```
> x <-
c(16.5,4.4,10.1,10.6,7.8,14.1,7.5,8.5,5.6,10.2,9.4,10.3,7.6,5.4,11.9,10.4,9.2,
8.0)
> x
 [1] 16.5  4.4 10.1 10.6  7.8 14.1  7.5  8.5  5.6 10.2  9.4 10.3  7.6  5.4
 11.9 10.4  9.2  8.0


 > wilcox.test(x,mu=8,alternative="greater")

        Wilcoxon signed rank test with continuity correction

data:  x
V = 109.5, p-value = 0.06189
alternative hypothesis: true location is greater than 8

Warning messages:
1: In wilcox.test.default(x, mu = 8, alternative = "greater") :
  cannot compute exact p-value with ties
2: In wilcox.test.default(x, mu = 8, alternative = "greater") :
  cannot compute exact p-value with zeroes


> library(exactRankTests)

> wilcox.exact(x,mu=8,alternative="greater")

        Exact Wilcoxon signed rank test

data:  x
V = 109.5, p-value = 0.06155
alternative hypothesis: true mu is greater than 8



> wilcox.exact(x,mu=8,conf.int=TRUE,alternative="two.sided")

        Exact Wilcoxon signed rank test

data:  x
V = 109.5, p-value = 0.1231
alternative hypothesis: true mu is not equal to 8
95 percent confidence interval:
  7.75 10.65
sample estimates:
(pseudo)median
         9.225
```

Install first from mirror

```
> x
 [1] 16.5  4.4 10.1 10.6  7.8 14.1  7.5  8.5  5.6 10.2  9.4 10.3
 7.6  5.4 11.9 10.4  9.2  8.0


> perm.test(x,mu=8,exact=TRUE,tol=0.001,alternative="greater")

        1-sample Permutation Test using rounded scores)

data:  x
T = 33.2, p-value = 0.0402
alternative hypothesis: true mu is greater than 8
```

```
> phyper(2,10,10,10)
[1] 0.01150707

> 1-phyper(7,10,10,10)
[1] 0.01150707

> phyper(2,10,10,10) + (1.0-phyper(7,10,10,10))
[1] 0.02301414

> xmatrix <- matrix(c(2,8,8,2),nrow=2)

> fisher.test(xmatrix,alternative="two.sided")

        Fisher's Exact Test for Count Data

data:  xmatrix
p-value = 0.02301
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.004177187 0.763628520
sample estimates:
odds ratio
0.07546953
```

```
> y <- c(10.2, 10.5, 10.3, 10.8,9.8,10.6,10.7,10.2,10.0,10.6)
> x <- c(9.4,9.8,10.1,10.2,9.9,9.7,10.3,9.6,9.8,9.5,9.6)
> library(exactRankTests)
> wilcox.exact(y,x,alternative="two.sided",exact=T,paired=F,conf.int=T)

        Exact Wilcoxon rank sum test

data:  y and x
W = 98.5, p-value = 0.001143
alternative hypothesis: true mu is not equal to 0
95 percent confidence interval:
 0.3 0.9
sample estimates:
difference in location
                  0.55
```

W_{xy}

$153.5 - \frac{(10)(11)}{2} = 98.5$

W_S = Wilcoxon

W_{xy} = Mann-Whitney

Ansari Bradley

```
> x <- c(210, 217, 236, 240, 195, 215, 200, 197, 223, 227)
> y <- c(221, 210, 215, 202, 204, 196, 225, 227, 215, 217)
> ansari.test(x,y,alternative="two.sided")

        Ansari-Bradley test

data:  x and y
AB = 46.5, p-value = 0.22
alternative hypothesis: true ratio of scales is not equal to 1

Warning message:
In ansari.test.default(x, y, alternative = "two.sided") :
  cannot compute exact p-value with ties
> ansari.exact(x,y,alternative="two.sided")

        Ansari-Bradley test

data:  x and y
AB = 46.5, p-value = 0.1979
alternative hypothesis: true ratio of scales is not equal to 1
```

Symmetry

Untitled

```
> x
 [1] 16.5  4.4 10.1 10.6  7.8 14.1  7.5  8.5  5.6 10.2  9.4 10.3  7.6  5.4 11.9
[16] 10.4  9.2  8.0

> library(lawstat)
> symmetry.test(x)

        Test of Symmetry - MGG Test

data:  x
Test Statistic = 0.0113, p-value = 0.9909


> utils:::menuInstallPkgs()
trying URL
'http://cran.cnr.Berkeley.edu/bin/windows/contrib/2.14/exactRankTests_0.8-22.zip'
Content type 'application/zip' length 116367 bytes (113 Kb)
opened URL
downloaded 113 Kb

package 'exactRankTests' successfully unpacked and MD5 sums checked

The downloaded packages are in
        C:\Users\mreiser\AppData\Local\Temp\RtmpkNxOgw\downloaded_packages


> library(exactRankTests)
Loading required package: survival

Attaching package: 'survival'

The following object(s) are masked from 'package:boot':

    aml

 Package 'exactRankTests' is no longer under development.
 Please consider using package 'coin' instead.

Warning message:
package 'exactRankTests' was built under R version 2.14.2


> perm.test(x,mu=8,exact=TRUE,tol=0.001,alternative="greater")

        1-sample Permutation Test using rounded scores)

data:  x
T = 33.2, p-value = 0.0402
alternative hypothesis: true mu is greater than 8

>
```

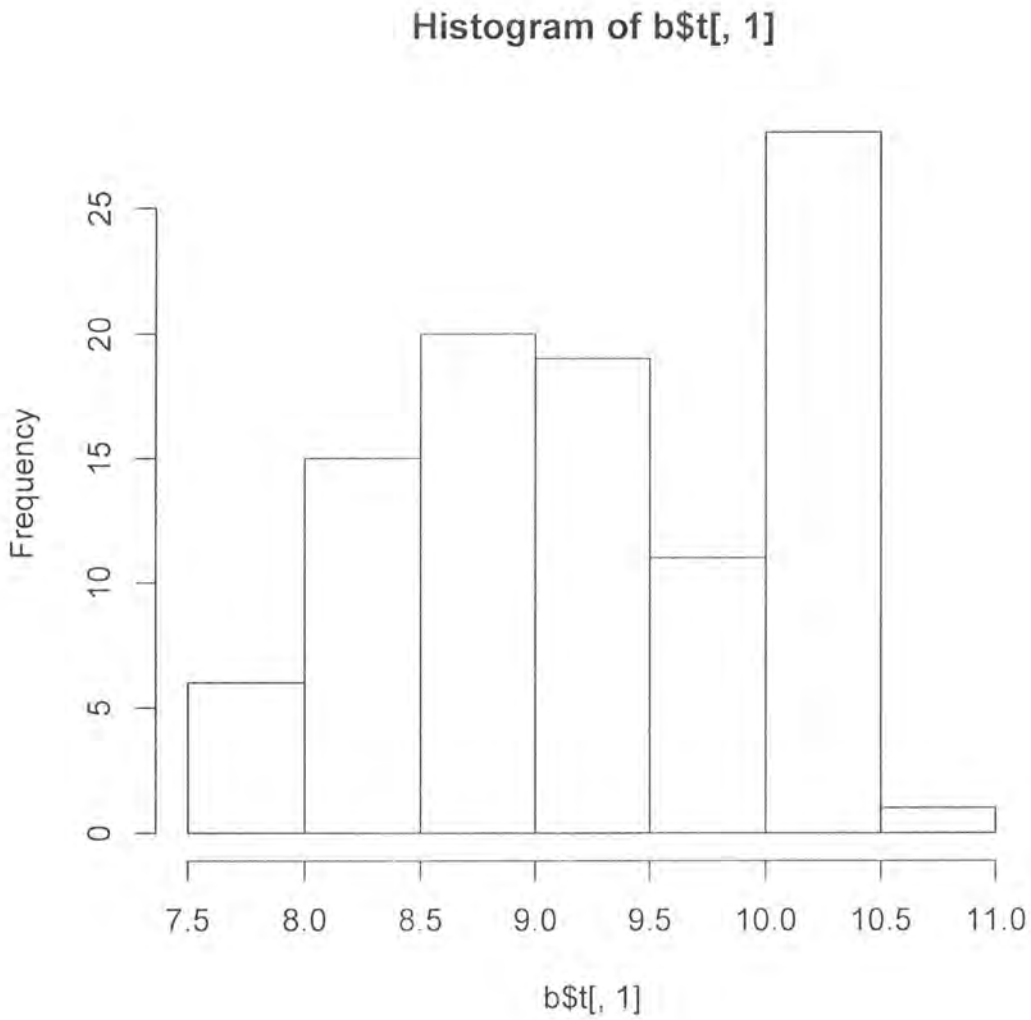
install forest

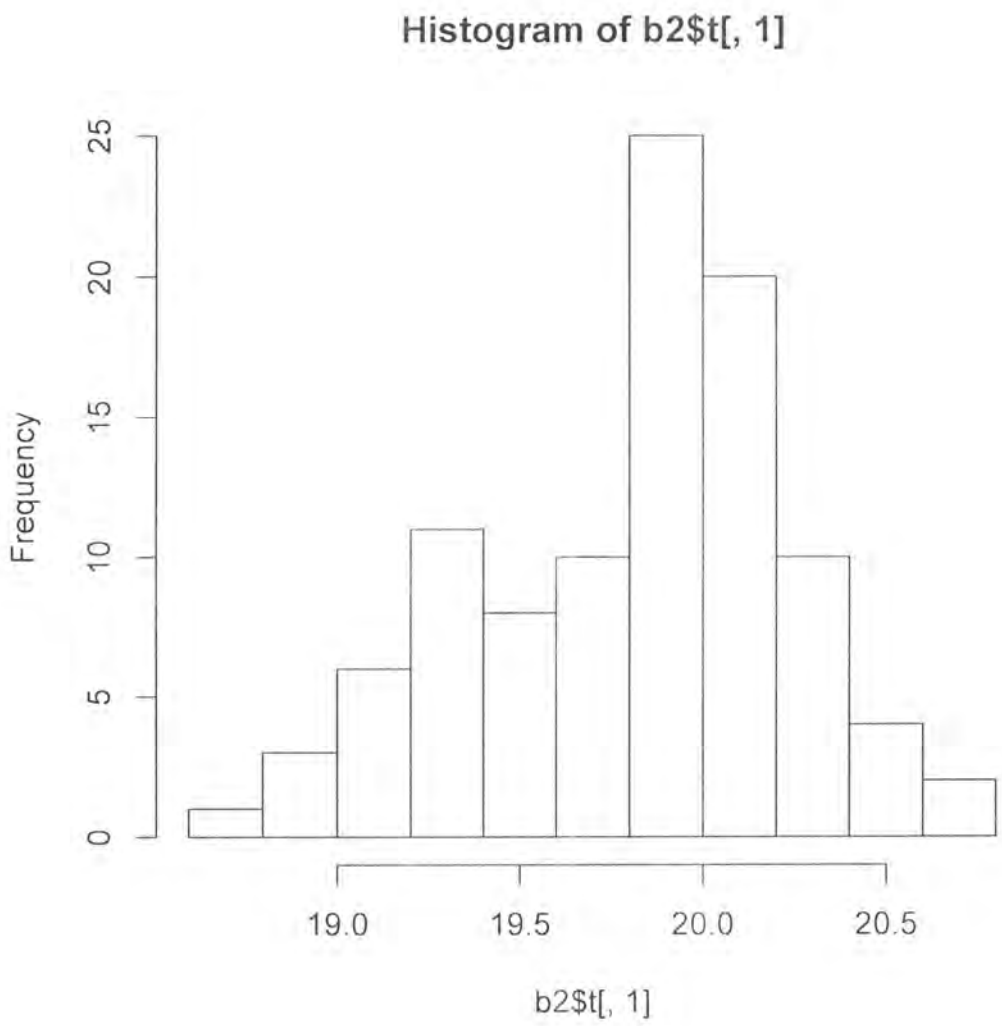
```
> x
 [1] 16.5  4.4 10.1 10.6  7.8 14.1  7.5  8.5  5.6 10.2  9.4 10.3  7.6  5.4 11.9
[16] 10.4  9.2  8.0

> library(boot)

> psample <- sample(x)

> median(x)
[1] 9.3
> median(psample)
[1] 9.3

> psample <- sample(x,replace=T)
> psample
 [1]  7.5 14.1  5.6  8.0 11.9 10.1  8.5 10.2 10.3  5.4  7.5 10.4  5.6  5.6  9.2
[16]  5.6  8.5 11.9

> median(psample)
[1] 8.5

> psample2 <- sample(x,replace=T)
> psample2
 [1] 16.5  7.5  8.0  7.8  8.0 10.3 10.3  5.4  7.5  5.6 10.6 11.9 11.9 10.1  8.5
[16]  5.4 10.3  7.5
> median(psample2)
[1] 8.25

> resamples <- lapply(1:20, function(i) sample(x,replace=T))
> resamples
[[1]]
 [1]  7.5 10.4  7.8  8.5  8.0 10.6 10.1  8.5 10.4 11.9  9.2 16.5  4.4 10.4  9.4
[16]  5.4 10.3  7.5

[[2]]
 [1]  9.2  7.6  7.6  7.6 10.1 10.2  7.8  5.6  5.4  4.4 10.1 14.1 10.2  9.4 16.5
[16]  5.4  4.4 10.6

[[3]]
 [1] 10.4  8.5 10.3 14.1  4.4  5.6  7.5  9.2  7.6  9.4 14.1  9.2  8.5 10.3 10.1
[16]  8.5  4.4  9.2

[[4]]
 [1]  8.0 11.9 10.3  7.8 11.9 10.3 10.4  7.6  5.4  7.8  5.4  9.2 16.5  4.4 14.1
[16] 10.1  8.5  8.0

[[5]]
 [1]  4.4 10.2 10.2  7.5  4.4  8.5  7.6 10.6 14.1  5.4 14.1 16.5 10.4 10.2 10.1
[16] 10.2  9.2  9.2

[[6]]
 [1]  7.5 10.3 10.2 14.1  7.6  5.6 10.4 11.9 14.1  8.0 10.6 10.2 16.5 10.3 10.2
[16]  5.6  5.6  4.4

[[7]]
 [1]  5.4  9.2 14.1  7.5  5.4  4.4  8.0 14.1 14.1 14.1 10.4  7.6  4.4 10.2 10.6
[16]  8.0 11.9  4.4

[[8]]
 [1] 11.9  7.8  9.2  4.4  5.4  4.4  7.6 14.1  9.2  7.8 16.5  9.4 14.1  7.8  9.4
[16] 10.2 10.6  5.4

[[9]]
```

```
 [1]  8.5  9.4 10.1 10.6 10.2 16.5  7.5 10.4 16.5 10.1  4.4 10.4  5.6  8.0  9.4
[16] 10.4  7.8  8.5

[[10]]
 [1]  8.5  7.8 14.1  5.4  4.4  8.5 14.1  5.4 11.9 10.3  7.8  9.4  7.8 10.2  8.0
[16]  9.2 10.4  9.2

[[11]]
 [1]  7.8  8.5  7.8 10.2  4.4  9.2 16.5  9.4  5.6 14.1 16.5  9.2 10.2 10.1  9.2
[16] 10.4  9.2 10.4

[[12]]
 [1] 11.9  9.4  5.6  9.2 16.5 14.1  4.4  7.5  9.2 10.1  9.2  9.2 10.4 11.9 10.1
[16] 10.6 14.1 10.4

[[13]]
 [1] 10.4  4.4  9.2  4.4  7.5  5.4  9.2 10.6  7.5  7.8 10.4  5.6 11.9 10.6 11.9
[16] 10.2  5.4 10.2

[[14]]
 [1] 14.1  4.4 16.5  5.4  4.4 16.5  7.5  5.4 10.4 10.3 10.6 10.2 10.4  7.8  7.6
[16]  8.0 14.1  5.6

[[15]]
 [1] 10.6 10.2 10.2 10.6 14.1  9.2 10.1 14.1  9.2 10.3 10.4  9.4  8.0 10.3  7.6
[16]  8.0 11.9  8.0

[[16]]
 [1] 10.6 10.6  7.5 10.2  7.8  5.6  8.5 16.5  8.5 16.5  4.4 10.3  8.0  8.0  9.4
[16]  8.0  7.5 10.3

[[17]]
 [1]  9.4  9.4 10.3 10.3  4.4 10.1  7.5  8.0  7.5 10.1  7.5 10.4 10.6  7.6  8.0
[16] 10.3  5.4 10.1

[[18]]
 [1]  5.4  8.5 10.6  7.5  7.5  7.8  9.4 10.4 10.3  9.2 10.1  8.0 11.9  9.4  4.4
[16]  7.5  9.4 10.4

[[19]]
 [1]  9.2 10.1  5.4 11.9 10.6 10.6  5.6  7.6  9.2  9.2 10.3 11.9  4.4  9.2 10.1
[16] 10.4 10.3  8.5

[[20]]
 [1] 14.1  9.2 16.5  8.0  4.4 10.6 16.5  5.4  9.4  7.5  5.4  5.6 16.5  7.8  5.4
[16]  9.4 10.1  5.4

> r.median <- sapply(resamples,median)

> r.median
 [1]  9.30  8.50  9.20  8.85 10.15 10.20  8.60  9.20  9.75  8.85  9.30 10.10
[13]  9.20  9.10 10.20  8.50  9.40  9.30  9.65  8.60

> sort(r.median)
 [1]  8.50  8.50  8.60  8.60  8.85  8.85  9.10  9.20  9.20  9.20  9.30  9.30
[13]  9.30  9.40  9.65  9.75 10.10 10.15 10.20 10.20

> quantile(r.median, c(0.025, 0.975))
 2.5% 97.5%
  8.5  10.2

> d <- c(3,2,2)
> x[d]
```

```
[1] 10.1  4.4  4.4

> samplemedian <- function(data, d) {return(median(data[d]))}

> b=samplemedian(x,d)
> b
[1] 4.4

> b = boot(x,samplemedian, R=20)
> b

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x, statistic = samplemedian, R = 20)


Bootstrap Statistics :
    original  bias    std. error
t1*      9.3   -0.47   0.8110098

> sort(b$t[,1])
 [1]  7.80  7.80  7.80  7.90  8.15  8.15  8.50  8.50  8.50  8.60  8.60  8.60
[13]  9.30  9.40  9.40  9.75  9.75  9.75 10.10 10.25

> pvalue <- mean(c(b$t[,1]) <= 8)
> pvalue
[1] 0.2

> b = boot(x,samplemedian, R=100)

> quantile(b$t[,1], c(0.025, 0.975))
 2.5% 97.5%
  7.9  10.4

> pvalue <- mean(c(b$t[,1]) <= 8)
> pvalue
[1] 0.06

> hist(b$t[,1])

> y <- rnorm(200, 20, 3)
> b2 = boot(y,samplemedian, R=100)
> b2

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = y, statistic = samplemedian, R = 100)


Bootstrap Statistics :
    original     bias    std. error
t1* 19.90868 -0.1138694   0.4184884

> hist(b2$t[,1])

> quantile(b2$t[,1], c(0.025, 0.975))
    2.5%    97.5%
18.95668 20.53951
>
```

column 1 of vector t in data result b


Histogram of b$t[, 1]
Frequency
0
5
10
15
20
25
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0
b$t[, 1]


