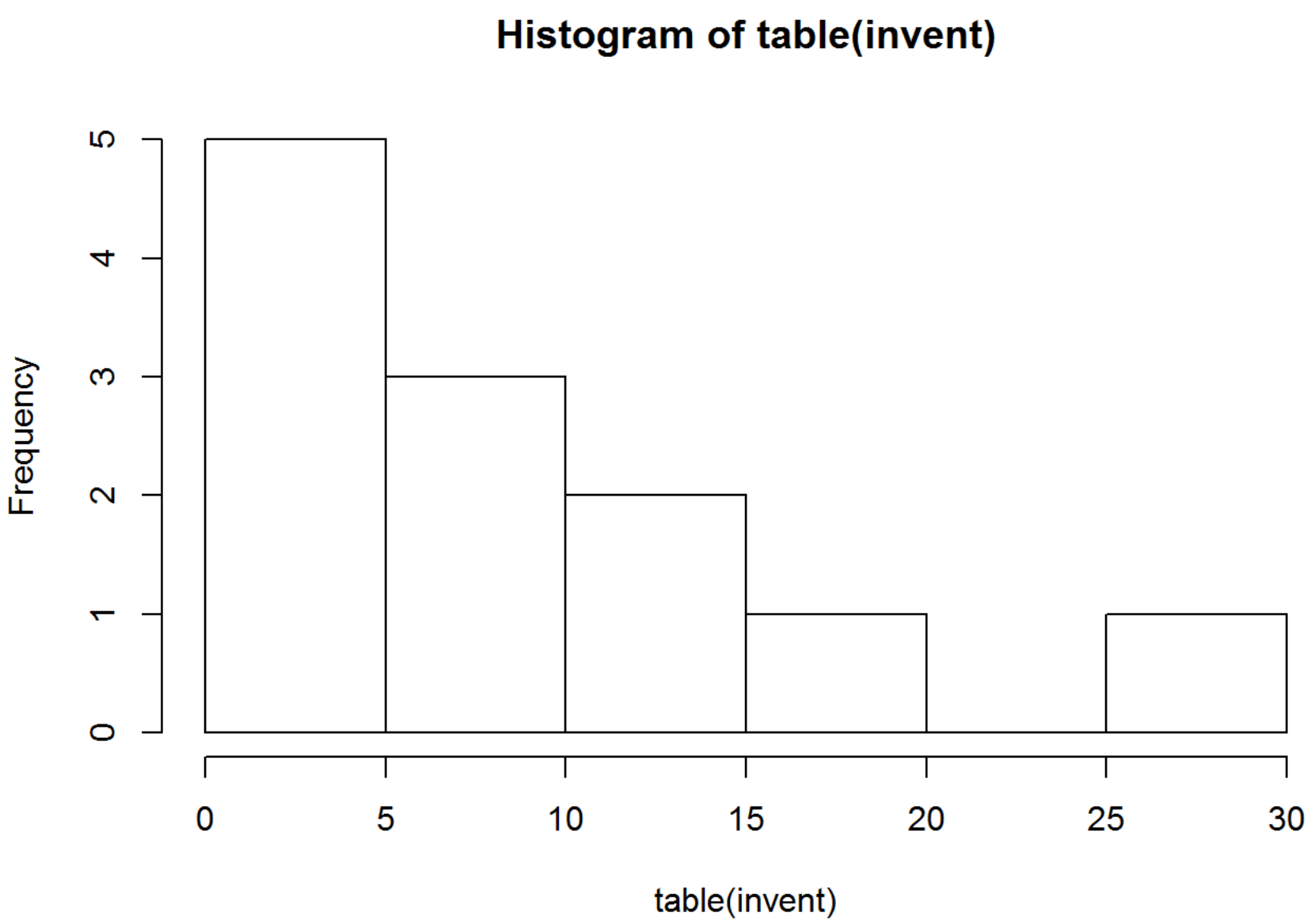
```
## Warning: package 'sfsmisc' was built under R version 3.2.3
```

```
#Tabulating the given
table(invent)
```

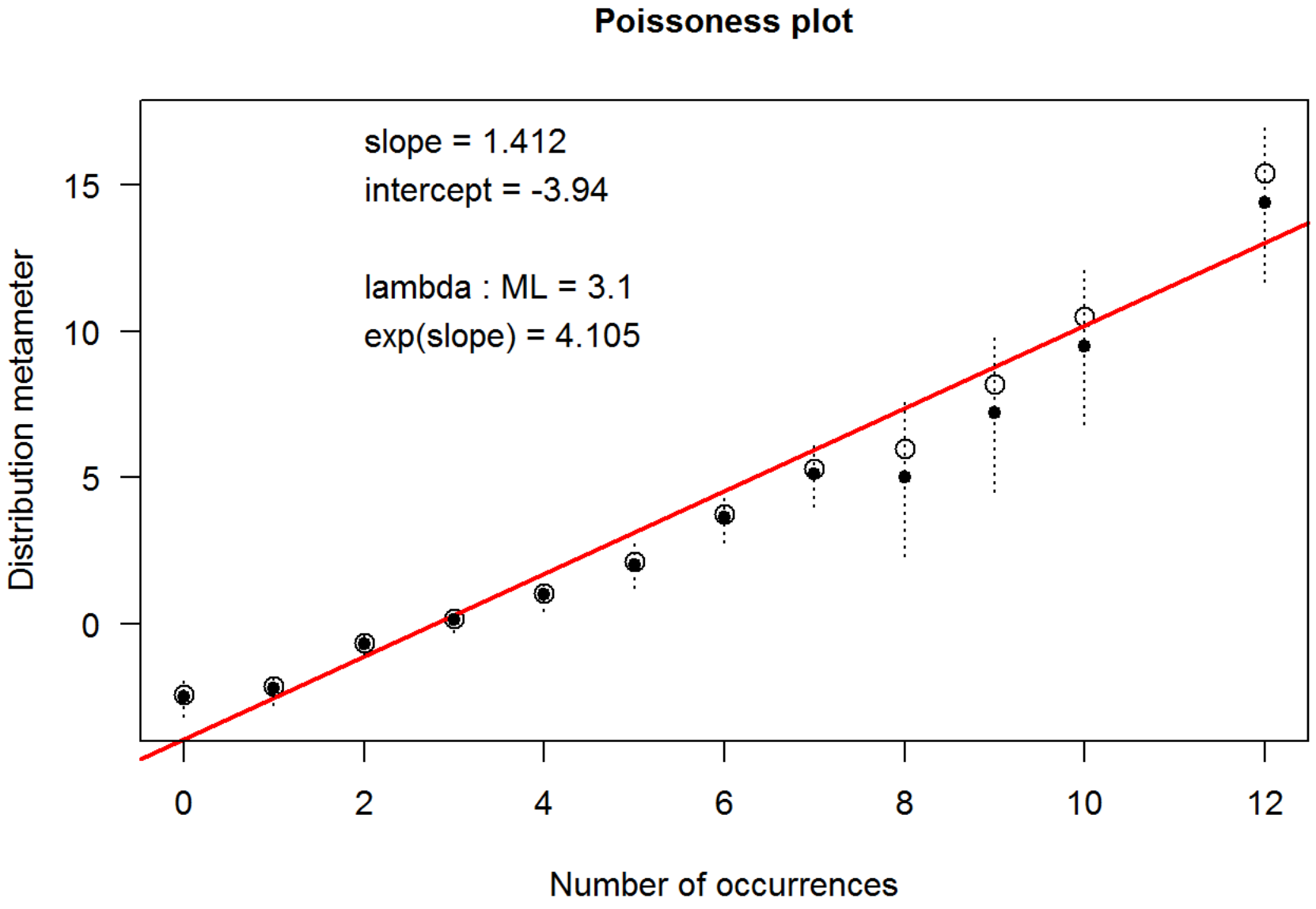
```
## invent
##  0  1  2  3  4  5  6  7  8  9 10 12
##  9 12 26 20 12  7  6  4  1  1  1  1
```

```
#histogram
```

```
hist(table(invent))
```



```
#We see that it is quite similair ot poisson distribution, so we can say that poisson distribut
ion might be appropriate for these count data


#Poisson Plot
distplot(table(invent))
```




```
#pLOT OF FREEMAN TUKEY RESIDUALS


ft.res<-FTres(table(invent), sapply(as.array(table(invent)),function(x){(1.142*x-3.94)}))
```

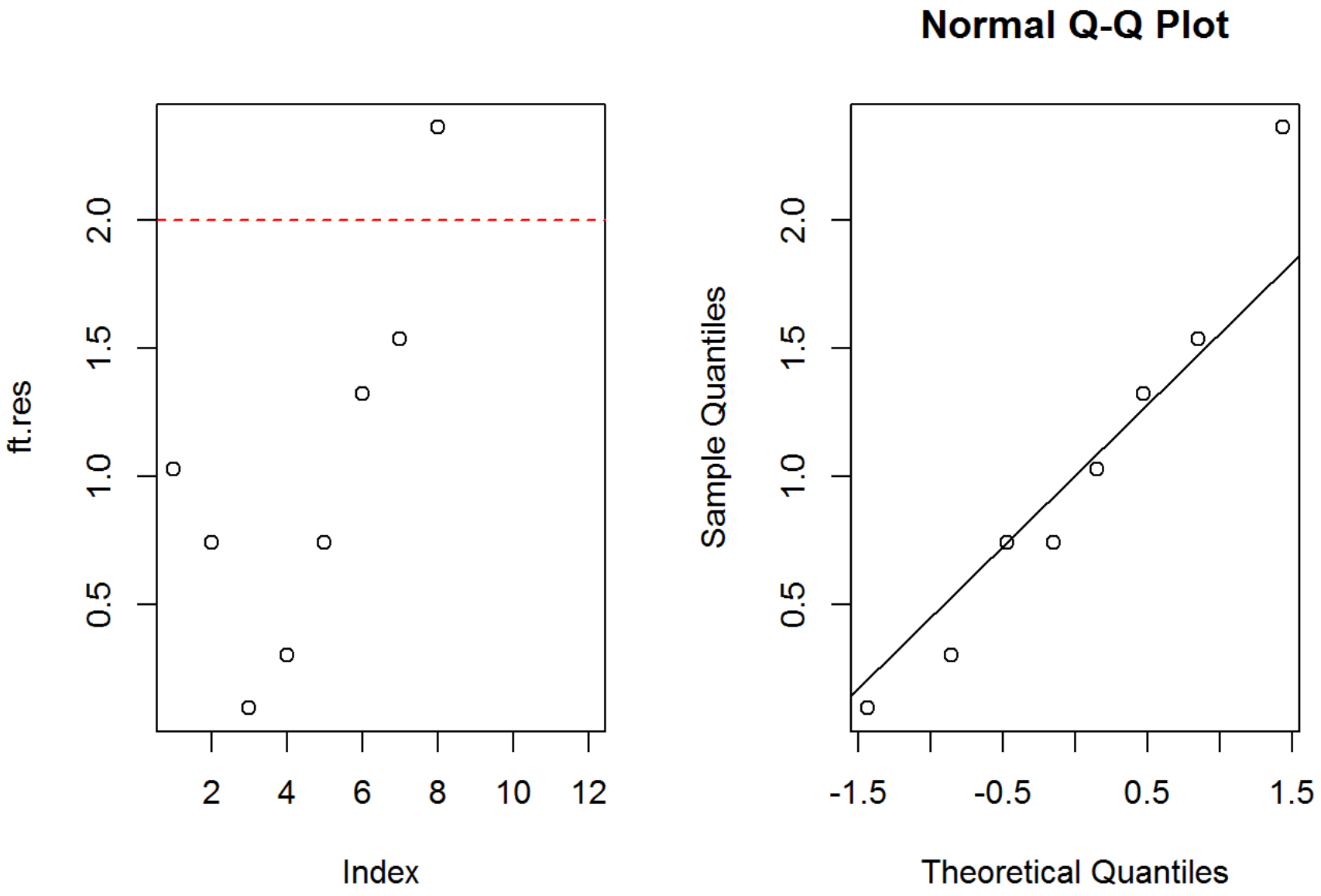
```
## Warning in sqrt(4 * fit[i] + 1): NaNs produced
```

```
## Warning in sqrt(4 * fit[i] + 1): NaNs produced
```

```
## Warning in sqrt(4 * fit[i] + 1): NaNs produced
```

```
## Warning in sqrt(4 * fit[i] + 1): NaNs produced
```

```
par(mfrow=c(1,2))
plot(ft.res)
abline(h=c(-2,0,2),lty=c(2,1,2),col=c(2,1,2))
qqnorm(ft.res)
qqline(ft.res)
```




```
# SOme of the points can be seen to deviate from the plot
#Freeman  -Tukey residuals has a normal distribution
```

c Freeman tukey residuals are given by √(ni) +√(ni+1) - √(4mi+1) for a estimated frequency mi and observed frequency ni

The above residuals are used to stabilize the variance in the transformed data. QQ plot shows the normal distribution AND THE FREEMAN TUKEY variables are more clear.

