Appendix for Advanced Statistical Method HW3

2021-21116 Taeyoung Chang

Compare the performance of R and Rcpp

We shall compare the elapsed time to solve problem2 of HW3 "Reproduce Figure 4.2"

```
library(tictoc)
```

Original codes using R

Codes for finding MLE by Newton's method

```
# l'(theta) where l(theta) is loglikelihood function
cauchy_lprime<-function(theta, x){
  n=length(x)
  sum=0
  for(i in 1:n){
    numerator=2*(x[i]-theta)
    denominator=1+(x[i]-theta)^2
    sum = sum + numerator/denominator
  }
  return(sum)
}

# l''(theta) where l(theta) is loglikelihood function
cauchy_ldprime<-function(theta,x){
  n=length(x)
  sum=0
  for(i in 1:n){
    numerator = 2*((x[i]-theta)^2-1)
    denominator = (1+(x[i]-theta)^2)^2
    sum = sum + numerator/denominator
  }
  return(sum)
}

# Function to find mle using Newton's method
cauchy_mle<-function(x, tol){
  # Use initial value as median since location parameter of cauchy distribution is population median
  theta.current=median(x)
  iter=0
  while(TRUE){
    theta.new = theta.current - cauchy_lprime(theta.current, x) / cauchy_ldprime(theta.current, x)
    if(abs(cauchy_lprime(theta.new,x))<tol ) break
    theta.current = theta.new
    iter=iter+1
    # Print a message if the algorithm converges too slow
```

```r
    if(iter>1e+5) {
      print('too slow for iteration to converge')
      break
    }
  }
  return(theta.new)
}
```

Codes for simulating 10000 cauchy samples of size 20 and making lists of MLE and Observed Information bound.

```r
# We will generate 10000 datasets.
N=10000

# List of MLEs and observed Information bounds will be stored in X
X=matrix(0, ncol=2, nrow=N)
colnames(X)<-c('MLE', 'Obs.Info.Bound')

# sample size
n=20

#error tolerance
tol=1e-6

set.seed(100)


tic("R - Newton's method")
for(i in 1:N){
  # Generate sample of size n from cauchy(0,1)
  x=rcauchy(n, location=0, scale=1)
  mle = cauchy_mle(x, tol)
  # Too large value of theta may be a wrong answer
  if(abs(mle)>1e+03) {
    print("Abnormal value of mle is yielded")
    # Draw another sample and find mle again
    while(abs(mle)>1e+03){
      x=rcauchy(n, location=0, scale=1)
      mle=cauchy_mle(x, tol)
    }
  }
  # Calculate observed information bound
  obs_info_bound = -1/ cauchy_ldprime(mle, x)
  X[i,]=c(mle, obs_info_bound)
}
```

```
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "too slow for iteration to converge"
```

```
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
## [1] "too slow for iteration to converge"
## [1] "too slow for iteration to converge"
## [1] "Abnormal value of mle is yielded"
## [1] "Abnormal value of mle is yielded"
```

```
toc()
```

```
## R - Newton's method: 4.284 sec elapsed
```

```
X=as.data.frame(X)
# Sorting the list by the order statistics of observed information bound
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##                MLE Obs.Info.Bound
## 6155 -31.97291766    -3.33141318
## 5206  15.33740838    -1.81613613
## 8575  -6.82551482    -0.96347451
## 2856  -0.23648680     0.03724461
## 6964  -0.08596687     0.04185940
## 6361   0.38495939     0.04203593
## 5730  -0.22412033     0.04216019
## 3055   0.20981631     0.04228430
## 5542  -0.03992384     0.04234671
## 8977   0.24383388     0.04259172
```

