STAT 428 Statistical Computing

Homework 3 Solutions

Department of Statistics, University of Illinois at Urbana-Champaign

Feb 26th 2017

Problem 1 (Two-Sample T-test vs Wilcoxon Ranked Sum Test)

For Problem 1, we wish to test

$$H_0: \ \mu_x = \mu_y \quad vs \quad H_a: \ \mu_x > \mu_y$$

Two-Sample T-test and **Wilcoxon Ranked Sum Test** given in the description part of problem 1 can be carried out directly using the functions `t.test()` and `wilcox.test()` in *R*.

Basically, **power** is the probability of rejecting H_0 when H_a is true.

$$power = P(reject \ H_0 \mid H_A \ is \ true)$$

type 1 error rate is the probability of rejecting H_0 when H_0 is true.

$$type \ I \ error = P(reject \ H_0 \mid H_0 \ is \ true)$$

The algorithm of estimating **power** and **type 1 error rate** is given below:

Step 1: Draw n samples from F_x and F_y respectively;

Step 2: Decide if you should support or reject H_0 based on samples you drawed using Two-Sample T-test or Wilcoxon Ranked Sum Test;

Step 3: Repeat Step 1-2 m times and compute the proportion of rejecting H_0.

Notes:

In this problem, if $F_x = F_y$, we are estimating type 1 error rate. Otherwise, we are estimating power.

```
#####Function for a one-time simulation
###Input:
#dist: distribution, which can be normal ("norm"), exponential ("exp") or chi-square ("chisq")
#alpha: level of the test, default is 0.05
#n: sample size
#delta: -delta is the mean under the alternative. Specify when dist = "norm"
#lambda: rate of exponential distribution. Specify when dist = "exp"
#df: degree of freedom of chi-square distribution. Specify when dist = "chisq"
###Output:
#rej.fun1: reject function for t-test
#rej.fun2: reject function for wilcoxon test

two.sample.test <- function(delta, alpha = 0.05, dist, lambda, df, n){
    #Generate random samples
    X = switch(dist,
        "norm" = rnorm(n,0,1),
        "exp" = rexp(n,1),
        "chisq" = rchisq(n,df=1))
```

```
    Y = switch(dist,
        "norm" = rnorm(n,-delta,1),
        "exp" = rexp(n,lambda),
        "chisq" = rchisq(n,df=df))
    #Perform two-sample t-test
    t.test.result = t.test(X, Y, alternative = "greater", val.equal = T)
    wilcox.test.result = wilcox.test(X, Y, alternative = "greater", exact = FALSE)
    #p-values
    rej.fun1 = as.numeric(t.test.result$p.value<alpha)
    rej.fun2 = as.numeric(wilcox.test.result$p.value<alpha)
    return(c(rej.fun1, rej.fun2))
}

#Function to estimate type 1 error and power
rej.rate <- function(delta, alpha = 0.05, dist, lambda, df, n){
    iter = 1000 #Number of iterations
    rej.fun = switch(dist,
        "norm" = replicate(iter, two.sample.test(delta = delta, dist = "norm", n = n)),
        "exp" = replicate(iter, two.sample.test(lambda = lambda, dist = "exp", n = n)),
        "chisq" = replicate(iter, two.sample.test(df = df, dist = "chisq", n = n)))
    rej.rate = apply(rej.fun, FUN = mean, MARGIN = 1)
    return(rej.rate)
}
```

(a)

$X \sim N(0,1)$ and $Y \sim N(-\Delta,1)$.

Thus, we are essentially testing

$$H_0: \ \Delta = 0 \quad vs \quad H_a: \ \Delta > 0.$$

```
#(a) Normal distribution case
#Estimate type 1 error and power
library(knitr)
n = c(10,20,50,100,200,500) #sample sizes
delta = c(0,0.1,0.2,0.5,1,2) #normal mean
par = cbind(rep(n,6),matrix(apply(as.matrix(delta),1,rep,6), ncol = 1))
# rej.rate.estimate.norm = mapply(FUN = rej.rate, delta = par[,2], n = par[,1], dist = "norm")
load("rej.rate.estimate.norm.Rdata")
t.est<-matrix(rej.rate.estimate.norm, ncol=length(delta))[seq(1,12,2),]
w.est<-matrix(rej.rate.estimate.norm, ncol=length(delta))[seq(2,12,2),]
rownames(t.est) = as.character(n)
colnames(t.est) = as.character(delta)
rownames(w.est) = as.character(n)
colnames(w.est) = as.character(delta)
knitr::kable(t.est, caption = "Normal Distribution (t test)")
```