13.

```
r1<-c(16,13.6,16.2,14.2,9.3,15.1,10.6,12,11.3,10.5,7.7,10.6)
r2<-c(30.4,27.3,32.4,24.1,27.3,21,19.2,22,19.4,14.9,11.4,18)
r3<-c(34.8,37.1,40.3,30.3,35,38.1,26.2,30.6,25.8,18.1,12.3,17.9)
r4<-c(37.2,41.8,42.1,34.6,38.8,34,30,31.8,27.9,18.9,13,17.9)
r5<-c(35.3,40.6,42.9,32.5,38.6,38.9,30.9,32.4,28.5,19.5,12.5,17.9)
r6<-c(39.2,41.4,43.9,35.4,37.5,39.6,32.4,31.1,28.1,22.2,13.7,18.9)
r7<-c(39.7,44.3,45.5,38.7,42.4,41.4,35.5,31.5,27.8,21.9,14.4,19.9)

df<-rbind(r1,r2,r3,r4,r5,r6,r7)
colnames(df)<-c(111,211,311,412,512,612,721,821,921,1022,1122,1222)
rownames(df)<-c(95,175,250,350,500,675,1000)
results<-medpolish(df)
```

```
## 1: 174.4
## 2: 162.35
## Final: 161.5375
```

```
results
```

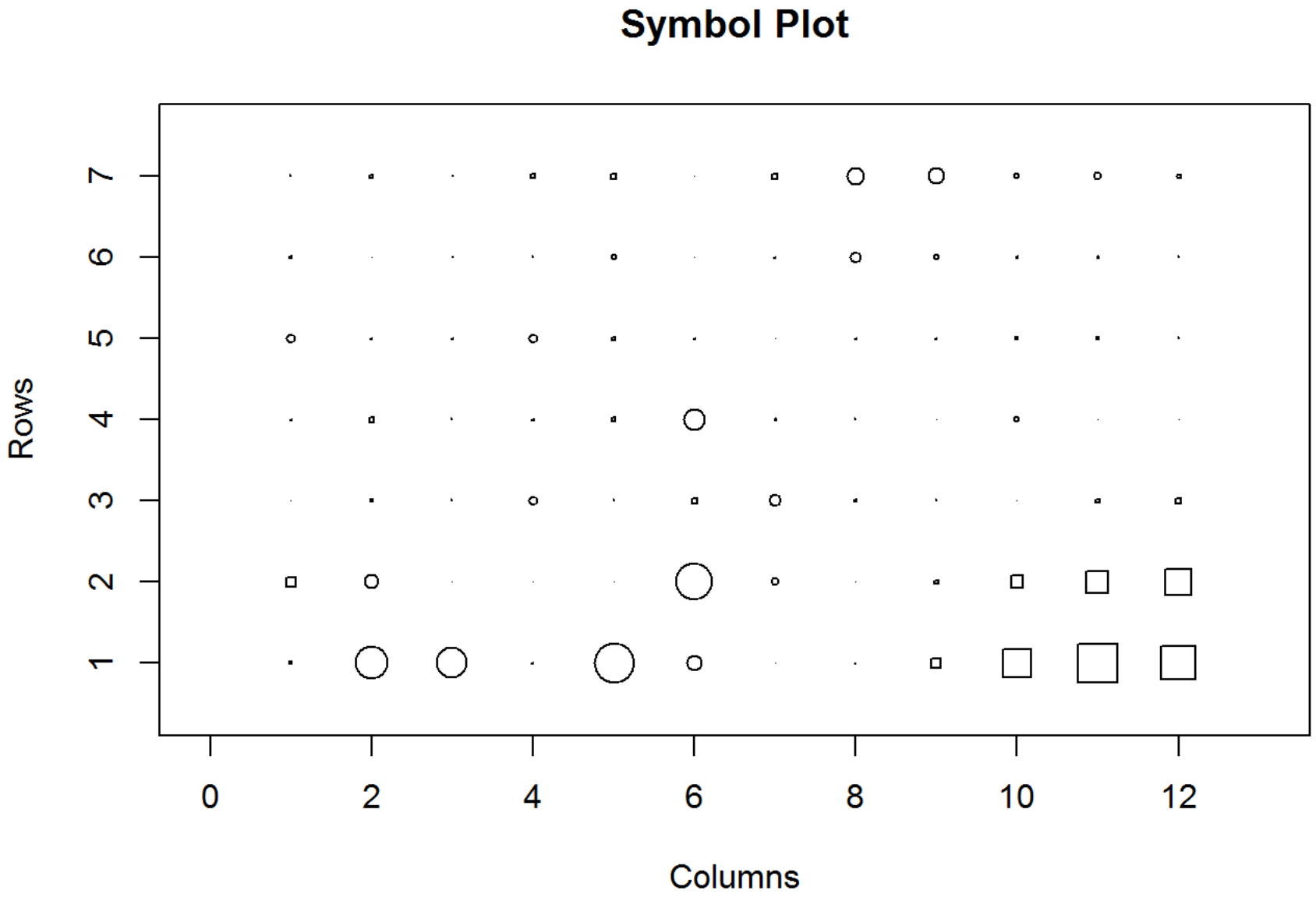
```
##
## Median Polish Results (Dataset: "df")
##
## Overall: 33.0125
```

```
##
## Row Effects:
##       95      175      250      350      500      675     1000
## -20.1375  -9.9625  -2.0500   0.0000   0.2125   1.3750   3.0000
##
## Column Effects:
##      111      211      311      412      512      612      721      821