Histogram of b2$t[, 1]
Frequency
0
5
10
15
20
25
19.0
19.5
20.0
20.5
b2$t[, 1]

Permutation Test
Two Sample

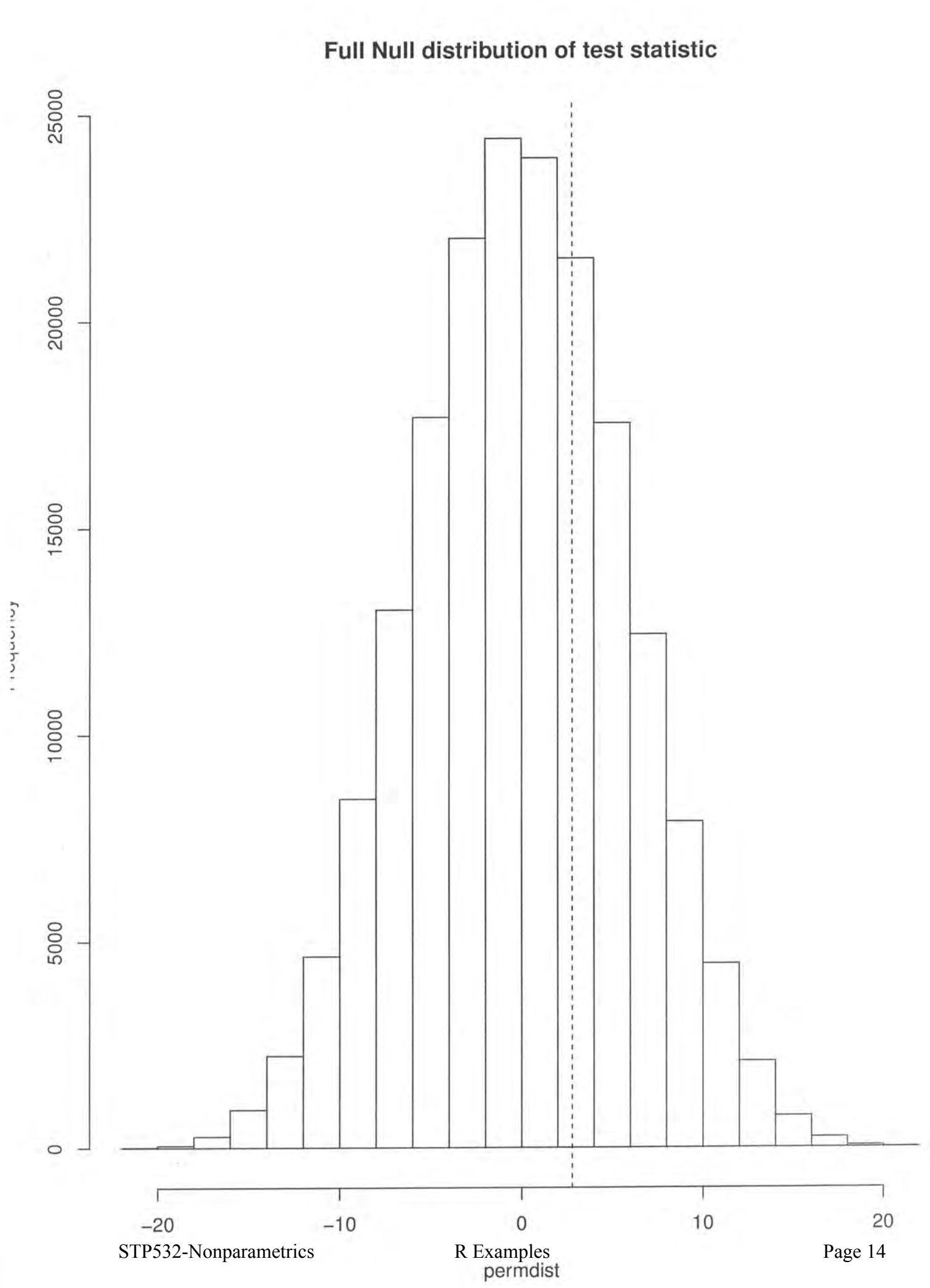
```
> A <- combn(5,2)
> A
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    2    2    2    3    3     4
[2,]    2    3    4    5    3    4    5    4    5     5
> x <- c(35,25)
> y <- c(50,30,70)
>
  > mean(y)-mean(x)
[1] 20
> xandy <- c(x,y)
> xandy
[1] 35 25 50 30 70
> d <- A[,1]
> mean(xandy[d])-mean(xandy[-d])
[1] -20

> permdist <- rep(NA,10)
> sum1and2 <- rep(NA,10)
> for(k in 1:10)
  + {d <- A[,k]
     +  permdist[k] <- mean(xandy[d])-mean(xandy[-d])
     +  sum1and2[k] <- sum(xandy[d])
     + }
> permdist
[1] -20.0000000   0.8333333 -15.8333333  17.5000000  -7.5000000 -24.1666667   9.1666667  -3.3333333  30.0000000
13.3333333
> sum1and2
[1]  60  85  65 105  75  55  95  80 120 100
>
  > permdistS <- sort(permdist)
> sum1and2S <- sort(sum1and2)
> permdistS
[1] -24.1666667 -20.0000000 -15.8333333  -7.5000000  -3.3333333   0.8333333   9.1666667  13.3333333  17.5000000
30.0000000
>
  > sum1and2S
[1]  55  60  65  75  80  85  95 100 105 120
```

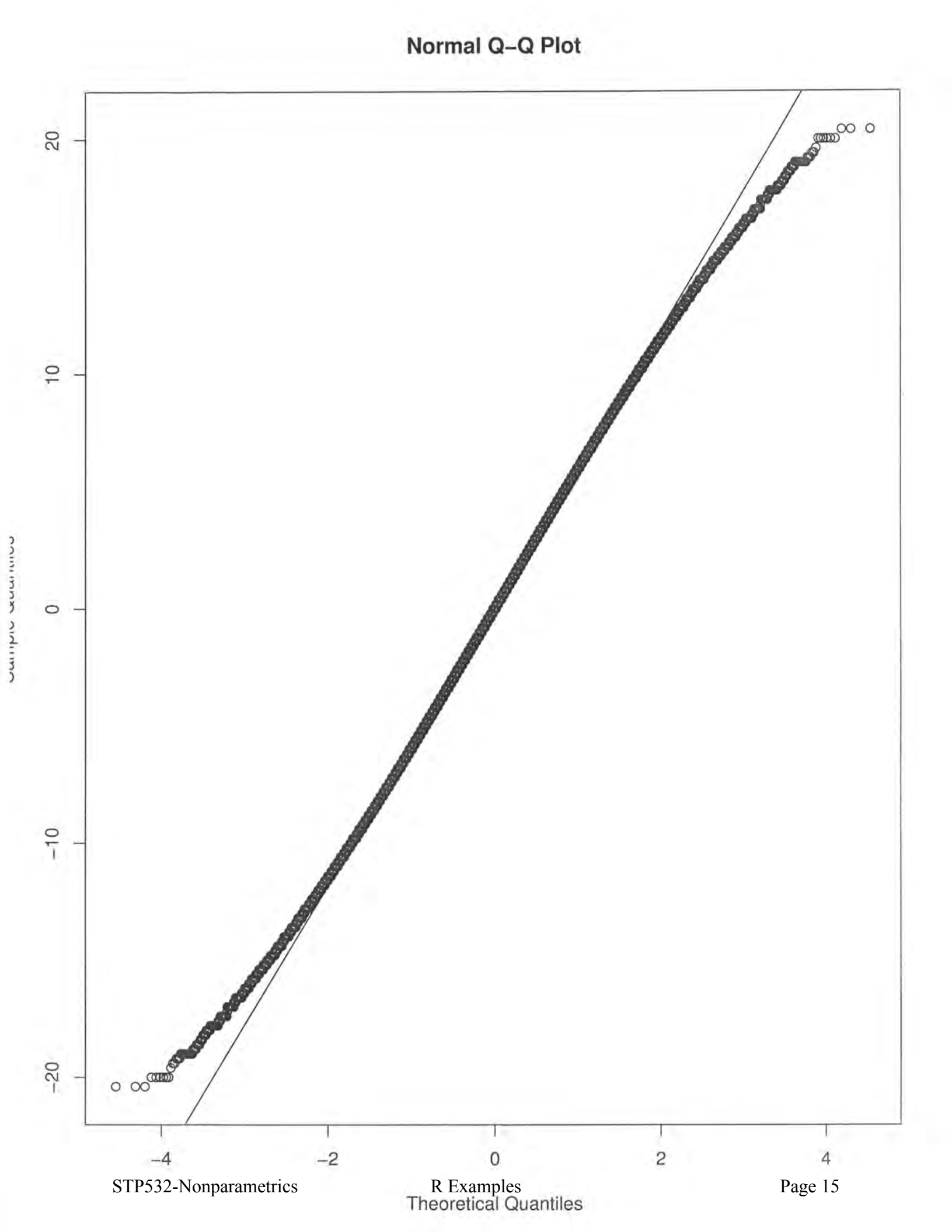
Permutation Test Full Distribution

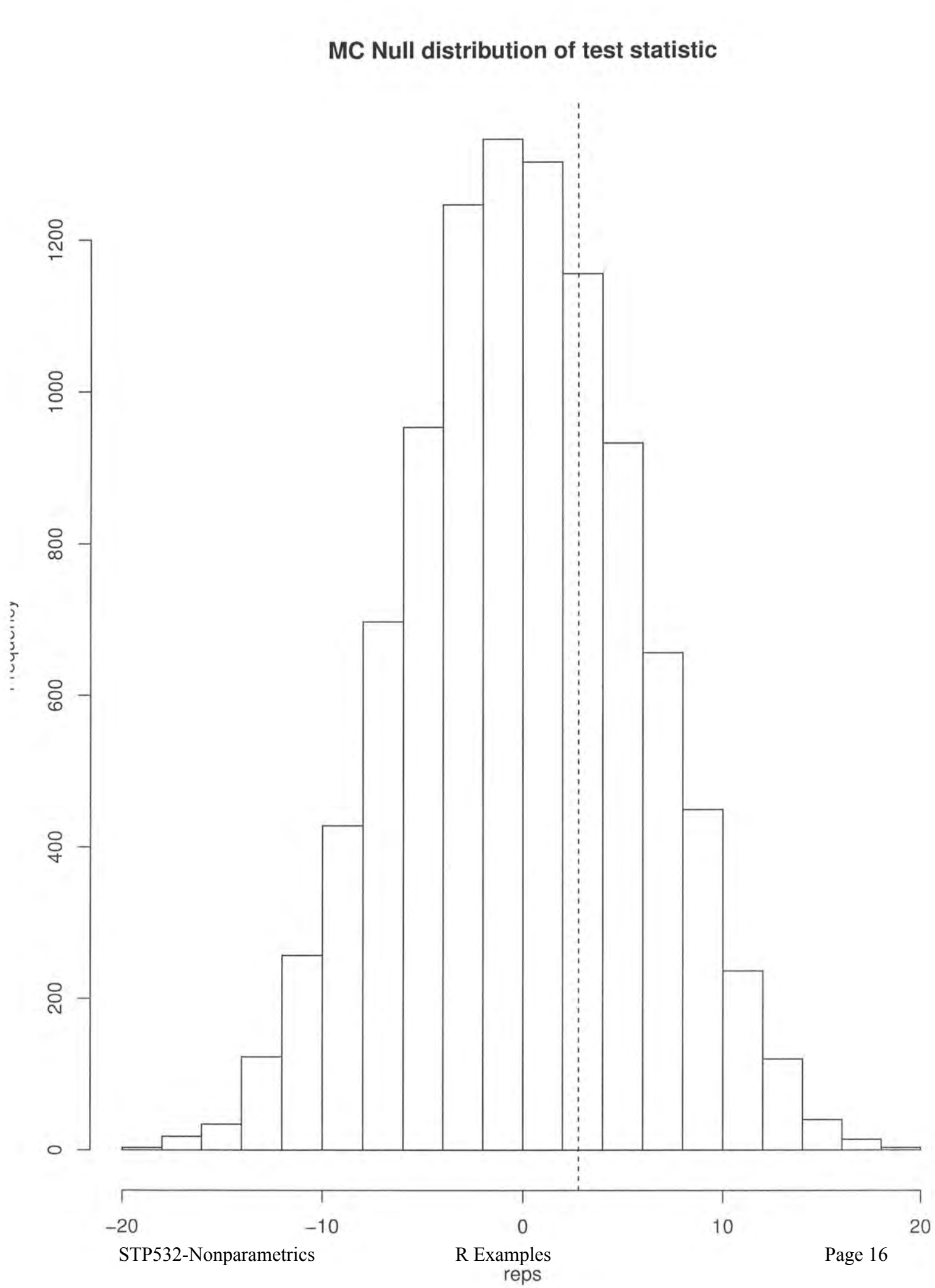
```
> size <- choose(20,10)
> A <- combn(20,10)

> permdist <- rep(NA,size)
> x <- c(210, 217, 236, 240, 195, 215, 200, 197, 223, 227)
> y <- c(221,210,215,202,204,196,225,227,215,217)
> xandy <- c(x,y)
> mean(x)-mean(y)
[1] 2.8
> ts <- mean(x)-mean(y)

> for (k in 1:size)
  + {d <- A[,k]
     + permdist[k] <- mean(xandy[d])-mean(xandy[-d])
     + }


> pvalue = mean(abs(permdist) >= abs(ts) )

> pvalue
[1] 0.6502414
>
```

Monte Carlo Permutation Test

```
  > R <- 9999
> k <- 1:length(xandy)

> reps <- numeric(R)
> for (i in 1:R) {
  + d <- sample(k, size=length(x), replace=FALSE)
  + xi <- xandy[d]
  + yi <- xandy[-d]
  + reps[i] <- mean(xi) - mean(yi)
  + }

> pvalue <- mean(abs(c(ts,reps)) >= abs(ts))
> pvalue
[1] 0.6449

> hist(reps, main="MC Null distribution of test statistic")
> abline(v=ts, lty=2)
> hist(permdist, main="Full Null distribution of test statistic")
> abline(v=ts, lty=2)
>
```


Full Null distribution of test statistic
25000
20000
15000
10000
5000
0
−20
−10
0
10
20
permdist

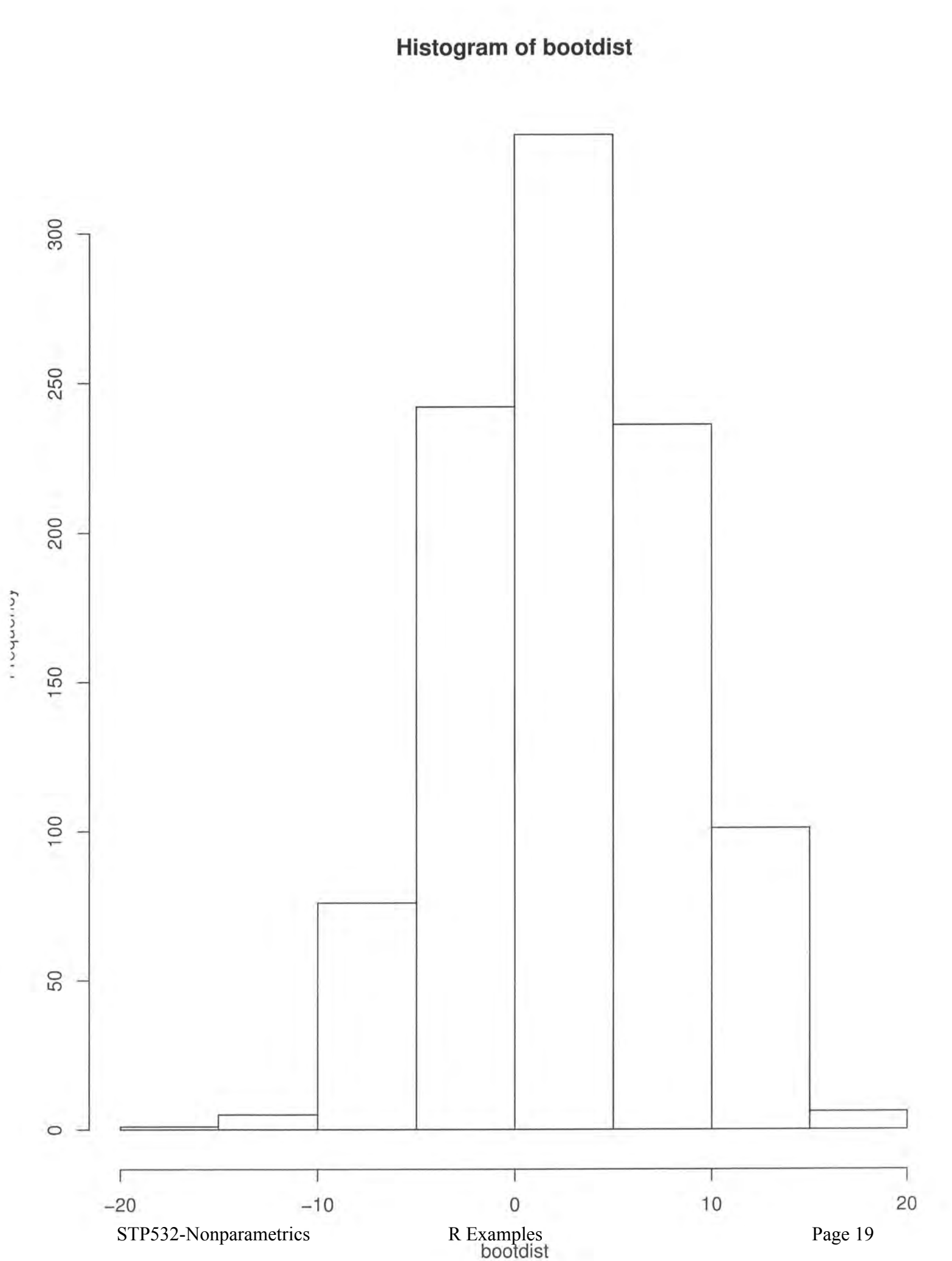

Normal Q–Q Plot
Sample Quantiles
Theoretical Quantiles
20
10
0
−10
−20
−4
−2
0
2
4