Table 1: Normal Distribution (t test)

	0	0.1	0.2	0.5	1	2
10	0.061	0.081	0.102	0.261	0.677	0.999
20	0.058	0.100	0.141	0.454	0.925	1.000
50	0.051	0.113	0.275	0.800	0.999	1.000

	0	0.1	0.2	0.5	1	2
100	0.050	0.185	0.392	0.965	1.000	1.000
200	0.055	0.255	0.651	1.000	1.000	1.000
500	0.056	0.481	0.943	1.000	1.000	1.000

```
knitr::kable(w.est, caption = "Normal Distribution (wilcoxon test)")
```

Table 2: Normal Distribution (wilcoxon test)

	0	0.1	0.2	0.5	1	2
10	0.055	0.074	0.098	0.246	0.645	0.996
20	0.056	0.091	0.137	0.450	0.906	1.000
50	0.056	0.115	0.266	0.785	0.999	1.000
100	0.047	0.170	0.383	0.957	1.000	1.000
200	0.049	0.251	0.648	1.000	1.000	1.000
500	0.054	0.462	0.928	1.000	1.000	1.000

```
knitr::kable(round(sqrt(t.est*(1-t.est)/1000),3), caption = "standard error (t test)")
```

Table 3: standard error (t test)

	0	0.1	0.2	0.5	1	2
10	0.008	0.009	0.010	0.014	0.015	0.001
20	0.007	0.009	0.011	0.016	0.008	0.000
50	0.007	0.010	0.014	0.013	0.001	0.000
100	0.007	0.012	0.015	0.006	0.000	0.000
200	0.007	0.014	0.015	0.000	0.000	0.000
500	0.007	0.016	0.007	0.000	0.000	0.000

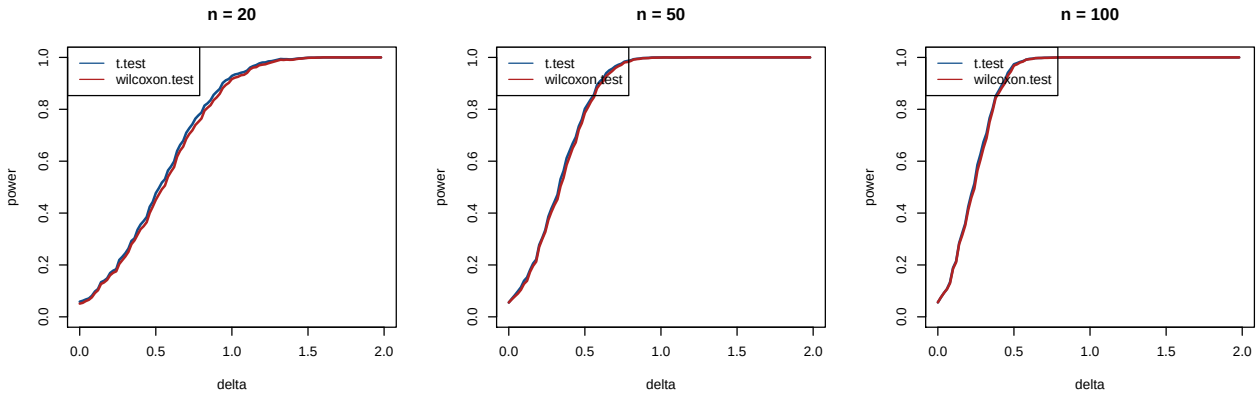
```
knitr::kable(round(sqrt(w.est*(1-w.est)/1000),3), caption = "standard error (wilcoxon test)")
```

Table 4: standard error (wilcoxon test)