##   3.8375   7.0125   9.3500   1.0500   4.2500   5.2125  -2.3250  -1.0500
##      921     1022     1122     1222
##  -5.1125 -12.8625 -20.0125 -15.1125
##
## Residuals:
##          111     211     311     412     512     612     721     821
## 95   -0.7125 -6.2875 -6.0250  0.2750 -7.8250 -2.9875  0.0500  0.1750
## 175   3.5125 -2.7625  0.0000  0.0000  0.0000 -7.2625 -1.5250  0.0000
## 250   0.0000 -0.8750 -0.0125 -1.7125 -0.2125  1.9250 -2.4375  0.6875
## 350   0.3500  1.7750 -0.2625  0.5375  1.5375 -4.2250 -0.6875 -0.1625
## 500  -1.7625  0.3625  0.3250 -1.7750  1.1250  0.4625  0.0000  0.2250
## 675   0.9750  0.0000  0.1625 -0.0375 -1.1375  0.0000  0.3375 -2.2375
## 1000 -0.1500  1.2750  0.1375  1.6375  2.1375  0.1750  1.8125 -3.4625
##          921    1022    1122    1222
## 95    3.5375 10.4875 14.8375 12.8375
## 175   1.4625  4.7125  8.3625 10.0625
## 250  -0.0500  0.0000  1.3500  2.0500
## 350   0.0000 -1.2500  0.0000  0.0000
## 500   0.3875 -0.8625 -0.7125 -0.2125
## 675  -1.1750  0.6750 -0.6750 -0.3750
## 1000 -3.1000 -1.2500 -1.6000 -1.0000
```

```