Codes for finding MLE using Dekker's method and simulating the same thing.

```
# To use a function `swap`
library(seqinr)

cauchy_mle<-function(x, tol){
  # Start an algorithm with initial interval [a,b]
  # Here [a,b] is chosen as [-k, k] where k=|med(X_i)|+3
  # For bisection method to begin, l'(a)l'(b) < 0 should be satisfied.
  a = -(abs(median(x))+3)
  theta.past = a
  theta.current=abs(median(x))+3
  iter=0
  while(cauchy_lprime(theta.past, x)*cauchy_lprime(theta.current, x ) > 0){
    # If bisection l'(a)l'(b) < 0 is not satisfied then
    # give small fluctuation to a and b to attain the condition
    warning("the initial values does not saitsfy starting condition
    of bisection method. Shifting it a little...")
    theta.past = theta.past + rnorm(1)
    theta.current =  theta.current + rnorm(1)
    a=theta.past
    # If this procedure takes too much iteration, return NA
    iter=iter+1
    if(iter>1e+3) {
      print('initial value shifing is too complicated')
      return(NA)
```

```
      break
    }
  }
  iter=0
  while(TRUE){
    # Calculate the ratio which is used instead of l''(theta)
    # This is a difference between Newton's method and secant method
    ratio = (cauchy_lprime(theta.current, x)-cauchy_lprime(theta.past, x)) / (theta.current-theta.past)

    # For current bisection interval [a,b], calculate middle point m
    middle=(a + theta.current)/2

    # Propose a updated theta value , which is mainly done by a secant method
    # If two previous theta value are the same, then secant method cannot be used
    # so that middle point is proposed as new theta.
    proposal = ifelse(cauchy_lprime(theta.current, x)-cauchy_lprime(theta.past, x) !=0 ,
                      theta.current-cauchy_lprime(theta.current, x) / ratio, middle)

    # Determine an updated theta. Proposed theta becomes updated theta if it lies between m and b
    # Other wise, middle point becomes an updated theta
    if(middle<theta.current){
      theta.new=ifelse(proposal<=theta.current & proposal>=middle, proposal, middle)
    }
    else{
      theta.new=ifelse(proposal<=middle & proposal>=theta.current, proposal, middle)
    }

    # End the iteration if " |l'(theta)| < error tolerance "  is attained.
    if(abs(cauchy_lprime(theta.new,x))<tol ) break

    # Setting a bisection interval [a,b] for the next step
    if(cauchy_lprime(theta.new, x)*cauchy_lprime(a, x) > 0) a=theta.current
    if(abs(cauchy_lprime(theta.new,x)) > abs(cauchy_lprime(a, x)) ) swap(a, theta.new)

    theta.past=theta.current
    theta.current=theta.new

    # Print a message if the algorithm converges too slow
    iter=iter+1
    if(iter>1e+5) {
      print('too slow for iteration to converge')
      break
    }

  }
  return(theta.new)
}

# We will generate 10000 datasets.
N=10000

# List of MLEs and observed Information bounds will be stored in X
X=matrix(0, ncol=2, nrow=N)
```

```r
colnames(X)<-c('MLE', 'Obs.Info.Bound')

# sample size
n=20

#error tolerance
tol=1e-6

set.seed(100)

tic("R - Dekker's method with sample size 20")

for(i in 1:N){
  # Generate sample of size n from cauchy(0,1)
  x=rcauchy(n, location=0, scale=1)
  mle = cauchy_mle(x, tol)
  # Too large value of theta may be a wrong answer
  if(abs(mle)>1e+03) {
    print("Abnormal value of mle is yielded")
    # Draw another sample and find mle again
    while(abs(mle)>1e+03){
      x=rcauchy(n, location=0, scale=1)
      mle=cauchy_mle(x, tol)
    }
  }
  # Calculate observed information bound
  obs_info_bound = -1/ cauchy_ldprime(mle, x)
  X[i,]=c(mle, obs_info_bound)
}
```

```
## Warning in cauchy_mle(x, tol): the initial values does not saitsfy starting condition
##     of bisection method. Shifting it a little...

## Warning in cauchy_mle(x, tol): the initial values does not saitsfy starting condition
##     of bisection method. Shifting it a little...

## Warning in cauchy_mle(x, tol): the initial values does not saitsfy starting condition
##     of bisection method. Shifting it a little...
```

```r
toc()
```

```
## R - Dekker's method with sample size 20: 3.242 sec elapsed
```