	0	0.1	0.2	0.5	1	2
10	0.007	0.008	0.009	0.014	0.015	0.002
20	0.007	0.009	0.011	0.016	0.009	0.000
50	0.007	0.010	0.014	0.013	0.001	0.000
100	0.007	0.012	0.015	0.006	0.000	0.000
200	0.007	0.014	0.015	0.000	0.000	0.000
500	0.007	0.016	0.008	0.000	0.000	0.000

```
par(mfrow=c(1,3))
for(n in c(20,50,100)){
plot1<-apply(as.matrix(seq(0,2,0.03),1), 1, rej.rate, alpha = 0.05, dist="norm", n=n)
x=seq(0,2,0.03)
y1=plot1[1,]
y2=plot1[2,]
powersmooth=ksmooth(x,y1,bandwidth=.1)
```

```
plot(powersmooth$x,powersmooth$y, xlab="delta",ylab="power",xlim=c(0,2), ylim=c(0,1),lwd=2,type="l",col=
powersmooth=ksmooth(x,y2,bandwidth=.1)
lines(powersmooth$x,powersmooth$y, xlab="delta",ylab="power", ylim=c(0,1),lwd=2,col="firebrick")
legend("topleft",col = c('dodgerblue4','firebrick'), lty = 1, c("t.test", "wilcoxon.test"))
}
```



(b)

$X \sim exp(1)$ and $Y \sim exp(\lambda)$.

Thus, we are essentially testing

$$H_0 : \ \lambda = 1 \quad vs \quad H_a : \ \lambda > 1.$$

```
#(b) Exponential case
n = c(10,20,50,100,200,500) #sample sizes
lambda = c(1,1.2,1.5,3,5,10)
par = cbind(rep(n,6),matrix(apply(as.matrix(lambda),1,rep,6), ncol = 1))
# rej.rate.estimate.exp = mapply(FUN = rej.rate, lambda = par[,2], n = par[,1], dist = "exp")
load("rej.rate.estimate.exp.Rdata")
t.est<-matrix(rej.rate.estimate.exp, ncol=length(lambda))[seq(1,12,2),]
w.est<-matrix(rej.rate.estimate.exp, ncol=length(lambda))[seq(2,12,2),]
rownames(t.est) = as.character(n)
colnames(t.est) = as.character(lambda)
rownames(w.est) = as.character(n)
colnames(w.est) = as.character(lambda)
knitr::kable(t.est, caption = "Exponential Distribution (t test)")
```

Table 5: Exponential Distribution (t test)

	1	1.2	1.5	3	5	10
10	0.045	0.103	0.199	0.662	0.902	0.973
20	0.047	0.126	0.304	0.947	0.999	1.000
50	0.055	0.233	0.633	1.000	1.000	1.000
100	0.046	0.352	0.890	1.000	1.000	1.000
200	0.044	0.576	0.994	1.000	1.000	1.000
500	0.055	0.900	1.000	1.000	1.000	1.000

```
knitr::kable(w.est, caption = "Exponential Distribution (wilcoxon test)")
```

Table 6: Exponential Distribution (wilcoxon test)

	1	1.2	1.5	3	5	10
10	0.044	0.100	0.177	0.637	0.884	0.987
20	0.055	0.124	0.256	0.889	0.998	1.000
50	0.050	0.186	0.544	1.000	1.000	1.000
100	0.048	0.293	0.794	1.000	1.000	1.000
200	0.055	0.475	0.973	1.000	1.000	1.000
500	0.042	0.790	1.000	1.000	1.000	1.000

```
knitr::kable(round(sqrt(t.est*(1-t.est)/1000),3), caption = "standard error (t test)")
```

Table 7: standard error (t test)