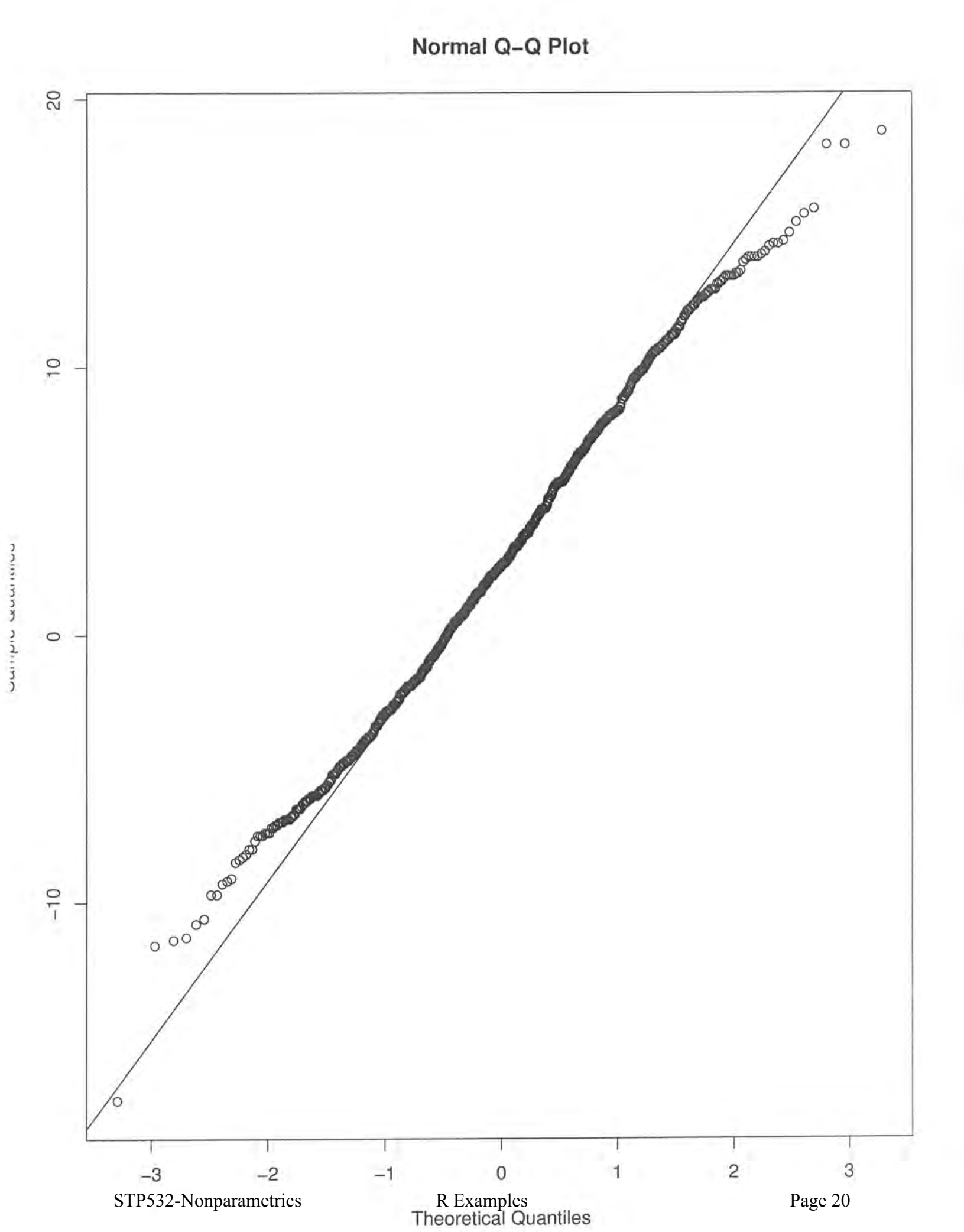

MC Null distribution of test statistic
Frequency
0
200
400
600
800
1000
1200
−20
−10
0
10
20
reps

```
> ts <- t.test(x,y,var.equal=TRUE)$statistic
> ts
t
0.4718208

> for (i in 1:R) {
  + d <- sample(k,size=length(x), replace=FALSE)
  + xi <- xandy[d]
  + yi <- xandy[-d]
  + reps[i] <- t.test(xi,yi,var.equal=TRUE)$statistic
  + }
> pvalue <- mean(abs(c(ts,reps)) >= abs(ts))
> pvalue
[1] 0.6439
```

Bootstrap Test of Differences

```
> ts <- mean(x)-mean(y)
> R<-1000
> bootdist <- rep(NA,R)

> for (i in 1:R) {
  + xsample <- sample(x,size=length(x), replace=TRUE)
  + ysample <- sample(y,size=length(y), replace=TRUE)
  + bootdist[i] <- mean(xsample)-mean(ysample)
  + }

> est.bias <- mean(bootdist)- ts
> est.bias
[1] -0.0652

> pvalue = mean(abs(bootdist) >= abs(ts))
> pvalue
[1] 0.648
> hist(bootdist)
> qqnorm(bootdist)
> qqline(bootdist)
```


Histogram of bootdist
Frequency
0
50
100
150
200
250
300
−20
−10
0
10
20
bootdist


Normal Q–Q Plot
Sample Quantiles
Theoretical Quantiles
20
10
0
−10
−3
−2
−1
0
1
2
3

```
> x
[1] 210 217 236 240 195 215 200 197 223 227
> y
[1] 221 210 215 202 204 196 225 227 215 217
> ts <- mean(x)-mean(y)

> R<-1000
> bootdist <- rep(NA,R)
> for (i in 1:R) {
  + xsample <- sample(x,size=length(x), replace=TRUE)
  + ysample <- sample(y,size=length(y), replace=TRUE)
  + bootdist[i] <- mean(xsample)-mean(ysample)
  + }
> est.bias <- mean(bootdist)- ts
> est.bias
[1] -0.1263
> pvalue = mean(abs(bootdist) >= abs(ts))
> pvalue
[1] 0.633

> quantile(bootdist,c(0.025,0.975))
2.5%   97.5%
  -7.8025 14.1025

> thrustdata <- data.frame(x,y)
> thrustdata
x   y
1  210 221
2  217 210
3  236 215
4  240 202
5  195 204
6  215 196
7  200 225
8  197 227
9  223 215
10 227 217

> thrustdata$x
[1] 210 217 236 240 195 215 200 197 223 227

> mean(thrustdata$x)
[1] 216
> mean(x)
[1] 216

> xsize=10
> ysize=10

> meandiff <- function(data,d1,d2) {d1=sample(1:xsize,size=xsize,replace=TRUE)
                                    + d2=sample(1:ysize,size=ysize,replace=TRUE)
                                    + return(mean(data$x[d1])- mean(data$y[d2]))}
> bootresult3 <- boot(thrustdata,meandiff,R=1000)
> bootresult

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
  boot(data = thrustdata, statistic = meandiff2, R = 1000)


Bootstrap Statistics :
  original  bias    std. error
t1*      2.8  0.1161    6.459086

> boot.ci(boot.out=bootresult3, type=c("perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
  boot.ci(boot.out = bootresult3, type = c("perc", "bca"))

Intervals :
  Level     Percentile            BCa
95%   ( -8.1000,  14.4898 )   (-19.1000,   5.3000 )
Calculations and Intervals on Original Scale
Warning : BCa Intervals used Extreme Quantiles
Some BCa intervals may be unstable
Warning message:
  In norm.inter(t, adj.alpha) : extreme order statistics used as endpoints
```

Non parametric
Bootstrap
Two Sample

```
> quantile(bootresult3$t[,1], c(0.025,0.975))
2.5% 97.5%
  -8.10 14.11


> bootresult4 <- boot(thrustdata,meandiff,R=10000)
> boot.ci(boot.out=bootresult4, type=c("perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
  boot.ci(boot.out = bootresult4, type = c("perc", "bca"))

Intervals :
  Level     Percentile            BCa
95%   ( -8.1000,  14.0000 )   (-12.7176,   8.6000 )
Calculations and Intervals on Original Scale
> quantile(bootresult4
          + $t[,1], c(0.025,0.975))
2.5% 97.5%
  -8.1  14.0
```

MCsimulation

```
# In each simulation step, the limits of the estimated confidence intervals
# are saved in vectors l and r
# vector("numeric",100) creates vectors containing 100 0s.
# Alternatively, you can also choose to create vectors consisting of NAs
# In this way, it is easier to distinguish between places that
# have been calculated during simulation and those
# that have not yet been filled during simulation
# 100 simulations -> vectors of length 100

l<-rep(NA,100)
r<-rep(NA,100)

## The simulations are run using a for-loop
## In each step of the for-loop, a new set of 100 individuals is sampled
n<-100

for(i in 1:100)
{# This command is purely informative: in this way, we know
# at which simulation run R is
print(i)
# set.seed declares the seed for the random generator
# If we run the for-loop the next time, results will be the same
# Note that this command is not necessary
set.seed(i)
a1<-sample(1:5815,n,replace=F)
# The lower or left limit of the confidence interval is saved in the vector l
# The upper or right limit of the confidence interval is saved in the vector r
# The i-th places in the vectors correspond to the results of the i-th simulation
l[i]<-t.test(x[a1],alternative="two.sided",mu = 0,conf.level=0.95)$conf.int[1]
r[i]<-t.test(x[a1],alternative="two.sided",mu = 0,conf.level=0.95)$conf.int[2]
}

# proportion of the 100 confidence intervals containing the population mean
mean((l<=mean(x,na.rm=T))&(mean(x,na.rm=T)<=r))


# Graphical display of results
win.graph()
plot(c(l,r),c(1:100,1:100),type="n",xlab="Confidence interval",ylab="Sample")
s<-seq(1,100,length=100)
for (i in 1:100)
{ifelse((l[s][i]<=mean(x,na.rm=T))&(mean(x,na.rm=T)<=r[s][i]),b<-1,b<-2)
lines(c(l[s][i],r[s][i]),c(s[i],s[i]),col=b)}
lines(rep(mean(x,na.rm=T),100),1:
```

MC Simulation
Confidence Intervals

```
#Declare sizes of arrays
> lbound <- rep(NA,1000)
> ubound <- rep(NA,1000)

# Generate 20 random values from normal distribution with mean 20 and standard
deviation 3.
# Store the 20 random values in vector x
> x <- rnorm(20,10,3)

#Perform parametric t-test on values in vector x
> t.test(x,alternative="two.sided",mu=10,conf.level=0.95)

        One Sample t-test

data:  x
t = 1.1152, df = 19, p-value = 0.2787
alternative hypothesis: true mean is not equal to 10
95 percent confidence interval:
  9.177157 12.699750
sample estimates:
mean of x
 10.93845

# Perform t-test on values in vector x and store results in "result"
> result <-t.test(x,alternative="two.sided",mu=10,conf.level=0.95)
> result

        One Sample t-test

data:  x
t = 1.1152, df = 19, p-value = 0.2787
alternative hypothesis: true mean is not equal to 10
95 percent confidence interval:
  9.177157 12.699750
sample estimates:
mean of x
 10.93845

# Print confidence interval information from t-test result

> result$conf.int
[1]  9.177157 12.699750  # The lower bound is in conf.int[1] and the upper
bound is in conf.int[2]
attr(,"conf.level")
[1] 0.95

> test<-c(3,4,5,6,7)
> mean(test)
[1] 5
```

```
> mean(5<=test)
[1] 0.6  #This is the proportion greater than or equal to 5
> mean(4>=test)
[1] 0.4

# Define truevalue
> truevalue <- 10

# Perform loop which draws random sample and calculates t-test 1000 times
# 1000 values of lower bound are stored in lbound, and 1000 values of upper
# bound are stored in ubound.  lbound and ubound are vectors.
# This is known as a Monte Carlo simulation.
> for(i in 1:1000)
+ {
+ x=rnorm(20,10,3)
+ lbound[i] <-
t.test(x,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[1]
+ ubound[i] <-
t.test(x,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[2]
+ }

# list lbound
> lbound
   [1]  7.957920  8.790323 10.189575  8.487396  9.571972  9.795656  8.084613
   9.188037  8.277876  9.255778  8.311459  9.152787  8.101265  8.401827
   9.197274  7.686529
  [17]  8.527014  9.903367  8.716144  8.609114  7.771853  8.203858  8.575425
  8.513876  9.375396 10.079283  8.076695  7.501178  8.700142  7.948405
  8.694294 10.284342
  [33]  9.857134  8.405588  8.479591  7.753647  8.405383  8.124858  8.376854
  7.770710  8.944675  9.462978  8.808710  9.421645  8.625798  6.969113
  8.863250  9.137385
  [49]  9.722309  9.332438  7.795967  8.007758  9.966627  8.868293 10.112902
  8.808400  8.846001  9.306550  8.771606  8.308229  8.083660  8.918448
  7.740417  8.594694
  [65]  9.728190  8.391705  8.999075  9.921952  7.830063 10.293732  9.217921
  9.345825  8.909696  8.890652  9.387541  8.103143  8.583834  9.699739
  8.150368  9.222402
  [81]  8.999077  8.085976  8.445892  9.124127  9.297745  9.003012  9.043864
  7.297671  8.694806  9.296255  7.741647  9.096819 10.580842  8.723817
  8.688717  7.544071
  [97]  8.517997  8.120702  9.625649  7.531577  8.879674  7.769847  8.944646
  7.563848  7.919152  7.171218 10.114149  8.440346  8.687198  9.898009
  9.268543 10.622972
 [113]  7.567201  8.701773  8.513382  7.780970  8.658238  8.631987  8.349972
 8.804409  9.170282  9.564289  8.259650  8.471758  7.177760  7.109145
 6.896967  9.881404

# With this syntax, the mean fuction will produce the proportion of the 1000
# confidence intervals that contain the truevalue
> mean((lbound<=truevalue)&(truevalue<=ubound))
```

```
[1] 0.944    # This is the confidence inteval coverage

# Calculate margin of error for confidence interval on proportion of
confidence
# intervals that contain the truevalue
> M <- 1.96*sqrt(0.05*0.95/1000)
> M
[1] 0.01350837

# Calculate lower limit for confidence interval on proportion
> .95 - M

[1] 0.9364916

# Calculate upper limit for confidence interval on proportion
> .95 + M
[1] 0.9635084
#We can see that the coverage, 0.944 is contained in the confidence interval
(0.9365, 0.9635)

# Now run the simulation with random values from uniform distribution
> for(i in 1:1000)
+ {
+ x=runif(20,min=5, max=15)
+
+ }
> for(i in 1:1000)
+ {
+ y=runif(20,min=5, max=15)
+ lbound[i] <-
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[1]
+ ubound[i] <-
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[2]
+ }
> mean((lbound<=truevalue)&(truevalue<=ubound))
[1] 0.948 #The coverage

# Now run simultion for random values from chi-square distribution with df=10
and n=20
> y=rchisq(20,10)
> y
 [1] 17.926402 15.366606  6.513746 16.959809  7.307377 10.433493 14.241665
 8.726017  7.622191  5.175172  8.651299 10.686322  5.752736  5.650084
 10.418913  4.942965
[17] 18.320712 11.380706 12.173928  9.278392
> for(i in 1:1000)
+ {
+ y=rchisq(20,10)
+ lbound[i] <-
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[1]
+ ubound[i] <-
```

```
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[2]
+ }
> mean((lbound<=truevalue)&(truevalue<=ubound))
[1] 0.942

# Now run simulation for random values from chi-square distribution with df=10
and n=10
> for(i in 1:1000)
+ {
+ y=rchisq(10,10)
+ lbound[i] <-
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[1]
+ ubound[i] <-
t.test(y,alternative="two.sided",mu=10,conf.level=0.95)$conf.int[2]
+ }
> mean((lbound<=truevalue)&(truevalue<=ubound))
[1] 0.929

# Obtain confidence interval from Monte Carlo bootstrap distribution using
percentile method
# for one sample.
> x <- rnorm(100,10,3)
> library(boot)
> bresult = boot(x,samplemedian,R=100)
> lb=quantile(bresult$t[,1], 0.025)
> ub=quantile(bresult$t[,1], 0.975)
> mean((lb<=truevalue)&(ub>=truevalue))
[1] 1 # This confidence interval contains the truevalue
> lb
    2.5%
9.723806
> ub
   97.5%
10.84293

# Now run simulation for bootstrap confidence interval using percentile method
# Each bootstrap confidence interval is based on 100 samples.  The simulation
# obtains 1000 confidence intervals, so 100,000 bootstrap distributions are
generated.
# This is a Monte Carlo simulation of the Monte Carlo bootstrap
> for(i in 1:1000)
+ {
+ x<-rnorm(100,10,3)
+ bresult = boot(x,samplemedian,R=100)
+ lbound[i] <- quantile(bresult$t[,1], 0.025)
+ ubound[i] <- quantile(bresult$t[,1], 0.975)
+ }
> mean((lbound<=truevalue)&(truevalue<=ubound))
[1] 0.931  #The coverage is too low

> set.seed(456422)
```

```
> for(i in 1:1000)
+ {
+ x<-rnorm(100,10,3)
+ bresult = boot(x,samplemedian,R=100)
+ lbound[i] <- quantile(bresult$t[,1], 0.025)
+ ubound[i] <- quantile(bresult$t[,1], 0.975)
+ }
> mean((lbound<=truevalue)&(truevalue<=ubound))
[1] 0.924  #The coverage is too low again

#define constants for a simulation
> nreps<-1000
> numboots <-100
> mu<-10
> sigma <-3
> alpha=0.05
> n<-100

#Run simulation of bootstrap using script file
> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.934 #Coverage is still too low

> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.921

> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.936

> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.923

> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.93

> source("z:/work/stp532/R Runs/simulationscript.R")
[1] 0.93

# Set new seed for random number generation
> set.seed(7374994)

# Set working directory
> setwd("z:/work/stp532/R Runs")

# Run simulation from script in working directory
> source("simulationscript.R")
[1] 0.933



> x <- rnorm(50,10,3)
```

```
> bresult <- boot(x,samplemedian,R=100)

# Obtain bootstrap confidence interval from boot.ci using BCa method
> boot.ci(bresult, type=c("perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 100 bootstrap replicates

CALL :
boot.ci(boot.out = bresult, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   ( 8.563, 10.217 )   ( 6.682, 10.093 )
Calculations and Intervals on Original Scale
Some percentile intervals may be unstable
Some BCa intervals may be unstable

# Save boot.ci results to bciresults
> bciresult <- boot.ci(bresult, type=c("perc","bca"))

# Print bciresult
> bciresult
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 100 bootstrap replicates

CALL :
boot.ci(boot.out = bresult, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   ( 8.563, 10.217 )   ( 6.682, 10.093 )
Calculations and Intervals on Original Scale
Some percentile intervals may be unstable
Some BCa intervals may be unstable

# BCa confidence intervals are in bciresults$bca
> bciresult$bca
     conf
[1,] 0.95 1.02 95.46 6.682167 10.09338

# Lower bound of BCa confidence interval is in position 4 of vector bca
# upper bound of BCa confidence interval is in position 5 of vector bca
> lb <- bciresult$bca[4]
> ub <- bciresult$bca[5]

# lb and ub are scalars here because the confidence interval was calculated
# for only one sample
> lb
6.682167
> ub
10.09338
```

```
for(i in 1:nreps)
{
x<-rnorm(n,mu,sigma)
bresult = boot(x,samplemedian,R=numboots)
lbound[i] <- quantile(bresult$t[,1], alpha/2)
ubound[i] <- quantile(bresult$t[,1], 1-alpha/2)
}
coverage <- mean((lbound<=truevalue)&(truevalue<=ubound))
print(coverage)
```