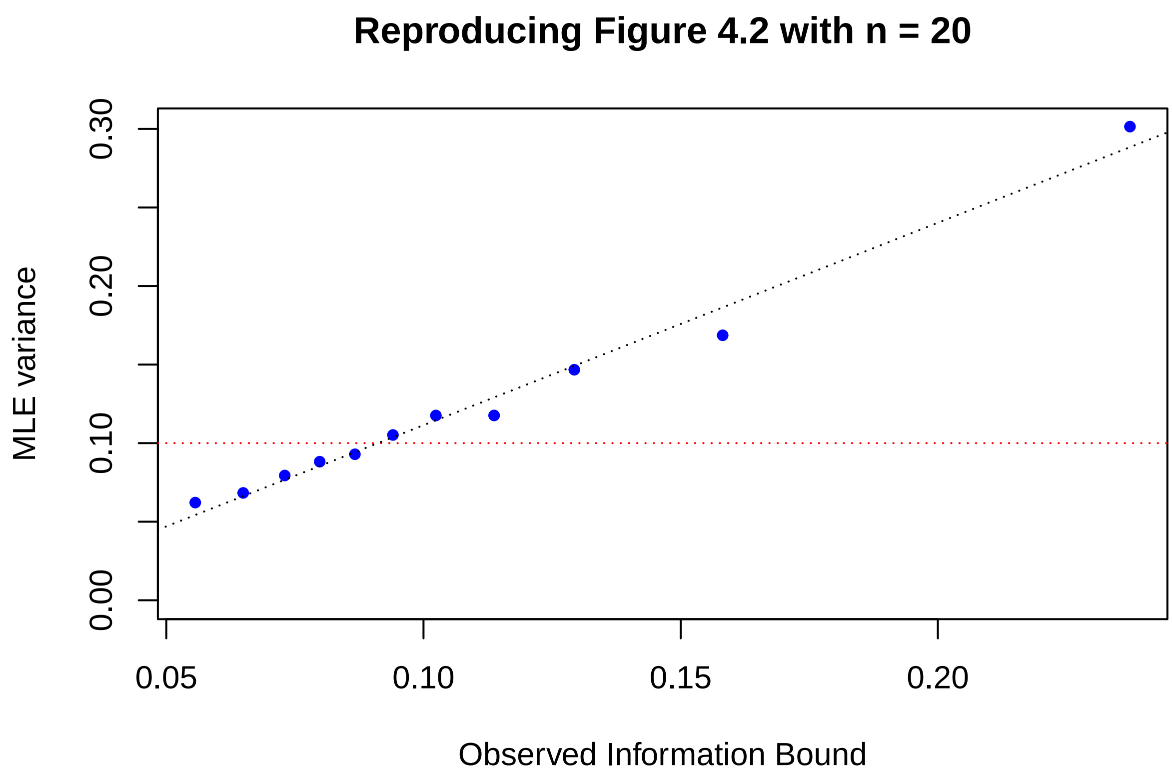
```r
X=as.data.frame(X)
# Sorting the list by the order statistics of observed information bound
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##                MLE Obs.Info.Bound
## 8568  0.152569546     0.03937138
## 6490 -0.244318638     0.04103515
## 8520  0.006857701     0.04189009
## 3908  0.243447185     0.04359162
## 2124  0.383324723     0.04378321
## 2221  0.184737462     0.04431370
```

```
## 8465  0.199806715     0.04458801
## 3624  0.313135109     0.04483765
## 7976  0.601368498     0.04499235
## 2507  0.294786221     0.04532065
```

Codes for producing the graph similar as Figure 4.2

```
DrawFigure<-function(X, n){
  percent=c(0,5,15,25,35,45,55,65,75,85,95,100)
  N=nrow(X)
  last.indices=N*percent/100
  var_theta=rep(0, 11)
  med_infobound=rep(0,11)
  for(i in 2:12){
    indices=(last.indices[i-1]+1):last.indices[i]
    thetas=X[indices, 1]
    infobounds=X[indices,2]
    var_theta[i-1]=var(thetas)
    med_infobound[i-1]=median(infobounds)
  }
  S=as.data.frame(cbind(var_theta, med_infobound))

  # Recreate Figure 4.2
  plot(S$med_infobound, S$var_theta, col='blue', pch=20,
       xlab='Observed Information Bound', ylab='MLE variance',
       main=paste0('Reproducing Figure 4.2 with n = ',as.character(n)),
       ylim=c(0,round(max(S$var_theta),3))
       )

  # CRLB(theta) which does not depend on the data
  unconditional.variance= 1/ (1/2 * n)
  abline(h=unconditional.variance, lty='dotted', col='red')

  # linear regression fit of Var(theta) ~ Med(Obs.Info.Bound)
  fit<-lm(var_theta~med_infobound, data=S)
  abline(fit$coefficients, lty='dotted')
}

DrawFigure(X, n)
```