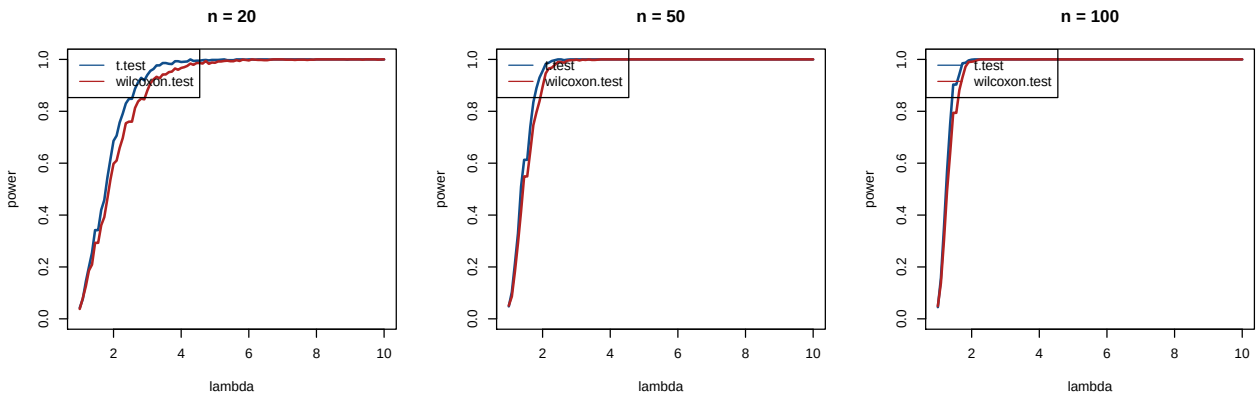
	1	1.2	1.5	3	5	10
10	0.007	0.010	0.013	0.015	0.009	0.005
20	0.007	0.010	0.015	0.007	0.001	0.000
50	0.007	0.013	0.015	0.000	0.000	0.000
100	0.007	0.015	0.010	0.000	0.000	0.000
200	0.006	0.016	0.002	0.000	0.000	0.000
500	0.007	0.009	0.000	0.000	0.000	0.000

```
knitr::kable(round(sqrt(w.est*(1-w.est)/1000),3), caption = "standard error (wilcoxon test)")
```

Table 8: standard error (wilcoxon test)

	1	1.2	1.5	3	5	10
10	0.006	0.009	0.012	0.015	0.010	0.004
20	0.007	0.010	0.014	0.010	0.001	0.000
50	0.007	0.012	0.016	0.000	0.000	0.000
100	0.007	0.014	0.013	0.000	0.000	0.000
200	0.007	0.016	0.005	0.000	0.000	0.000
500	0.006	0.013	0.000	0.000	0.000	0.000

```
par(mfrow=c(1,3))
for(n in c(20,50,100)){
x=seq(1,10,0.1)
plot2<-mapply(FUN = rej.rate, lambda = x, n = n, dist = "exp")
y1=plot2[1,]
y2=plot2[2,]
powersmooth=ksmooth(x,y1,bandwidth=.1)
plot(powersmooth$x,powersmooth$y, xlab="lambda",ylab="power",xlim=c(1,10), ylim=c(0,1),lwd=2,type="l",c
powersmooth=ksmooth(x,y2,bandwidth=.1)
lines(powersmooth$x,powersmooth$y, xlab="lambda",ylab="power", ylim=c(0,1),lwd=2,col="firebrick")
legend("topleft",col = c('dodgerblue4','firebrick'), lty = 1, c("t.test", "wilcoxon.test"))
}
```



(c)

$X \sim \chi^2(1)$ and $Y \sim \chi^2(\upsilon)$.

Thus, we are essentially testing

$$H_0 : \upsilon = 1 \quad vs \quad H_a : \upsilon < 1.$$

```
#(c) Chi-square case
n = c(10,20,50,100,200,500) #sample sizes
df = c(0.1,0.2,0.3,0.5,0.8,1)
par = cbind(rep(n,6),matrix(apply(as.matrix(df),1,rep,6), ncol = 1))
# rej.rate.estimate.chisq = mapply(FUN = rej.rate, df = par[,2], n = par[,1], dist = "chisq")
load("rej.rate.estimate.chisq.Rdata")
t.est<-matrix(rej.rate.estimate.chisq, ncol=length(df))[seq(1,12,2),]
w.est<-matrix(rej.rate.estimate.chisq, ncol=length(df))[seq(2,12,2),]
rownames(t.est) = as.character(n)
colnames(t.est) = as.character(df)
rownames(w.est) = as.character(n)
colnames(w.est) = as.character(df)
knitr::kable(t.est, caption = "Chisquare Distribution (t test)")
```

Table 9: Chisquare Distribution (t test)

	0.1	0.2	0.3	0.5	0.8	1
10	0.754	0.622	0.425	0.241	0.090	0.049
20	0.946	0.836	0.705	0.437	0.117	0.049
50	0.999	0.988	0.945	0.664	0.182	0.045
100	1.000	1.000	0.999	0.902	0.283	0.059
200	1.000	1.000	1.000	0.993	0.451	0.066
500	1.000	1.000	1.000	1.000	0.768	0.057

```
knitr::kable(w.est, caption = "Chisquare Distribution (wilcoxon test)")
```

Table 10: Chisquare Distribution (wilcoxon test)