Kruskal - Wallis

```
R version 2.13.1 (2011-07-08)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]
```

YOY = Young of Year
Gizzard Shad
Kokosing Lake Ohio

```
> library(coin)
Loading required package: survival
Loading required package: splines

Attaching package: 'survival'

The following object(s) are masked _by_ '.GlobalEnv':

    cancer

Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
```

```
> library(multcomp)
>
```

```
> ### Length of YOY Gizzard Shad from Kokosing Lake, Ohio,
> ### sampled in Summer 1984, Hollander & Wolfe (1999), Table 6.3, page 200
> YOY <- data.frame(length = c(46, 28, 46, 37, 32, 41, 42, 45, 38, 44,
+ 42, 60, 32, 42, 45, 58, 27, 51, 42, 52,
+ 38, 33, 26, 25, 28, 28, 26, 27, 27, 27,
+ 31, 30, 27, 29, 30, 25, 25, 24, 27, 30),
+ site = factor(c(rep("I", 10), rep("II", 10),
+ rep("III", 10), rep("IV", 10))))
```

Data in data.frame

```
> kruskal.test(length~site,data=YOY)

        Kruskal-Wallis rank sum test

data:  length by site
Kruskal-Wallis chi-squared = 22.8524, df = 3, p-value = 4.335e-05
```

```
> YOYresult <- oneway_test(length ~ site, data = YOY,
+ ytrafo = function(data) trafo(data, numeric_trafo = rank),
+ xtrafo = function(data) trafo(data, factor_trafo = function(x)
+ model.matrix(~x - 1) %*% t(contrMat(table(x), "Tukey"))),
+ teststat = "quad")
> YOYresult

        Asymptotic K-Sample Permutation Test

data:  length by site (I, II, III, IV)
chi-squared = 22.8524, df = 3, p-value = 4.335e-05
```

Kruskall-Wallis test

```
> print(pvalue(YOYresult, method = "single-step"))
[1] 4.334659e-05
```

```
> YOYresult <- oneway_test(length ~ site, data = YOY,
+ ytrafo = function(data) trafo(data, numeric_trafo = rank),
+ xtrafo = function(data) trafo(data, factor_trafo = function(x)
+ model.matrix(~x - 1) %*% t(contrMat(table(x), "Tukey"))),
+ teststat = "max")
> YOYresult

        Asymptotic K-Sample Permutation Test

data:  length by site (I, II, III, IV)
maxT = 3.664, p-value = 0.001502
```

not Kruskal-Wallis test

```
> print(pvalue(YOYresult, method = "single-step"))

II - I   0.944857606
III - I  0.011956774
IV - I   0.010423581
III - II 0.001681912
IV - II  0.001433905
IV - III 0.999931627
```

Pair-wise contrasts from joint ranks

```
> trafo(YOY, numeric_trafo = rank)
   length site.I site.II site.III site.IV
1    35.5      1       0        0       0
2    14.0      1       0        0       0
3    35.5      1       0        0       0
4    24.0      1       0        0       0
5    21.5      1       0        0       0
6    27.0      1       0        0       0
7    29.5      1       0        0       0
8    33.5      1       0        0       0
9    25.5      1       0        0       0
10   32.0      1       0        0       0
11   29.5      0       1        0       0
12   40.0      0       1        0       0
13   21.5      0       1        0       0
14   29.5      0       1        0       0
15   33.5      0       1        0       0
16   39.0      0       1        0       0
17    9.5      0       1        0       0
18   37.0      0       1        0       0
19   29.5      0       1        0       0
20   38.0      0       1        0       0
21   25.5      0       0        1       0
22   23.0      0       0        1       0
23    5.5      0       0        1       0
24    3.0      0       0        1       0
25   14.0      0       0        1       0
26   14.0      0       0        1       0
27    5.5      0       0        1       0
28    9.5      0       0        1       0
29    9.5      0       0        1       0
30    9.5      0       0        1       0
31   20.0      0       0        0       1
32   18.0      0       0        0       1
33    9.5      0       0        0       1
34   16.0      0       0        0       1
35   18.0      0       0        0       1
36    3.0      0       0        0       1
37    3.0      0       0        0       1
38    1.0      0       0        0       1
39    9.5      0       0        0       1
40   18.0      0       0        0       1
attr(,"assign")
[1] 1 2 2 2 2
>
```

Transform length measurements to ranks

```
>  trafo(YOY, factor_trafo = function(x)
+ model.matrix(~x - 1) %*% t(contrMat(table(x), "Tukey")))
    length site.II - I site.III - I site.IV - I site.III - II site.IV - II site.IV - III
1       46          -1           -1          -1             0            0             0
2       28          -1           -1          -1             0            0             0
3       46          -1           -1          -1             0            0             0
4       37          -1           -1          -1             0            0             0
5       32          -1           -1          -1             0            0             0
6       41          -1           -1          -1             0            0             0
7       42          -1           -1          -1             0            0             0
8       45          -1           -1          -1             0            0             0
9       38          -1           -1          -1             0            0             0
10      44          -1           -1          -1             0            0             0
11      42           1            0           0            -1           -1             0
12      60           1            0           0            -1           -1             0
13      32           1            0           0            -1           -1             0
14      42           1            0           0            -1           -1             0
15      45           1            0           0            -1           -1             0
16      58           1            0           0            -1           -1             0
17      27           1            0           0            -1           -1             0
18      51           1            0           0            -1           -1             0
19      42           1            0           0            -1           -1             0
20      52           1            0           0            -1           -1             0
21      38           0            1           0             1            0            -1
22      33           0            1           0             1            0            -1
23      26           0            1           0             1            0            -1
24      25           0            1           0             1            0            -1
25      28           0            1           0             1            0            -1
26      28           0            1           0             1            0            -1
27      26           0            1           0             1            0            -1
28      27           0            1           0             1            0            -1
29      27           0            1           0             1            0            -1
30      27           0            1           0             1            0            -1
31      31           0            0           1             0            1             1
32      30           0            0           1             0            1             1
33      27           0            0           1             0            1             1
34      29           0            0           1             0            1             1
35      30           0            0           1             0            1             1
36      25           0            0           1             0            1             1
37      25           0            0           1             0            1             1
38      24           0            0           1             0            1             1
39      27           0            0           1             0            1             1
40      30           0            0           1             0            1             1
attr(,"assign")
[1] 1 2 2 2 2 2 2
```

K - Sample

```
R version 2.13.1 (2011-07-08)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> library(coin)
Loading required package: survival
Loading required package: splines

Attaching package: 'survival'

The following object(s) are masked _by_ '.GlobalEnv':

    cancer

Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4



> library(multcomp)
>


> ### Length of YOY Gizzard Shad from Kokosing Lake, Ohio,
> ### sampled in Summer 1984, Hollander & Wolfe (1999), Table 6.3, page 200
> YOY <- data.frame(length = c(46, 28, 46, 37, 32, 41, 42, 45, 38, 44,
+ 42, 60, 32, 42, 45, 58, 27, 51, 42, 52,
+ 38, 33, 26, 25, 28, 28, 26, 27, 27, 27,
+ 31, 30, 27, 29, 30, 25, 25, 24, 27, 30),
+ site = factor(c(rep("I", 10), rep("II", 10),
+ rep("III", 10), rep("IV", 10))))
```

Data in data.frame

```
> trafo(YOY, numeric_trafo = rank)
   length site.I site.II site.III site.IV
1    35.5      1       0        0       0
2    14.0      1       0        0       0
3    35.5      1       0        0       0
4    24.0      1       0        0       0
5    21.5      1       0        0       0
6    27.0      1       0        0       0
7    29.5      1       0        0       0
8    33.5      1       0        0       0
9    25.5      1       0        0       0
10   32.0      1       0        0       0
11   29.5      0       1        0       0
12   40.0      0       1        0       0
13   21.5      0       1        0       0
14   29.5      0       1        0       0
15   33.5      0       1        0       0
16   39.0      0       1        0       0
17    9.5      0       1        0       0
18   37.0      0       1        0       0
19   29.5      0       1        0       0
20   38.0      0       1        0       0
21   25.5      0       0        1       0
22   23.0      0       0        1       0
23    5.5      0       0        1       0
24    3.0      0       0        1       0
25   14.0      0       0        1       0
26   14.0      0       0        1       0
27    5.5      0       0        1       0
28    9.5      0       0        1       0
29    9.5      0       0        1       0
30    9.5      0       0        1       0
31   20.0      0       0        0       1
32   18.0      0       0        0       1
33    9.5      0       0        0       1
34   16.0      0       0        0       1
35   18.0      0       0        0       1
36    3.0      0       0        0       1
37    3.0      0       0        0       1
38    1.0      0       0        0       1
39    9.5      0       0        0       1
40   18.0      0       0        0       1
attr(,"assign")
[1] 1 2 2 2 2
>
```

Transform length measurements to ranks

K-Sample Rank Test
Asymptotic

```
> xandg <- data.frame(x,g)
> xandg
    x                                    g
1  2.9                       Normal subjects
2  3.0                       Normal subjects
3  2.5                       Normal subjects
4  2.6                       Normal subjects
5  3.2                       Normal subjects
6  3.8 Subjects with obstructive airway disease
7  2.7 Subjects with obstructive airway disease
8  4.0 Subjects with obstructive airway disease
9  2.4 Subjects with obstructive airway disease
10 2.8                 Subjects with asbestosis
11 3.4                 Subjects with asbestosis
12 3.7                 Subjects with asbestosis
13 2.2                 Subjects with asbestosis
14 2.0                 Subjects with asbestosis
> NDWD <-oneway_test(x ~ g, data = xandg,
+ ytrafo = function(data) trafo(data, numeric_trafo = rank),
+ xtrafo = function(data) trafo(data, factor_trafo = function(x)
+ model.matrix(~x - 1) %*% t(contrMat(table(x), "Tukey"))),
+ teststat = "max"
+ )
> NDWD

        Asymptotic K-Sample Permutation Test

data:  x by
         g (Normal subjects, Subjects with obstructive airway disease, Subjects with asbestosis)
maxT = 0.84, p-value = 0.6779

> print(pvalue(NDWD, method = "single-step"))

Subjects with obstructive airway disease - Normal subjects          0.8199772
Subjects with asbestosis - Normal subjects                          0.9720254
Subjects with asbestosis - Subjects with obstructive airway disease 0.6779163
```

~~Steel Dwass~~
Paired Comparisons
Monte Carlo
p-value

```
> library(coin)
> library(multcomp)
> ### Length of YOY Gizzard Shad from Kokosing Lake, Ohio,
> ### sampled in Summer 1984, Hollander & Wolfe (1999), Table 6.3, page 200
> YOY <- data.frame(length = c(46, 28, 46, 37, 32, 41, 42, 45, 38, 44,
+                              42, 60, 32, 42, 45, 58, 27, 51, 42, 52,
+                              38, 33, 26, 25, 28, 28, 26, 27, 27, 27,
+                              31, 30, 27, 29, 30, 25, 25, 24, 27, 30),
+                   site = factor(c(rep("I", 10), rep("II", 10),
+                                   rep("III", 10), rep("IV", 10))))
>
> ### Nemenyi-Damico-Wolfe-Dunn test (joint ranking)
> ### Hollander & Wolfe (1999), page 244
> ### (where Steel-Dwass results are given)
> NDWD <- oneway_test(length ~ site, data = YOY,
+                     ytrafo = function(data) trafo(data, numeric_trafo = rank),
+                     xtrafo = function(data) trafo(data, factor_trafo = function(x)
+                         model.matrix(~x - 1) %*% t(contrMat(table(x), "Tukey"))),
+                     teststat = "max", distribution = approximate(B = 90000),
+                     )
> NDWD

        Approximative K-Sample Permutation Test

data:  length by site (I, II, III, IV)
maxT = 3.664, p-value = 0.0005889

> ### global p-value
> print(pvalue(NDWD))
[1] 0.0005888889
99 percent confidence interval:
 0.0004014296 0.0008310070

>
> ### sites (I = II) != (III = IV) at alpha = 0.05 (default was 0.01) (page 244)
> print(pvalue(NDWD, method = "single-step"))

II - I   0.9479666667
III - I  0.0088111111
IV - I   0.0069000000
III - II 0.0007111111
IV - II  0.0005777778
IV - III 0.9999444444
>
```

```
> load("/media/2C20-718A/stp532/R Runs/YOYworkspace.RData")
> YOY
   length site
1      46    I
2      28    I
3      46    I
4      37    I
5      32    I
6      41    I
7      42    I
8      45    I
9      38    I
10     44    I
11     42   II
12     60   II
13     32   II
14     42   II
15     45   II
16     58   II
17     27   II
18     51   II
19     42   II
20     52   II
21     38  III
22     33  III
23     26  III
24     25  III
25     28  III
26     28  III
27     26  III
28     27  III
29     27  III
30     27  III
31     31   IV
32     30   IV
33     27   IV
34     29   IV
35     30   IV
36     25   IV
37     25   IV
38     24   IV
39     27   IV
40     30   IV
> kruskal.test(length~site,data=YOY)

        Kruskal-Wallis rank sum test

data:  length by site
Kruskal-Wallis chi-squared = 22.8524, df = 3, p-value = 4.335e-05
```

15.5 t_{10}
2
15.5 t_{10}
5
3.5 t_3
7
9.5 t_7
13.5 t_9
6
12
9.5 t_7
20
3.5 t_3
9.5 t_7
13.5 t_9
19
1
17
9.5 t_7
18