Now, repeat the same process with n=100

```
# List of MLEs and observed Information bounds will be stored in X
X=matrix(0, ncol=2, nrow=N)
colnames(X)<-c('MLE', 'Obs.Info.Bound')

# sample size
n=100

#error tolerance
tol=1e-6

set.seed(100)

tic("R - Dekker's method with sample size 100")

for(i in 1:N){
  # Generate sample of size n from cauchy(0,1)
  x=rcauchy(n, location=0, scale=1)
  mle = cauchy_mle(x, tol)
  # Too large value of theta may be a wrong answer
  if(abs(mle)>1e+03) {
    print("Abnormal value of mle is yielded")
    # Draw another sample and find mle again
    while(abs(mle)>1e+03){
      x=rcauchy(n, location=0, scale=1)
      mle=cauchy_mle(x, tol)
    }
  }
  # Calculate observed information bound
```

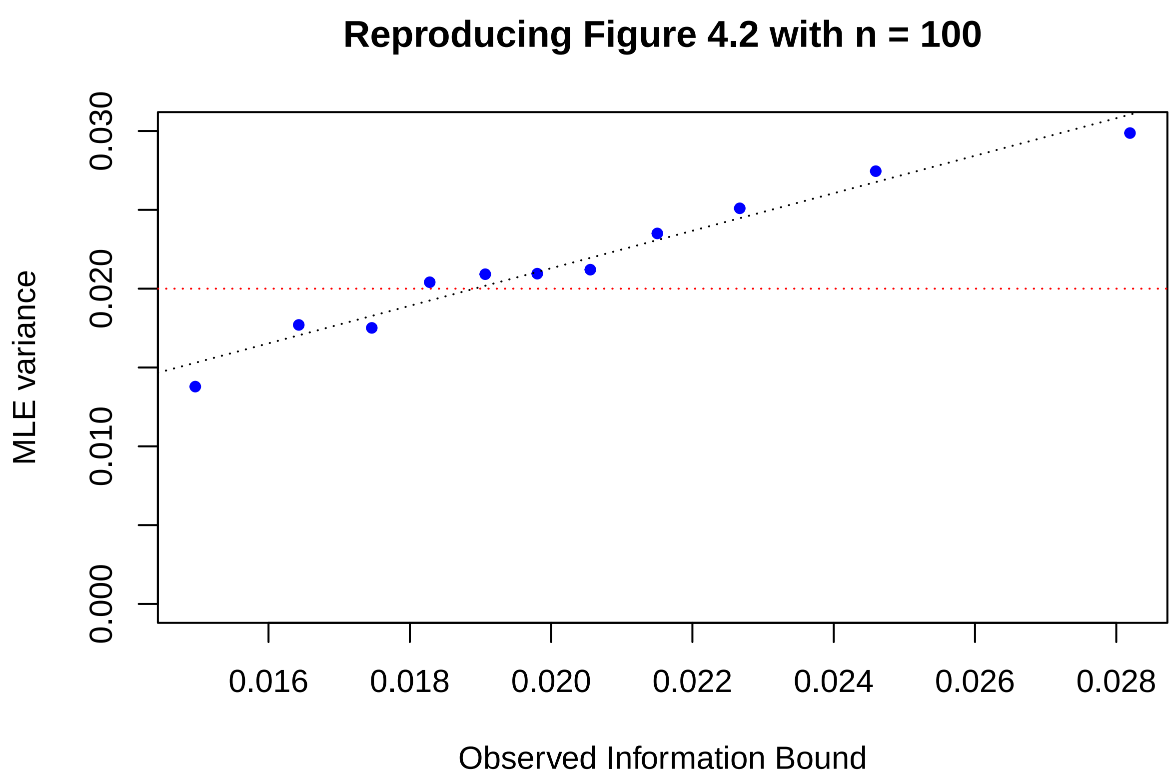
```
  obs_info_bound = -1/ cauchy_ldprime(mle, x)
  X[i,]=c(mle, obs_info_bound)
}

toc()
```

```
## R - Dekker's method with sample size 100: 7.563 sec elapsed
```

```
X=as.data.frame(X)
# Sorting the list by the order statistics of observed information bound
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##                MLE Obs.Info.Bound
## 8269  0.103122404     0.01170760
## 3807  0.047390131     0.01227333
## 2657  0.147277084     0.01228737
## 302   0.146465417     0.01244905
## 3671  0.012381201     0.01246538
## 5655  0.004597226     0.01251603
## 7423  0.045802189     0.01256538
## 8301  0.045404875     0.01270758
## 6780 -0.155045171     0.01287148
## 9111  0.090526844     0.01318322
```

```
DrawFigure(X, n)
```