	0.1	0.2	0.3	0.5	0.8	1
10	0.994	0.921	0.740	0.386	0.123	0.048
20	1.000	0.994	0.947	0.679	0.157	0.043

	0.1	0.2	0.3	0.5	0.8	1
50	1.000	1.000	1.000	0.946	0.265	0.053
100	1.000	1.000	1.000	0.999	0.457	0.056
200	1.000	1.000	1.000	1.000	0.698	0.064
500	1.000	1.000	1.000	1.000	0.964	0.045

```
knitr::kable(round(sqrt(t.est*(1-t.est)/1000),3), caption = "standard error (t test)")
```

Table 11: standard error (t test)

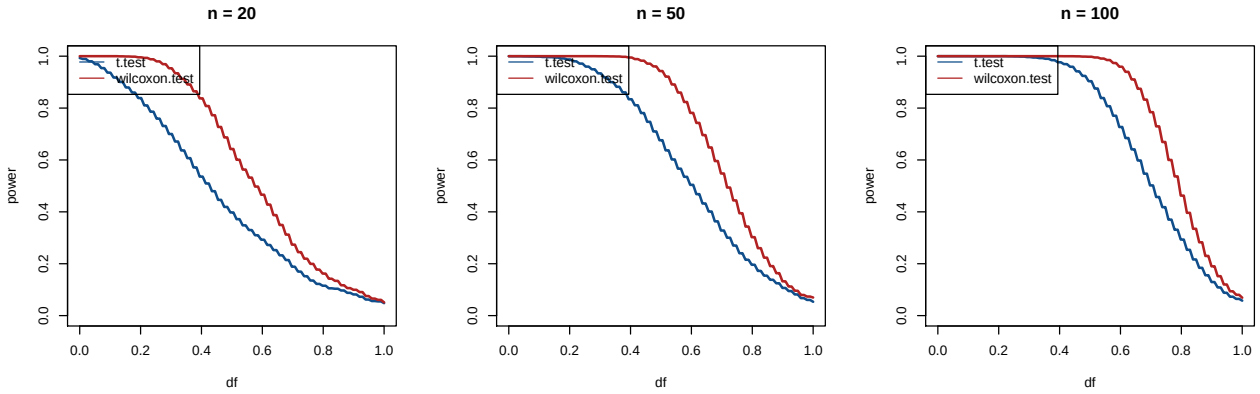
	0.1	0.2	0.3	0.5	0.8	1
10	0.014	0.015	0.016	0.014	0.009	0.007
20	0.007	0.012	0.014	0.016	0.010	0.007
50	0.001	0.003	0.007	0.015	0.012	0.007
100	0.000	0.000	0.001	0.009	0.014	0.007
200	0.000	0.000	0.000	0.003	0.016	0.008
500	0.000	0.000	0.000	0.000	0.013	0.007

```
knitr::kable(round(sqrt(w.est*(1-w.est)/1000),3), caption = "standard error (wilcoxon test)")
```

Table 12: standard error (wilcoxon test)

	0.1	0.2	0.3	0.5	0.8	1
10	0.002	0.009	0.014	0.015	0.010	0.007
20	0.000	0.002	0.007	0.015	0.012	0.006
50	0.000	0.000	0.000	0.007	0.014	0.007
100	0.000	0.000	0.000	0.001	0.016	0.007
200	0.000	0.000	0.000	0.000	0.015	0.008
500	0.000	0.000	0.000	0.000	0.006	0.007

```
par(mfrow=c(1,3))
for(n in c(20,50,100)){
x=seq(0,1,0.02)
plot3<-mapply(FUN = rej.rate, df = x, n = n, dist = "chisq")
y1=plot3[1,]
y2=plot3[2,]
powersmooth=ksmooth(x,y1,bandwidth=.1)
plot(powersmooth$x,powersmooth$y, xlab="df",ylab="power",xlim=c(0,1), ylim=c(0,1),lwd=2,type="l",col="d
powersmooth=ksmooth(x,y2,bandwidth=.1)
lines(powersmooth$x,powersmooth$y, xlab="df",ylab="power", ylim=c(0,1),lwd=2,col="firebrick")
legend("topleft",col = c('dodgerblue4','firebrick'), lty = 1, c("t.test", "wilcoxon.test"))
}
```