$t_3 = 2$

$t_7 = 4$

$t_9 = 2$

$t_{10} = 2$

Bootstrap Chi-square Test

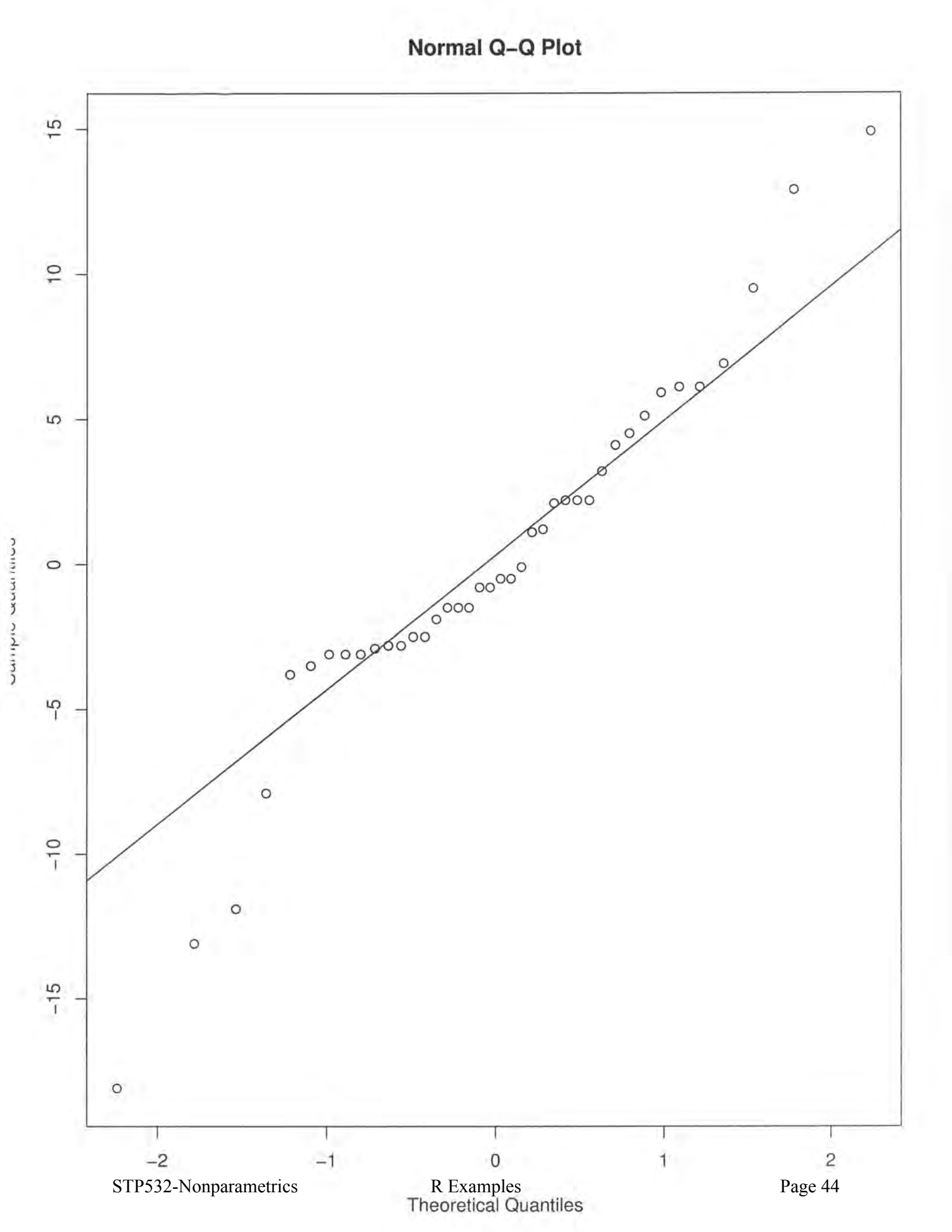
```
> observed <- c(157,69,35,17,1,1)
> H0probs <- c(.4704,.3829,.1246,.0203,.0017,.0001)
> chisq.test(observed,p=H0probs)

        Chi-squared test for given probabilities

data:  observed
X-squared = 75.3225, df = 5, p-value = 7.967e-15

Warning message:
In chisq.test(observed, p = H0probs) :
  Chi-squared approximation may be incorrect
> 1-pchisq(75.3225,4)
[1] 1.665335e-15

> teststat <- rep(NA,100000)
> for (i in 1:100000)
+ {counts <- rmultinom(1,280,H0probs)
+ suppressWarnings(result <- chisq.test(counts,p=H0probs))
+ teststat[i]=result$statistic}

> quantile(teststat,0.95)
     95%
12.76381

> qchisq(.95,4)
[1] 9.487729

> mean(teststat > 75.32)
[1] 0.00034

> 1-pchisq(75.3225,4)
[1] 1.665335e-15

> chisq.test(observed,p=H0probs,simulate.p.value=TRUE,B=100000)

        Chi-squared test for given probabilities with simulated p-value (based on 1e+05
replicates)

data:  observed
X-squared = 75.3225, df = NA, p-value = 0.00032
```

KS test for Residuals

```
> YOY
   length site
1      46    I
2      28    I
3      46    I
4      37    I
5      32    I
6      41    I
7      42    I
8      45    I
9      38    I
10     44    I
11     42   II
12     60   II
13     32   II
14     42   II
15     45   II
16     58   II
17     27   II
18     51   II
19     42   II
20     52   II
21     38  III
22     33  III
23     26  III
24     25  III
25     28  III
26     28  III
27     26  III
28     27  III
29     27  III
30     27  III
31     31   IV
32     30   IV
33     27   IV
34     29   IV
35     30   IV
36     25   IV
37     25   IV
38     24   IV
39     27   IV
40     30   IV
```

```
> result <- lm(length~site,data=YOY)
>
> summary(result)

Call:
lm(formula = length ~ site, data = YOY)

Residuals:
    Min      1Q  Median      3Q     Max
-18.100  -2.825  -0.650   3.425  14.900

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   39.900      2.067  19.305  < 2e-16 ***
siteII         5.200      2.923   1.779 0.083685 .
siteIII      -11.400      2.923  -3.900 0.000403 ***
siteIV       -12.100      2.923  -4.140 0.000200 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.536 on 36 degrees of freedom
Multiple R-squared: 0.5882,     Adjusted R-squared: 0.5539
F-statistic: 17.14 on 3 and 36 DF,  p-value: 4.421e-07

> result$residual
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17
  6.1 -11.9   6.1  -2.9  -7.9   1.1   2.1   5.1  -1.9   4.1  -3.1  14.9 -13.1  -3.1  -0.1  12.9 -18.1
    21    22    23    24    25    26    27    28    29    30    31    32    33    34    35    36    37
  9.5   4.5  -2.5  -3.5  -0.5  -0.5  -2.5  -1.5  -1.5  -1.5   3.2   2.2  -0.8   1.2   2.2  -2.8  -2.8

> qqnorm(result$residual)
> qqline(result$residual)

> ksfcn <- function(x) {
+ mu.hat <- mean(x)
+ sigma.hat <- sd(x)
+ d.hat <- ks.test(x, "pnorm", mean = mu.hat,
+ sd = sigma.hat)$statistic
+ return(as.numeric(d.hat))
+ }



> ksfcn(result$residual)
[1] 0.1725448
Warning message:
In ks.test(x, "pnorm", mean = mu.hat, sd = sigma.hat) :
  cannot compute correct p-values with ties
>
> d.hat <- ksfcn(result$residual)
```

```
> n <- length(x)
> nsim <- 4999
> d.star <- double(nsim)
> for (i in 1:nsim) {
+ x.star <- rnorm(n)
+ d.star[i] <- ksfcn(x.star)
+ }

> hist(d.star)
> abline(v = d.hat, lty = 2)

> ## simulation-derived P-value
> pval <- (sum(d.star > d.hat) + 1) / (nsim + 1)
> print(pval)
[1] 0.0034

> print(nsim / (nsim + 1) * sqrt(pval * (1 - pval) / nsim))
[1] 0.0008231357

> cat("Calculation took", proc.time()[1], "seconds\n")
Calculation took 46.954 seconds

> library(nortest)

> lillie.test(result$residual)

        Lilliefors (Kolmogorov-Smirnov) normality test

data:  result$residual
D = 0.1725, p-value = 0.004184

> lillie.test(x)

        Lilliefors (Kolmogorov-Smirnov) normality test

data:  x
D = 0.1725, p-value = 0.004184

> shapiro.test(result$residual)

        Shapiro-Wilk normality test

data:  result$residual
W = 0.9466, p-value = 0.05812
```


Normal Q–Q Plot
Sample Quantiles
Theoretical Quantiles
15
10
5
0
−5
−10
−15
−2
−1
0
1
2

Parametric Bootstrap; Resampling Residuals

```
> YOY
   length site
1      46    I
2      28    I
3      46    I
4      37    I
5      32    I
6      41    I
7      42    I
8      45    I
9      38    I
10     44    I
11     42   II
12     60   II
13     32   II
14     42   II
15     45   II
16     58   II
17     27   II
18     51   II
19     42   II
20     52   II
21     38  III
22     33  III
23     26  III
24     25  III
25     28  III
26     28  III
27     26  III
28     27  III
29     27  III
30     27  III
31     31   IV
32     30   IV
33     27   IV
34     29   IV
35     30   IV
36     25   IV
37     25   IV
38     24   IV
39     27   IV
40     30   IV
> lm(length~site,data=YOY)

Call:
lm(formula = length ~ site, data = YOY)

Coefficients:
(Intercept)       siteII      siteIII       siteIV
       39.9          5.2        -11.4        -12.1

> model.matrix(length~site,data=YOY)
   (Intercept) siteII siteIII siteIV
1            1      0       0      0
2            1      0       0      0
3            1      0       0      0
```

YOY data

Fitting linear model with site as factor variable

```
4           1     0     0     0
5           1     0     0     0
6           1     0     0     0
7           1     0     0     0
8           1     0     0     0
9           1     0     0     0
10          1     0     0     0
11          1     1     0     0
12          1     1     0     0
13          1     1     0     0
14          1     1     0     0
15          1     1     0     0
16          1     1     0     0
17          1     1     0     0
18          1     1     0     0
19          1     1     0     0
20          1     1     0     0
21          1     0     1     0
22          1     0     1     0
23          1     0     1     0
24          1     0     1     0
25          1     0     1     0
26          1     0     1     0
27          1     0     1     0
28          1     0     1     0
29          1     0     1     0
30          1     0     1     0
31          1     0     0     1
32          1     0     0     1
33          1     0     0     1
34          1     0     0     1
35          1     0     0     1
36          1     0     0     1
37          1     0     0     1
38          1     0     0     1
39          1     0     0     1
40          1     0     0     1
attr(,"assign")
[1] 0 1 1 1
attr(,"contrasts")
attr(,"contrasts")$site
[1] "contr.treatment"
```

Model Matrix

Parameters are

$$\beta = \begin{bmatrix} \mu_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix}$$

α_2, α_3, and α_4 are contrasts to group #1

```
> X <- model.matrix(length~site,data=YOY)
> result <- lm(length ~ site, data=YOY)
> summary(result)

Call:
lm(formula = length ~ site, data = YOY)

Residuals:
     Min       1Q   Median       3Q      Max
-18.100  -2.825  -0.650   3.425  14.900

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  39.900      2.067  19.305  < 2e-16 ***
siteII        5.200      2.923   1.779 0.083685 .
siteIII     -11.400      2.923  -3.900 0.000403 ***
siteIV      -12.100      2.923  -4.140 0.000200 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.536 on 36 degrees of freedom
Multiple R-squared: 0.5882,     Adjusted R-squared: 0.5539
F-statistic: 17.14 on 3 and 36 DF,  p-value: 4.421e-07

> betahat <- summary(result)$coefficients
> residse <- summary(result)$sigma
> residse
[1] 6.536011
> betaH0 <- c(betahat[1,1],0,0,0)
> betaH0
[1] 39.9  0.0  0.0  0.0
> fitted <- X%*%betaH0
> fitted
    [,1]
1  39.9
2  39.9
3  39.9
4  39.9
5  39.9
6  39.9
7  39.9
8  39.9
9  39.9
10 39.9
11 39.9
12 39.9
13 39.9
14 39.9
15 39.9
16 39.9
17 39.9
18 39.9
19 39.9
20 39.9
21 39.9
22 39.9
23 39.9
24 39.9
25 39.9
26 39.9
27 39.9
28 39.9
29 39.9
30 39.9
31 39.9
32 39.9
33 39.9
34 39.9
35 39.9
```

```
36 39.9
37 39.9
38 39.9
39 39.9
40 39.9
> eij=residse*rnorm(40)
> ystar=fitted + eij
> ystar
        [,1]
1  43.87726
2  45.68410
3  32.46982
4  34.85002
5  27.45609
6  48.20363
7  35.70089
8  36.87915
9  33.35181
10 35.71766
11 41.04655
12 46.62377
13 56.22244
14 32.85068
15 39.66760
16 52.03192
17 45.73826
18 33.85259
19 42.17782
20 38.26784
21 39.04723
22 47.09291
23 32.84212
24 22.57487
25 44.59671
26 37.47449
27 46.31368
28 34.89105
29 24.55999
30 32.28296
31 42.83744
32 46.71215
33 27.00037
34 35.57958
35 41.50844
36 33.46632
37 40.23862
38 42.26843
39 34.57130
40 33.72118
> sitevar <- YOY$site
> sitevar
 [1] I   I   I   I   I   I   I   I   I   I   II  II  II  II  II  II  II  II  II  II  III III III
III III III III III III III IV  IV  IV  IV  IV
[36] IV  IV  IV  IV  IV
Levels: I II III IV
> bootstrapsample <- data.frame(ystar,sitevar)
```

$e_{ij} \sim N(0, \sigma^2)$ $\sigma^2 = \hat{\sigma}_\varepsilon^2$

A single bootstrap sample under H_0

```
> newresult <- lm(ystar~sitevar,data=bootstrapsample)
> summary(newresult)

Call:
lm(formula = ystar ~ sitevar, data = bootstrapsample)

Residuals:
     Min       1Q   Median       3Q      Max
-13.593  -4.068  -1.489   4.620  13.374

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.4190     2.2628  16.536   <2e-16 ***
sitevarII     5.4289     3.2001   1.696   0.0984 .
sitevarIII   -1.2514     3.2001  -0.391   0.6981
sitevarIV     0.3713     3.2001   0.116   0.9083
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.156 on 36 degrees of freedom
Multiple R-squared: 0.1236,     Adjusted R-squared: 0.0506
F-statistic: 1.693 on 3 and 36 DF,  p-value: 0.1858

> fstat <-summary(newresult <- lm(ystar~sitevar,data=bootstrapsample))$fstatistic[1]
> fstat
   value
1.692839
> teststat <- rep(NA,10000)
> for(i in 1:10000)
+ { eij=residse*rnorm(40)
+ ystar=fitted + eij
+ bootstrapsample <- data.frame(ystar,sitevar)
+ teststat[i] <-summary(newresult <- lm(ystar~sitevar,data=bootstrapsample))$fstatistic[1]
+ }
> summary(teststat)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.000358 0.407900 0.803600 1.052000 1.419000 9.619000
> quantile(teststat,0.95)
     95%
2.873367
> qf(.95,3,36)
[1] 2.866266
> e <-resid(result)
> teststat <- rep(NA,10000)
> for(i in 1:10000)
+ { eij=sample(e,replace=T)
+ ystar=fitted + eij
+ bootstrapsample <- data.frame(ystar,sitevar)
+ teststat[i] <-summary(newresult <- lm(ystar~sitevar,data=bootstrapsample))$fstatistic[1]
+ }
> summary(teststat)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.000294 0.401000 0.806300 1.047000 1.434000 9.196000
> quantile(teststat,0.95)
     95%
```

fit linear model to bootstrap sample

Results from single bootstrap sample fitting $H_0: \gamma_c = 0$

Parametric Bootstrap

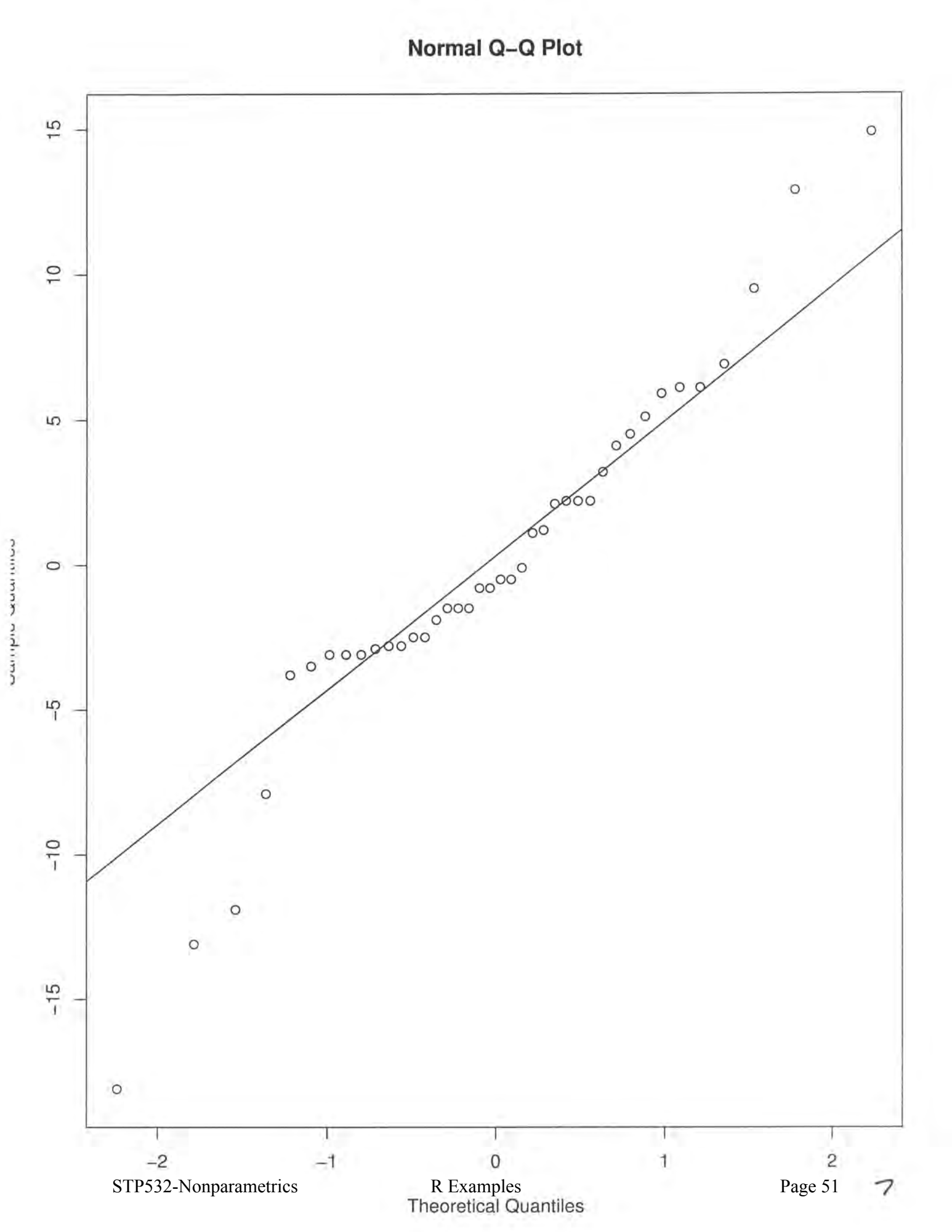
Bootstrap quantile

F distribution quantile

Resampling Residuals; Assumes residuals are exchangeable

```
2.78226
> teststat <- rep(NA,10000)
> for(i in 1:10000)
+ { eij=sample(e,replace=T)
+ ystar=fitted + eij
+ bootstrapsample <- data.frame(ystar,sitevar)
+ teststat[i] <-summary(newresult <- lm(ystar~sitevar,data=bootstrapsample))$fstatistic[1]
+ }
> summary(teststat)
      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
 0.001584  0.399300  0.798000  1.042000  1.410000 11.040000
> quantile(teststat,0.95)
     95%
2.802953

> qqnorm(e)
> qqline(e)
```