The time elapsed to calculate MLE and reproduce the figure using R are given as
* R - Newton's method : 3.949 sec elapsed
* R - Dekker's method with sample size 20 : 3.098 sec elapsed
* R - Dekker's method with sample size 100 : 7.484 sec elapsed

Alternative way which uses Rcpp

```
remove(list=ls())

library(Rcpp)
library(RcppArmadillo)

sourceCpp('./rcpp_mle.cpp')
```

Now we can use cpp version of functions defined above. We expect the calculation will be done much faster.

```
set.seed(100)

N=10000
n=20

tic("Rcpp - Newton's method")
X=MLEwithCRLB(N,n)
```

```
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Too slow for iteration to convege
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
## Too slow for iteration to convege
## Too slow for iteration to convege
## Abnormal value of MLE is yielded
## Abnormal value of MLE is yielded
```

```
toc()
```

```
## Rcpp - Newton's method: 0.029 sec elapsed
```

```
X=as.data.frame(X)
colnames(X)<-c('MLE', 'Obs.Info.Bound')
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##                MLE Obs.Info.Bound
## 6155 -31.97291766    -3.33141318
## 5206  15.33740838    -1.81613613
## 8576  -6.82551482    -0.96347451
## 2856  -0.23648680     0.03724461
## 6964  -0.08596687     0.04185940
## 6361   0.38495939     0.04203593
## 5730  -0.22412033     0.04216019
## 3055   0.20981631     0.04228430
## 5542  -0.03992384     0.04234671
## 8978   0.24383388     0.04259172
```

```r
set.seed(100)

tic("Rcpp - Dekker's method with sample size 20")
X=MLEwithCRLB(N, n, "Dekker")
```

```
## The initial values do not satisfy starting condition of bisection method. Shifting it a little...
## The initial values do not satisfy starting condition of bisection method. Shifting it a little...
```

```r
toc()
```