How large the sample size should be in order to use normal approximation?

```
set.seed(183)
n.check = matrix(nrow = 3, ncol = 20, data = 0)
for(n in 1:20){
    X = rnorm(n,0,1)
    Y = rnorm(n,0,1)
    n.check[1,n] = wilcox.test(X, Y, alternative = "greater", exact = TRUE)$p.value
    n.check[2,n] = wilcox.test(X, Y, alternative = "greater", exact = FALSE)$p.value
}
n.check[3,]<-abs(n.check[1,]-n.check[2,])
knitr::kable(round(n.check[,1:10],4),col.names = as.character(1:10))
```

1	2	3	4	5	6	7	8	9	10
0.5	1.0000	0.5	0.5571	0.7262	0.9794	0.7325	0.1172	0.4657	0.6020
0.5	0.9736	0.5	0.5574	0.7346	0.9773	0.7385	0.1136	0.4648	0.6043
0.0	0.0264	0.0	0.0002	0.0084	0.0021	0.0060	0.0037	0.0009	0.0023

When $n \geq 8$, the absolute error between exact and approximate p-value is smaller than 0.005, so we can replace the exact p-value with the approximated one.

Conclusion

1. For normal distribution and exponential distribution, the first column of the table are estimates of **type 1 error rate** and others are estimates of **power**. For chi-square distrubution, the last column of the table are estimates of **type 1 error rate** and others are estimates of **power**.
2. The **type 1 error rate** will get closer to 0.05 when increasing n. The **power** will get closer to 1 when increasing n. Also, when the value of the parameter under the alternative hypothesis is far away from the one under the null hypothesis, the **power** will be greater.
3. Comparing t and W
   From plots, we can see that
   a) For normal distribution, **Two-Sample T-test** performed better than **Wilcoxon Ranked Sum Test** because the t reference distribution for the t-statistic is exact.
   b) For exponential distribution, **Two-Sample T-test** performed better than **Wilcoxon Ranked Sum Test**.
   c) For chi-square distribution, **Wilcoxon Ranked Sum Test** performed better than **Two-Sample T-test**. Because when the normality assumption doesn't hold, a non-parametric test will be more robust.

Problem 2: (Comparison of Confidence Intervals)

The algorithm of estimating the **coverage probability** is given below:

Step 1: Draw n samples from F_x;

Step 2: Construct Standard Normal Bootstrap Interval or Percentile Bootstrap Confidence Interval based on samples you drawed;

Step 3: Repeat Step 1-2 m times and compute the proportion of intervals that covered the true median of F_x.

Similarly, to get the **mean lengths of intervals**, we just need to take the mean of lengths of m intervals we got in Step 3.