symbolPlot<-function(mat){
  result<-medpolish(mat)
```

```
  res<-c(result$residuals)
  genNos<-expand.grid(1:7,1:12)
  plotvar<-cbind(genNos$Var2,genNos$Var1,res)
  pos<-plotvar[plotvar[,3]>=0,]
  max<-sum(abs(pos[,3]))
  symbols(pos[,1],pos[,2],squares = 3*(abs(pos[,3]/(max))),inches = FALSE,xlab="Columns",ylab="
Rows",main="Symbol Plot")
  pos<-plotvar[plotvar[,3]<0,]
  symbols(pos[,1],pos[,2],circles = 3*(abs(pos[,3]/(max))),inches = FALSE,add = TRUE)
}



symbolPlot(df)
```

```
## 1: 174.4
## 2: 162.35
## Final: 161.5375
```

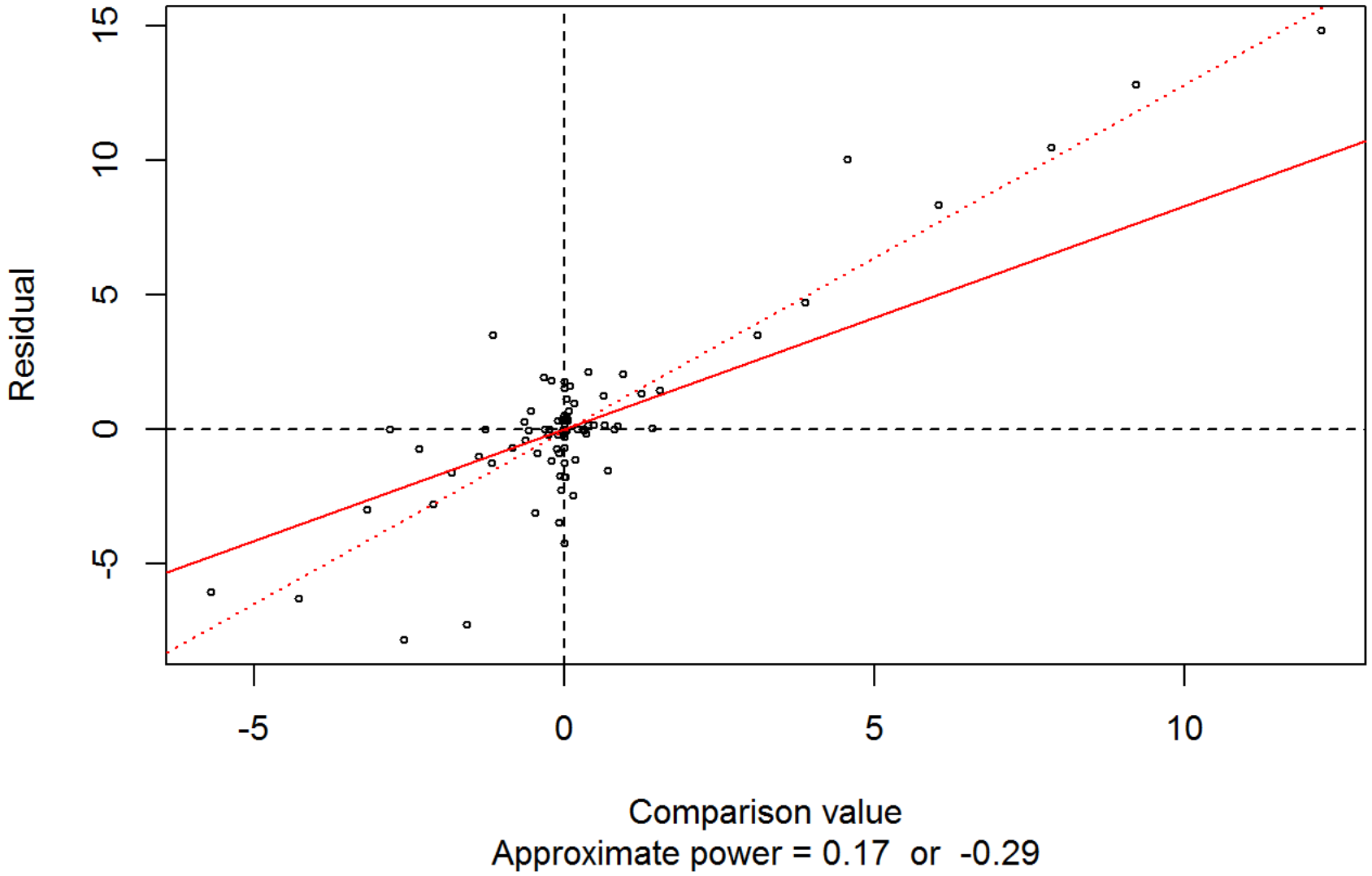



```
#The plot shows the data clearly with better visualization of variability in data
```

```r
#b


AnalogR<- 1-((sum(abs(results$residuals))) /(sum(abs(df-results$overall))))
print(paste("Variability Measure, Analog R Square :",AnalogR))
```

```
## [1] "Variability Measure, Analog R Square : 0.808064755680974"
```

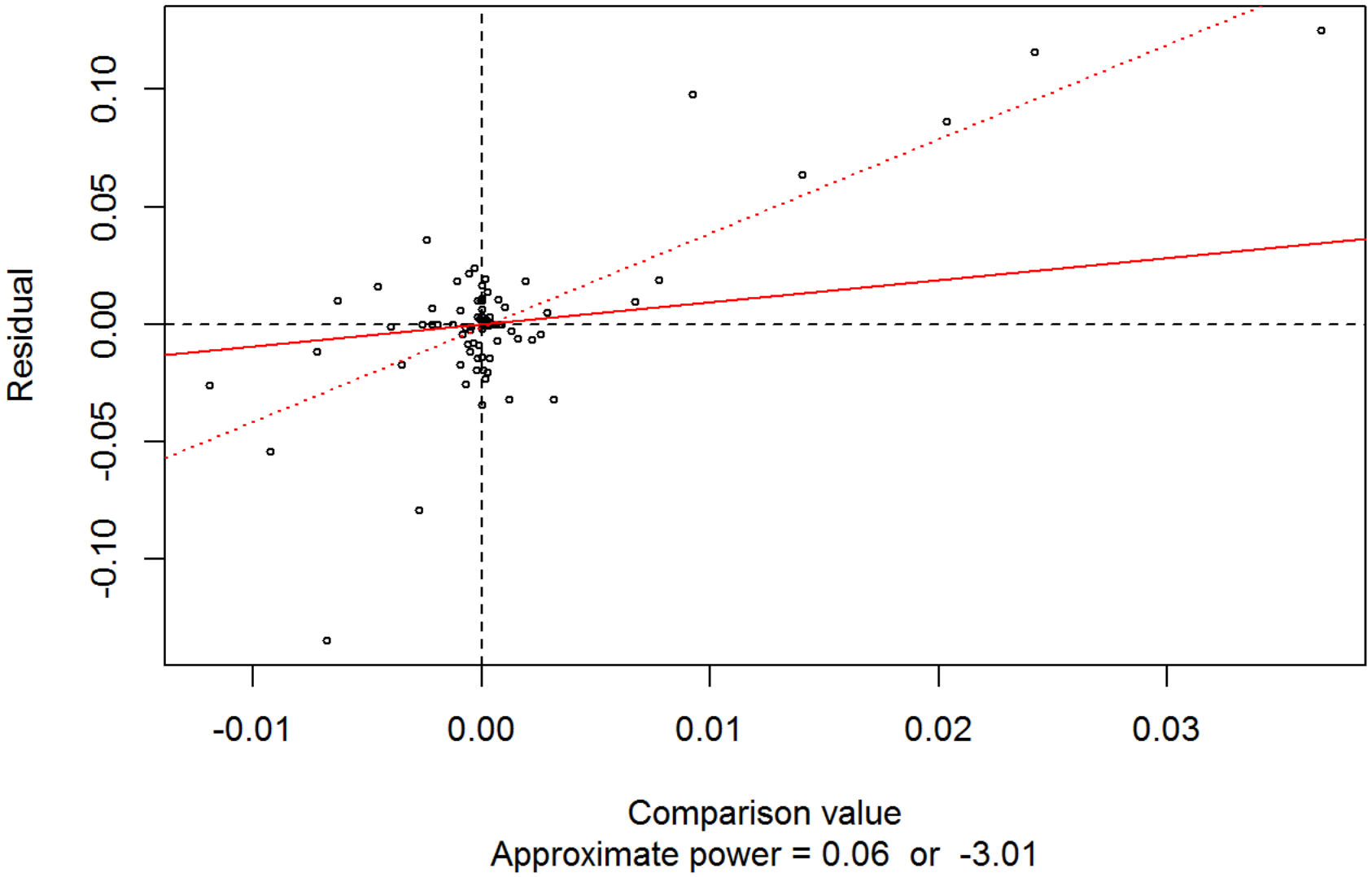
```r
#c


diag.MP <- function(fit){
  source("rrline.r")
  fit.comp <- matrix(fit$row,ncol=1) %*% matrix(fit$col,nrow=1)/fit$overall