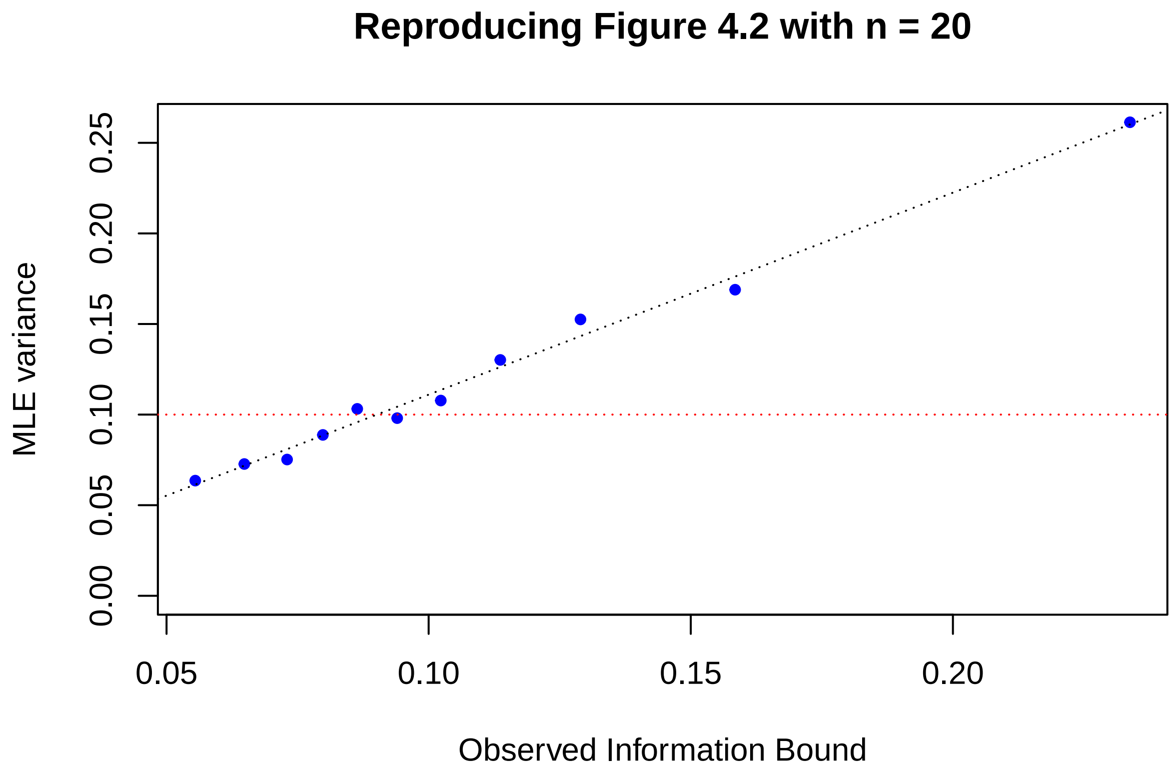
```
## Rcpp - Dekker's method with sample size 20: 0.102 sec elapsed
```

```r
X=as.data.frame(X)
colnames(X)<-c('MLE', 'Obs.Info.Bound')
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##              MLE Obs.Info.Bound
## 2507  0.38366352     0.03715269
## 8568  0.03502611     0.04112196
## 5679  0.27129790     0.04216386
## 3057  0.15168847     0.04223548
## 8520  0.05562770     0.04296338
## 8718  0.03258876     0.04317644
## 7104  0.49361179     0.04352892
## 5697 -0.05020748     0.04369098
## 2124  0.38332472     0.04378321
## 6484  0.15516144     0.04394068
```

```r
DrawFigure<-function(X, n){
  percent=c(0,5,15,25,35,45,55,65,75,85,95,100)
  N=nrow(X)
  last.indices=N*percent/100
  var_theta=rep(0, 11)
  med_infobound=rep(0,11)
  for(i in 2:12){
    indices=(last.indices[i-1]+1):last.indices[i]
    thetas=X[indices, 1]
    infobounds=X[indices,2]
    var_theta[i-1]=var(thetas)
    med_infobound[i-1]=median(infobounds)
  }
  S=as.data.frame(cbind(var_theta, med_infobound))

  # Recreate Figure 4.2
  plot(S$med_infobound, S$var_theta, col='blue', pch=20,
       xlab='Observed Information Bound', ylab='MLE variance',
       main=paste0('Reproducing Figure 4.2 with n = ',as.character(n)),
       ylim=c(0,round(max(S$var_theta),3))
       )

  # CRLB(theta) which does not depend on the data
  unconditional.variance= 1/ (1/2 * n)
  abline(h=unconditional.variance, lty='dotted', col='red')

  # linear regression fit of Var(theta) ~ Med(Obs.Info.Bound)
```

```
  fit<-lm(var_theta~med_infobound, data=S)
  abline(fit$coefficients, lty='dotted')
}

DrawFigure(X, n)
```




```
n=100

set.seed(100)

tic("Rcpp - Dekker's method with sample size 100")
X=MLEwithCRLB(N, n, "Dekker")
toc()
```