The standard error for the coverage probability estimates is given by

$$se(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{m}},$$

where $\hat{p}$ is the estimate of the coverage probability.

```
###Function for resample
boot.resample.med <- function(x,n){
    xb = sample(x,n,replace = TRUE)
    return(median(xb))
}

###Function to construct CI
###Input:
#dist: distribution, which can be standard normal ("norm"), exponential ("exp"), Cauchy ("cauchy") or u
#alpha: level of the test, default is 0.10
#n: sample size, which can be n = 20, 50 or 100
###Output:
#A 2-by-2 matrix.
ci.bootstrap <- function(n, alpha = 0.10, dist){
    B = 10000 #Number of replications
    #Generate random samples from certain distributions
    X = switch(dist,
        "norm" = rnorm(n),
        "exp" = rexp(n,1),
        "cauchy" = rcauchy(n),
        "unif" = runif(n,0,1))
    thetahat = median(X)
    #Resample
    thetahat.boot = replicate(B, boot.resample.med(X,n))
    se.thetahat = sd(thetahat.boot)
    zval = qnorm(0.975,0,1)
    #CI
    L1 = thetahat - zval*se.thetahat
    U1 = thetahat + zval*se.thetahat
    #Percentile interval
    thetahat.boot = sort(thetahat.boot)
    L2 = thetahat.boot[250]
    U2 = thetahat.boot[9750]
    return(matrix(c(L1,L2,U1,U2), nrow = 2))
}
```

```r
#Function to estimate the coverage rate and lengths of intervals
ci.coverage <- function(n, alpha = 0.10, dist){
    iter = 4000 #Number of iterations
    #True median
    med = switch(dist,
        "norm" = qnorm(0.5,0,1),
        "exp" = qexp(0.5,1),
        "cauchy" = qcauchy(0.5),
        "unif" = qunif(0.5,0,1))
    result = replicate(iter,ci.bootstrap(n = n, dist = dist))
    coverage1 = sum((med>result[1,1,])&(med<result[1,2,]))/iter
    coverage2 = sum((med>result[2,1,])&(med<result[2,2,]))/iter
    ci.length1 = mean(result[1,2,] - result[1,1,])
    ci.length2 = mean(result[2,2,] - result[2,1,])
    return(matrix(c(coverage1,ci.length1,coverage2,ci.length2), nrow = 2))
}

#Compare coverage prob and mean length of the intervals
n = c(20,50,100)
#norm<-apply(as.matrix(n), MARGIN = 1, FUN = ci.coverage, dist = "norm")
load("norm.Rdata")
load("exp.Rdata")
load("cauchy.Rdata")
load("unif.Rdata")
colnames(norm)<-as.character(n)
rownames(norm)<-c("bs_cover_prob","bs_length","order_bs_cover_prob","order_bs_length")
colnames(exp)<-as.character(n)
rownames(exp)<-c("bs_cover_prob","bs_length","order_bs_cover_prob","order_bs_length")
colnames(cauchy)<-as.character(n)
rownames(cauchy)<-c("bs_cover_prob","bs_length","order_bs_cover_prob","order_bs_length")
colnames(unif)<-as.character(n)
rownames(unif)<-c("bs_cover_prob","bs_length","order_bs_cover_prob","order_bs_length")
#exp<-apply(as.matrix(n), MARGIN = 1, FUN = ci.coverage, dist = "exp")
#cauchy<-apply(as.matrix(n), MARGIN = 1, FUN = ci.coverage, dist = "cauchy")
#unif<-apply(as.matrix(n), MARGIN = 1, FUN = ci.coverage, dist = "unif")
knitr::kable(round(norm,4), caption = "Normal Distribution")
```

Table 14: Normal Distribution

	20	50	100
bs_cover_prob	0.9400	0.9450	0.9400
bs_length	1.1023	0.7097	0.5028
order_bs_cover_prob	0.9500	0.9475	0.9450
order_bs_length	1.0464	0.7014	0.4945

```r
knitr::kable(round(exp,4), caption = "Exponential Distribution")
```

Table 15: Exponential Distribution

	20	50	100
bs_cover_prob	0.9375	0.9350	0.9675
bs_length	0.9069	0.5804	0.4040

	20	50	100
order_bs_cover_prob	0.9425	0.9525	0.9650
order_bs_length	0.8583	0.5700	0.3950

```
knitr::kable(round(cauchy,4), caption = "Cauchy Distribution")
```

Table 16: Cauchy Distribution

	20	50	100
bs_cover_prob	0.9750	0.9525	0.9425
bs_length	1.7228	0.9610	0.6578
order_bs_cover_prob	0.9625	0.9550	0.9500
order_bs_length	1.6223	0.9529	0.6482

```
knitr::kable(round(unif,4), caption = "Uniform Distribution")
```

Table 17: Uniform Distribution

	20	50	100
bs_cover_prob	0.9200	0.9275	0.9250
bs_length	0.3994	0.2760	0.1922
order_bs_cover_prob	0.9375	0.9525	0.9325
order_bs_length	0.3780	0.2682	0.1905

```
min_p<-min(norm[c(1,3),],exp[c(1,3),],cauchy[c(1,3),],unif[c(1,3),])
sqrt(min_p*(1-min_p)/4000)
```

```
## [1] 0.004289522
```

Conclusions

1. The coverage probability given by Standard Normal Bootstrap Interval and Percentile Bootstrap Confidence Interval are both close to 0.9 when increasing n. Confidence Intervals given by Percentile Bootstrap method are slightly narrower than the ones constructed by Standard Normal Bootstrap method. Since both method gave similar coverage probability, we will recommend the Percentile Bootstrap method, which gave a narrower interval.

2. Since that the standard error is given by

$$se(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{m}},$$

$$\sqrt{\frac{\hat{p}(1-\hat{p})}{m}} \approx \sqrt{\frac{0.9 \times (1-0.9)}{m}} \leq 0.005 \Rightarrow m >= 3600,$$

where m is number of iterations.