  plot(fit.comp, fit$res,xlab="Comparison value",ylab="Residual",cex=0.5)
  abline(v=0,h=0,lty=2)
  ls <- lm(c(fit$res)~c(fit.comp))
  abline(ls,col="red",lty=3)
  rr <- run.rrline(fit.comp,fit$res,iter=10)
  abline(rr$a, rr$b, col="red")
  pwr1 <- 1 - rr$b
  pwr2 <- 1 - ls$coef[2]
  title("",paste("Approximate power =",format(round(pwr1,2))," or ", format(round(pwr2,2))))
}
diag.MP(results)
```




```
#d

df.t <- df^(0.17)
results.t <- medpolish(df.t)
```

```
## 1: 1.76713
## 2: 1.487822
## Final: 1.47599
```

```
diag.MP(results.t)
```




```
analog_r2<- 1-((sum(abs(results.t$residuals))) /(sum(abs(df.t-results.t$overall))))
print(paste("Analog R square after Re-Expression : ",analog_r2))
```

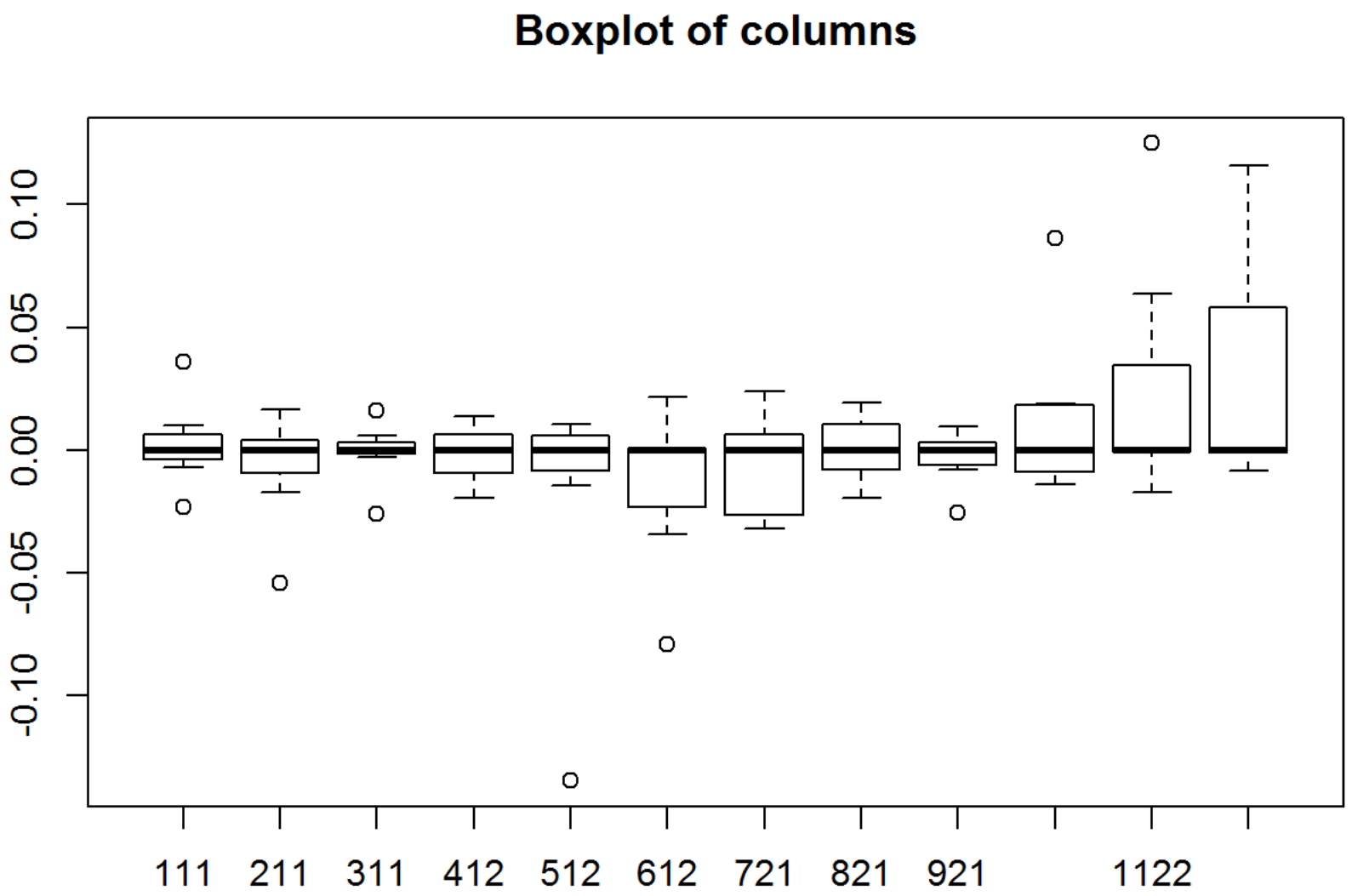
```
## [1] "Analog R square after Re-Expression :  0.847646504570185"
```

```
#e



stem(results.t$residuals,2)
```

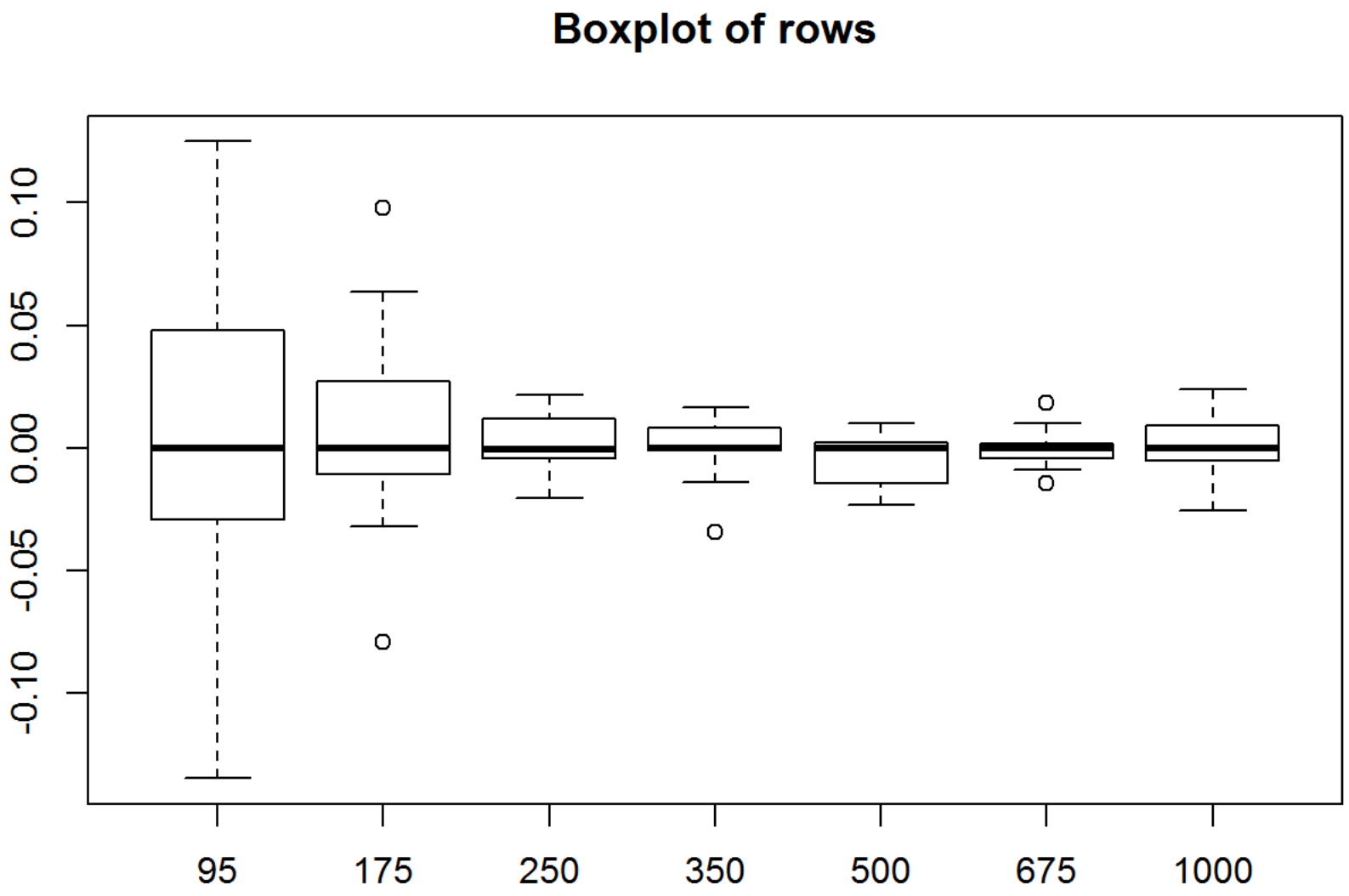
```
##
##   The decimal point is 2 digit(s) to the left of the |
##
##   -12 | 5
##   -10 |
##    -8 |
##    -6 | 9
##    -4 | 4
##    -2 | 422663100
##    -0 | 774442298877644322111100
##     0 | 000000000000001122333566779000001467899
##     2 | 146
##     4 |
##     6 | 4
##     8 | 68
##    10 | 6
##    12 | 5
```