Normal QQ plot


Normal Q–Q Plot
15
10
5
0
−5
−10
−15
−2
−1
0
1
2
Theoretical Quantiles

```
> fitted <- X%*%betahat
> fitted
   Estimate Std. Error  t value     Pr(>|t|)
1      39.9   2.066868 19.30457 1.407684e-20
2      39.9   2.066868 19.30457 1.407684e-20
3      39.9   2.066868 19.30457 1.407684e-20
4      39.9   2.066868 19.30457 1.407684e-20
5      39.9   2.066868 19.30457 1.407684e-20
6      39.9   2.066868 19.30457 1.407684e-20
7      39.9   2.066868 19.30457 1.407684e-20
8      39.9   2.066868 19.30457 1.407684e-20
9      39.9   2.066868 19.30457 1.407684e-20
10     39.9   2.066868 19.30457 1.407684e-20
11     45.1   4.989861 21.08357 8.368532e-02
12     45.1   4.989861 21.08357 8.368532e-02
13     45.1   4.989861 21.08357 8.368532e-02
14     45.1   4.989861 21.08357 8.368532e-02
15     45.1   4.989861 21.08357 8.368532e-02
16     45.1   4.989861 21.08357 8.368532e-02
17     45.1   4.989861 21.08357 8.368532e-02
18     45.1   4.989861 21.08357 8.368532e-02
19     45.1   4.989861 21.08357 8.368532e-02
20     45.1   4.989861 21.08357 8.368532e-02
21     28.5   4.989861 15.40446 4.030988e-04
22     28.5   4.989861 15.40446 4.030988e-04
23     28.5   4.989861 15.40446 4.030988e-04
24     28.5   4.989861 15.40446 4.030988e-04
25     28.5   4.989861 15.40446 4.030988e-04
26     28.5   4.989861 15.40446 4.030988e-04
27     28.5   4.989861 15.40446 4.030988e-04
28     28.5   4.989861 15.40446 4.030988e-04
29     28.5   4.989861 15.40446 4.030988e-04
30     28.5   4.989861 15.40446 4.030988e-04
31     27.8   4.989861 15.16498 2.001963e-04
32     27.8   4.989861 15.16498 2.001963e-04
33     27.8   4.989861 15.16498 2.001963e-04
34     27.8   4.989861 15.16498 2.001963e-04
35     27.8   4.989861 15.16498 2.001963e-04
36     27.8   4.989861 15.16498 2.001963e-04
37     27.8   4.989861 15.16498 2.001963e-04
38     27.8   4.989861 15.16498 2.001963e-04
39     27.8   4.989861 15.16498 2.001963e-04
40     27.8   4.989861 15.16498 2.001963e-04
> for(i in 1:10000)
+ { eij=sample(e,replace=T)
+ ystar=fitted + eij
+ bootstrapsample <- data.frame(ystar,sitevar)
+ teststat[i] <- lm(ystar~sitevar,data=bootstrapsample)$coef[2]
+ }
> summary(teststat)
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -5.510   3.290   5.200   5.206   7.080  18.040
```

```
> quantile(teststat,c(0.025,0.975))
    2.5%    97.5%
-0.26000 10.68025
>
> summary(result)

Call:
lm(formula = length ~ site, data = YOY)

Residuals:
    Min      1Q  Median      3Q     Max
-18.100  -2.825  -0.650   3.425  14.900

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   39.900      2.067  19.305  < 2e-16 ***
siteII         5.200      2.923   1.779 0.083685 .
siteIII      -11.400      2.923  -3.900 0.000403 ***
siteIV       -12.100      2.923  -4.140 0.000200 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.536 on 36 degrees of freedom
Multiple R-squared: 0.5882,     Adjusted R-squared: 0.5539
F-statistic: 17.14 on 3 and 36 DF,  p-value: 4.421e-07

> confint(result,2)
              2.5 %  97.5 %
siteII -0.7281048 11.1281
>
```

Bootstrap Confidence Interval based on resampling residuals; Percentile Method.

Result for original data

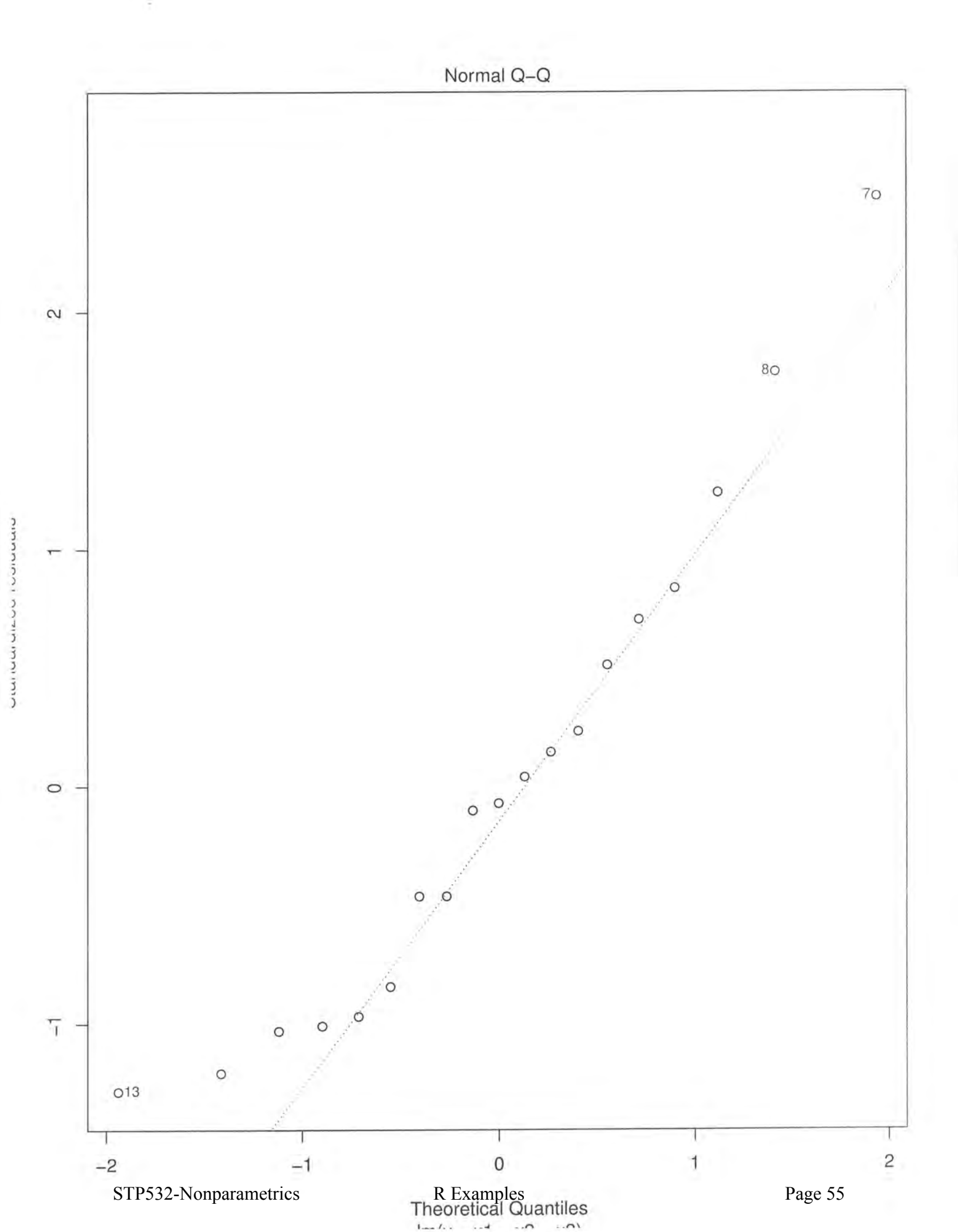
parametric confidence interval, t-distribution

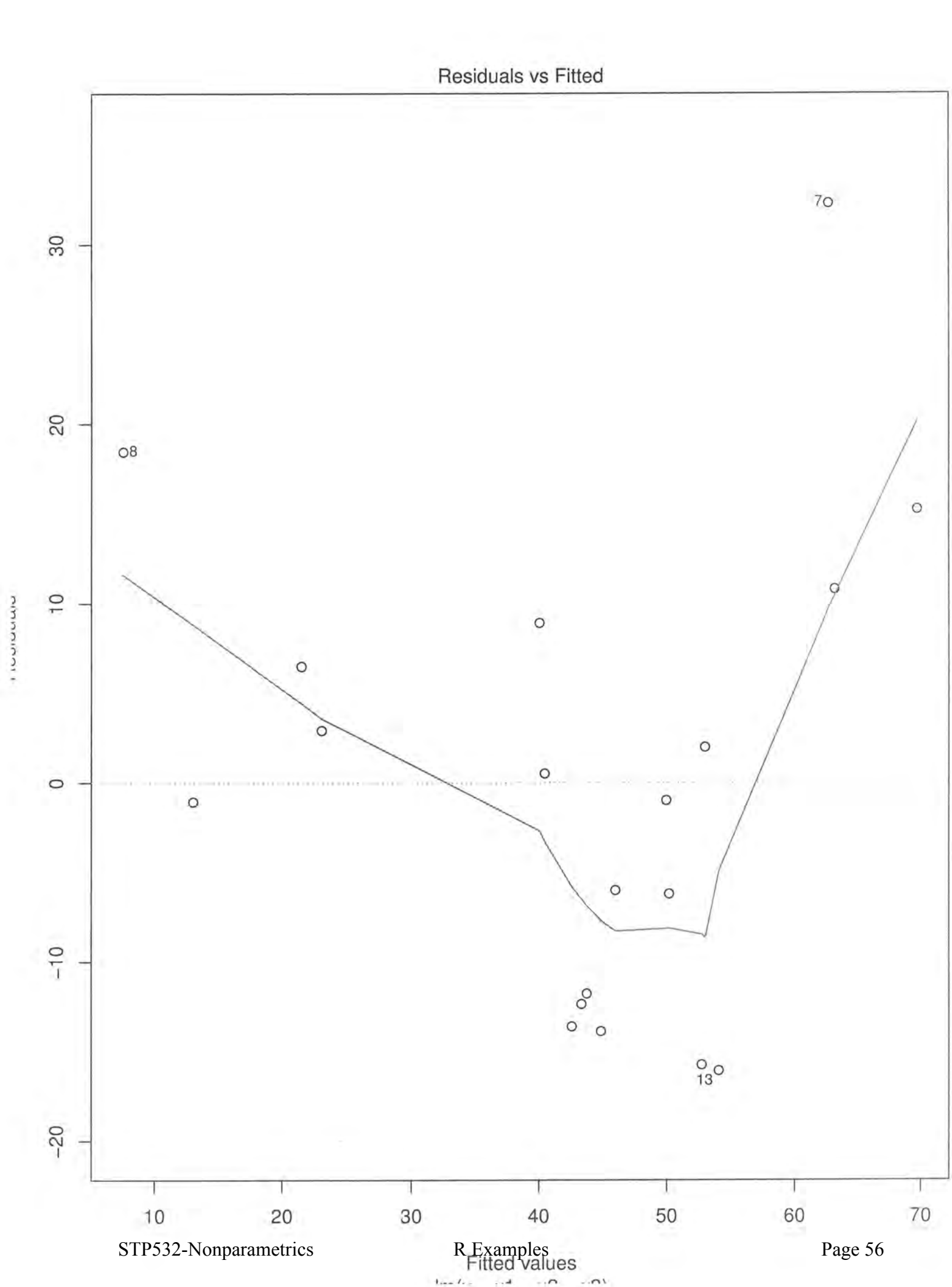
Bootstrap Regression

```
> CH09PR13 <- read.table("/mnt/snap/work/qba530/Data Sets/Chapter
                                                9 Data Sets/CH09PR13.txt", quote="\"")
>   View(CH09PR13)
> y=v4

> attach(CH09PR13)

> CH09PR13
   V1 V2 V3 V4
1  49 45 36 45
2  55 30 28 40
3  85 11 16 42
4  32 30 46 40
5  26 39 76 43
6  28 42 78 27
7  95 17 24 36
8  26 63 80 42
9  74 25 12 52
10 37 32 27 35
11 31 37 37 55
12 49 29 34 47
13 38 26 32 28
14 41 38 45 30
15 12 38 99 26
16 44 25 38 47
17 29 27 51 44
18 40 37 32 54
19 31 34 40 36
```

Y = V1 = arterial pressure in lung
X1 = V2 = rate of blood pumping
X2 = V3 = ejection rate of blood pumping
X3 = V4 = blood gas measure

```
> y=V1
> y
 [1] 49 55 85 32 26 28 95 26 74 37 31 49 38 41 12 44 29 40 31
> x1=V2
> x2=V3
> x3=V4
> mlrfit <- lm(y~x1+x2+x3)

> mlrfit

Call:
lm(formula = y ~ x1 + x2 + x3)

Coefficients:
(Intercept)           x1           x2           x3
   87.18750     -0.56448     -0.51315     -0.07196


> plot(mlrfit)

Hit <Return> to see next plot:
```


Normal Q–Q
7
8
13
2
1
0
−1
−2
−1
0
1
2
Theoretical Quantiles


Residuals vs Fitted
7
8
13
30
20
10
0
−10
−20
10
20
30
40
50
60
70
Fitted values

```
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
> X <- model.matrix(y~x1+x2+x3, data=CH09PR13)
> X
   (Intercept) x1 x2 x3
1            1 45 36 45
2            1 30 28 40
3            1 11 16 42
4            1 30 46 40
5            1 39 76 43
6            1 42 78 27
7            1 17 24 36
8            1 63 80 42
9            1 25 12 52
10           1 32 27 35
11           1 37 37 55
12           1 29 34 47
13           1 26 32 28
14           1 38 45 30
15           1 38 99 26
16           1 25 38 47
17           1 27 51 44
18           1 37 32 54
19           1 34 40 36
attr(,"assign")
[1] 0 1 2 3
> betahat <- summary(mlrfit)$coefficients
> fitted = X%*%betahat
> e <- resid(mlrfit)
>
> betastar <- matrix(NA,1000,4)

> betastar[1,]
[1] NA NA NA NA


>
> for(i in 1:1000) {
+  eij=sample(e,replace=T)
+ ystar=fitted + eij
+ bootstrapsample <- data.frame(ystar,x1,x2,x3)
+ newresult <- lm(ystar~x1 + x2 +x3,data=bootstrapsample)
+ betastar[i,1] <-newresult$coef[1]
+ betastar[i,2] <-newresult$coef[2]
+ betastar[i,3] <-newresult$coef[3]
+ betastar[i,4] <-newresult$coef[4]
+ }
```

Residual Resampling

```
> summary(mlrfit)

Call:
lm(formula = y ~ x1 + x2 + x3)

Residuals:
    Min      1Q  Median      3Q     Max
-16.075 -12.064  -0.988   7.707  32.315

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 87.18750   21.55246   4.045  0.00106 **
x1          -0.56448    0.42791  -1.319  0.20691
x2          -0.51315    0.22449  -2.286  0.03723 *
x3          -0.07196    0.45457  -0.158  0.87633
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.42 on 15 degrees of freedom
Multiple R-squared: 0.6141,     Adjusted R-squared: 0.5369
F-statistic: 7.957 on 3 and 15 DF,  p-value: 0.002083

> confint(mlrfit)
                  2.5 %      97.5 %
(Intercept) 41.2495093 133.12549088
x1          -1.4765554   0.34759875
x2          -0.9916493  -0.03465126
x3          -1.0408608   0.89694494
> quantile(betastar[,2], c(0.025, 0.975))
      2.5%      97.5%
-1.2343345  0.1989153
> quantile(betastar[,3], c(0.025, 0.975))
      2.5%      97.5%
-0.9158399 -0.1172219
> quantile(betastar[,4], c(0.025, 0.975))
      2.5%      97.5%
-0.8608367  0.7128770
> mean(betastar[,2])
[1] -0.5704193
> mean(betastar[,3])
[1] -0.5045995
> mean(betastar[,4])
[1] -0.06363205
```

Need to be corrected for bias

```
> CH09PR13[sample(19,19,replace=T),]
      V1 V2 V3 V4
3     85 11 16 42
10    37 32 27 35
12    49 29 34 47
12.1  49 29 34 47
11    31 37 37 55
10.1  37 32 27 35
11.1  31 37 37 55
12.2  49 29 34 47
4     32 30 46 40
12.3  49 29 34 47
10.2  37 32 27 35
15    12 38 99 26
13    38 26 32 28
9     74 25 12 52
7     95 17 24 36
5     26 39 76 43
12.4  49 29 34 47
18    40 37 32 54
15.1  12 38 99 26
```

```
> datamat <- cbind(y,x1,x2,x3)
> datamat
        y x1 x2 x3
 [1,] 49 45 36 45
 [2,] 55 30 28 40
 [3,] 85 11 16 42
 [4,] 32 30 46 40
 [5,] 26 39 76 43
 [6,] 28 42 78 27
 [7,] 95 17 24 36
 [8,] 26 63 80 42
 [9,] 74 25 12 52
[10,] 37 32 27 35
[11,] 31 37 37 55
[12,] 49 29 34 47
[13,] 38 26 32 28
[14,] 41 38 45 30
[15,] 12 38 99 26
[16,] 44 25 38 47
[17,] 29 27 51 44
[18,] 40 37 32 54
[19,] 31 34 40 36
```