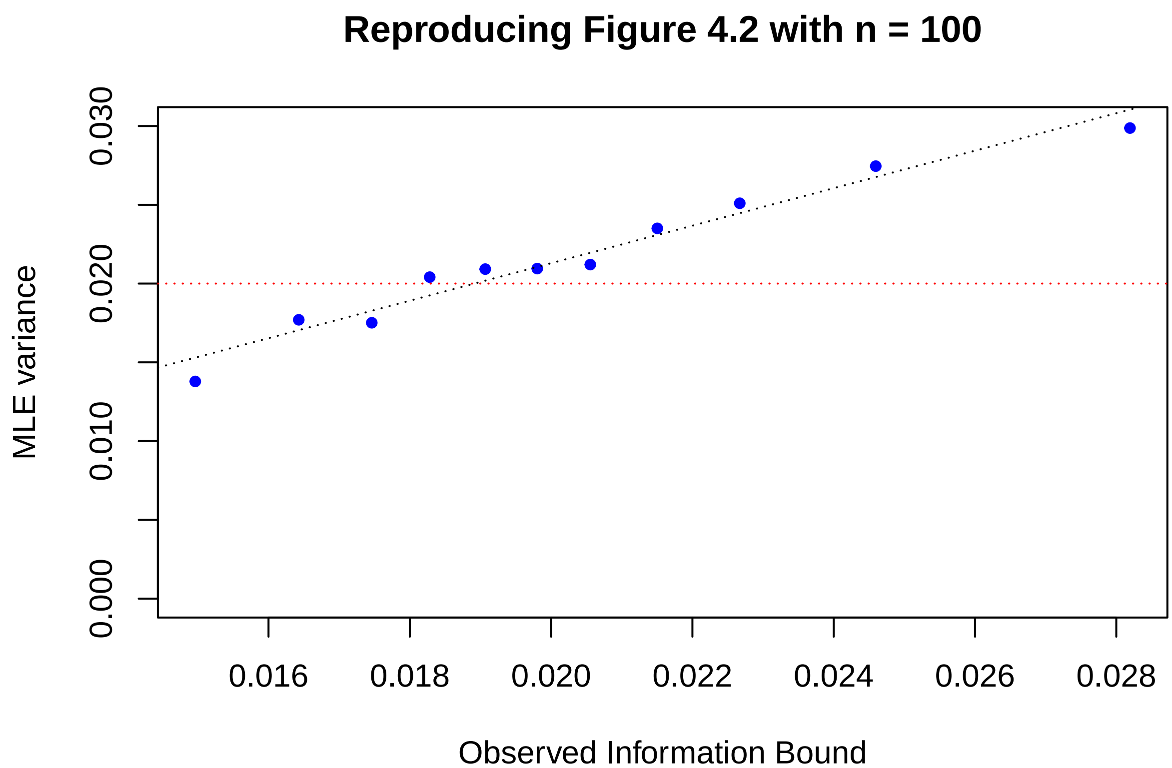
```
## Rcpp - Dekker's method with sample size 100: 0.243 sec elapsed
```

```
X=as.data.frame(X)
colnames(X)<-c('MLE', 'Obs.Info.Bound')
X=X[order(X$Obs.Info.Bound),]
head(X, 10)
```

```
##                MLE Obs.Info.Bound
## 8269  0.103122404     0.01170760
## 3807  0.047390131     0.01227333
## 2657  0.147277084     0.01228737
## 302   0.146465417     0.01244905
## 3671  0.012381201     0.01246538
## 5655  0.004597226     0.01251603
## 7423  0.045802189     0.01256538
## 8301  0.045404875     0.01270758
```

```
## 6780 -0.155045171     0.01287148
## 9111  0.090526844     0.01318322
```

```
DrawFigure(X, n)
```



Recall that time elapsed to calculate MLE and reproduce the figure using R are given as
* R - Newton's method : 3.949 sec elapsed
* R - Dekker's method with sample size 20 : 3.098 sec elapsed
* R - Dekker's method with sample size 100 : 7.484 sec elapsed

Here, Rcpp has much better performance to do the same task.
* Rcpp - Newton's method : 0.046 sec elapsed
* Rcpp - Dekker's method with sample size 20 : 0.107 sec elapsed
* Rcpp - Dekker's method with sample size 100 : 0.232 sec elapsed