```
# 5 is an outlier in the above stem leaf plot



```

```
#f
boxplot(results.t$residuals, main = "Boxplot of columns")
```



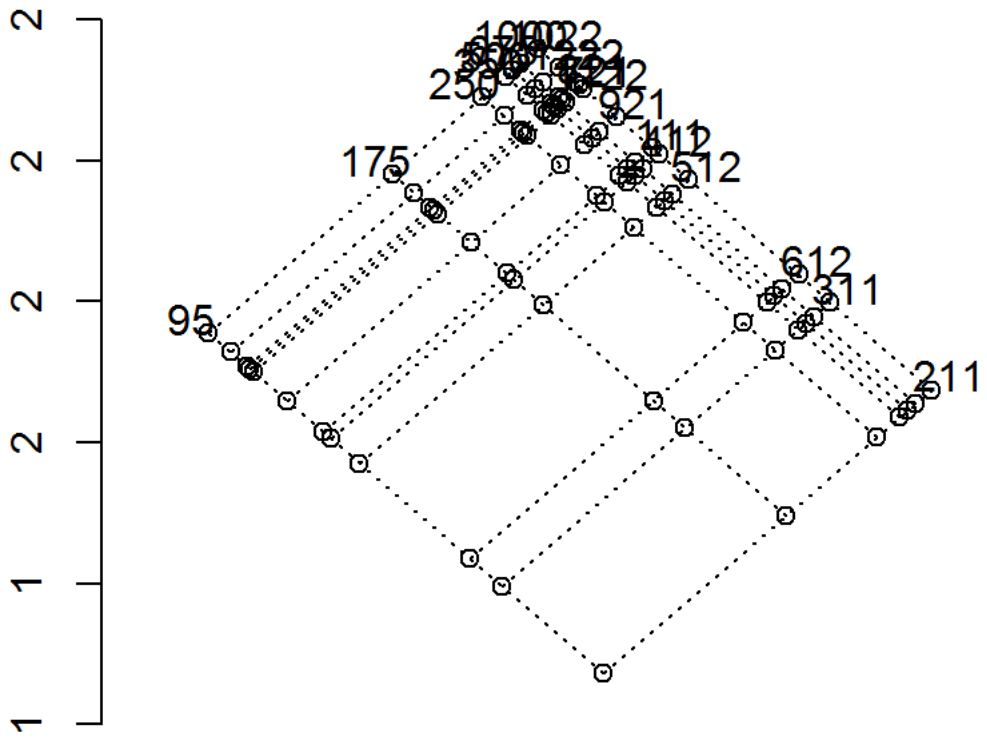
```
boxplot(t(results.t$residuals), main = "Boxplot of rows")
```




```
#g
```

```r
forgetitplot <- function(outmpol,outlim=0,...) {
  # outmpol is output of medpolish in library(eda) or library(stats)
  # be sure to assign dimnames to matrix being polished
  oldpar <- par()
  par(fig=c(0,.7,0,1))
  nc <- length(outmpol$col)
  nr <- length(outmpol$row)
  a <- rep(outmpol$row,nc)
  b <- rep(outmpol$col,rep(nr,nc))
  sqrt2 <- sqrt(2)
  ab <- cbind((a-b)/sqrt2,(a+b)/sqrt2)
  xrange <- range(ab[,1]) + c(-.1,.1)*(max(ab[,1])-min(ab[,1]))
  yrange <- range(ab[,2]) + c(-.1,.1)*(max(ab[,2])-min(ab[,2]))
  dx <- (xrange[2]-xrange[1])/50
  dy <- (yrange[2]-yrange[1])/50
  plot(ab[,1],ab[,2],axes=F,xlim=xrange,ylim=yrange,xlab="",ylab="",...)