Case Resampling

```
> for(i in 1:1000) {
+ datastar <-datamat[sample(19,19,replace=T),]
+ newframe <- as.data.frame(datastar)
+ newresult <- lm(y~x1 + x2 +x3,data=newframe)
+ betastar[i,1] <-newresult$coef[1]
+ betastar[i,2] <-newresult$coef[2]
+ betastar[i,3] <-newresult$coef[3]
+ betastar[i,4] <-newresult$coef[4]
+ }
> mean(betastar[,2])
[1] -0.6201279
> mean(betastar[,3])
[1] -0.5170812
> mean(betastar[,4])
[1] -0.08808362
> quantile(betastar[,2], c(0.025, 0.975))
      2.5%      97.5%
-1.8844253  0.2597327
> quantile(betastar[,3], c(0.025, 0.975))
      2.5%      97.5%
-0.9809736 -0.1332903
> quantile(betastar[,4], c(0.025, 0.975))
      2.5%      97.5%
-1.1246083  0.6745642
> summary(mlrfit)
Call:
lm(formula = y ~ x1 + x2 + x3)

Residuals:
    Min      1Q  Median      3Q     Max
-16.075 -12.064  -0.988   7.707  32.315

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 87.18750   21.55246   4.045  0.00106 **
x1          -0.56448    0.42791  -1.319  0.20691
x2          -0.51315    0.22449  -2.286  0.03723 *
x3          -0.07196    0.45457  -0.158  0.87633
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14.42 on 15 degrees of freedom
Multiple R-squared: 0.6141,     Adjusted R-squared: 0.5369
F-statistic: 7.957 on 3 and 15 DF,  p-value: 0.002083

> confit(mlrfit)
Error: could not find function "confit"
> confint(mlrfit)
                  2.5 %       97.5 %
(Intercept) 41.2495093 133.12549088
x1          -1.4765554   0.34759875
x2          -0.9916493  -0.03465126
x3          -1.0408608   0.89694494
```

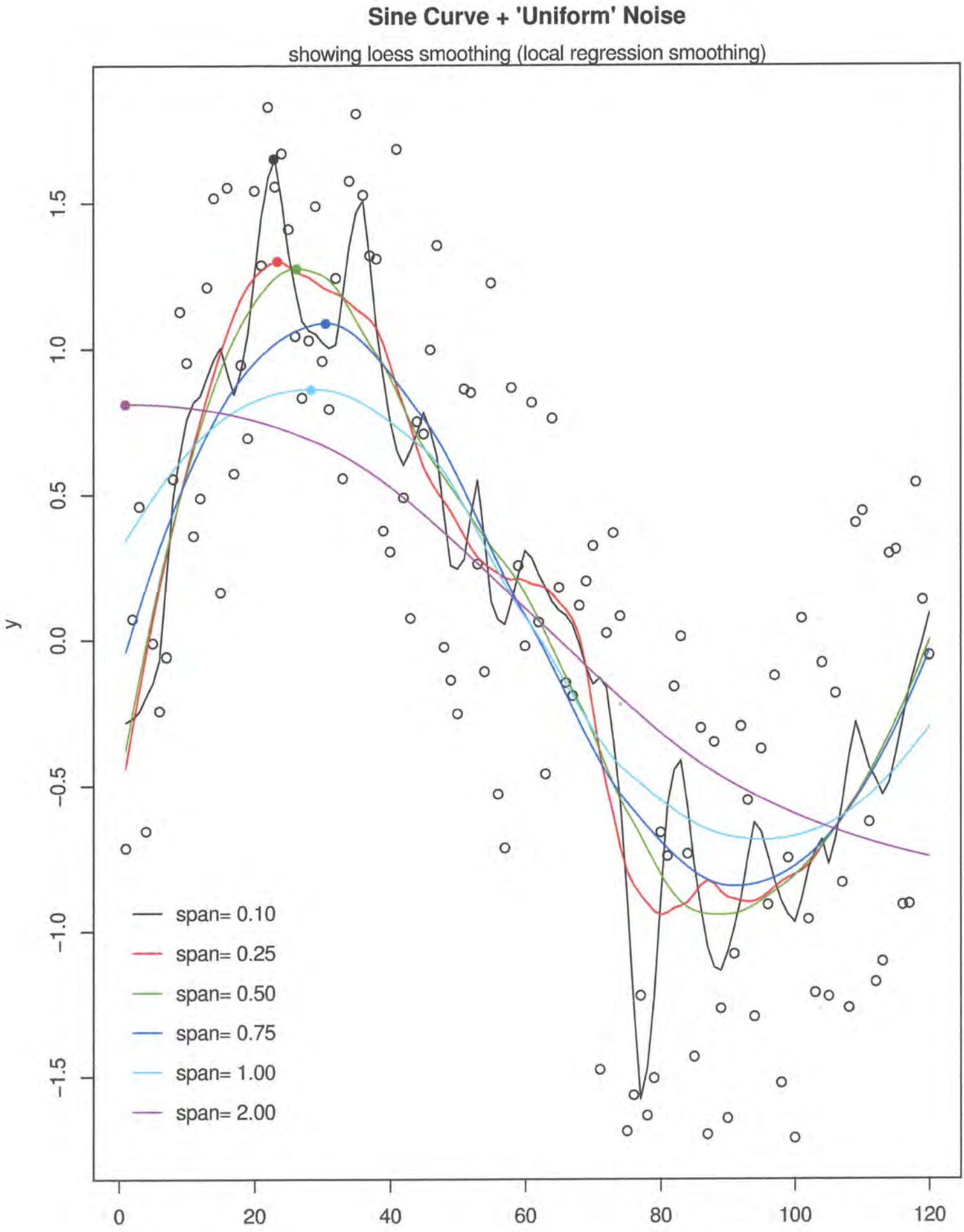
Need to be corrected for bias.

```
> bootmlr <- function(data, indices ){
+ indices <- sample(1:19,replace=TRUE)
+ datastar <- data[indices,]
+ bsresult <- lm(V1 ~ V2 + V3 + V4, data=datastar)
+ coefficients(bsresult) }
> lungboot <- boot(CH09PR13,bootmlr, 1999)
> lungboot

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = CH09PR13, statistic = bootmlr, R = 1999)


Bootstrap Statistics :
       original      bias    std. error
t1* 108.9530243 -19.3487073  28.7874865
t2*  -2.3250160   1.6919410   0.5802653
t3*  -0.2449749  -0.2722142   0.2086077
t4*   0.3428814  -0.4309740   0.4497326
> boot.ci(lungboot, index=2,  type=c("norm", "perc", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = lungboot, type = c("norm", "perc", "bca"),
    index = 2)

Intervals :
Level      Normal              Percentile            BCa
95%   (-5.154, -2.880 )   (-1.971,  0.261 )   (-3.220, -3.075 )
Calculations and Intervals on Original Scale
Warning : BCa Intervals used Extreme Quantiles
Some BCa intervals may be unstable
Warning message:
In norm.inter(t, adj.alpha) : extreme order statistics used as endpoints

> confint(mlrfit)
                 2.5 %       97.5 %
(Intercept) 41.2495093 133.12549088
x1          -1.4765554   0.34759875
x2          -0.9916493  -0.03465126
x3          -1.0408608   0.89694494
> boot.ci(lungboot, index=3,  type=c("norm", "perc", "bca"))
```

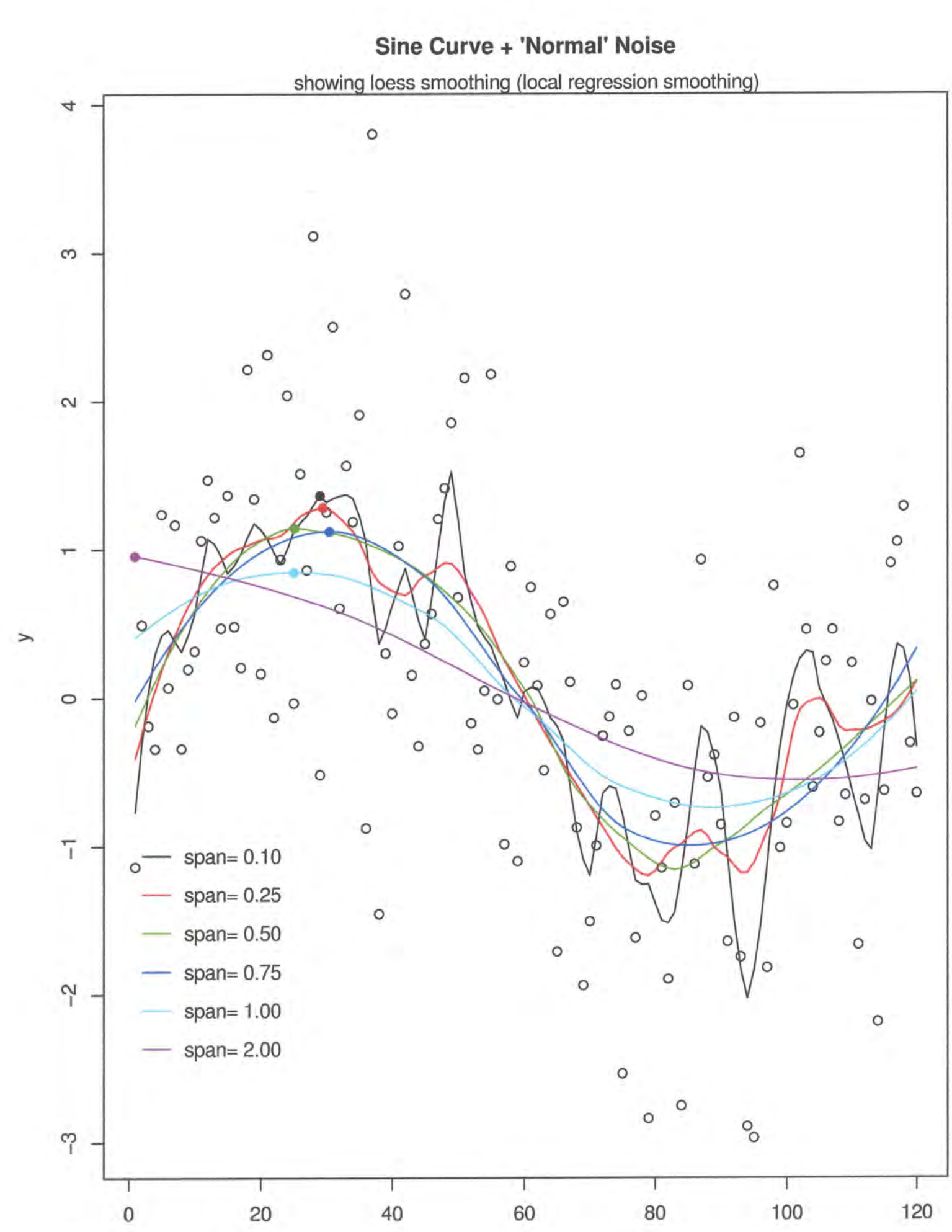
```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = lungboot, type = c("norm", "perc", "bca"),
    index = 3)

Intervals :
Level      Normal             Percentile            BCa
95%   (-0.3816,  0.4361 )   (-0.9935, -0.1326 )   (-0.3229,  0.4554 )
Calculations and Intervals on Original Scale
Warning : BCa Intervals used Extreme Quantiles
Some BCa intervals may be unstable
Warning message:
In norm.inter(t, adj.alpha) : extreme order statistics used as endpoints
> boot.ci(lungboot, index=4,  type=c("norm", "perc", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = lungboot, type = c("norm", "perc", "bca"),
    index = 4)

Intervals :
Level      Normal             Percentile            BCa
95%   (-0.1076,  1.6553 )   (-1.0742,  0.7426 )   ( 0.0297,  1.6971 )
Calculations and Intervals on Original Scale
Warning : BCa Intervals used Extreme Quantiles
Some BCa intervals may be unstable
Warning message:
In norm.inter(t, adj.alpha) : extreme order statistics used as endpoints
```

Loess

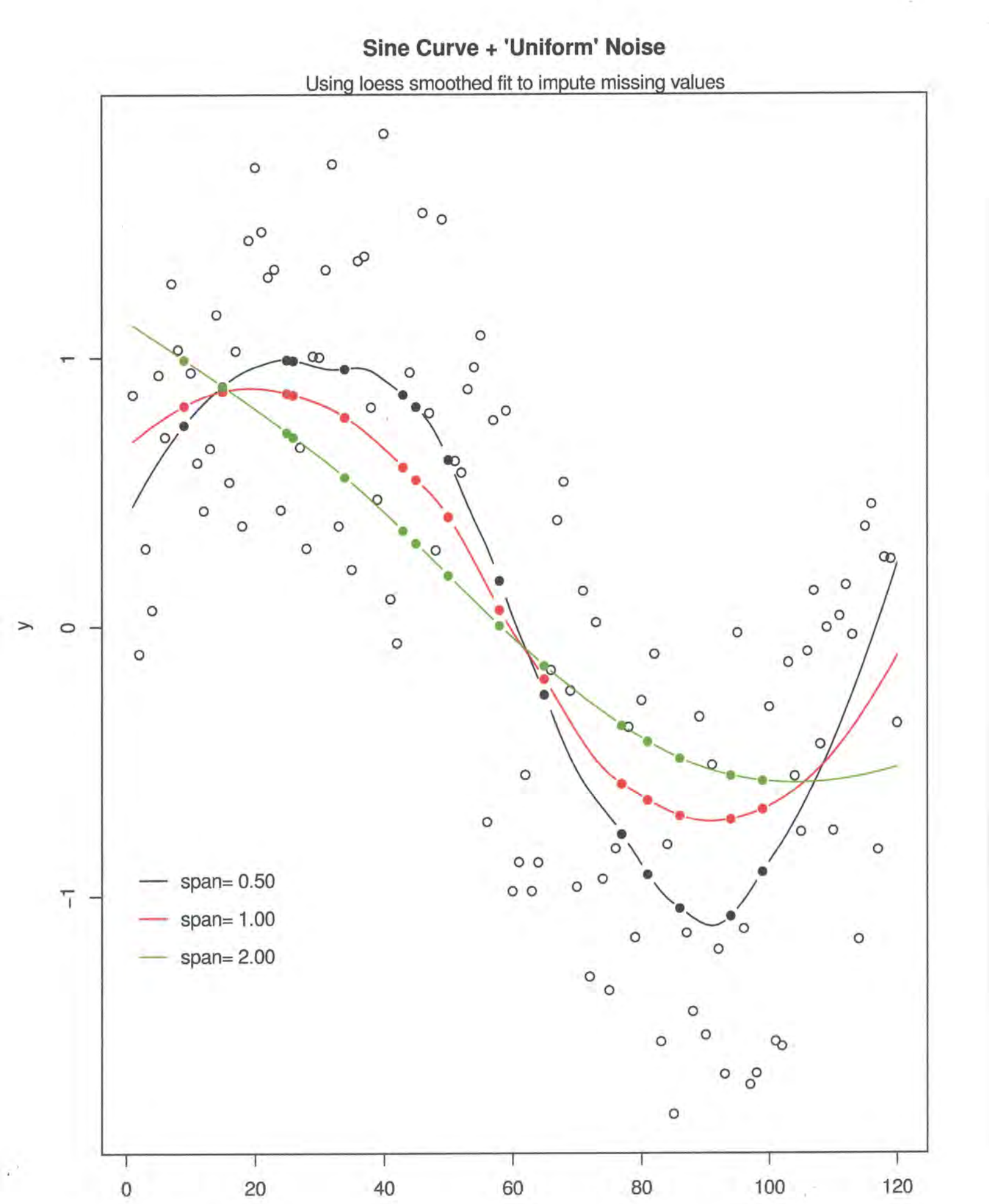
```
# efg, 6 Oct 2004
# Stowers Institute for Medical Research

# Make example reproducible
set.seed(19)

period <- 120

# Create sine curve with noise
x <- 1:120
y <- sin(2*pi*x/period) + runif(length(x),-1,1)

# Plot points on noisy curve
plot(x,y, main="Sine Curve + 'Uniform' Noise")
mtext("showing loess smoothing (local regression smoothing)")

spanlist <- c(0.10, 0.25, 0.50, 0.75, 1.00, 2.00)
for (i in 1:length(spanlist))
{
  y.loess <- loess(y ~ x, span=spanlist[i], data.frame(x=x,
y=y))
  y.predict <- predict(y.loess, data.frame(x=x))

  # Plot the loess smoothed curve
  lines(x,y.predict,col=i)

  # Find peak point on smoothed curve
  peak <- optimize(function(x, model)
                   predict(model, data.frame(x=x)),
                           c(min(x),max(x)),
                           maximum=TRUE,
                           model=y.loess)
  # Show position of smoothed curve maximum
  points(peak$maximum,peak$objective, pch=FILLED.CIRCLE<-19,
col=i)
}

legend (0,-0.8,
        c(paste("span=", formatC(spanlist, digits=2,
format="f"))),
        lty=SOLID<-1, col=1:length(spanlist), bty="n")
```


Sine Curve + 'Uniform' Noise
showing loess smoothing (local regression smoothing)
y
x
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100
120
span= 0.10
span= 0.25
span= 0.50
span= 0.75
span= 1.00
span= 2.00

```
# efg, 6 Oct 2004
# Stowers Institute for Medical Research

# Make example reproducible
set.seed(19)

period <- 120;

# Create sine curve with noise
x <- 1:120
y <- sin(2*pi*x/period) + rnorm(length(x))

# Plot points on noisy curve
plot(x,y, main="Sine Curve + 'Normal' Noise")
mtext("showing loess smoothing (local regression smoothing)")

spanlist <- c(0.10, 0.25, 0.50, 0.75, 1.00, 2.00)
for (i in 1:length(spanlist))
{
  y.loess <- loess(y ~ x, span=spanlist[i], data.frame(x=x,
y=y))
  y.predict <- predict(y.loess, data.frame(x=x))

  # Plot the loess smoothed curve
  lines(x,y.predict,col=i)

  # Find peak point on smooted curve
  peak <- optimize(function(x, model)
                   predict(model, data.frame(x=x)),
                          c(min(x),max(x)),
                          maximum=TRUE,
                          model=y.loess)

  # Show position of smoothed curve maximum
  points(peak$maximum,peak$objective, pch=FILLED.CIRCLE<-19,
col=i)
}

legend (0,-0.8,
        c(paste("span=", formatC(spanlist, digits=2,
format="f"))),
        lty=1, col=1:length(spanlist), bty="n")
```


Sine Curve + 'Normal' Noise
showing loess smoothing (local regression smoothing)
y
span= 0.10
span= 0.25
span= 0.50
span= 0.75
span= 1.00
span= 2.00
4
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120

```
# efg, 15 April 2005
# Stowers Institute for Medical Research

# Make example reproducible
set.seed(19)

period <- 120

FullList <- 1:120
x <- FullList

# "randomly" make 15 of the points "missing"
MissingList <- sample(x,15)
x[MissingList] <- NA

# Create sine curve with noise
y <- sin(2*pi*x/period) + runif(length(x),-1,1)

# Plot points on noisy curve
plot(x,y, main="Sine Curve + 'Uniform' Noise")
mtext("Using loess smoothed fit to impute missing values")

spanlist <- c(0.50, 1.00, 2.00)
for (i in 1:length(spanlist))
{
  y.loess <- loess(y ~ x, span=spanlist[i], data.frame(x=x,
y=y))
  y.predict <- predict(y.loess, data.frame(x=FullList))

  # Plot the loess smoothed curve showing gaps for missing data
  lines(x,y.predict,col=i)

  # Show imputed points to fill in gaps
  y.Missing <-  predict(y.loess, data.frame(x=MissingList))
  points(MissingList, y.Missing, pch=FILLED.CIRCLE<-19, col=i)
}

legend (0,-0.8,
        c(paste("span=", formatC(spanlist, digits=2,
format="f"))),
        lty=SOLID<-1, col=1:length(spanlist), bty="n")
```


Sine Curve + 'Uniform' Noise
Using loess smoothed fit to impute missing values
y
1
0
−1
0
20
40
60
80
100
120
span= 0.50
span= 1.00
span= 2.00

```
> kidneyfunction <- read.table("/home/snap/work/qba530/Data Sets/Chapter  9 Data
Sets/kidneyfunction.txt", header=T, quote="\"")
>   View(kidneyfunction)
> kidneyfunction
      y   x1 x2  x3
1   132 0.71 38  71
2    53 1.48 78  69
3    50 2.21 69  85
4    82 1.43 70 100
5   110 0.68 45  59
6   100 0.76 65  73
7    68 1.12 76  63
8    92 0.92 61  81
9    60 1.55 68  74
10   94 0.94 64  87
11  105 1.00 66  79
12   98 1.07 49  93
13  112 0.70 43  60
14  125 0.71 42  70
15  108 1.00 66  83
16   30 2.52 78  70
17  111 1.13 35  73
18  130 1.12 34  85
19   94 1.38 35  68
20  130 1.12 16  65
21   59 0.97 54  53
22   38 1.61 73  50
23   65 1.58 66  74
24   85 1.40 31  67
25  140 0.68 32  80
26   80 1.20 21  67
27   43 2.10 73  72
28   75 1.36 78  67
29   41 1.50 58  60
30  120 0.82 62 107
31   52 1.53 70  75
32   73 1.58 63  62
33   57 1.37 68  52
```

Y = kidney function

X_1 = creatine concentration

X_2 = age

X_3 = weight

```
> slrfit <- lm(y ~ x1, data=kidneyfunction)

> summary(slrfit)

Call:
lm(formula = y ~ x1, data = kidneyfunction)

Residuals:
     Min       1Q   Median       3Q      Max
-41.769 -11.547   2.787  10.897  37.565
```

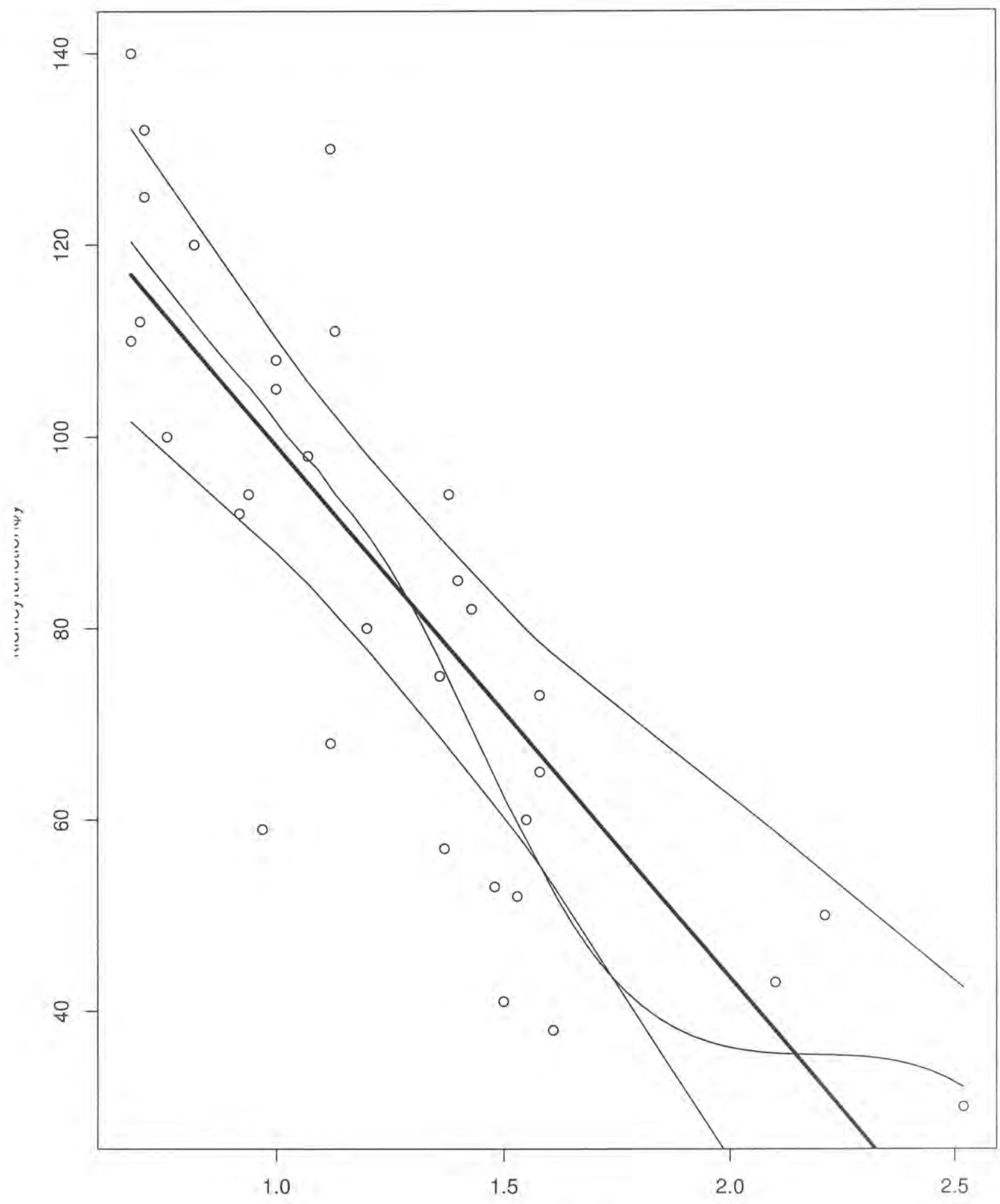
```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  154.662      9.861  15.684 2.72e-16 ***
x1           -55.560      7.437  -7.471 2.04e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.9 on 31 degrees of freedom
Multiple R-squared: 0.6429,     Adjusted R-squared: 0.6314
F-statistic: 55.81 on 1 and 31 DF,  p-value: 2.041e-08


> anova(slrfit)
Analysis of Variance Table

Response: y
           Df Sum Sq Mean Sq F value    Pr(>F)
x1         1  19927   19927  55.811 2.041e-08 ***
Residuals 31  11068     357
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> slryhat <- predict(slrfit,se=TRUE)
> slryhat
$fit
        1         2         3         4         5         6         7         8         9
10        11        12
115.21435  72.43339  31.87482  75.21138 116.88114 112.43637  92.43488 103.54682  68.54422
102.43562  99.10204  95.21286
        13        14        15        16        17        18        19        20        21
22        23        24
115.76995 115.21435  99.10204  14.65132  91.87928  92.43488  77.98936  92.43488 100.76883
65.21063  66.87742  76.87817
        25        26        27        28        29        30        31        32        33
116.88114  87.99011  37.98639  79.10056  71.32220 109.10279  69.65541  66.87742  78.54496

$se.fit
 [1]  5.191146  3.707495  7.860868  3.551291  5.365615  4.909128  3.428464  4.104007  3.974609
4.016828  3.778425  3.551291
[13]  5.248893  5.191146  3.778425 10.001450  3.408238  3.428464  3.428464  3.428464  3.893061
4.241202  4.104007  3.473341
[25]  5.365615  3.310272  7.126082  3.389525  3.778425  4.587639  3.893061  4.104007  3.408238
$df
[1] 31
$residual.scale
[1] 18.89571

> slrupper <- slryhat$fit + 2.8843*slryhat$se
> slrlower <- slryhat$fit - 2.8843*slryhat$se
```

Working-Hotelling Intervals

$\hat{Y}_h \pm W \hat{\sigma}_{\hat{Y}_h}$

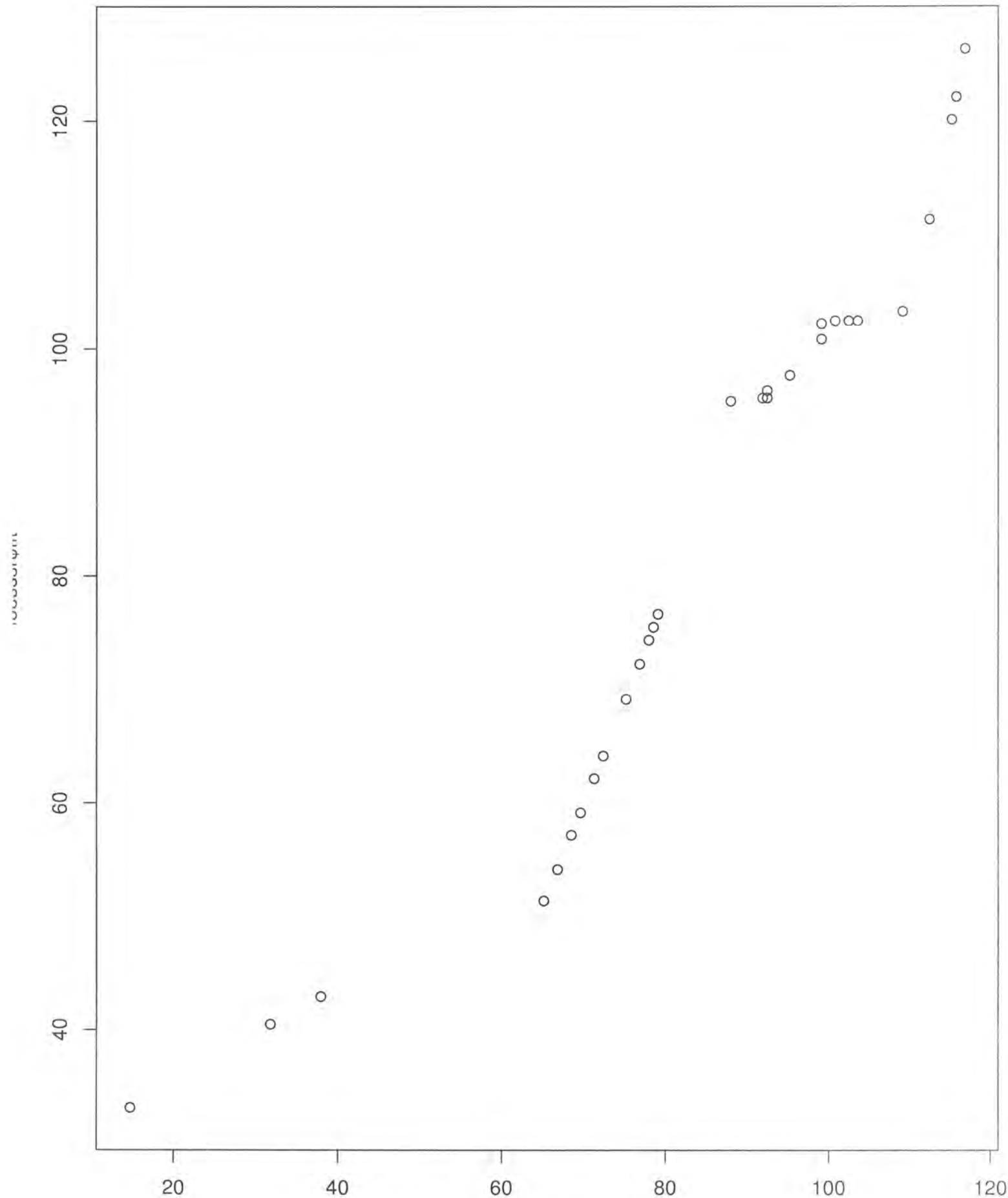
$W^2 = 2F(1-\alpha; df(\text{Model}), df(\text{Error}))$

```
> loesssr <- loess( y ~ x1 , span=.6, degree=2, data=kidneyfunction)


> loesssr
Call:
loess(formula = y ~ x1, data = kidneyfunction, span = 0.6, degree = 2)
Number of Observations: 33
Equivalent Number of Parameters: 6.12
Residual Standard Error: 17.96
> loesssr$fit
 [1] 120.10411  64.12973  40.47215  69.12440 126.35196 111.30072 102.42869  96.31749  57.16682
95.69277  95.65933 100.83425
[13] 122.10862 120.10411  95.65933  33.10980 102.18783 102.42869  74.32039 102.42869  95.38568
51.35274  54.12012  72.21353
[25] 126.35196  97.65848  42.91424  76.59776  62.14652 103.25039  59.14484  54.12012  75.45356

> loessyhat <- loesssr$fit

> plot(kidneyfunction$x1,kidneyfunction$y)
> lines(lowess(kidneyfunction$x1,slrupper))
> lines(lowess(kidneyfunction$x1,slrlower))
> lines(loess.smooth(kidneyfunction$x1,kidneyfunction$y))
> lines(kidneyfunction$x1,slryhat$fit,lty=1, lwd=3)



> SSE.1 <- sum(residuals(slrfit)^2)
> SSE.2 <- sum(residuals(loesssr)^2)
> F <- ((SSE.1 - SSE.2)/(6.12-2)) /(SSE.2/(33-6.12))
> pf(F, 6.12-2, 33-6.12, lower.tail=FALSE)
[1] 0.09229549
```


140
120
100
80
60
40
1.0
1.5
2.0
2.5
kidneyfunction$x1

```
> olsfit <- lm(y ~ x1 + x2 + x3, data=kidneyfunction)
> summary(olsfit)

Call:
lm(formula = y ~ x1 + x2 + x3, data = kidneyfunction)

Residuals:
    Min      1Q  Median      3Q     Max
-28.668  -7.002   1.518   9.905  16.006

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 120.0473    14.7737   8.126 5.84e-09 ***
x1          -39.9393     5.6000  -7.132 7.55e-08 ***
x2           -0.7368     0.1414  -5.211 1.41e-05 ***
x3            0.7764     0.1719   4.517 9.69e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.46 on 29 degrees of freedom
Multiple R-squared: 0.8548,     Adjusted R-squared: 0.8398
F-statistic: 56.92 on 3 and 29 DF,  p-value: 2.885e-12

> loessfit <- loess( y ~ x1 + x2 + x3, span=.6, degree=2, data=kidneyfunction)
> loessfit
Call:
loess(formula = y ~ x1 + x2 + x3, data = kidneyfunction, span = 0.6,
    degree = 2)

Number of Observations: 33
Equivalent Number of Parameters: 17.65
Residual Standard Error: 23.98
> loessfit$fit
 [1] 131.40470  57.90003  46.21424  79.87604 115.34473 100.86482  85.33569  94.81134
57.08460 106.41513 104.32592 101.83985 116.23601 123.72807
[15] 104.31990  29.12750  90.38730 118.43410  82.44848 110.50964  66.33323  49.87268
62.14259  64.83609 140.10893  78.75218  40.58844  69.52253
[29]  75.74448 132.15599  54.50860  69.66752  54.54537

> loessyhat <- loessfit$fit

> olsyhat <- predict(olsfit, se=TRUE)
```

```
> lower <- olsyhat$fit - 2.4224*olsyhat$se
> upper <- olsyhat$fit + 2.4224*olsyhat$se
> result <- data.frame(lower,upper,loessyhat)
> result
          lower     upper loessyhat
1  110.0522457 127.58560 131.40470
2   48.3522605  65.73196  57.90003
3   32.7515215  61.12849  46.21424
4   75.9617058 102.04268  79.87604
5   94.9469781 116.13844 115.34473
6   88.4574849 108.50667 100.86482
7   57.5652735  78.90531  85.33569
8   93.2186506 109.28175  94.81134
9   58.7463093  72.24594  57.08460
10  93.4652565 112.33399 106.41513
11  84.7922354 100.84453 104.32592
12 103.1410138 123.69406 101.83985
13  96.8172963 117.17045 116.23601
14 106.5686367 123.62223 123.72807
15  87.3704901 104.47762 104.31990
16   0.2981575  32.26510  29.12750
17  97.3831594 114.23193  90.38730
18 105.8293459 126.69211 118.43410
19  82.2641500 101.61709  82.44848
20 100.0037642 127.98457 110.50964
21  72.0232411  93.31853  66.33323
22  28.6354481  52.92832  49.87268
23  59.0366031  72.50636  62.14259
24  82.3317860 104.29318  64.83609
25 121.2439280 141.60702 140.10893
26  95.9161250 121.41992  78.75218
27  26.8870106  49.69863  40.58844
28  51.0753711  69.48862  69.52253
29  56.2024249  71.77938  75.74448
30 108.7945477 140.59396 132.15599
31  58.6273433  72.56825  54.50860
32  50.9395028  66.39002  69.66752
33  44.6952804  66.51271  54.54537
```


120
100
80
60
40
20
40
60
80
100
120
slryhat$fit