  segments((min(a)-outmpol$col)/sqrt2, (min(a)+outmpol$col)/sqrt2,
           (max(a)-outmpol$col)/sqrt2, (max(a)+outmpol$col)/sqrt2,lty=3)
  segments((outmpol$row-min(b))/sqrt2, (outmpol$row+min(b))/sqrt2,
           (outmpol$row-max(b))/sqrt2, (outmpol$row+max(b))/sqrt2,lty=3)
  # segments((outmpol$row)/sqrt2-min(b), (outmpol$row)/sqrt2+min(b),
  # (outmpol$row)/sqrt2-max(b), (outmpol$row)/sqrt2+max(b),lty=3)
  yrowloc <- rep(max(b),nr)
  xrowloc <- outmpol$row
  # text((xrowloc-yrowloc)/sqrt2-dx,dy+(xrowloc+yrowloc)/sqrt2,format(1:nr))
  text((xrowloc-yrowloc)/sqrt2-dx,dy+(xrowloc+yrowloc)/sqrt2,
       names(sort(outmpol$row)))
  xcolloc <- rep(max(a),nc)
  ycolloc <- outmpol$col
  # text(dx+(xcolloc-ycolloc)/sqrt2,dy+(xcolloc+ycolloc)/sqrt2,format(1:nc))
```

```
  text(dx+(xcolloc-ycolloc)/sqrt2,dy+(xcolloc+ycolloc)/sqrt2,
       names(sort(outmpol$col)))
  ynames <- format(round(outmpol$overall + sqrt2*pretty(ab[,2])))
  axis(2,at=pretty(ab[,2]),labels=ynames)
  # add vertical lines when there is an outlier
  if(abs(outlim) > 1e-4) {
    out.index <- which(abs(outmpol$res) > outlim, arr.ind=T)
    # find (r,c) for outlier indices
    zz.x <- outmpol$row[out.index[,1]]
    zz.y <- outmpol$col[out.index[,2]]
    # outlier points at (zz.x-zz.y)/sqrt2, (zz.x+zz.y)/sqrt2
    # draw segment from here to end of residual
    segments((zz.x-zz.y)/sqrt2, (zz.x+zz.y)/sqrt2,
             (zz.x-zz.y)/sqrt2, (zz.x+zz.y)/sqrt2 + outmpol$res[out.index])
  }
  par <- oldpar
  invisible()
}
forgetitplot(results.t)
```




```
# from the plot

#i Factor 95 is more infulential than the others
#ii  At plant 311 Co2 level 1000 had maximum effect
```

```r
#h


#Bootstrap estimates



bootstrap <- function(mat, R)
{
  row.est <- matrix(0, nrow = R, ncol = nrow(mat))
  col.est <- matrix(0, nrow = R, ncol = ncol(mat))
  overall.est <- c()
  res <- medpolish(mat)
  for(i in 1:R)
  {
    sample.dat <- sample(mat, nrow(mat)*ncol(mat), replace = T)
    samp.mat <- matrix(sample.dat, nrow = nrow(mat), ncol = ncol(mat))
    bs <- medpolish(samp.mat)
    overall.est[i] <- bs$overall
    row.est[i,] <- bs$row
    col.est[i,] <- bs$col
  }
  r.est <- apply(row.est, 2, mean)
  c.est <- apply(col.est, 2, mean)

  rsd.est <- apply(row.est, 2, sd)
  csd.est <- apply(col.est, 2, sd)
```

```
  overall <- mean(overall.est)
  overallsd <- sd(overall.est)
  return(list(row = r.est, col = c.est, overall = overall, rowsd=rsd.est,colsd=csd.est,osd = ov
erallsd ))
}

bs <- bootstrap(df, 1000)
```

```
print(bs)
```

```
## $row
## [1] -0.3909438 -0.2707250 -0.3047375 -0.3128438 -0.3928875  0.1096812
## [7] -0.4524125
##
## $col
##  [1] -0.2085312 -0.1819562 -0.4244687 -0.3441187 -0.4650813 -0.2327562
##  [7] -0.0839375 -0.4536750 -0.1524937 -0.4388687 -0.4168938 -0.1898125
##
## $overall
## [1] 28.10316
##
## $rowsd
## [1] 4.547916 4.523209 4.501224 4.477641 4.574253 4.521811 4.488434
##
## $colsd
##  [1] 5.413394 5.475253 5.786246 5.661184 5.448890 5.536700 5.585822
##  [8] 5.535339 5.539726 5.748807 5.699131 5.588509
##
## $osd
## [1] 2.719414
```